Accelerate Machine Learning Using TensorFlow and SYCL on OpenCL Devices

Mehdi Goli, Luke Iwanski, Andrew Richards

May 2017

Agenda

- Introduction to SYCL™, TensorFlow™ and Eigen
- Goals & Challenges
- Implementation Status
- Benchmarks
- Next Steps

Motivation

- Machine Learning is back!
  - More complex concepts can be applied
  - Deep calculation networks can be trained and executed faster
  - Thanks to heterogeneous platforms
- ML is widely used in many different areas
  - Pattern recognition, classification, content generation, optimization, driving cars, decision making

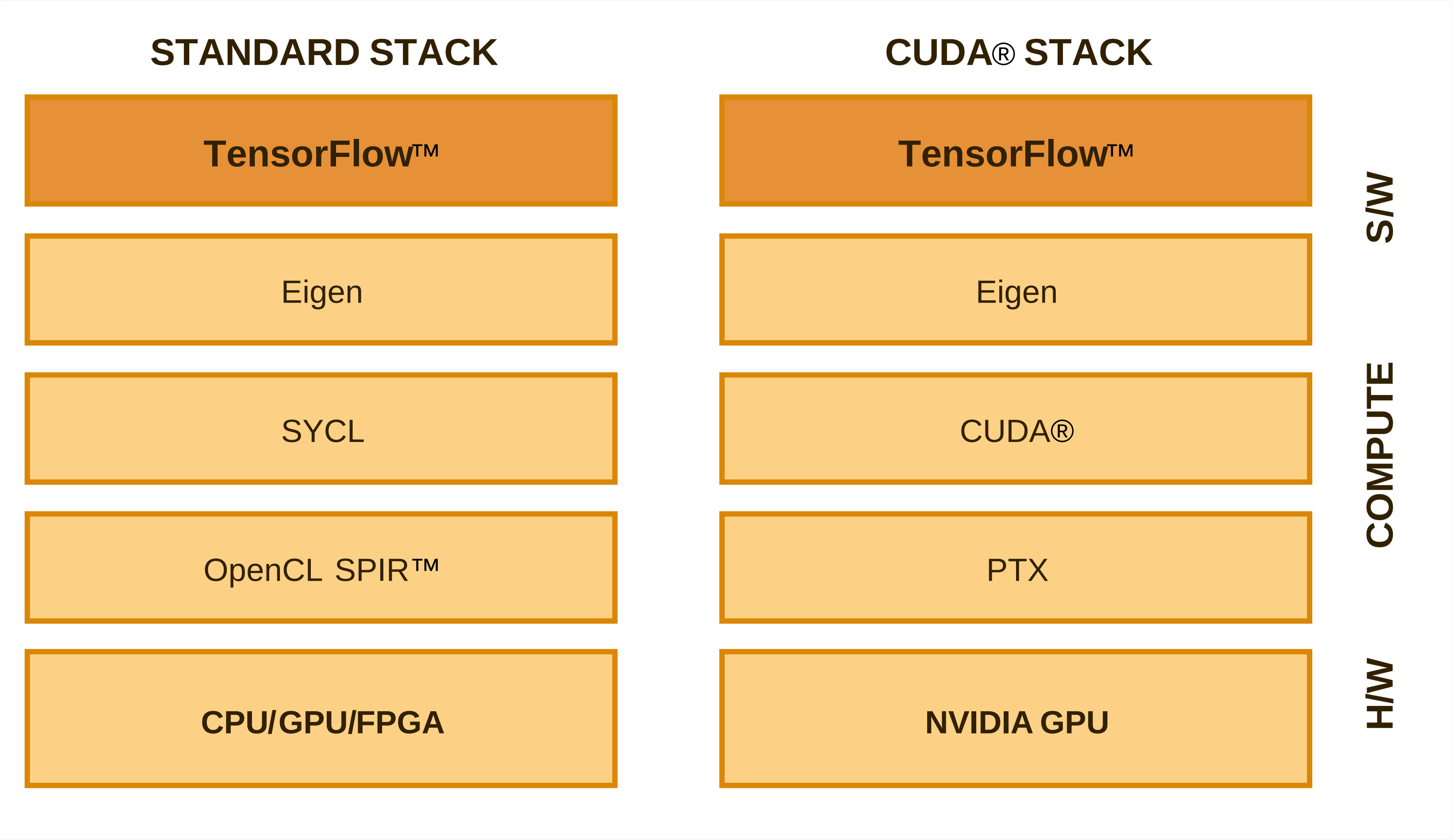
Motivation

- Available frameworks are dominated by CUDA®
  - Lack of OpenCL™ support
    - Does not support multiple architectures
    - Does not support performance portability
  - Embedded system challenges
    - Huge computation and data transfer demands
    - Storage, power and memory resource constraints
    - High efficiency and accuracy

SYCL Programming Model

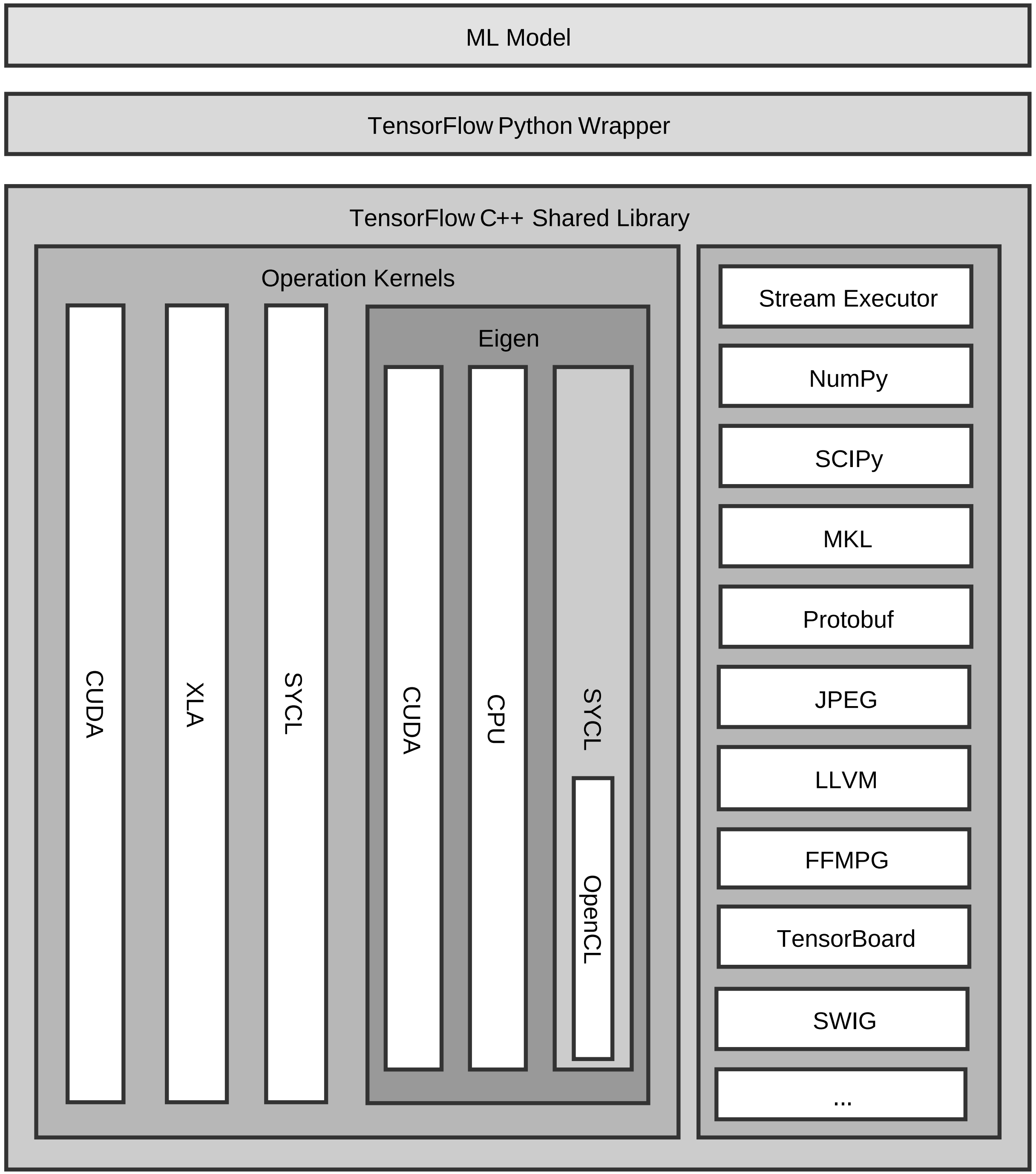
- A royalty-free, open standard from The Khronos Group™
- ComputeCpp implementation used for this project
  - TriSYCL - alternative implementation
- Enables better cross-platform performance portability
- Modern C++ supported
- Single-source programming model

Where SYCL Fits

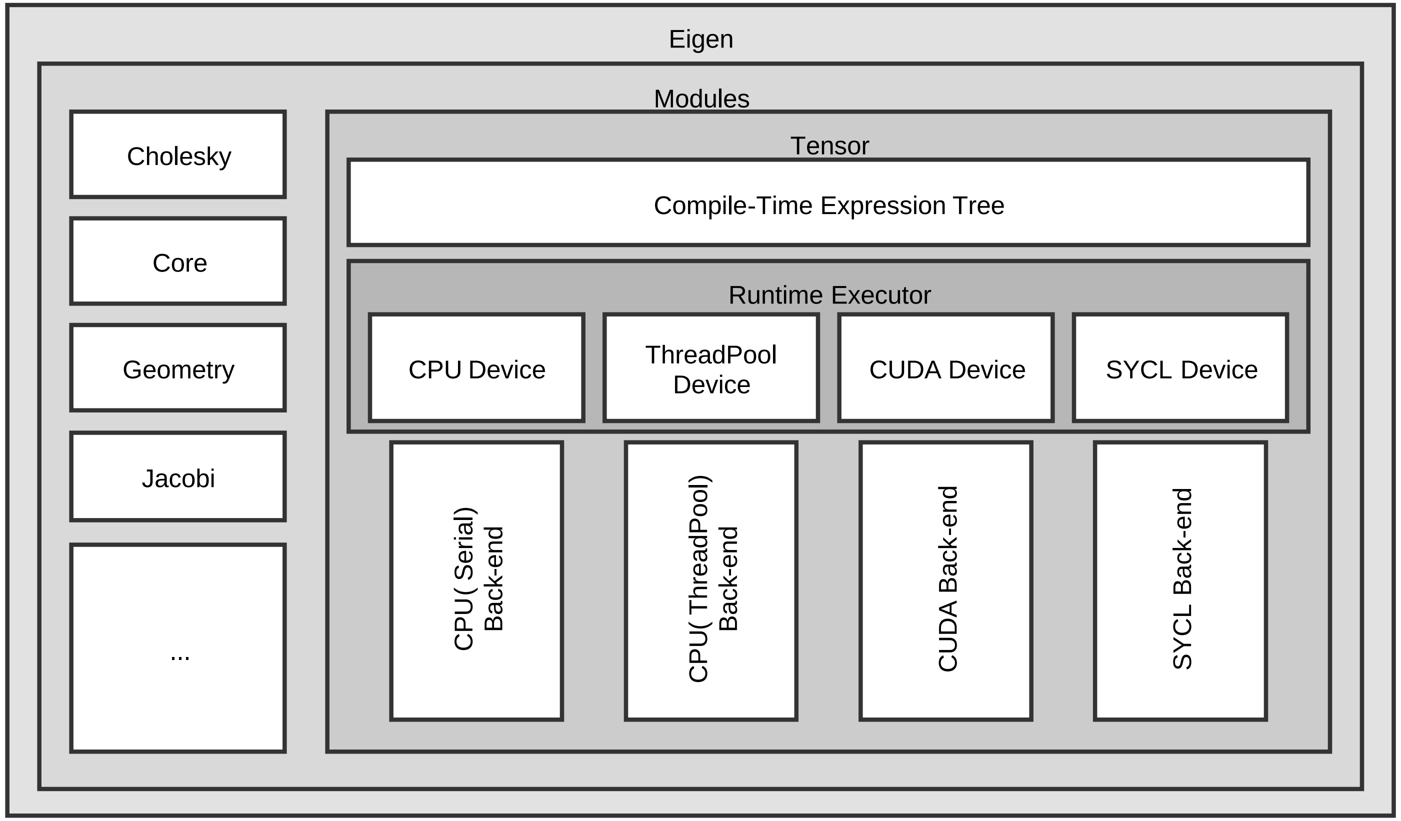


TensorFlow

- Modern data-flow framework by Google
- Front-end: graph-based model
- Tensor (input / output data)
- Operations (computation kernels)
- Back-ends:
  - Eigen (main)
  - CuDNN: NVIDIA neural network library
  - Embedded built-in operations



Eigen



- C++ high-performance linear algebra library.
- Modular
- Headers only
- Expression template meta-programming technique
- Back-ends:
  - CPU
  - NVIDIA CUDA
  - and now SYCL

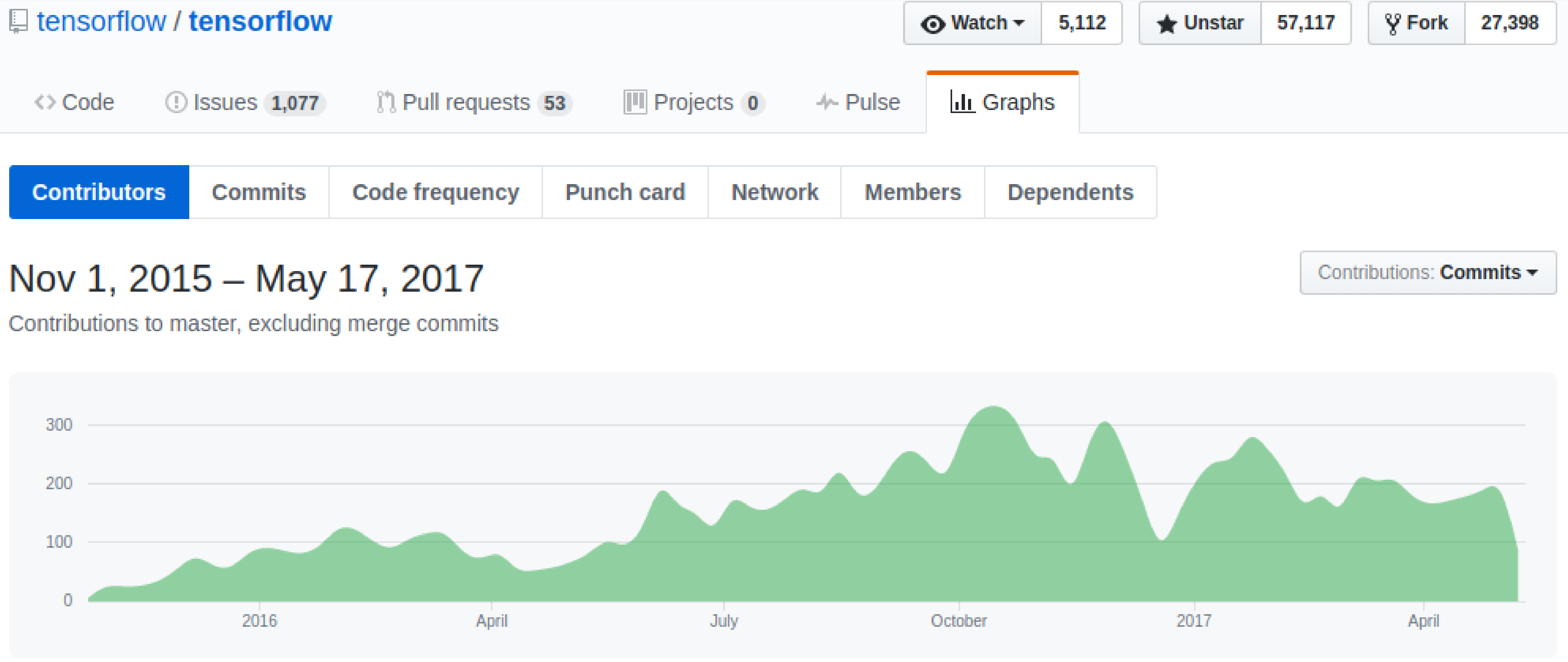
The Goal

- Functional OpenCL 1.2 back-end in TensorFlow
  - An OpenCL 1.2 back-end for Eigen is also needed
  - Integration must be **non-intrusive**
    - Should not change the front-end interface
    - Should re-use the existing code base as much as possible
    - Should not break any other modules
- TensorFlow integration without **any** major changes

The Challenge

- **Eigen**
  - Heavily uses C++ template meta-programming
    - The expression tree model
  - Single-source programming model used for CUDA and CPU
  - Standard scalar pointer used for existing back-ends (persistent device pointer)
  - Explicit data transfer interface between host and device
- **TensorFlow**
  - Complex, multi-layered framework design
  - CUDA design used for main heterogeneous back-end
  - Under active development – new features are added on a weekly basis

TensorFlow on GitHub



Why SYCL?

- **SYCL is an open standard**, enabling portability across a wide range of devices
- SYCL can dispatch device kernels from a C++ application, similar to CUDA
- OpenCL 1.2 does not support C++ directly, so adding OpenCL support to TensorFlow would require reimplementation of the back-end – maintenance overhead
- Expression of the tree-based kernel fusion is challenging without embedding a custom compiler
- Single-source programming model
  - No need to implement separate kernel code for each operation
- Re-use of the existing template code for both host and device is possible

Work Performed

- Conversion of raw pointers to accessors at compile-time:
  - The expression tree is recreated, with SYCL buffers in place of raw pointers
  - The expression tree is then traversed, in order to re-instantiate the expression tree on the SYCL device
    - Pointers to data in host memory are replaced with the corresponding accessors to SYCL buffers

Work Performed

- TensorFlow operation registration for SYCL
- Reuse of Eigen operations

```
namespace tensorflow {
REGISTER5(UnaryOp, CPU, "Sqrt", functor::sqrt, float, Eigen::half, double,
          complex64, complex128);
#if GOOGLE_CUDA
REGISTER3(UnaryOp, GPU, "Sqrt", functor::sqrt, float, Eigen::half, double);
#endif
#ifdef TENSORFLOW_USE_SYCL
REGISTER2(UnaryOp, SYCL, "Sqrt", functor::sqrt, float, double);
#endif // TENSORFLOW_USE_SYCL
```

Work Performed

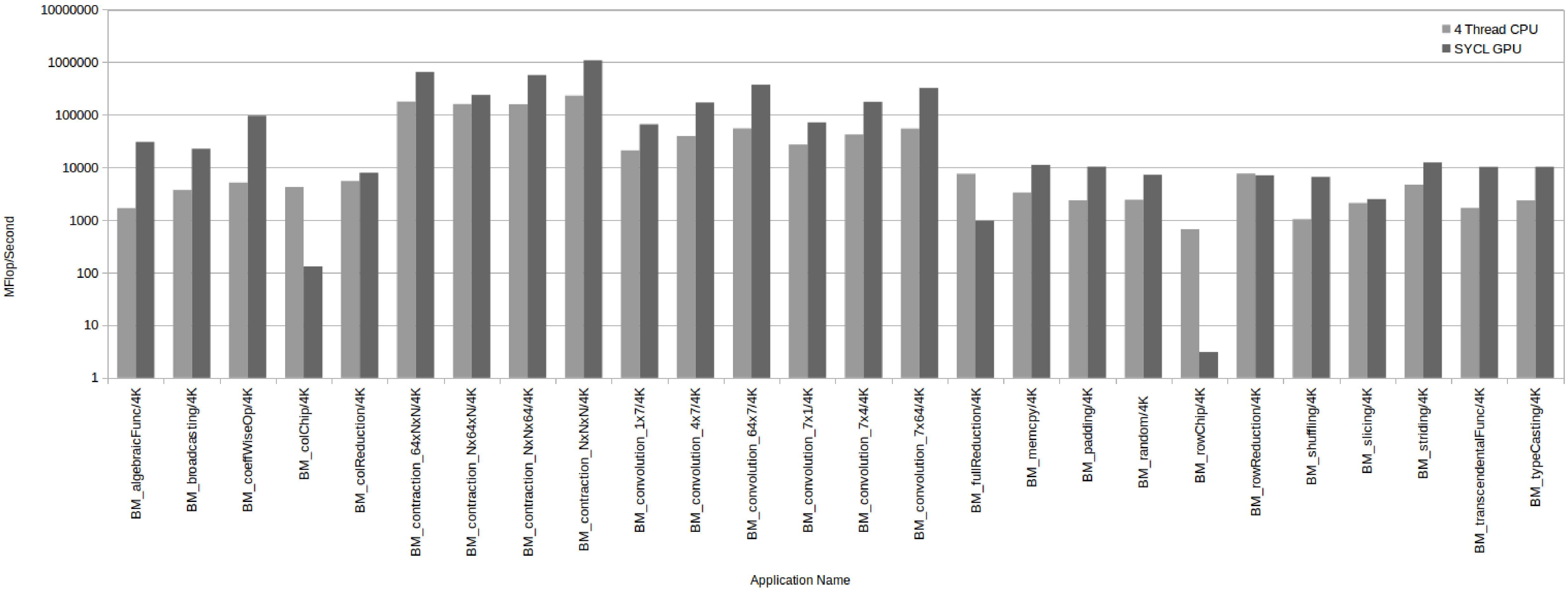
- TensorFlow operation registration for SYCL
- Reuse of Eigen operations

```cpp
//   Coefficient-wise unary operations:
//   Device: E.g., CPUDevice, GPUDevice or SYCLDevice.
//   Functor: defined in cwise_ops.h. E.g., functor::sqrt.
template <typename Device, typename Functor>
class UnaryOp : public OpKernel {
 public:
  typedef typename Functor::in_type Tin;   // Input scalar data type.
  typedef typename Functor::out_type Tout;// Output scalar data type.
  // Tin may be different from Tout. E.g., abs: complex64 -> float
  explicit UnaryOp(OpKernelConstruction* ctx) : OpKernel(ctx) {
    auto in = DataTypeToEnum<Tin>::v();
    auto out = DataTypeToEnum<Tout>::v();
    OP_REQUIRES_OK(ctx, ctx->MatchSignature({in}, {out}));
  }
  void Compute(OpKernelContext* ctx) override {
    const Tensor& inp = ctx->input(0);
    Tensor* out = nullptr;
    if (std::is_same<Tin, Tout>::value) {
      OP_REQUIRES_OK(ctx, ctx->forward_input_or_allocate_output(
                              {0}, 0, inp.shape(), &out));
    } else {
      OP_REQUIRES_OK(ctx, ctx->allocate_output(0, inp.shape(), &out));
    }
    functor::UnaryFunctor<Device, Functor>()(
        ctx->eigen_device<Device>(),
        out->flat<Tout>(), inp.flat<Tin>());
  }
};
```

Where we are

- We have **most** of the Eigen back-end implemented
- We are working on performance improvements
- SYCL support in TensorFlow is approaching full support for Inception-v3
  - Most of the model's operations run on SYCL devices
- We are in the process of **upstreaming** our changes

Intel® Core™ i7-6700K CPU 4.00GHz VS AMD Radeon™ R9 Nano



What Next?

- Current SYCL support in Eigen and TensorFlow is at an initial release level
- Progressing towards feature completion in both
- Performance improvements
- Benchmarking with ML models
- Targeting more platforms
- Continuing to push changes to the upstream repositories
- We'll keep you posted!

Thanks! Questions?

luke@codeplay.com

https://github.com/lukeiwanski/tensorflow
https://bitbucket.org/mehdi_goli/opencl
http://sycl.tech/