XILINX ALL PROGRAMMABLE™

TensorFlow SYCL with triSYCL

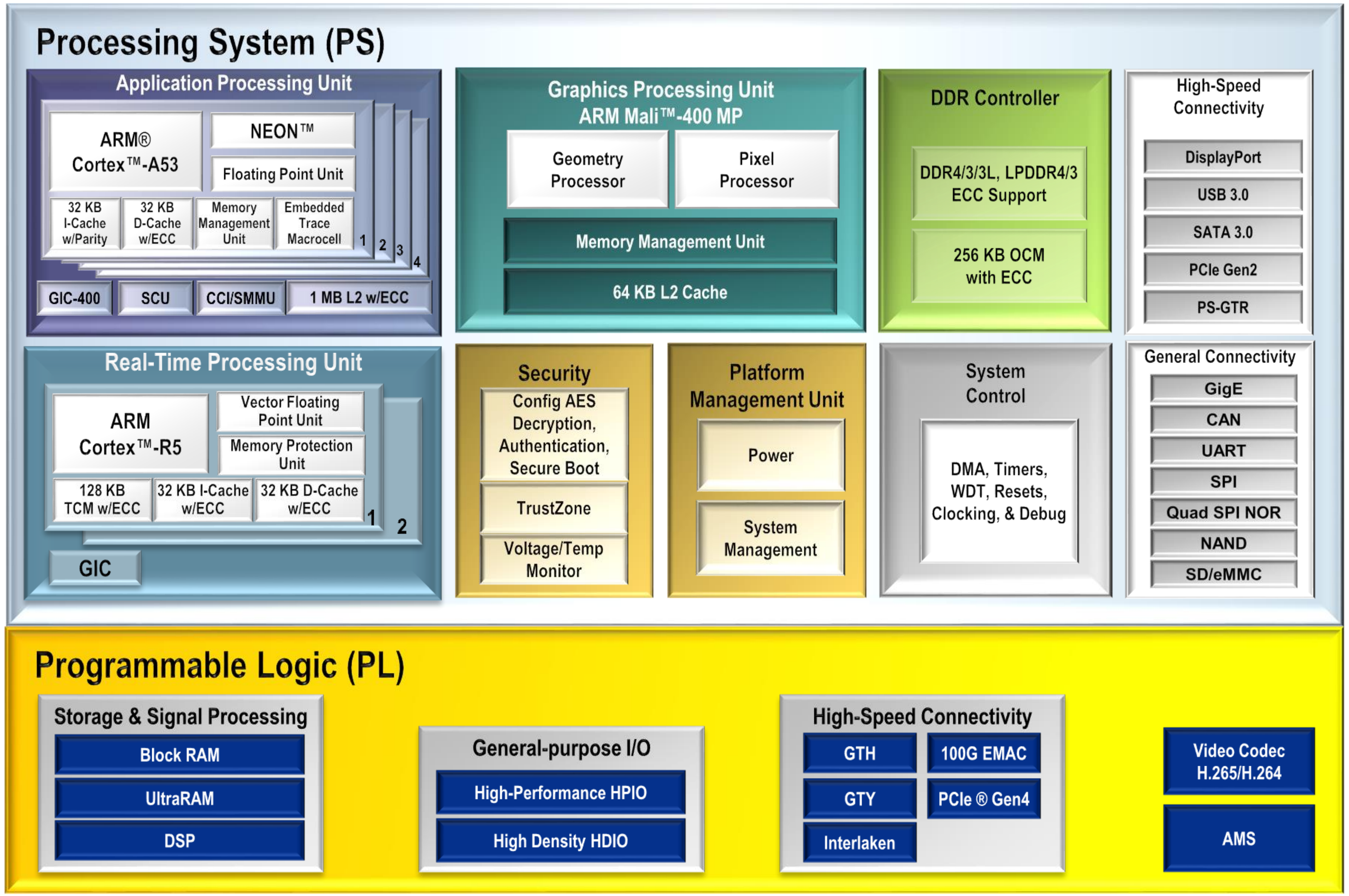
Xilinx Research Labs
Khronos booth @SC17 2017/11/12—19

TensorFlow

- Library for dataflow programming
- Symbolic math library with multidimensional data arrays (tensors)
- Used for machine learning applications
- Developed by Google for research & production
- Open-source software since 2015
- Only supported CUDA at that time… ☹

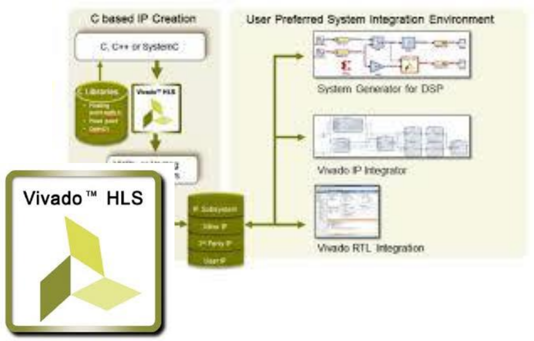
https://www.tensorflow.org/

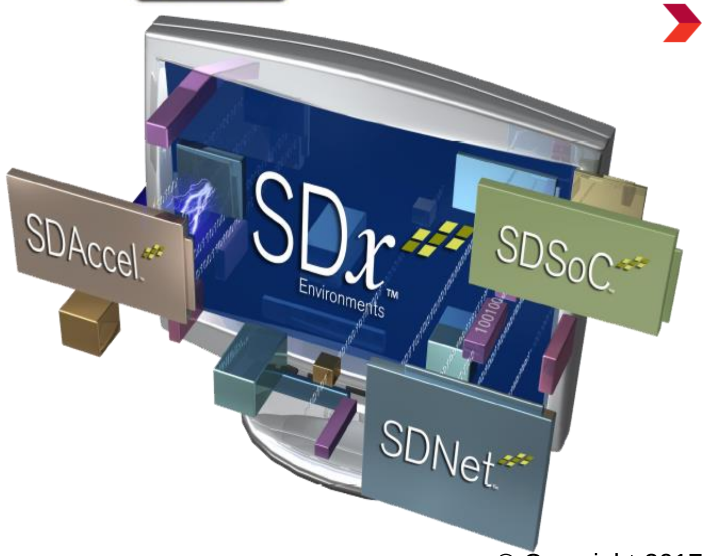
Zynq UltraScale+ MPSoC overview: All Programmable…



…Xilinx Zynq UltraScale+ MPSoC programming

- Vivado
  - Hardware basic block integration
  - RTL (Verilog & VHDL) programming
- Vivado HLS
  - C & C++ high-level synthesis
  - Down to LUT, DSP & BRAM...
- SDAccel
  - Khronos Group OpenCL
- SDSoC
  - C & C++ with `#pragma`





- SDNet
  - Generate routers from network protocol description
- Various libraries
  - OpenCV, DNN...
- Linux
  - Usual CPU multicore programming
- OpenAMP
  - Real-time ARM R5

PROMOTER MEMBERS

KHRONOS GROUP

Over 100 members worldwide

Any company is welcome to join

AMD, Apple, arm, EPIC GAMES, Google, HUAWEI, Imagination, intel, NOKIA, QUALCOMM, NVIDIA, SAMSUNG, SONY, VALVE, VeriSilicon

3D Incorporated, Adobe, AIMOTIVE, Ajou University 1973, amazon.com, antilatency, AXELL CORPORATION, AXIS COMMUNICATIONS, pinecone, BASE MARK, BINOMIAL

BLIZZARD ENTERTAINMENT, BROADCOM, BRENWILL, cadence, CANONICAL, CEVA, CAICT, codeplay, Continental

Coordinate Embedded Solution provider, COREAVI, DASSAULT SYSTEMES, DELL, CATAPULT Digital, DMP DIGITAL MEDIA PROFESSIONALS, ETRI Electronics and Telecommunications Research Institute, Fraunhofer, FUTUREMARK CORPORATION, Gaijin entertainment, HARMAN

HITACHI Inspire the Next, htc, hugh perk, HYPEREAL, igalia, IKEA Communications AB, IKEA, Imperial College London, 財團法人資訊工業策進會 INSTITUTE FOR INFORMATION INDUSTRY, ITRI Industrial Technology Research Institute

KDAB, knu KYUNGPOOK NATIONAL UNIVERSITY, LG, Linaro, logitech, Los Alamos NATIONAL LABORATORY EST. 1943, LUNARG, matrox, MAXON, MEDIATEK, Mentor Graphics

Microsoft, MIT Lincoln Laboratory, MITSUBISHI ELECTRIC, moz://a, mobica, National Taiwan University, National Tsing Hua University, NEC, NIHON UNIVERSITY, Nintendo, NXP

oculus, ON Semiconductor, OSU Oregon State University, owlii, OXIDE, Panasonic, PIXAR, PLUTO VR, Qt The Qt Company, RAZER, RENESAS

Rockwell Collins, 서울대학교 SEOUL NATIONAL UNIVERSITY, sensics, Silicon Studio, SIRU INNOVATIONS, socionext, SPREADTRUM, STREAM COMPUTING Performance Engineers, STARBREEZE, SYNOPSYS, TAKUMI

TAMPERE UNIVERSITY OF TECHNOLOGY, TU WIEN TECHNISCHE UNIVERSITÄT WIEN, Tencent, TEXAS INSTRUMENTS, thinci, Think Silicon, tobii, TOSHIBA, umbra, unity, Alma Mater Studiorum A.D. 1088

UCL, University of BRISTOL, UNIVERSITY OF TORONTO, University of Windsor, UX3D, 兆芯, Visteon, vmware, vrg., XILINX, zSpace

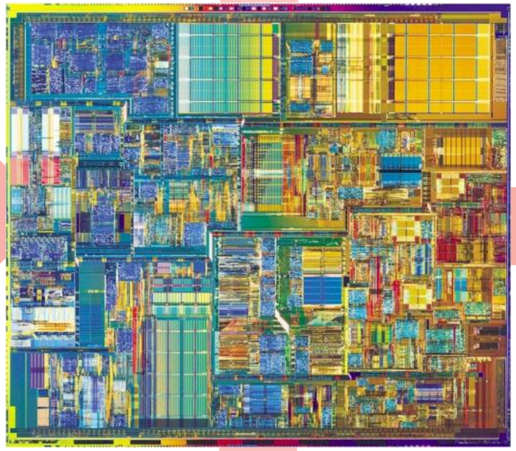
Khronos standards for heterogeneous systems

Connecting Software to Silicon

3D for the Web
- Real-time apps and games in-browser
- Efficiently delivering runtime 3D assets

Vision and Neural Networks
- Tracking and odometry
- Scene analysis/understanding
- Neural Network inferencing

Parallel Computation
- Machine Learning acceleration
- Embedded vision processing
- High Performance Computing (HPC)



Real-time 2D/3D
- Virtual and Augmented Reality
- Cross-platform gaming and UI
- CG Visual Effects
- CAD and Product Design
- Safety-critical displays

Remember C++ ?

2-line description by Bjarne Stroustrup

- Direct mapping to hardware
- Zero-overhead abstraction

⇒Unique existing position in embedded system to control the full stack!!!

- C++ rebooted in 2011
  – 1 new version every 3 years
  – Shipping what is implemented
- Easier
  – Simpler to do simple things
- More powerful
  – Possible to do more complex things

Even better with modern C++ (C++14, C++17, C++2a)

- Huge library improvements, parallelism...
- Simpler syntax, type inference in constructors…

```
std::vector my_vector { 1, 2, 3, 4, 5 };
// Display each element
for (auto e : my_vector)
  std::cout << e;
// Increment each element
for (auto &e : my_vector)
  e += 1;
```

Modern C++ : like Python but with speed and type safety

- Python 3.x (interpreted):

```
def add(x, y):
  return x + y
print(add(2, 3))       # Output: 5
print(add("2", "3"))   # Output: 23
print(add(2, "Boom")) # Fails at run-time :-(
```

- Same in C++14 but compiled + static compile-time type-checking:

```
auto add = [] (auto x, auto y) { return x + y; };
std::cout << add(2, 3) << std::endl;          // 5
std::cout << add("2"s, "3"s) << std::endl; // 23
std::cout << add(2, "Boom"s) << std::endl; // Does not compile :-)
```

- Automatic type inference for terse generic programming and type safety
  – Without ~~template~~ keyword!

Generic variadic lambdas & operator interpolation

```
#include <iostream>
#include <string>
using namespace std::string_literals;
// Define an adder on anything.
// Use new C++14 generic variadic lambda syntax
auto add = [] (auto... args) {
  // Use new C++17 operator folding syntax
  return (... + args);
};
int main() {
 std::cout << "The result is: " << add(1, 2, 3) << std::endl;
 std::cout << "The result is: " << add("begin"s, "end"s) << std::endl;
}
```

- Terse generic programming and type safety
  – Without ~~template~~ keyword!

Complete example of matrix addition in OpenCL SYCL

```
#include <CL/sycl.hpp>
#include <iostream>
using namespace cl::sycl;
constexpr size_t N = 2;
constexpr size_t M = 3;
using Matrix = float[N][M];

// Compute sum of matrices a and b into c
int main() {
 Matrix a = { { 1, 2, 3 }, { 4, 5, 6 } };
 Matrix b = { { 2, 3, 4 }, { 5, 6, 7 } };

 Matrix c;

 {// Create a queue to work on default device
  queue q;
  // Wrap some buffers around our data
  buffer A { &a[0][0], range { N, M } };
  buffer B { &b[0][0], range { N, M } };
  buffer C { &c[0][0], range { N, M } };
  // Enqueue some computation kernel task
  q.submit([&](handler& cgh) {
   // Define the data used/produced
   auto ka = A.get_access<access::mode::read>(cgh);
   auto kb = B.get_access<access::mode::read>(cgh);
   auto kc = C.get_access<access::mode::write>(cgh);
   // Create & call kernel named "mat_add"
   cgh.parallel_for<class mat_add>(range { N, M },
      [=](id<2> i) { kc[i] = ka[i] + kb[i]; }
   );
  }); // End of our commands for this queue
 } // End scope, so wait for the buffers to be released
 // Copy back the buffer data with RAII behaviour.
 std::cout << "c[0][2] = " << c[0][2] << std::endl;
 return 0;
}
```

XILINX ALL PROGRAMMABLE.

SYCL 2.2 = pure C++17 DSEL

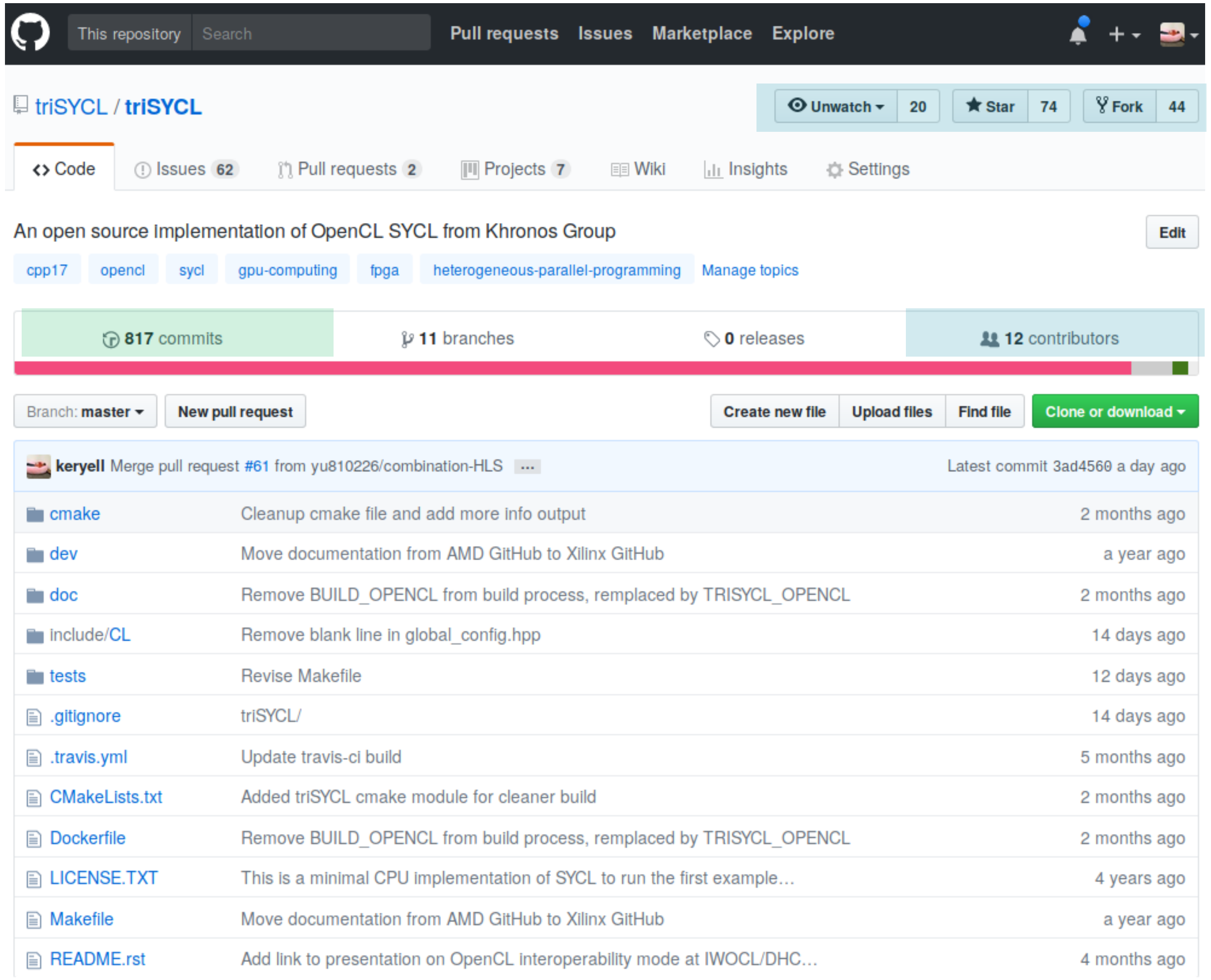
- Implement concepts useful for heterogeneous computing
- Asynchronous task graph
- Hierarchical parallelism & kernel-side enqueue
- Queues to direct computations on devices
- Single-source programming model
  - Take advantage of CUDA on steroids & OpenMP simplicity and power
  - Compiled for host *and* device(s)
  - Enabling the creation of C++ higher level programming models & C++ templated libraries
  - System-level programming (SYstemCL)
- Buffers to define location-independent storage
- Accessors to express usage for buffers and pipes: read/write/...
  - No explicit data motion
  - Automatic overlapping of communication/computation
- Hierarchical storage
  - Rely on C++ allocator to specify storage (SVM...)
  - Usual OpenCL-style global/local/private
- Most modern C++ features available for OpenCL
  - Programming interface based on abstraction of OpenCL components (data management, error handling...)
  - Provide OpenCL interoperability
- Directly executable DSEL
  - Host fall-back & emulation for free
  - No specific compiler needed for experimenting on host
  - Debug and symmetry for SIMD/multithread on host

Known implementations of SYCL

- ComputeCpp by Codeplay https://www.codeplay.com/products/computecpp
  - Most advanced SYCL 1.2 implementation
  - Outlining compiler generating SPIR
  - Run on any OpenCL device and CPU
  - Can run TensorFlow SYCL
- sycl-gtx https://github.com/ProGTX/sycl-gtx
  - Open source
  - No (outlining) compiler ➔ use some macros with different syntax
- triSYCL https://github.com/triSYCL/triSYCL

triSYCL

- Open Source SYCL 1.2/2.2
- Uses C++17 templated classes
- Used by Khronos to define the SYCL and OpenCL C++ standard
  – Languages are now too complex to be defined without implementing...
- On-going implementation started at AMD and now led by Xilinx
- https://github.com/triSYCL/triSYCL
- OpenMP for host parallelism
- Boost.Compute for OpenCL interaction
- Prototype of device compiler for Xilinx FPGA



An open source implementation of OpenCL SYCL from Khronos Group

keryell Merge pull request #61 from yu810226/combination-HLS … Latest commit 3ad4560 a day ago

cmake	Cleanup cmake file and add more info output	2 months ago
dev	Move documentation from AMD GitHub to Xilinx GitHub	a year ago
doc	Remove BUILD_OPENCL from build process, remplaced by TRISYCL_OPENCL	2 months ago
include/CL	Remove blank line in global_config.hpp	14 days ago
tests	Revise Makefile	12 days ago
.gitignore	triSYCL/	14 days ago
.travis.yml	Update travis-ci build	5 months ago
CMakeLists.txt	Added triSYCL cmake module for cleaner build	2 months ago
Dockerfile	Remove BUILD_OPENCL from build process, remplaced by TRISYCL_OPENCL	2 months ago
LICENSE.TXT	This is a minimal CPU implementation of SYCL to run the first example…	4 years ago
Makefile	Move documentation from AMD GitHub to Xilinx GitHub	a year ago
README.rst	Add link to presentation on OpenCL interoperability mode at IWOCL/DHC…	4 months ago

TensorFlow SYCL

- Initial TensorFlow version from Google supports CPU & nVidia GPU with CUDA
- Other devices with XLA compiler
- SYCL version started in 2015 by Codeplay
  - CUDA is single-source C++, SYCL too, easier to use than OpenCL C/C++
  - Joint effort by Codeplay, Google, Xilinx, Oracle…
  - Upstreamed directly in https://github.com/tensorflow/tensorflow
- Eigen: C++ library with mathematical & tensor operations
  - Use template metaprogramming to do kernel fusion
  - Extended with SYCL devices and SYCL memory management
- Tensorflow
  - Add SYCL devices
- Developed and tested with Codeplay ComputeCpp
  - Interesting to test with another SYCL implementation: triSYCL

Eigen

- Puts the “tensor” in TensorFlow

- C++ template library for linear algebra
  - Tensor module developed by Google and the SYCL extension by Codeplay
  - Single-source
  - Multiple devices available : Eigen thread pool, CUDA, SYCL
  - Lazy evaluation and kernel fusion built-in
  - Explicit scheduler
  - Follow CUDA low-level memory management ☹
  - 2 previous points do not fully take advantage of SYCL high-level concepts

- Worked with ComputeCPP and now triSYCL
  - Available upstream : https://bitbucket.org/eigen/eigen
  - Reuse triSYCL CMake module from the SYCL Parallel STL open-source project

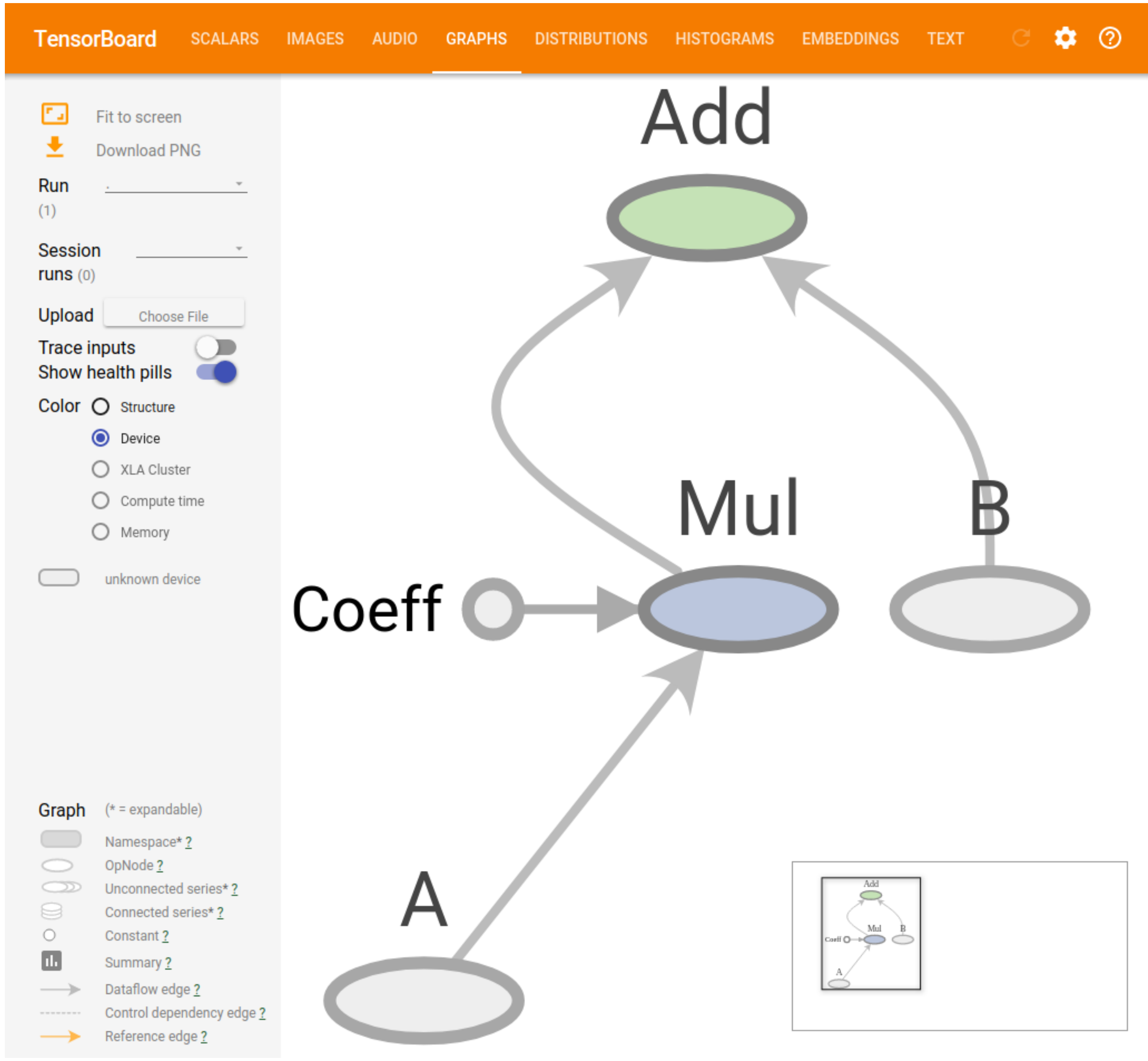
SYCL Eigen (computing (a + b) * b with tensors)

```
std::vector<cl::sycl::device> devices = Eigen::get_sycl_supported_devices();
QueueInterface queueInterface(devices[0]);
auto s_device = Eigen::SyclDevice(&queueInterface);

// Define the shape of the rank 3 tensors
IndexType sizeDim1 = 100, sizeDim2 = 20, sizeDim3 = 20;
Eigen::array<IndexType, 3> tensorRange = { { sizeDim1, sizeDim2, sizeDim3 } };
Eigen::Tensor<DataType, 3, DataLayout, IndexType> in1 { tensorRange };
Eigen::Tensor<DataType, 3, DataLayout, IndexType> in2 { tensorRange };
Eigen::Tensor<DataType, 3, DataLayout, IndexType> out { tensorRange };
Eigen::Tensor<DataType, 3, DataLayout, IndexType> out_host { tensorRange };

// Fill tensors with random values
in1.setRandom();
in2.setRandom();

// Allocate device memory for input and output tensors
auto gpu_in1_data = static_cast<DataType*>(s_device.allocate(in1.dimensions().TotalSize() * sizeof(DataType)));
auto gpu_in2_data = static_cast<DataType*>(s_device.allocate(in2.dimensions().TotalSize() * sizeof(DataType)));
auto gpu_out_data = static_cast<DataType*>(s_device.allocate(out.dimensions().TotalSize() * sizeof(DataType)));

// Create TensorMap from device memory
Eigen::TensorMap<Eigen::Tensor<DataType, 3, DataLayout, IndexType>> gpu_in1 { gpu_in1_data, tensorRange };
Eigen::TensorMap<Eigen::Tensor<DataType, 3, DataLayout, IndexType>> gpu_in2 { gpu_in2_data, tensorRange };
Eigen::TensorMap<Eigen::Tensor<DataType, 3, DataLayout, IndexType>> gpu_out { gpu_out_data, tensorRange };

// Copy the input data to the device
s_device.memcpyHostToDevice(gpu_in1_data, in1.data(), (in1.dimensions().TotalSize()) * sizeof(DataType));
s_device.memcpyHostToDevice(gpu_in2_data, in2.data(), (in2.dimensions().TotalSize()) * sizeof(DataType));
// c = (a + b) * b done on the sycl_device
gpu_out.device(s_device) = (gpu_in1 + gpu_in2) * gpu_in2;
// Copy the data back to the host
s_device.memcpyDeviceToHost(out.data(), gpu_out_data, (out.dimensions().TotalSize()) * sizeof(DataType));
// c = (a + b) * b done on the CPU
out_host = (in1 + in2) * in2;
```

TensorFlow SYCL example

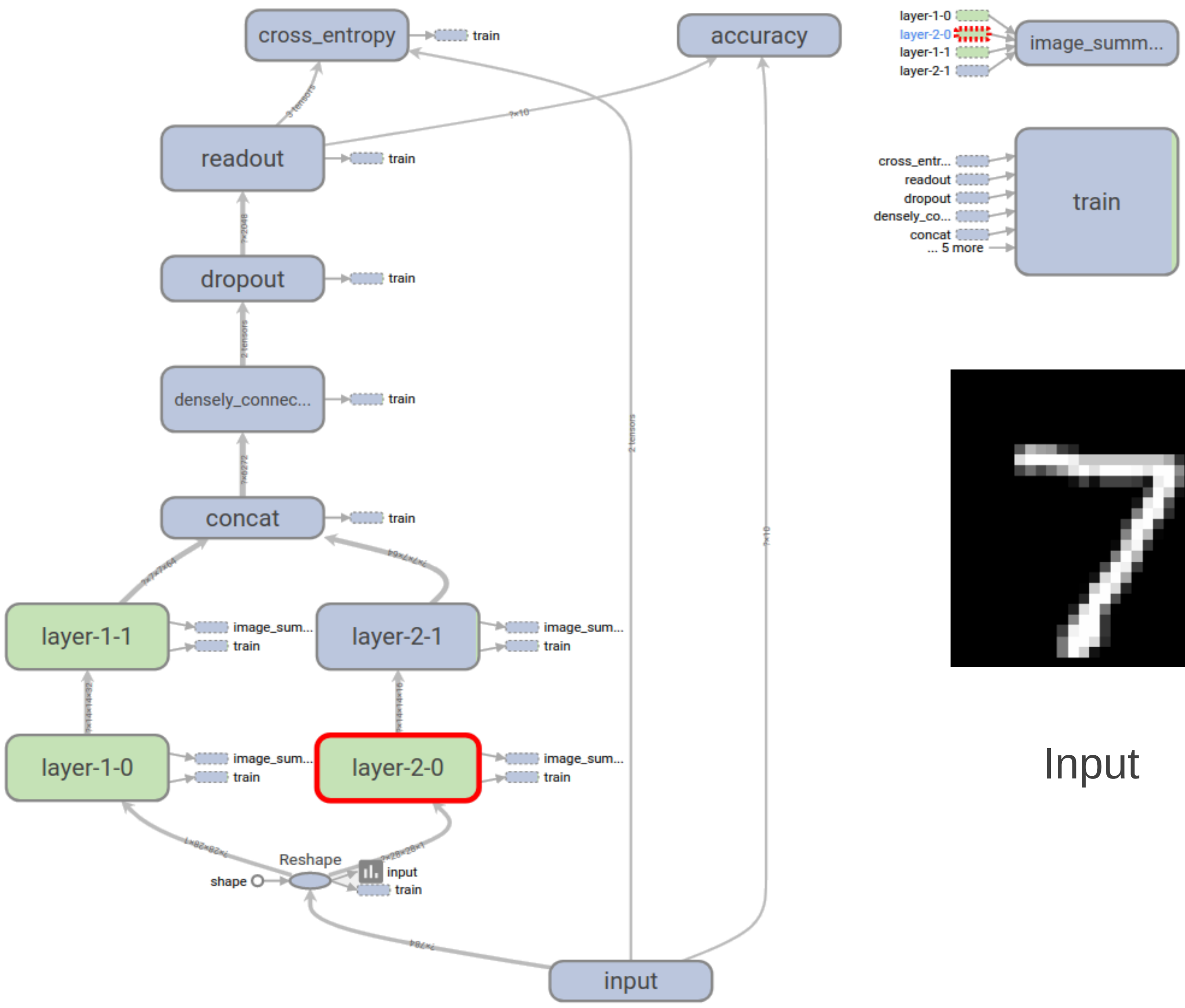
```
import tensorflow as tf
sess = tf.InteractiveSession()
file_writer = tf.summary.FileWriter('logs', sess.graph)
# To output a new version of the graph:
def ug():
  file_writer.add_graph(sess.graph)
  file_writer.flush()
coeff = tf.constant(3.0, tf.float32, name = "Coeff")
a = tf.placeholder(tf.float32, name = "A")
b = tf.placeholder(tf.float32, name = "B")
with tf.device(tf.DeviceSpec(device_type="SYCL")):
  product = tf.multiply(coeff, a, name = "Mul")
with tf.device(tf.DeviceSpec(device_type="CPU")):
  linear_model = tf.add(product, b, name = "Add")
print(sess.run(linear_model, {a : 3, b : 4.5 }))
ug()
```

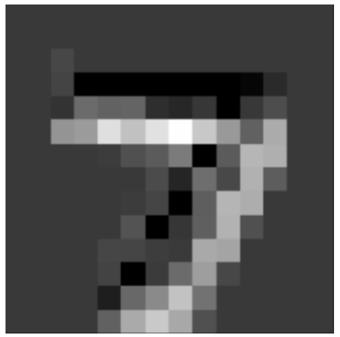
```
13.5
```



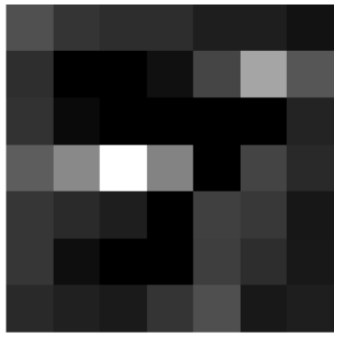
XILINX ALL PROGRAMMABLE.

TensorFlow SYCL example 2

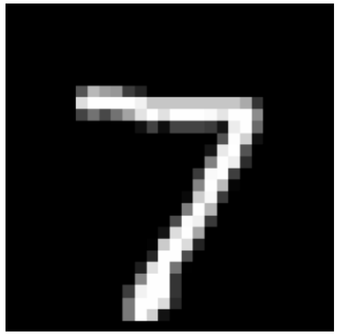




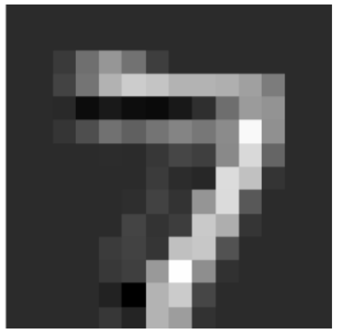
Layer 1-0



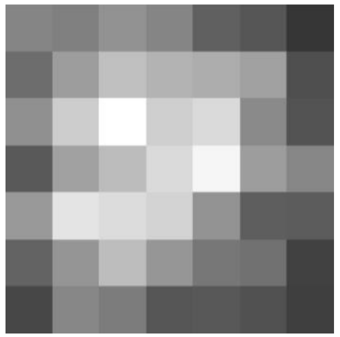
Layer 1-1



Input



Layer 2-0



Layer 2-1

TensorFlow SYCL example 2

```
[…]
def nn_layer1(input_tensor, weight_shape, bias_shape,
              layer_name, act=tf.nn.relu):
 with tf.device('/device:SYCL:0'):
   weights = weight_variable(weight_shape)
   biases = bias_variable(bias_shape)
   h_conv = act(conv2d(input_tensor, weights)
           + biases)
   h_pool = max_pool_2x2(h_conv)
   return h_pool

def nn_layer2(input_tensor, weight_shape, bias_shape,
              layer_name, act=tf.nn.relu):
  weights = weight_variable(weight_shape)
  biases = bias_variable(bias_shape)
  with tf.device('/device:SYCL:0'):
    h_pool = avg_pool_2x2(input_tensor)
  h_conv = act(conv2d(h_pool, weights) + biases)
  return h_conv

hidden10 = nn_layer1(x_image, [5, 5, 1, 32], [32],
                      'layer-1-0')
hidden20 = nn_layer1(x_image, [5, 5, 1, 16], [16],
                      'layer-2-0')
```

```
hidden11 = nn_layer1(hidden10, [5, 5, 32, 64], [64],
                     'layer-1-1')
hidden21 = nn_layer2(hidden20, [5, 5, 16, 64], [64],
                     'layer-2-1')
h_pool11_flat = tf.reshape(hidden11, [-1, 7*7*64])
h_pool21_flat = tf.reshape(hidden21, [-1, 7*7*64])
hidden_concat = tf.concat([h_pool11_flat,
                          h_pool21_flat], 1)
W_fc1 = weight_variable([(7 * 7 * 64) + (7 * 7 * 64),
                        2048])
b_fc1 = bias_variable([2048])
h_fc1 = tf.nn.relu(tf.matmul(hidden_concat, W_fc1) +
                   b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
W_fc2 = weight_variable([2048, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
diff = tf.nn.softmax_cross_entropy_with_logits(
         labels=y_, logits=y_conv)
cross_entropy = tf.reduce_mean(diff)

 […]
```

TensorFlow on FPGA

- TensorFlow CUDA is written with GPU target in mind…
- TensorFlow SYCL implementation
  - Keeps the TensorFlow single-source C++ operators
  - Changes the executors, memory management and host-device transfers
- SYCL brings functional portability on top of OpenCL
  - Unfortunately no performance portability across various architectures (FPGA…)
  - But there are SYCL & OpenCL standard ways to optimize to a given target

- But there are already optimized OpenCL DNN around…

OpenCL interoperability mode

- SYCL → Very generic parallel model
- Allows interaction with OpenCL/Vulkan/OpenGL without overhead
- Keeps the high-level features of SYCL
  - No explicit buffer transfer
  - Task and data dependency graphs
  - Templated C++ code

  …
- The user can call any existing OpenCL kernel
  - Even HLS C++ & RTL Xilinx FPGA kernels !
  - Avoid writing painful OpenCL C/C++ host code

OpenCL built-in kernels

- OpenCL built-in kernels are *very* common in FPGA world
- Written in Verilog/VHDL or Vivado HLS C++
  - But with SDAccel OpenCL kernel interface
- Typical use cases
  - Kernel libraries
  - Linear algebra
  - Machine learning
  - Computer vision
  - Direct access to hardware: wire-speed Ethernet…
- SYCL OpenCL interoperability mode can be used to simplify usage of these kernels

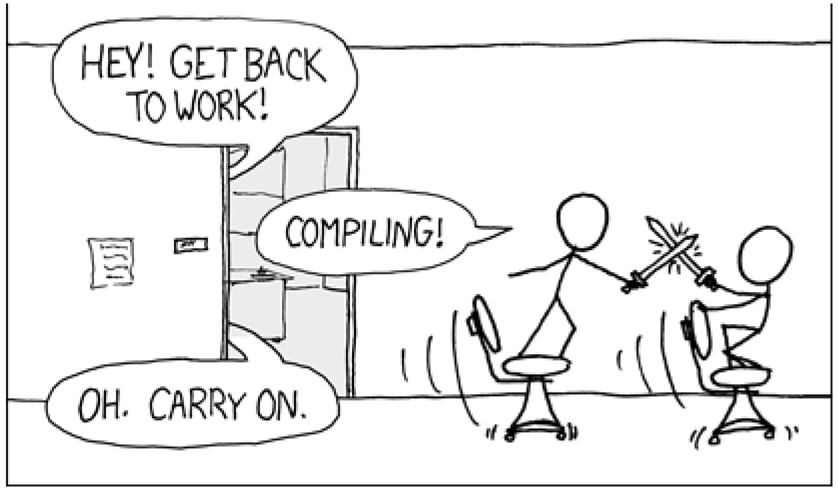
Using OpenCL interoperability mode in SYCL

```
#include <CL/sycl.hpp>
using namespace cl::sycl;

constexpr size_t N = 3;
using Vector = float[N];

int test_main(int argc, char *argv[]) {
  Vector a = { 1, 2, 3 };
  Vector b = { 5, 6, 8 };
  Vector c;
  // Construct the queue from the default OpenCL one
  queue q { boost::compute::system::default_queue() };
  // Create buffers from a & b vectors
  buffer<float> A { std::begin(a), std::end(a) };
  buffer<float> B { std::begin(b), std::end(b) };
  { // A buffer of N float using the storage of c
    buffer<float> C { c, N };
    // Construct an OpenCL program from the source string
    auto program = boost::compute::program::create_with_source(R"(
     __kernel void vector_add(const __global float *a,
                              const __global float *b,
                              __global float *c, int offset) {
       c[get_global_id(0)] = a[get_global_id(0)] + b[get_global_id(0)]
                             + offset;
     } )", boost::compute::system::default_context());
  // Build a kernel from the OpenCL kernel
  program.build();
  // Get the OpenCL kernel
  kernel k { boost::compute::kernel { program, "vector_add" } };
```

```
// Launch the vector parallel addition
q.submit([&](handler &cgh) {
/* The host-device copies are managed transparently by
    these accessors: */
  cgh.set_args(A.get_access<access::mode::read>(cgh),
               B.get_access<access::mode::read>(cgh),
               C.get_access<access::mode::write>(cgh),
               cl::sycl::cl_int { 42 } );
  cgh.parallel_for(N, k);
});
//< End of our commands for this queue
}
//< Buffer C goes out of scope and copies back values to c
std::cout << std::endl << "Result:" << std::endl;
for (auto e : c) std::cout << e << " "; std::cout << std::endl;
return 0;
}
```

Adding OpenCL interoperability to Eigen

- Goal → Introduce the ability to use native OpenCL kernels in Eigen
- Using OpenCL kernels :
  - Use existing optimised kernels (for FPGA)
  - Target specific accelerators (FPGA too ☺)
- SYCL OpenCL interoperability mode allows that possibility !
- Implemented as a new Eigen operation
  - Takes an arbitrary number of inputs
  - User-provided OpenCL file and kernel name
  - Accepts binary or OpenCL source file
  - Can use dynamically SDx xocc on Xilinx platform
    - Beware of the compilation time at the first run ☺



```
nativeOCL(void** arg_list, size_t arg_num, std::string kernel_name, std::string file_name, bool is_bin)
```

Eigen code using new OpenCL interoperability mode

```
arg1.setRandom();
arg2.setRandom();
arg3.setRandom();

auto arg1_device_ptr = static_cast<float*>(sycl_device.allocate(arg1.dimensions().TotalSize()*sizeof(float)));
auto arg2_device_ptr = static_cast<float*>(sycl_device.allocate(arg2.dimensions().TotalSize()*sizeof(float)));
auto arg3_device_ptr = static_cast<float*>(sycl_device.allocate(arg2.dimensions().TotalSize()*sizeof(float)));
sycl_device.memcpyHostToDevice(arg1_device_ptr, arg1.data(),(arg1.dimensions().TotalSize())*sizeof(float));
sycl_device.memcpyHostToDevice(arg2_device_ptr, arg2.data(),(arg2.dimensions().TotalSize())*sizeof(float));
sycl_device.memcpyHostToDevice(arg3_device_ptr, arg3.data(),(arg3.dimensions().TotalSize())*sizeof(float));

auto kernel_res_device_ptr = static_cast<float*>(sycl_device.allocate(kernel_res.dimensions().TotalSize()*sizeof(float)));

Eigen::TensorMap<Eigen::Tensor<float, 3, DataLayout, IndexType>> arg1_device_map(arg1_device_ptr, arg1.dimensions());
Eigen::TensorMap<Eigen::Tensor<float, 3, DataLayout, IndexType>> kernel_res_device_map(kernel_res_device_ptr, kernel_res.dimensions());

const void* arg_tab[2];
arg_tab[0] = arg2_device_ptr;
arg_tab[1] = arg3_device_ptr;

host_res = arg1 + (arg2 * arg3);
kernel_res_device_map.device(sycl_device) = arg1_device_map.nativeOCL(arg_tab, 2, "vector_add", "/path/to/file.cl", false);

sycl_device.memcpyDeviceToHost(kernel_res.data(), kernel_res_device_ptr, (kernel_res.dimensions().TotalSize())*sizeof(float));
```

The OpenCL Kernel :

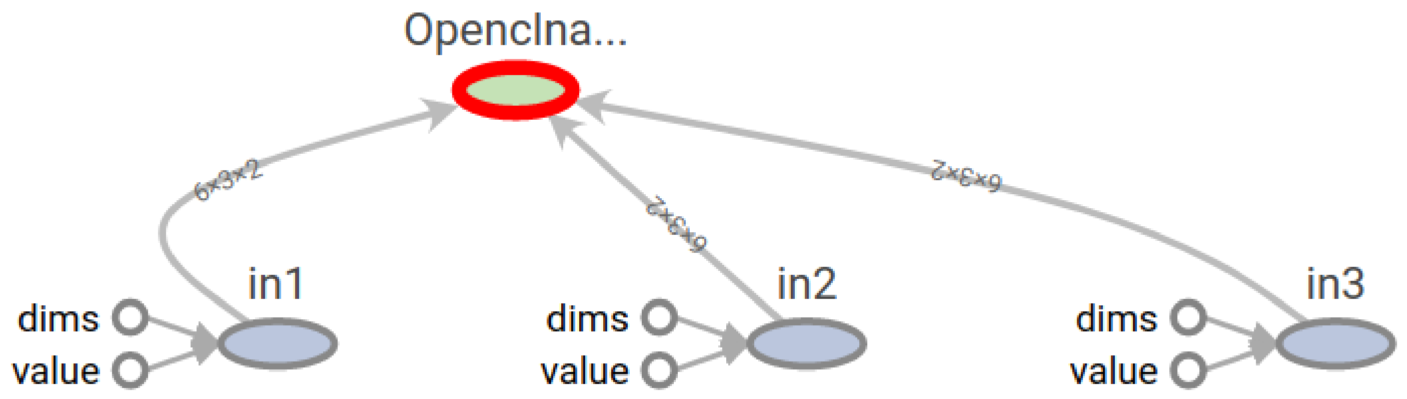
```
__kernel void vector_add(__global float* a, const __global float *b,
                         const __global float* c, const __global float* d) {
    a[get_global_id(0)] = b[get_global_id(0)] + (c[get_global_id(0)] * d[get_global_id(0)]);
}
```

OpenCL interoperability with Tensorflow

- A Tensorflow operation was also added
  - Uses the Eigen operation in the back-end
  - We get a Python interface for free!

```
conf = tf.ConfigProto(allow_soft_placement=False)
sess = tf.InteractiveSession(config=conf)

with tf.device('/cpu:0'):
  in1 = tf.fill([6,3,2], 18.0, name="in1")
  in2 = tf.fill([6,3,2], 12.0, name="in2")
  in3 = tf.fill([6,3,2], 2.0, name="in3")

with tf.device('/device:SYCL:0'):
  result = tf.user_ops.ocl_native_op(input_list=[in1, in2, in3], output_type=tf.float32, shape=[6,3,2],
                                     file_name="/path/to/kernel.cl", kernel_name="vector_add", is_binary=False)
print(result.eval())
```

TensorBoard graph :

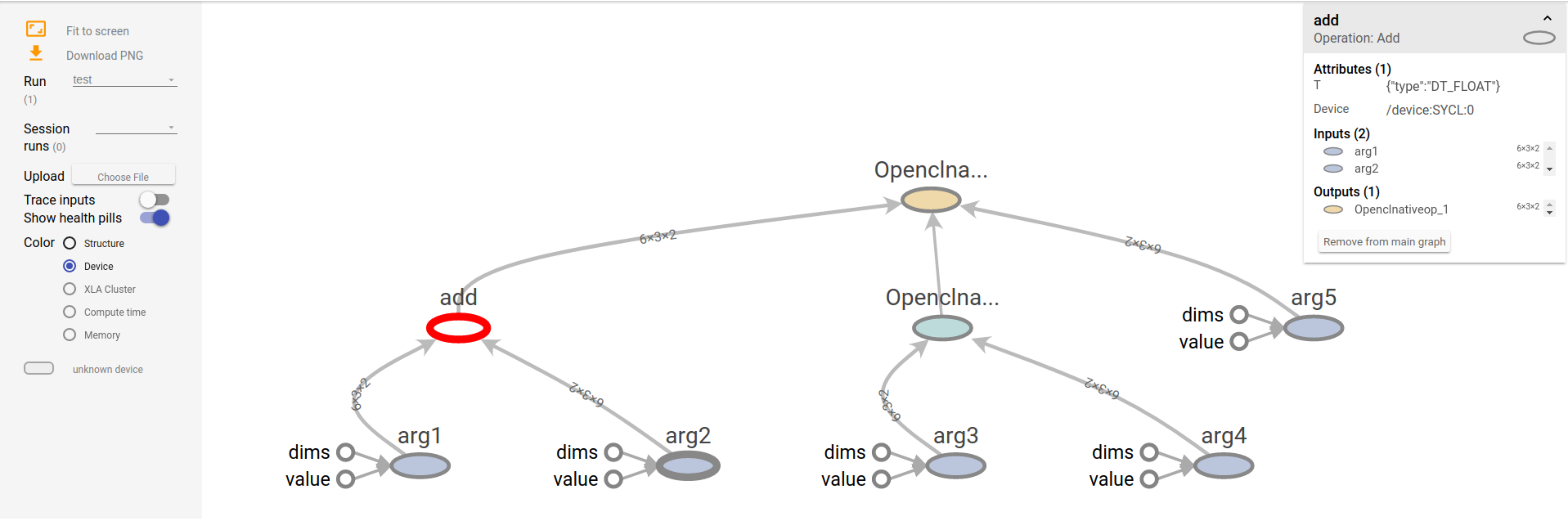


Multi-SYCL device & OpenCL kernels

- SYCL:0 → Host
- SYCL:1 → FPGA (Xilinx OpenCL)
- SYCL:2 → CPU (Intel OpenCL)

```
import tensorflow as tf

def testOclOp(self):
  conf = tf.ConfigProto(allow_soft_placement=False, device_count={'SYCL': 3})
  sess = tf.InteractiveSession(config=conf)

  with tf.device('/cpu:0'):
    arg1 = tf.fill([6,3,2], 11.5, name="arg1")
    arg2 = tf.fill([6,3,2], 10.5, name="arg2")
    arg3 = tf.fill([6,3,2], 5.0, name="arg3")
    arg4 = tf.fill([6,3,2], 2.0, name="arg4")
    arg5 = tf.fill([6,3,2], 2.0, name="arg5")

  with tf.device('/device:SYCL:0'):
    add_node = arg1 + arg2

  with tf.device('/device:SYCL:2'):
    mul_node = tf.user_ops.ocl_native_op(input_list=[arg3, arg4], output_type=tf.float32, shape=[6,3,2],
                                         file_name="/path/to/VecMul.cl", kernel_name="vector_mul", is_binary=False)

  with tf.device('/device:SYCL:1'):
    result = tf.user_ops.ocl_native_op(input_list=[add_node, mul_node, arg5], output_type=tf.float32, shape=[6,3,2],
                                       file_name="/path/to/VecAddMul.xclbin", kernel_name="vector_add_mul", is_binary=True)

 res = sess.run([result])
 print(res[0])
 writer = tf.summary.FileWriter('/tmp/tensorflow/logs/test', sess.graph)
 writer.close()
```

Multi-SYCL device & OpenCL kernels



Run: test (1)

Session runs (0)

Upload: Choose File

Trace inputs [off]

Show health pills [✓ on]

Color: Structure [✓ Device] XLA Cluster Compute time Memory

unknown device

add
Operation: Add

Attributes (1)

T	{"type":"DT_FLOAT"}
Device	/device:SYCL:0

Inputs (2)

arg1	6×3×2
arg2	6×3×2

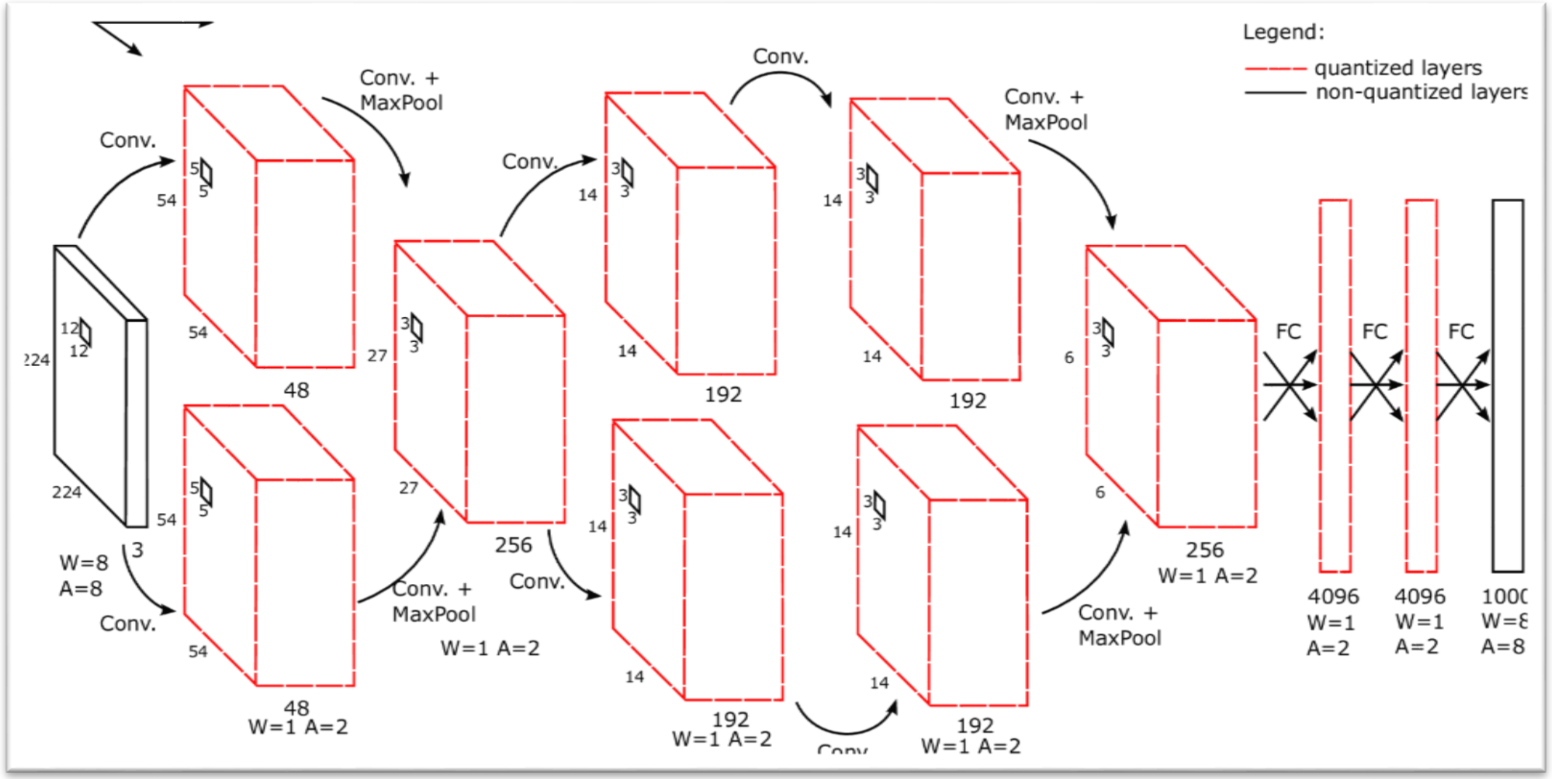
Outputs (1)

Openclnativeop_1	6×3×2

Remove from main graph

OpenCL interoperability in TensorFlow: enable new features

- Can use smaller data types than the ones available in TensorFlow
- DoReFa-Net (Pruned) – AlexNet like
  - 3915 Images/sec inference
  - 1b Weights, 2b Activations
  - 8.54 TOPS @ 109Mhz
  - 0.432msec latency



  - Amazon AWS F1 instance - 1x Xilinx VU9P FPGA
  - Host source: C++, with Khronos OpenCL C++ bindings
  - Kernel source: Xilinx Vivado HLS C++ with OpenCL-compatible kernel API

TensorFlow SYCL has minimal change compare to CUDA

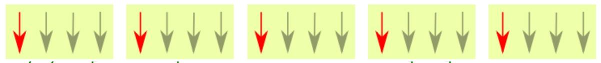
- SYCL execution model is based on OpenCL similar to CUDA
- Eigen operators unchanged from CUDA to SYCL
  - Use same coding style with explicit work-item & work-group management
  - From CUDA low-level thread blocks and `__syncthread()`
- Not efficient in triSYCL on CPU because it is pure C++
  - No de-SPMD operation like in PoCL or ComputeCpp…
  - Require 1 CPU thread/work-item just in case of barrier
  - triSYCL has no way to figure out there is a barrier or not

Example of CPU-unfriendly explicit work-item in SYCL

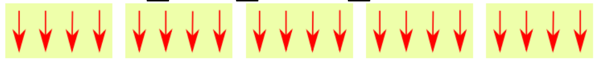
```
my_queue.submit([&] (handler &cgh) {
  // Use of local memory through an accessor
  using local_acc = cl::sycl::accessor<int, 1, cl::sycl::mode::read_write, cl::sycl::access::target::local>;
  local_acc local_accessor { cl::sycl::range<1> { 8 }, cgh };

  // Iterate over 8 work-groups of 8 work-items each
  cgh.parallel_for(nd_range<1> { range<1> { 8 }, range<1> { 8 } }, [=](nd_item<1> item) {
    int global_id = item.get_global(0);
    int local_id = item.get_local(0);
    // Fill the local memory
    local_accessor[local_id] = global_id;

    // Synchronise between work-items (bad on CPU)
    item.barrier(access::fence_space::local);          ☠

    // Use the memory filled before
    for (unsigned i = local_id - 1; i <= local_id + 1; i++) {
      if (i > 0 && i < item.get_local_range().size())
        output[global_id] += local_accessor[i];
    }
  });
});
```

∃ CPU-friendly hierarchical parallelism in SYCL

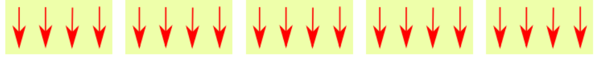
```
my_queue.submit([&](handler &cgh) {
  // Issue 5 work-groups of 4 work-items each
  cgh.parallel_for_work_group(range<1> { 5 }, range<1> { 4 }, [=](group<1> myGroup) {
```



```
    // [work-group code]
    // Variable shared between work-items
    int myLocal[4];
    // Issue parallel sets of 4 work-items
    parallel_for_work_item(myGroup, [=](item<1> myItem) {
```



```
      myLocal[myItem.get(0)] = myItem.get_linear_id();
    });
    // Implicit barrier here
    // Carry value across loops
    // Issue parallel sets of 4 work-items
    parallel_for_work_item(myGroup, [=](item<1> myItem) {
```



```
      // [work-item code]
      auto local_id = myItem.get(0);
      for (unsigned i = local_id - 1; i <= local_id + 1; i++) {
        if (i > 0 && i < myGroup.get(0))
          output[myItem.get_linear_id()] += myLocal[i];
      }
    });
  });
});
```

What’s Next ?

- Finish setting up Jenkins node in Google infrastructure
  - Insure that other commits do not break SYCL & triSYCL port
  - Already a ComputeCpp node
  - Issue with firewall at Xilinx for now
- Improve triSYCL/HLS/SDAccel integration
- Continue integrating OpenCL interoperability in Eigen/Tensorflow
- Optimise the host execution further
- Add FPGA-tailored features inside SYCL & TensorFlow
  - Arbitrary precision and fixed point types

Conclusion

- Upstreamed TensorFlow can use CUDA and SYCL for accelerators
- TensorFlow SYCL opens TensorFlow to Khronos realm
- SYCL brings pure modern C++ abstraction for heterogeneous computing
- Codeplay ComputeCpp is the way to go for TensorFlow SYCL on GPU
- But open-source triSYCL is making progress…
  - triSYCL for CPU: 78 failing tests among 1152
  - triSYCL for accelerators & FPGA: not mature enough for direct Eigen & TensorFlow
  - triSYCL OpenCL native node allows explicit access to OpenCL kernels
    - Enable TensorFlow interconnection with OpenCL-compatible accelerators
    - Can be defined as source, binary or built-in kernels
    - Allows connection with optimized OpenCL ABI machine-learning libraries

XILINX
ALL PROGRAMMABLE™