Yu F, Zhao JC, Cui HM *et al*. VTensor: Using virtual tensors to build a layout-oblivious AI programming framework. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(5): 1074−1097 Sept. 2023. DOI: 10.1007/s11390-022-1457-6

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

Feng Yu (俞　峰), Jia-Cheng Zhao (赵家程), Hui-Min Cui (崔慧敏), Xiao-Bing Feng (冯晓兵), Jingling Xue (薛京灵)

Research Questions

Q1. Poor Maintainability

```
The total number of lines of code to implement the MklAvgPoolingOp is about 500.
class MklAvgPoolingOp {
  void Compute(OpKernelContext* ctx) {
    Tensor input = ctx->input(0);
    MklDnnShape shape;
    // Resolve the second input tensor to MklDnnShape and perform an
    // integrity check on the input tensor, which takes about 40 lines of code.
    ResolveAndCheckIntegrity(input, shape, ctx);

    // Extracts dimension information and attributes of AvgPool operator, which
    // takes up 190 lines.
    PoolParameters pool_params;
    ExtractPoolParameters(ctx, &pool_params, input, shape);

    // Infer output tensor shape, which takes up 30 lines.
    memory::dims output_dims_mkl_order;
    InferOutShape(pool_params, &output_dims_mkl_order);

    // The attributes and dimension information of operators are represented
    // by MKL-DNN data structure, which takes up 70 lines.
    memory::dims filter_dims, strides, padding_left, padding_right;
    PoolParamsToAttributes(&pool_params, &filter_dims, &strides,
                           &padding_left, &padding_right, is_pool2d);
    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, …);

    // Takes up 10 lines, Omit other parameters.
    MklPoolingParams fwdParams(src_dims, output_dims_mkl_order, filter_dims, …);
    MklPoolingFwdPrimitive* pooling_fwd =
                MklPoolingFwdPrimitiveFactory::Get(fwdParams);

    // allocate output tensor, which takes up 90 lines.
    AllocateOutputTensor(0, out_tf_shape, output_tensor);
    AllocateMklShapeTensor(1, out_mkl_shape, shape_tensor);

    // Perform data conversion operations, occupying 70 lines.
    if (input_md.format != pooling_fwd->GetSrcMemoryFormt()) {
      CheckReorderToOpMem(input_md, required_layout);
    }
    // execute pooling
    pooling_fwd->Execute(src_data, output_tensor);
  }
};
```

Fig.1. Layout-aware programming for AvgPool in TensorFlow (with the layout-dependent lines shown in red).

Library have layout conventions, while applications have no restrictions on layout, so a lot of layout-related code needs to be developed.

Q2. Lost opportunity for layout optimization.

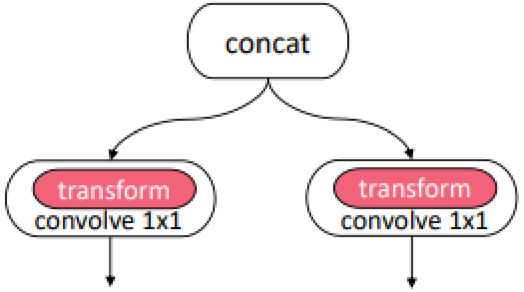
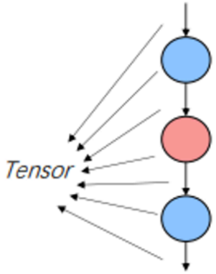


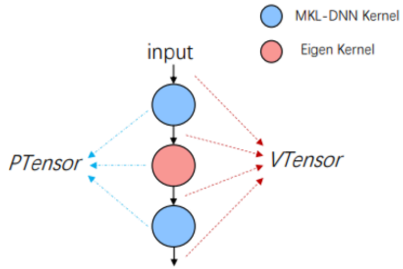
Fig.2. Layout conversion is done by operators, because the layout of the application can only be determined at runtime.

Lost layout optimization opportunities, such as redundant layout transition operations.。

Insights & Solution



Adopt the idea of polymorphism in the object-oriented model！！



Insights

```
The total number of lines of code to implement the MklAvgPoolingOp is about 500.
class MklAvgPoolingOp {
  void Compute(OpKernelContext* ctx) {
    Tensor input = ctx->input(0);
    MklDnnShape shape;
    // Resolve the second input tensor to MklDnnShape and perform an
    // integrity check on the input tensor, which takes about 40 lines of code.
    ResolveAndCheckIntegrity(input, shape, ctx);

    // Extracts dimension information and attributes of AvgPool operator, which
    // takes up 190 lines.
    PoolParameters pool_params;
    ExtractPoolParameters(ctx, &pool_params, input, shape);

    // Infer output tensor shape, which takes up 30 lines.
    memory::dims output_dims_mkl_order;
    InferOutShape(pool_params, &output_dims_mkl_order);

    // The attributes and dimension information of operators are represented
    // by MKL-DNN data structure, which takes up 70 lines.
    memory::dims filter_dims, strides, padding_left, padding_right;
    PoolParamsToAttributes(&pool_params, &filter_dims, &strides,
                           &padding_left, &padding_right, is_pool2d);
    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, …);

    // Takes up 10 lines, Omit other parameters.
    MklPoolingParams fwdParams(src_dims, output_dims_mkl_order, filter_dims, …);
    MklPoolingFwdPrimitive* pooling_fwd =
            MklPoolingFwdPrimitiveFactory::Get(fwdParams);

    // allocate output tensor, which takes up 90 lines.
    AllocateOutputTensor(0, out_tf_shape, output_tensor);
    AllocateMklShapeTensor(1, out_mkl_shape, shape_tensor);

    // Perform data conversion operations, occupying 70 lines.
    if (input_md.format != pooling_fwd->GetSrcMemoryFormt()) {
      CheckReorderToOpMem(input_md, required_layout);
    }
    // execute pooling
    pooling_fwd->Execute(src_data, output_tensor);
  }
};
```

Fig.1. Layout-aware programming for AvgPool in TensorFlow (with the layout-dependent lines shown in red).

Virtual Tensor

The closer to the application, the developer is using the mathematical semantics of tensors.

Dynamic Resolver

Physical Tensor

The closer to the library, the developer is using the physical semantics of tensors.

```
The total number of lines of code for the AvgPoolOp class and the mkldnnAvgPoolInvoker
function is 140.
class AvgPoolOp {
  void Compute(OpKernelContext* ctx) {
    VTensor input = context->input(0);
    // Extracting parameters required by the AvgPool operator, which takes about 50 lines.
    VPoolParameters param;
    ExtractPoolParametersFromVTensor(ctx, &param, input);
    // Infer output tensor shape, which takes up 20 lines.
    InferOutShape(&param);
    // Allocate output tensor. The settings of other dimension names and sizes are omitted.
    VTensorOptions* vto(ctx, 0/*output index*/) ;
    VTensor* output_tensor = new VTensor({{'N', param.batch_size}, …}, vto);
    Dispatcher(ctx, {input}, {output_tensor},  &param);
  }
  void Dispatcher(OpKernelContext* ctx, vector<VTensor> input,
                                vector<VTensor*> output, Parameters* param) {
    ParametersAndDimensionCheck(ctx, input, output, param);
    auto invoker_dict = getDeviceInvokers(ctx->device());
    for (int priority = highest_priority; priority >= lowest_priority; priority--) {
      Invoker* invoker = invoker_dict.getInvoker(priority);
      if (invoker) { invoker(ctx, input, output, param); return;}
    }
  }
};
REGISTER(CPUDevice, "AvgPool", mkldnnAvgPoolInvoker)
void mkldnnAvgPoolInvoker(OpKernelContext* ctx, vector<VTensor> input,
                               vector<VTensor*> output, Parameters* param) {
  // Use VTensor to construct PTensor, the output PTensor is omitted here.
  PTensor* in_p = new VTensor(input[0]);
  LAYOUT p_layout = GetLibGuideLine("MKLDNN", in_p);
  in_p->require(p_layout);
  if (LibTagger("MKLDNN", {in_p}, {out_p}) return;
  MemoryDesc in_desc = in_p->getLibraryDesc("MKLDNN");
  MemoryDesc out_desc = out_p->getLibraryDesc("MKLDNN");
  // Converts information such as attributes and dimensions into data structures
  // acceptable to the MKL-DNN library, which takes up 40 lines.
  MklPoolingParams fwdParams;
  ConstructMklPoolParams(param, fwdParams, in_desc, out_desc);

  MklPoolingFwdPrimitive* pooling_fwd
                    = MklPoolingFwdPrimitiveFactory::Get(fwdParams);
  pooling_fwd->Execute(in_p->data(), out_p->data());
}
```

Fig.3. Layout-oblivious programming for AvgPool in VTensor (blue lines are auto-generated, orange lines are VTensor API calls).

Design and Experimentation

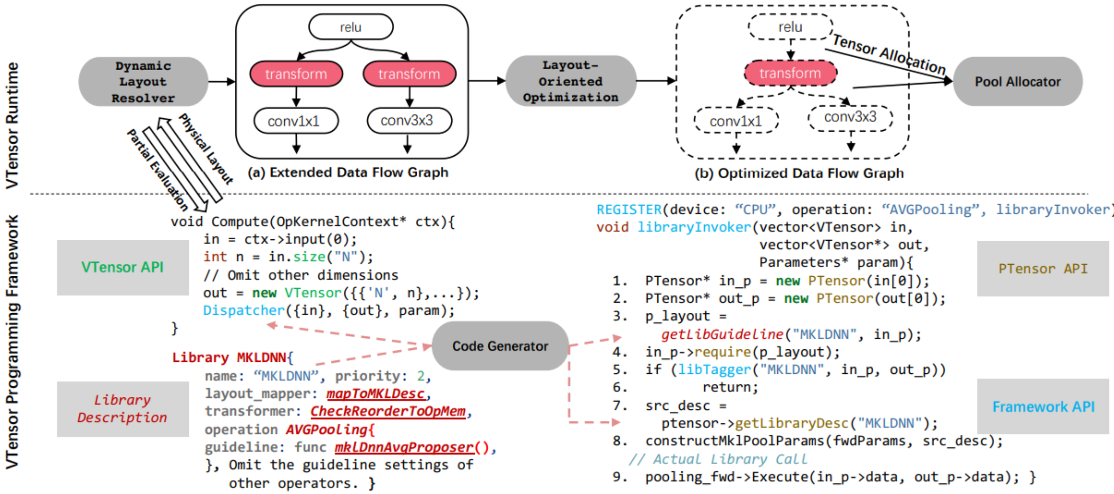


Fig.4. The VTensor framework overview.

Experimental results:

- VTensor can reduce the LOC of writing a new operation by 47.82% on average.
- VTensor improve the overall performance by 18.65%, on average.

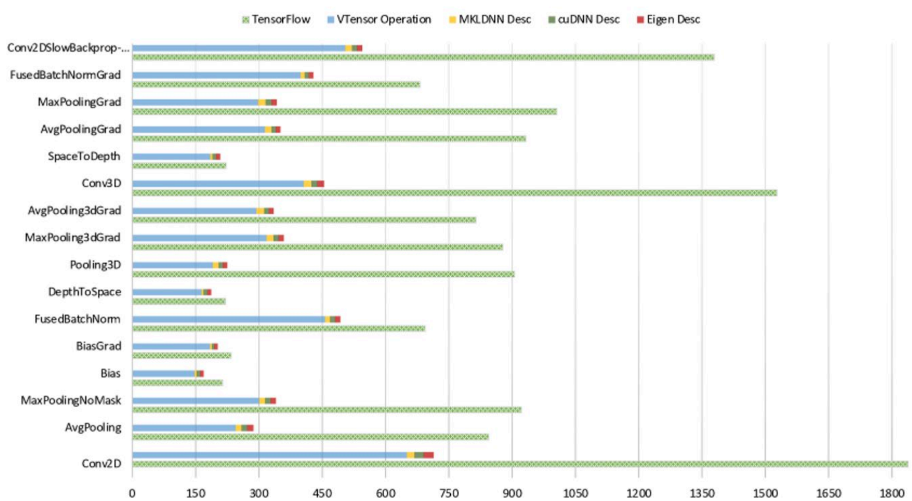


Fig.5. Comparison of LOC when writing an operator using VTensor/TensorFlow framework.

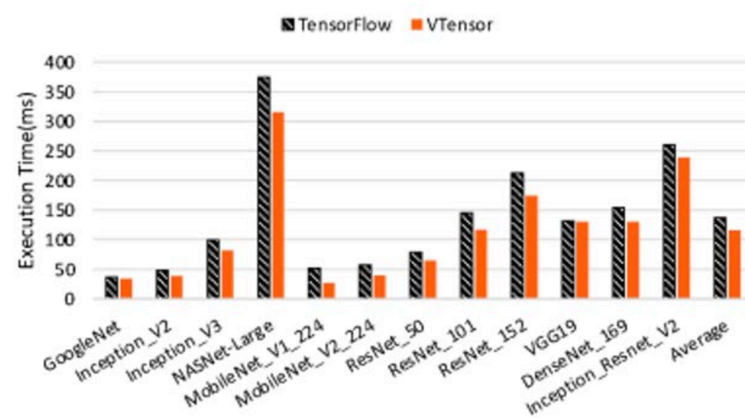


Fig.6. Inference latency of TensorFlow/VTensor on CPU platform.

Conclusions

- The library has a convention on the layout but the application has no restrictions, resulting in the following problems:
  - Poor Maintainability
  - Lost opportunity for layout optimization.
- Insight: The application layer uses the mathematical semantics of tensors, while the library uses the physical semantics of tensors.
  - Use the idea of polymorphism
- Contribution
  - Propose a new programming abstraction that decouples operator developers from tensor layouts.
  - Propose a new layout parsing mechanism for automatically mapping virtual tensors to physical tensors.
  - Taking advantage of the fact that VTensor layout resolution happens at runtime, uncovers new opportunities for layout optimization.

Conclusions

- Experimental results
  - VTensor can reduce the LOC of writing a new operation by 47.82% on average.
  - VTensor improve the overall performance by 18.65%, on average.
- Future work
  - Modify the layout optimization algorithm with the goal of minimizing layout transition time.
  - Abstracts the layout of sparse tensors and integrates the VTensor idea into the compiler as an intermediate representation.