TensorFlow acceleration for neural network inference using SYCL ecosystem

May 2018, Embedded Vision Summit

Andrew Richards
TensorFlow on SYCL & OpenCL

• This is open-source TensorFlow, ported to support SYCL
• You can build TensorFlow & run on any accelerator that supports the right OpenCL features
  • But you may need to adapt some operators to achieve performance
• This is the same source base as the CUDA version
• This work supports training & inference and tools like TensorBoard
• This is separate from projects that extract TensorFlow graphs and execute them on a different inference engine
Codeplay provides the standard tools for transforming AI prototypes into volume products

- Reduce power
- Reduce cost
- Standard programming models
- Safety qualification

Prototype to Product
Codeplay works in partnership to build industry standard solutions

Funding
Jez San: his company, Argonaut, created first videogames GPU (SuperFX)

Standards

Processors
ARM Mali
+ others unannounced
SYCL-TensorFlow

- Port of whole of TensorFlow to support SYCL and OpenCL
- Can take existing Python TensorFlow scripts and just run them as-is
- Training & Inference
- Profiling/tracing
- Custom operations, as they are added
- Open-source: can check out code from TensorFlow repository
- Python, C and C++ interfaces
- Continuous integration testing
- 37% of TensorFlow core operations, compared with CUDA's 54% and 24% with XLA
- (XLA falls back on non-XLA code for many ops)
TensorFlow/Eigen performance

Benchmarks taken from:
https://github.com/tensorflow/benchmarks
Getting TensorFlow using OpenCL/SYCL

You need to rebuild from source

• Checkout the TensorFlow source from github (currently Mali support is only in Codeplay’s github):
  • https://github.com/codeplaysoftware/tensorflow/tree/eigen_sycl
• Download the ComputeCpp for Arm package from Codeplay website:
  • http://www.codeplay.com/computecpp
• Download OpenCL implementation (with SPIR-V support) for your GPU or AI accelerator
• Compile and run:
  • Follow normal TensorFlow build options (using bazel)
  • Select OpenCL support during configuration
  • Need to provide the ComputeCpp directory
Mapping a neural network to SYCL

- **Convolution**
  - 11x11:96
  - ReLU
  - Normalization
  - Max Pooling

- **Convolution**
  - 5x5:256
  - Normalization
  - Max pooling

- **Convolution**
  - 3x3:384
  - ReLU

- **Convolution**
  - 3x3:384
  - ReLU

- **Convolution**
  - 3x3:256
  - ReLU

- **Fully connected**
  - Matrix multiply: 4096
  - ReLU

- **Fully connected**
  - Matrix multiply: 4096
  - ReLU

- **Fully connected**
  - Matrix multiply

- **Convolution**
  - RelU, Normalization, Max Pooling

- **Matrix multiply**
  - SYCL-BLAS

- **Fully connected**
  - Matrix multiply

- **Convolution hardware**
  - SYCL code
  - OpenCL

- **Matrix multiply hardware**
  - SYCL code
  - OpenCL

- **Fully connected**
  - SYCL code
  - OpenCL with kernel-fusion

- **Convolutions**
  - SYCL-DNN

- **Eigen**
  - SYCL code
TensorFlow on CUDA

- **Python Client**
- **C++ Client**
- **C API**
- **TensorFlow tensor Kernels (> 800 kernels)**
- **Matrix multiply**
- **Eigen Tensors**
- **Convolutions**
- **cuBLAS**
- **cuDNN**
- **CUDA**
- **Hand-coded assembly for NVIDIA GPUs**
- **NVIDIA GPUs**
TensorFlow on SYCL / OpenCL

Python Client

C++ Client

C API

TensorFlow tensor Kernels (> 800 kernels)

Matrix multiply

Convolutions

Eigen Tensors

SYCL-BLAS

SYCL-DNN

SYCL/ComputeCpp/triSYCL

OpenCL
## Optimizing TensorFlow

<table>
<thead>
<tr>
<th>SYCL-DNN</th>
<th>SYCL-BLAS</th>
<th>Eigen</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Highest impact on performance</td>
<td>• Currently, we actually use Eigen tensor-contract</td>
<td>• The generic algorithms must be adapted for each architecture:</td>
</tr>
<tr>
<td>• Algorithms need to be adapted to each new processor architecture</td>
<td>• The plan is to switch to SYCL-BLAS, making code-sharing easier</td>
<td>• Component-wise operations</td>
</tr>
<tr>
<td>• Or, alternatively, make use of hand-coded assembly</td>
<td>• Handles matrix multiplies</td>
<td>• Reduction</td>
</tr>
<tr>
<td>• Or, make use of convolution hardware</td>
<td>• Algorithms need to be adapted to each new processor architecture</td>
<td>• Partial reduction</td>
</tr>
<tr>
<td>• Some convolution operations make use of matrix multiplies (either SYCL-BLAS, or Eigen tensor contractions)</td>
<td>• Or, alternatively, make use of hand-coded assembly</td>
<td>• Shuffles and chipping</td>
</tr>
<tr>
<td></td>
<td>• Or, make use of matrix multiply hardware</td>
<td>• Tensor contractions</td>
</tr>
<tr>
<td></td>
<td>• Necessary to have performance for a very wide range of matrix shapes &amp; sizes</td>
<td>• All open-source</td>
</tr>
</tbody>
</table>

- Eigen
  - Source code is shared between CPU, CUDA and SYCL
  - Algorithms are written as C++ templates and can be configured per-device
SYCL-DNN: algorithms performance (inference)
SYCL-DNN: algorithms performance (training)
What’s next?

• Some further up-streaming required
  • Codeplay’s branch has the latest support
• Open-source and integrate SYCL-DNN
• Integrate SYCL-BLAS
• More accelerator processor support
• Setup more continuous integration testing
• It’s all open-source, open-standard, so easy to customize
Q&A