VTensor: Using Virtual Tensors to Build a Layout-Oblivious AI Programming Framework

Feng Yu
SKLCA, ICT, CAS & School of Computer Science and Technology, UCAS
yufeng@ict.ac.cn

Jiacheng Zhao
SKLCA, ICT, CAS & School of Computer Science and Technology, UCAS
zhaojiacheng@ict.ac.cn

Huimin Cui
SKLCA, ICT, CAS & School of Computer Science and Technology, UCAS
cuihm@ict.ac.cn

Xiaobing Feng
SKLCA, ICT, CAS & School of Computer Science and Technology, UCAS
fxb@ict.ac.cn

Jingling Xue
School of Computer Science and Engineering, UNSW Sydney
jingling@cse.unsw.edu.au

ABSTRACT

Tensors are a popular programming interface for developing AI algorithms. Representative AI programming frameworks require developers to be always aware of tensor layouts, thereby reducing their productivity in integrating an existing operation with a new library and/or writing a new operation. We propose VTensor, a layout-oblivious virtual tensor programming interface, together with a global layout inference mechanism to resolve the layout required by virtual tensors. Furthermore, VTensor leverages a layout-oriented optimization to globally minimize the number of layout conversion operations, together with a straggler-ware scheduling algorithm and a pool-based memory allocation scheme to globally allocate resources. VTensor yields significant speedup and LOC (Lines of Codes) reduction compared to TensorFlow.

1 INTRODUCTION

As AI technologies are quickly transforming almost every sphere of our lives, it is imperative to provide an AI programming framework that is easy to use and deploy across a variety of platforms. Ideally, with tensors, developers can easily refer to the logical dimensions of a data structure, without having to be concerned with the underlying physical layout.

However, in existing programming frameworks, developers are still required to be aware of the layout of a tensor all the time. Take TensorFlow [1] for example, developers are required to support different layouts when adding a new operation or porting an operation to a new platform. They must keep in mind which layout is supported by which library routine, and manually insert an appropriate layout transformation when necessary. Figure 1 shows a skeleton of the *avgpool* operator in TensorFlow, where the layout-dependent code segments are highlighted in red.

Researchers have noticed that tensor layouts are a performance-critical issue, and proposed a number of approaches to determine the optimal solutions [2]. However, these approaches still adopt the traditional layout-aware programming interfaces.

PACT '20, October 3–7, 2020, Virtual Event, GA, USA

ACM ISBN 978-1-4503-8075-1/20/10.
https://doi.org/10.1145/3410463.3414664

```
class MklAvgPoolingOp {
  void Compute(OpKernelContext* ctx) {
    Tensor input = ctx->input(0);
    MklDnnShape shape = GetMklShape(ctx, 1);

    int batch_size = shape.IsMklTensor() ? shape->GetDimension('N')
                        : GetTensorDim(input, tf_format, 'N');
    …
    memory::dims src_dims = shape.IsMklTensor() ? shape.GetSizesAsMklDims()
                        : TFShapeToMklDnnDimsInNCHW(input, tf_format);
    memory::desc input_md = shape.IsMklTensor() ? shape. GetMklLayout()
                        : memory::desc(src_dims, tf_format, …);
    …
    MklPoolingParams fwdParams(src_dims, …);
    MklPoolingFwdPrimitive* pooling_fwd =
              MklPoolingFwdPrimitiveFactory::Get(fwdParams);
    out_mkl_shape.SetMklLayout(&dst_pd);
    …
    AllocateOutputTensor(0, out_tf_shape, output_tensor);
    AllocateMklShapeTensor(1, out_mkl_shape, shape_tensor);
    …
    if (input_md.format != pooling_fwd->GetSrcMemoryFormt()) {
      …
      input_md.CheckReorderToOpMem(required_layout);
    }
    …
    pooling_fwd->Execute(src_data, dst_data);
  }
};
```

Figure 1: Layout-aware programming for the *avgpool* operator in TensorFlow (with the layout-dependent lines shown in red)

TensorFlow provides an ad-hoc mechanism for developers to maintain the layout information. This has three disadvantages:

- *Poor Maintainability.* As shown in Figure 1, when new operators or hardware platforms are introduced, developers have to maintain these layout-dependent code segments scattered throughout the framework.
- *Un-optimized Layout Transformations.* Under TensorFlow's ad-hoc layout processing mechanism, developers must explicitly insert layout transformation operations without globally considering the overall dataflow graph, thereby introducing un-optimized layout transformations.
- *Fragmented Memory Allocation.* Developers tend to introduce some temporary tensors during the layout transformations, with the corresponding memory when needed, thereby causing a large number of fragmented memory allocations.

2 VTENSOR FRAMEWORK

As shown in Figure 2, the VTensor framework consists of two components: a layout-oblivious programming interface and the VTensor runtime. The VTensor programming interface provides

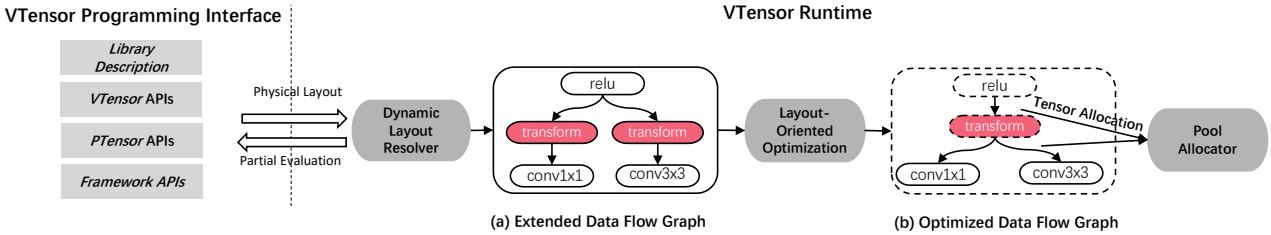

Figure 2: The VTensor framework.

four categories of programming interfaces to define an operation, describe a library, and illustrate how to invoke a library routine.

- *VTensor API* enables developers to implement an operator by writing the "Compute" function in order to access the virtual tensors.
- *PTensor API* allows developers to declare the corresponding physical tensors for virtual tensors.
- *Library Description* is required for each library to describe the layout mapping, the layout transformation handler, and some guidelines for selecting a layout.
- *Framework APIs* are provided for developers to register some handlers with the VTensor framework.

To resolve the layout for virtual tensors, we propose a "Dynamic Layout Resolver", which partially evaluates the dataflow graph to determine the physical layout for each node in the graph. Then it determines the locations required for layout transformations, and inserts the corresponding transformation operations as individual nodes into the dataflow graph, called the *extended dataflow graph*. Afterwards, we apply a pattern-based graph optimization to the extended dataflow graph to globally minimize the number of layout transformation nodes. Meanwhile, for efficient execution of the extended dataflow graph, VTensor uses offline profiling and a straggler-aware scheduling algorithm to globally allocate hardware resources to the computation nodes.

Finally, the VTensor resolver makes it possible to determine the number of temporary tensors in advance, so that we can use a centralized allocation scheme for all input/output tensors of each operation in order to avoid frequent fragmented memory allocation.

3 EVALUATION

To evaluate VTensor, we have implemented it in TensorFlow 1.14 and measured the inference latency time on the Intel Xeon E7-4820. For performance evaluation, we follow [3] to set TensorFlow's environment variables and parameters.

Compared with TensorFlow, as shown in Figure 3, VTensor has achieved a significant reduction in code sizes, from 9.95% to 70.97%, with an average of 46.6%. Since the layout-dependent codes in TensorFlow are extremely ad-hoc, developers have to explicitly write different library wrappers for layout selection and transformation. Furthermore, developers are required to write the *Compute* function for each library. In comparison, VTensor automatically inserts appropriate layout transformations when necessary, and shares the same *Compute* function for all library.

In comparison, VTensor achieves a performance improvement ranging from 10.82% to 47.96%, with an average of 27.5% compared

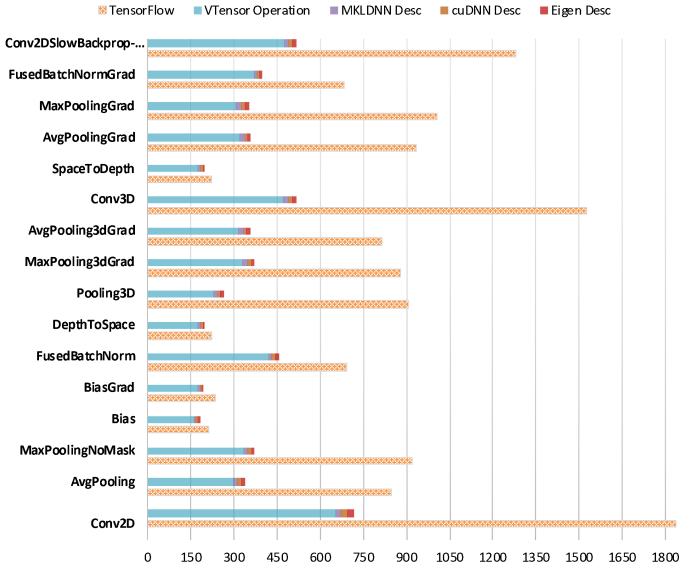

Figure 3: Comparison of LOC when writing an operator using VTensor/TensorFlow framework.

with TensorFlow. For networks with a large number of branches, VTensor can leverage layout-oriented optimizations to reduce the number of layout transformations and the straggler-aware algorithm can find more opportunities to exploit inter-op parallelism.

ACKNOWLEDGMENTS

This work is supported in part by the National Key R&D Program of China (2017YFB0202901), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030101), the National Natural Science Foundation of China (61802368, 61521092, 61432016, 61432018,61332009, 61702485, and 61872043), CCF-Tencent Open Research Fund, and an Australian Research Council grants (DP170103956 and DP180104069).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Andrew Anderson and David Gregg. 2018. Optimal DNN primitive selection with partitioned boolean quadratic programming. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 340–351.

[3] Niranjan Hasabnis. 2018. Auto-tuning tensorflow threading model for CPU backend. In *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*. IEEE, 14–25.