Standards-based stack for AI and HPC

**OpenCL**: C standard for accelerator programming. Widely used in machine vision.

**SYCL**: C++ standard for accelerator programming suitable for CUDA developers and big deep learning frameworks.

Combined standards give a huge ecosystem of deep learning and vision software.

<table>
<thead>
<tr>
<th>OpenCL-ecosystem</th>
<th>SYCL-ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCV</td>
<td>clBlast</td>
</tr>
<tr>
<td>ComputeCpp</td>
<td>triSYCL</td>
</tr>
</tbody>
</table>

- **SYCL**: Intel’s new open-source SYCL

**OpenCL**: C standard for accelerator programming. Widely used in machine vision.

**Processor-vendor-provided OpenCL**

**Open-source**: Pocl, Intel Neo, GPUopen

**AI processors**

<table>
<thead>
<tr>
<th></th>
<th>GPUs</th>
<th>FPGAs</th>
<th>DSPs</th>
<th>CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenCL</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SYCL</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OpenCL-ecosystem</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SYCL-ecosystem</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2019 Codeplay Software Ltd
Why OpenCL?

• OpenCL is supported on a huge range of processors
• OpenCL has a large computer vision and HPC ecosystem
• SPIR-V in OpenCL enables new accelerated languages
  • SPIR-V is a standardized compiler IR that enables e.g. AI graph compilers
• OpenCL can support lots of special-purpose accelerators
  • And we’re working to enable support for even more accelerator processors
• Gives access to a huge and growing ecosystem of software
• OpenCL is not performance-portable
  • But it enables performance-portable technologies on top
The SYCL Vision

• Bringing acceleration to mainstream C++ developers
• Make it easy to port from existing single-source programming models
• Supports a large ecosystem of open-source AI software
• Support the latest AI operators
• Enable C++ performance-portability techniques on top of OpenCL
Growing SYCL ecosystem

- [http://sycl.tech](http://sycl.tech)
- Growing open-source ecosystem
- SYCL-BLAS, SYCL-DNN, SYCL-ML
- TensorFlow, Eigen
- Renesas, Imagination Technologies, ARM, Intel, Xilinx, Red Hat
Porting from CUDA to SYCL

- Hand-port CUDA code to SYCL
  - Guides provided on Codeplay website

- Use a CUDA-to-SYCL refactoring tool
  - Currently in-development by Institute for Software (the Cevelop developers)

- Create a combined C++, CUDA, SYCL source file
  - Used in Eigen and TensorFlow
At the Institute for Software we have been developing plug-ins for Eclipse CDT for almost a decade. Some of this work has been contributed back to Eclipse (e.g. the majority of refactorings and the underlying infrastructure for writing refactorings), others like Linticator are available commercially, but the majority is open source (e.g. Metriculator) or free to use (e.g. CUTE). Over time, as more and more plug-ins were created, it became tedious to discover and install them one by one. So we decided to bundle them all into one single download, including the latest release of Eclipse CDT.
CUDA version

```cpp
#include <iostream>
#include <math.h>

// function to add the elements of two arrays
__global__ void add(int n, float *x, float *y)
{
    int index = threadIdx.x;
    int stride = blockDim.x;
    for (int i = index; i < n; i += stride)
        y[i] = x[i] + y[i];
}

cudaMallocManaged(&x, N*sizeof(float));
cudaMallocManaged(&y, N*sizeof(float));

add <<<1, 256 >>>(N, x, y);

cudaDeviceSynchronize();
```

SYCL version

```cpp
#include <CL/sycl.hpp>
#include <iostream>
#include <math.h>

// function to add the elements of two arrays
void add(cl::sycl::nd_item<1> item, int n, float *x, float *y)
{
    int index = item.get_local(0);
    int stride = item.get_local_range(0);
    for (int i = index; i < n; i += stride)
        y[i] = x[i] + y[i];
}

c::sycl::buffer<float> x(N);
c::sycl::buffer<float> y(N);

gpu_queue.submit([&](cl::sycl::handler &cgh) {
    auto aX = x.get_access<cl::sycl::access::mode::read>(cgh);
    auto aY = y.get_access<cl::sycl::access::mode::read_write>(cgh);
    cgh.parallel_for<class add_functor>(
        cl::sycl::nd_range<1>(cl::sycl::range<1>(256),
        cl::sycl::range<1>(256)),
        [=](cl::sycl::nd_item<1> it) {
            add(it, N, aX.get_pointer(), aY.get_pointer());
        });
});
```
CUDA to SYCL: Combined Source File

```c++
#include <CL/sycl.hpp>
#include <iostream>
#include <math.h>

// function to add the elements of two arrays
#ifdef __SYCL__
void add(cl::sycl::nd_item<1> item, int n, float *x, float *y)
{
    int index = item.get_local(0);
    int stride = item.get_local_range(0);
    __global__ void add(int n, float *x, float *y)
    {
        int index = threadIdx.x;
        int stride = blockDim.x;
        #else
        for (int i = index; i < n; i += stride)
            y[i] = x[i] + y[i];
    }
#else
    cl::sycl::buffer<float> x(N);
    cl::sycl::buffer<float> y(N);
    #else
    cudaMemcpyManaged(&x, N*sizeof(float));
    cudaMemcpyManaged(&y, N*sizeof(float));
    #endif
    gpu_queue.submit([&](cl::sycl::handler &cgh) {
        auto aX = x.get_access<cl::sycl::access::mode::read>(cgh);
        auto aY = y.get_access<cl::sycl::access::mode::read_write>(cgh);
        cgh.parallel_for<class add_functor>(
            cl::sycl::nd_range<1>(cl::sycl::range<1>(256),
            cl::sycl::range<1>(256)),
            [=](cl::sycl::nd_item<1> it) {
                add(it, N, aX.get_pointer(), aY.get_pointer());
            });
    });
#endif
```
# CUDA to SYCL: Modular Source Files

- Separation of device-specific details into a header file
- Core AI code in source files without device-specific code
- “Separation of Concerns”
- Standard C++ software engineering approach
Overhead of SYCL on top of OpenCL

Babelstream v2.0 by Bristol University benchmarks (Results from 2017)

Performance (Mbytes/sec - higher is better)

Current performance
- Generally, native CUDA/HIP performance can be expected
- Slightly larger kernel launch latency due to task graph overhead
  Not important for longer kernel runtimes (larger problems or slower GPUs)

Babelstream hipSYCL benchmarks

Source: http://uob-hpc.github.io/BabelStream
SYCL-BLAS

• Open-source BLAS library written using SYCL (like cuBLAS)
  • Similar C++ style to NVIDIA’s new CUTLASS tech https://github.com/NVIDIA/cutlass
  • Collaboration with BLAS experts at Universitat Jaume I, Castellion
  • Supports: Intel GPU, PowerVR, R-Car, AMD GPU, ARM Mali, Intel CPU, +others
  • Supports *kernel fusion* for high performance on complex algorithms
    • Reduces memory bandwidth
  • Adapts to be optimal on different GPUs and AI processors
    • Use C++ templates to let the same source code adapt to get best performance
    • Adapts the algorithm to different memory sizes, types, vector widths
    • Adapts the algorithm to different tensor shapes and sizes

https://github.com/codeplaysoftware/sycl-blas
SYCL-BLAS Operations

• BLAS1: Simple operations, bandwidth limited, but with fusion
  • SCAL, AXPY, NRM2, DOT, SCAL2OP, IAMAX, IAMIN, SCAL3OP, AXPY3OP, ROT, SWAP, COPY, ASUM, ICAMAX, ICAMIN

• BLAS2: Matrix * vector
  • With and without transpose on the matrix
  • GEMV, GER, TRMV, SYMV, SYR, SYR

• BLAS3: Matrix multiplies
  • GEMM (Matrix multiply): With and without transpose on both arguments
  • Batch GEMM

• All operations can be templated by base datatype
SYCL-BLAS Empirical Optimization

• Some processors have hand-optimized libraries integrated into SYCL-BLAS for specific operations

• Can tune the algorithms per-processor:
  • With and without on-chip local memory, work-group size, double buffering, avoid bank conflicts in on-chip memory, cache-line-size, top-level-tile-size, block-level-tile-size

• We continuously run a huge number of benchmarks with different optimization parameters for different platforms and different matrix sizes & shapes
SYCL-BLAS Kernel fusion

• These operations are all fused on device, so intermediate data is stored on-chip to reduce off-chip memory bandwidth

```c
_axpy(ex, size, alpha, gpu_vX, strd, gpu_vY, strd);
_asum(ex, size, gpu_vY, strd, gpu_vR);
_dot(ex, size, gpu_vX, strd, gpu_vY, strd, gpu_vS);
_nrm2(ex, size, gpu_vY, strd, gpu_vT);
_iamax(ex, size, gpu_vY, strd, gpu_vImax);
_iamin(ex, size, gpu_vY, strd, gpu_vImin);
_rot(ex, size, gpu_vX, strd, gpu_vY, strd, _cos, _sin);
_dot(ex, size, gpu_vX, strd, gpu_vY, strd, gpu_vU);
_swap(ex, size, gpu_vX, strd, gpu_vY, strd);
```
SYCL-DNN

- Open-source SYCL convolution library
- Replacement for cuDNN
- Adapts the algorithm according to the GPU (or DSP, FPGA...)
- Different accelerators benefit from different algorithms for different sizes & shape of convolution
- Algorithms: direct, direct-tiled, im2col, Winograd, GEMM (for 1x1)
- Supports both training and inference
- 2D convolutions, max&average pooling, 2D depth-wise convolutions, Relu, Tanh

https://github.com/codeplaysoftware/SYCL-DNN
SYCL-DNN: Auto-tuning per-processor

SYCL-DNN runs 5 algorithms across a range of common convolution sizes and shapes on each processor it supports and “learns” the best algorithm for each size and shape.

SYCL-DNN runs a range of different tile sizes and shapes on each algorithm and on each processor to “learn” the best tile sizes & shapes for each convolution.
Eigen

- Eigen is a C++ linear algebra library
- Open-source
- Developed by INRIA and Google
- Used in TensorFlow
- Supports kernel-fusion to reduce bandwidth
- Supports: CPU vectorization, CUDA and SYCL
TensorFlow

- Full-featured, open-source TensorFlow port: Keras, Tensorboard
- C++ and Python interface
- Training and inference
- Runs latest networks (e.g. Mask-RCNN)
What’s next?

OpenCL 1.2 & SYCL 1.2.1
• Enables AI & HPC software on large range of processor types
• Open conformance tests

SYCL 2019
• Support for a larger range of AI processors
• More developer-friendly features

Grow the ecosystem: more open-source, fill the gaps in the standards
• HPC, Deep learning, Computer vision

ISO C++
• Bring heterogeneous acceleration to the full ISO C++ standard
• MISRA C++ for automotive safe AI
Grow the open AI ecosystem with us!