Optimizing Loop Partitioning in TVM

Wuwei Lin*
wuweil@andrew.cmu.edu

Yingjing Lu*
yingjinl@andrew.cmu.edu

1 Major Changes

No major changes.

Minor change: We get rid of the testing on accelerator architecture and FPGA in our 125% goal due to TVM's RPC configuration issue. Another reason, as we will mention in the next section, due to TVM's lack of runtime exposition, program performance is hard to be measured through perf tools such as Pin-Tool for performance cost measures. Thus we implement and benchmark primarily on x86 architecture.

2 What You Have Accomplished So Far

According to our proposal, we want to implement a use case for our loop partition optimization first. So we used TVM to implement a GEMM on x86 CPU. We plan to use loop partition to support variable batch size. The current issue with variable batch size is that after loop tiling introduced nested conditional statements. We would like to optimize the GEMM with microkernels. For examples, we have found some existing optimized microkernels like 4xkx24 GEMM directly written in assembly with AVX instructions. We believe replacing the ordinary inner loop bodies with such microkernels can improve the performance. However, the nested conditional statements make such optimization impossible. We are still investigating the loop partitioning pass in order to find a way to eliminate such conditions. The idea is to find a set of iteration bounds that the conditions always hold true so that we can split the loop into multiple ones, some of which do not have conditional statements so that we can utilize the microkernels. We will continue with this part in the next step.

For the cost modeling part, what we are aiming for measuring optimization through two major metrics according to our proposal: the high level runtime, and the lower level data reuse. We were able to establish a framework that can benchmark TVM build programs through TVM's API for runtime and throughput. In trying to dive deeper searching a way to extract cache and data reuse results to estimate data reuse and cache utilization, we encountered some problems. We tried to attach the TVM optimized program with a customized tool that we build in order to accumulate the cache performance through APIs in PIN-Tool, however, we found tricky to isolate any particular executable that can be easily targeted by the PINN-Tool. To make our implementation more robust and expandable, we plan to accumulate data reuse within loops through further re-factoring the TVM's IR visitor to inspect the potential data reuse directly through the IR. We will be verifying that the implementation is correct through testing in our next step. (We have some testing toy results for benchmarking runtime for loop based matrix multiplication which can be found at the homepage of our website).

3 Meeting Your Milestone

Our plan by milestone is to nearly achieve our 75% goal which we will have a basic loop analysis plus a cost model. We think that we approximately meet that goal in enumerating some example workload and finding way to extract cost data. We suspect that for benchmarking data reuse, due to the restriction of TVM stack that only exposes the IR, we may not able to accurately estimate the data reuse in a multi-threaded environment due to complicated threading and scheduling. We may

Preprint. Under review.

choose to only benchmark the optimization for reuse only for single threaded case or we may not use that for our final evaluation.

4 Surprises

We don't observe anything surprising.

5 Revised Schedule

As we are aligned with our schedule, we choose to stick with the original schedule which is listed below:

Apr 13 - 19 **Milestone**
Expect to nearly achieve the 75% goal **(Lu, Lin)**
Expecting to achieve a basic working version of the loop analysis module that can export the loop execution procedure.
Expect the cost model to export the cost for the given procedure analysis.

Apr 20 - 26
Implement the cost exploration procedure. **(Lu)**
Implement the loop procedure optimization to leverage microkernels. **(Lin)**

Apr 27 - May 3
Extend the analysis to arbitrary input and potentially propose an extension on the TVM's tuning stack for tuning according to micro kernels **(Lu, Lin)**
Testing and summarize the results, write out the report. **(Lu, Lin)**

6 Resources Needed

Some example loop components in the deep learning tensor computation workloads. We have already found these examples.

TVM build and testing environment such as consumer x86 CPU, consumer grade GPU and enterprise GPU, FPGA, ARM processors. Some of which could be accessed through AWS.

7 Project Website

`https://yingjinglu.github.io/pages/compiler-proj.html`

8 Project Description

To achieve good performance by AOT compilation using TVM, the input shape need to be known at compile time. This is impractical because we often want to change the batch size or the shape in runtime. Allowing variables in the input shape relaxes this constraint. However, naive approach may often result in excessive amount of conditionals to enumerate all possibilities of the variable, and it is difficult to achieve good performance.

We would like to enhance loop partition algorithm to tackle dynamic shape. Loop partition is helpful for dynamic shape if it can partition the shape into suitable tiles so that we can utilize external highly-optimized microkernels, which often has strict requirements for the input shape. We would like to explore heuristics for loop partition, such as utilizing performance of microkernels for different shapes, to partition the loop in an efficient way. In addition, partitioning the loop may also introduce more possibilities for cache reuse optimization, pipelining across different loop segments and more parallelization.

* Denotes equal contribution

In this project, we intend to extend the existing TVM stack with a loop partitioning optimization module. The basic functionality of the system takes in a tensor computation loop with some given heuristics, and to partition the loop so that the loop runs faster / consumes less energy than the naive serial execution. Potential field of optimization includes parallel loop execution, tiling, scheduling, cache optimization and pipeline scheduling. A cost and exploration model for auto tuning performance. The optimization could be taken further if the target hardware environment is given where the algorithm can gear directly towards the configuration for full optimization.

9 Objectives

75% Objective: Construct appropriate test cases with dynamic batch size and Benchmark existing loop partition module in TVM, and analyze the bottleneck of the existing loop partition module. Extend the current loop partition module to support different loop partition strategies that leverage the information from the cost module. A cost module that takes in the procedure and analyze the number of execution cycle, cache, memory utilization and energy/time consumption given a general hardware environment such as a CPU environment or a x86 CPU environment.

100% Objective: Based on 75% goal. A loop partition analysis module that takes in the loop and conditionals in the loop, with variable input shape size, achieves better data reuse, parallelization than the naive loop pass for general x86 CPU and, potentially, GPU environment. The pass should be able to partially automatically explore the optimization space given the feedback cost from the cost module.

125% Objective: Based on 100% goal, A loop partition analysis module that can optimize loop partitioning given external microkernels. Combine the loop partition pass and the scheduling part of the TVM IR such that given the information of loop partition and the cost module, we can automatically tensorize the loop body to utilize external microkernels. A documented API as a potential extension of current TVM tuning stack.

10 Literature Search

The TVM paper that introduces the architecture, cost modules and existing features Chen et al. [2018a].

FlexTensor, an automatic scheduling exploration module for tensor computation based on TVM Zheng et al. [2020].

Learning to optimize tensor program: introduces a ML based tensor computation cost exploration model that utilizes boosting tree and treeGRU to efficiently explore the optimization search space without manual tuning Chen et al. [2018b].

11 Resources Needed

References

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, Carlsbad, CA, October 2018a. USENIX Association. ISBN 978-1-939133-08-3. URL `https://www.usenix.org/conference/osdi18/presentation/chen`.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems*, pages 3389–3400, 2018b.

Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 859–873, 2020.