codeplay®

Enabling AI to be Open, Safe & Accessible to All

SYCL in AI

Andrew

Mehdi

Alastair

Gordon

Codeplay Software

10 November 2021

Outline

- **Introduction: How it fits together**: Andrew Richards
- **From AI graph to SYCL/oneAPI**: Mehdi Goli (VP R&D)
  - ONNX flow: Simplified so you can see how it works
  - TensorFlow+Eigen flow: Large scale demonstration
- **From SYCL to the hardware**: Alastair Murray (VP Product)
  - Using a simple sample "AI accelerator" based on RISC-V cores
- **SYCL/oneAPI in Exascale HPC**: Gordon Brown (Principal PO)
  - This is using existing supercomputer GPUs from multiple vendors
- **Q&A**

The stack: deep-dive

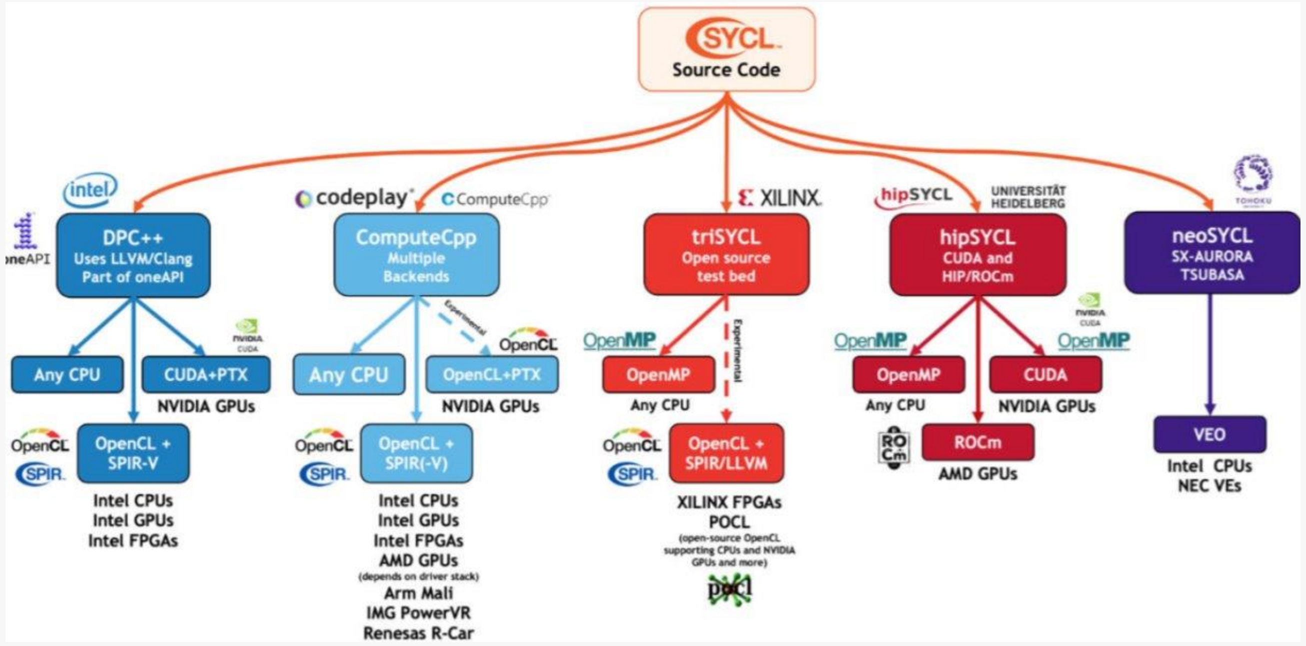
- We are going to go top-to-bottom from AI software down to heterogeneous hardware
- We are combining *scale, customization* and *performance*
  - **Scale**: supporting a large range of operations and large number of developers
  - **Customization**: you can write your own operations & design your own chips
  - **Performance**: you can use the full hardware features to achieve performance
- The examples will all be simplified:
  - Simplified AI software: so you can see how to write your own software
  - Simplified AI hardware: so you can see how to map to your own hardware

SYCL

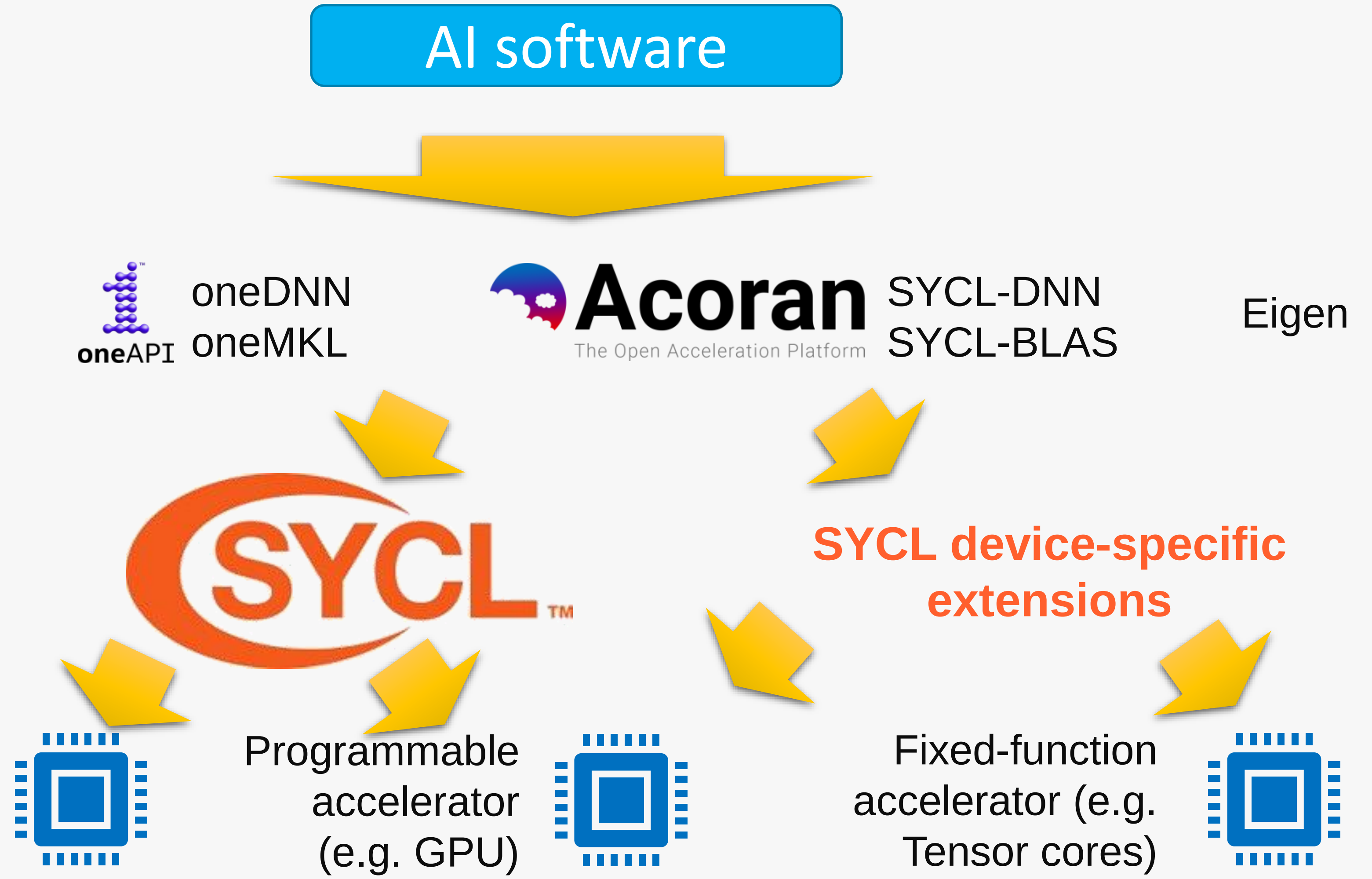
- Open standard API introduced by Khronos
  - Uses ISO standard C++ code
- Provides single-source programming model for accelerator processors
- Allow accessing both high-level and low-level code
- Suitable for graph model programming by tracking kernel dependency
- Multiple implementations
  - ComputeCPP
  - DPC++
  - hipSYCL
- Programming model
  - Kernel Scope
  - Command group scope
  - Application Scope

Support for different hardware

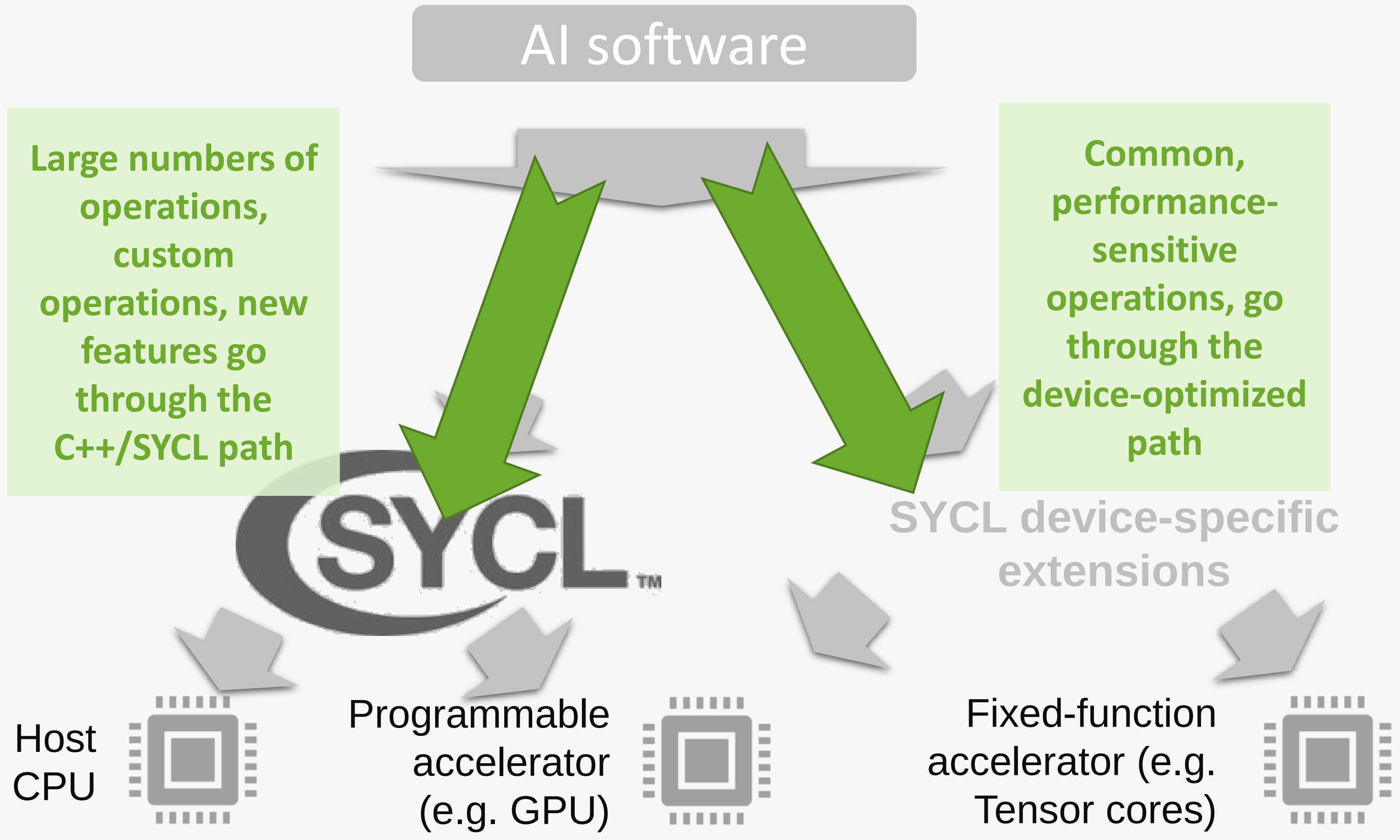
- Allows to use multiple implementations of SYCL
- Supports a wide variety of hardware
- Allows to reuse the optimized kernels across new platforms


SYCL Source Code
intel
oneAPI
DPC++ Uses LLVM/Clang Part of oneAPI
Any CPU
CUDA+PTX
NVIDIA GPUs
OpenCL + SPIR-V
Intel CPUs Intel GPUs Intel FPGAs
codeplay ComputeCpp
ComputeCpp Multiple Backends
Experimental
Any CPU
OpenCL+PTX
NVIDIA GPUs
OpenCL + SPIR(-V)
Intel CPUs Intel GPUs Intel FPGAs AMD GPUs (depends on driver stack) Arm Mali IMG PowerVR Renesas R-Car
XILINX
triSYCL Open source test bed
OpenMP
Any CPU
Experimental
OpenCL + SPIR/LLVM
XILINX FPGAs POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more)
pocl
hipSYCL
UNIVERSITÄT HEIDELBERG
hipSYCL CUDA and HIP/ROCm
OpenMP
Any CPU
CUDA
NVIDIA GPUs
ROCm
AMD GPUs
TOHOKU
neoSYCL SX-AURORA TSUBASA
VEO
Intel CPUs NEC VEs

A SYCL-Based AI Stack


Standards-based software
AI software
C++ and integrated libraries
oneAPI
oneDNN
oneMKL
Acoran
The Open Acceleration Platform
SYCL-DNN
SYCL-BLAS
Eigen
C++ platform
SYCL
SYCL device-specific extensions
Hardware
Host CPU
Programmable accelerator (e.g. GPU)
Fixed-function accelerator (e.g. Tensor cores)

Combining Scale with Performance


AI software
Large numbers of operations, custom operations, new features go through the C++/SYCL path
Common, performance-sensitive operations, go through the device-optimized path
SYCL
SYCL device-specific extensions
Host CPU
Programmable accelerator (e.g. GPU)
Fixed-function accelerator (e.g. Tensor cores)

What Codeplay Provide

- Ecosystem services:
  - Developer support & training for industry standards
  - Free tools to help build out an open ecosystem
  - Open-source software development & maintenance
- Silicon enablement:
  - Modular technology that lets you accelerate the ecosystem on your hardware
  - Documentation to show you how to map your hardware to the ecosystem
  - Services to help you with integration
- Functional safety:
  - To help bring this acceleration technology to applications that require safety

From AI graph to SYCL/oneAPI

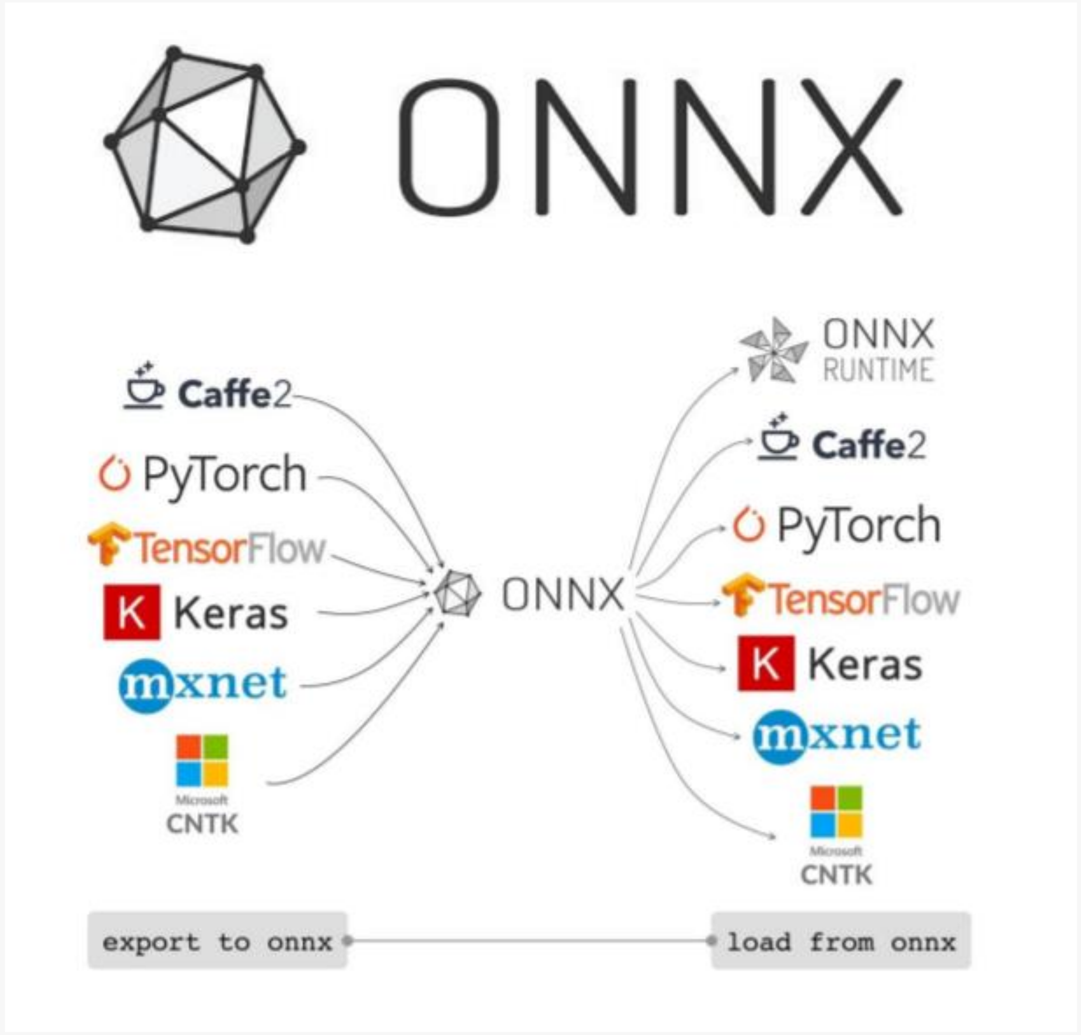
Mehdi Goli (VP R&D)

Deep Learning Challenges

- **Diversity** of Technologies and Techniques
- **Migrating** between DL framework
- Multiple implementations of Same algorithms
- **Maintainability** of various version of low-level backend/libraries integrated in existing high-level frameworks
- **Hardcoded** implementation of Inference engines for a restricted set of hardware

Open Neural Network Exchange (ONNX)

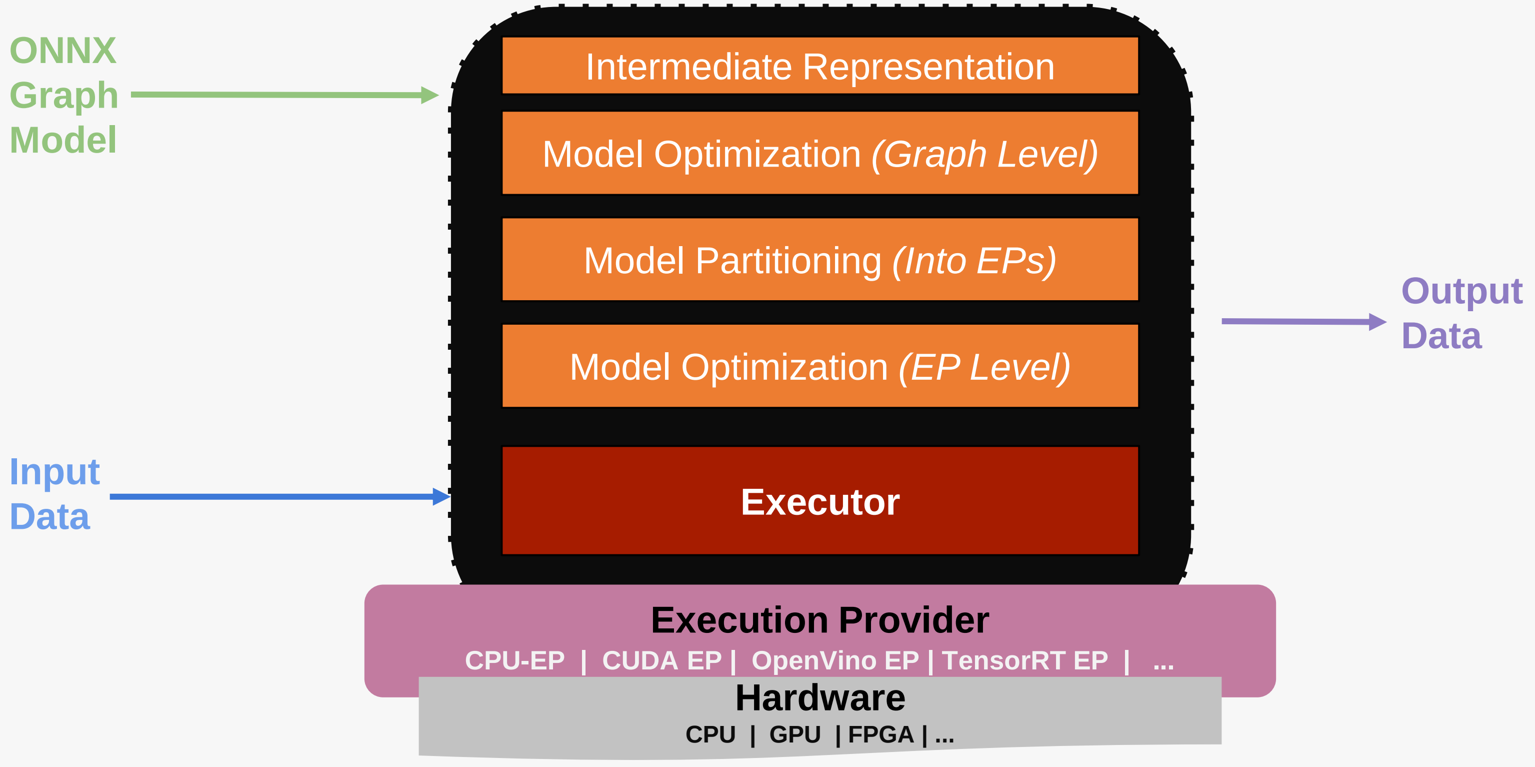
- Open-Source project for AI model
- Provide an **interoperable Open standard** format of ML and DL
- ONNX building Block
  - Computational Graph model
  - Standard date types
  - Built-in operators



ONNX Runtime

- ONNX Runtime
  - Implement ONNX specification as a runtime toolchain
  - Aims to Accelerate DL Inference
  - Targets high performance and interoperability across various platforms
  - Currently has a CPU, Cuda, OpenVINO, DNNL and TensorRT backend

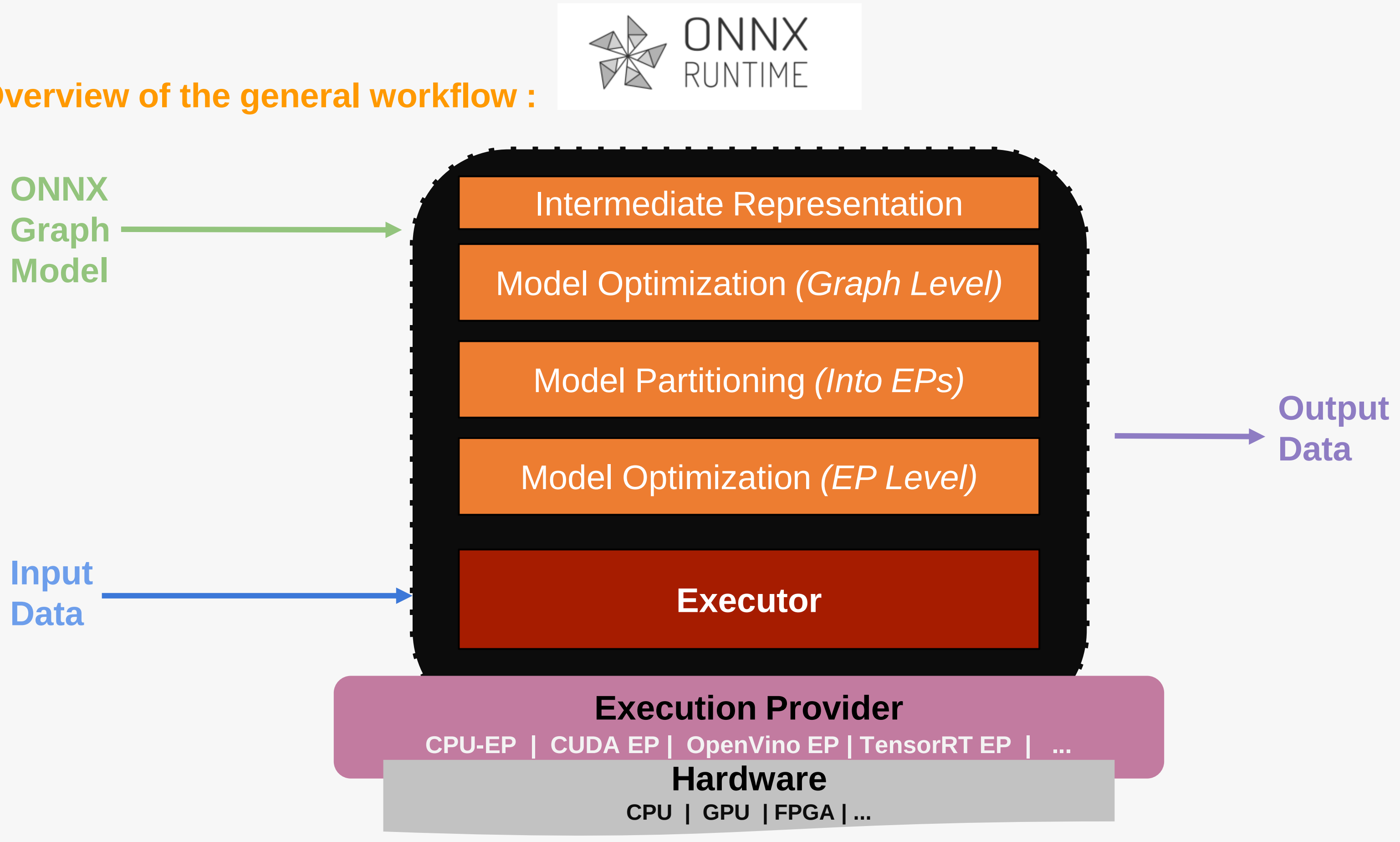
Overview of the general workflow :


ONNX Graph Model
Input Data
Output Data
Intermediate Representation
Model Optimization (Graph Level)
Model Partitioning (Into EPs)
Model Optimization (EP Level)
Executor
Execution Provider
CPU-EP | CUDA EP | OpenVino EP | TensorRT EP | ...
Hardware
CPU | GPU | FPGA | ...

ONNX Runtime

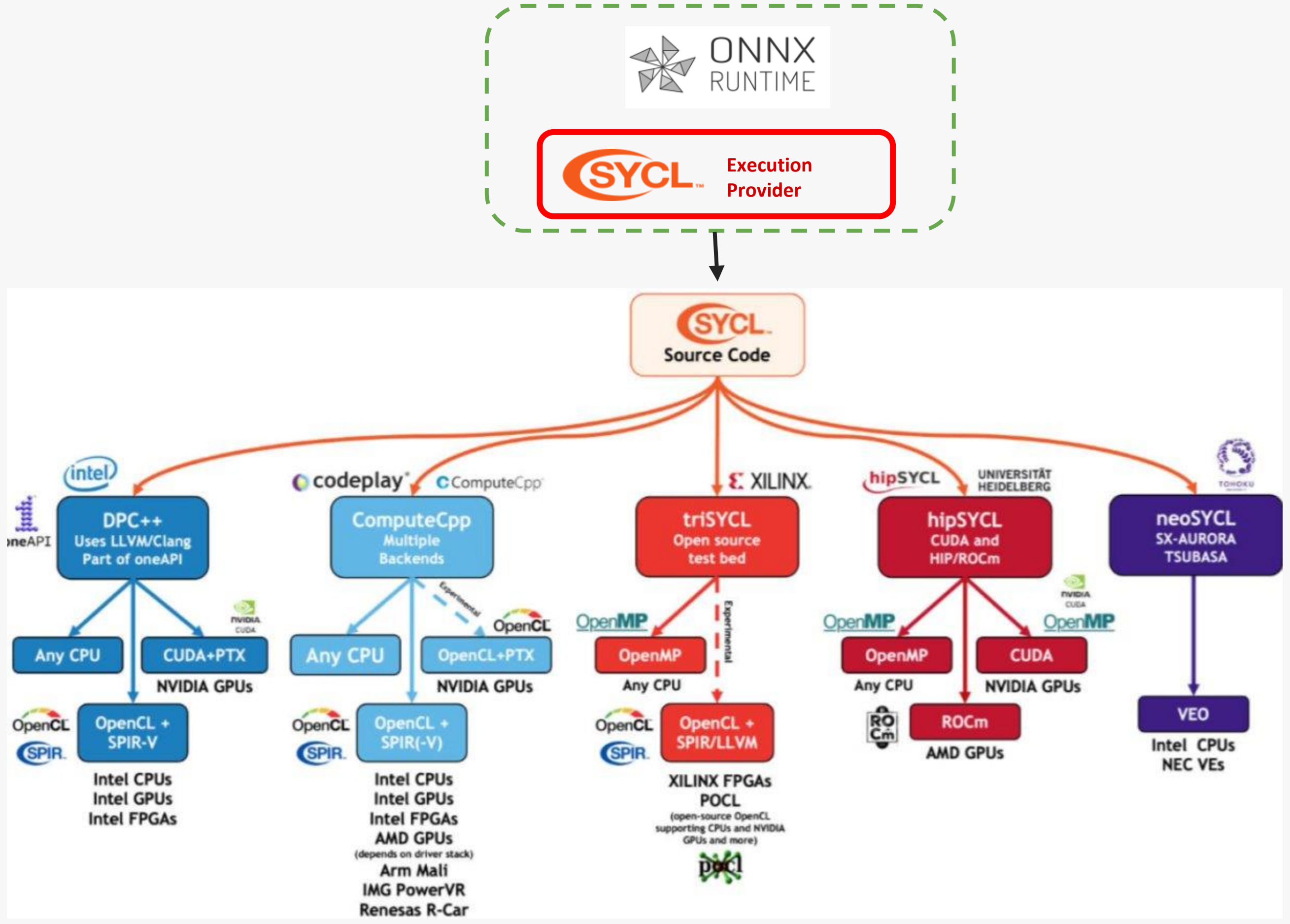
- Fallback onto the CPU execution provider for undefined operators
- Ability to add custom operations
- Graph level optimizations to improve the overall network
- Need to implement all the operators when supporting a new platform
- Inability to use existing optimized kernels across different platforms

Overview of the general workflow :


ONNX RUNTIME
ONNX Graph Model
Intermediate Representation
Model Optimization *(Graph Level)*
Model Partitioning *(Into EPs)*
Model Optimization *(EP Level)*
Input Data
Executor
Output Data
Execution Provider
CPU-EP | CUDA EP | OpenVino EP | TensorRT EP | ...
Hardware
CPU | GPU | FPGA | ...

SYCL Execution provider

- Allows to use multiple implementations of SYCL
- Allows to reuse the optimized kernels across new platforms
- Allows to use existing optimized libraries like **Eigen-SYCL**, **SYCL-BLAS**, **SYCL-DNN**, **oneMKL**, and **oneDNN**
- Takes advantage of the graph optimizations available in ONNX Runtime

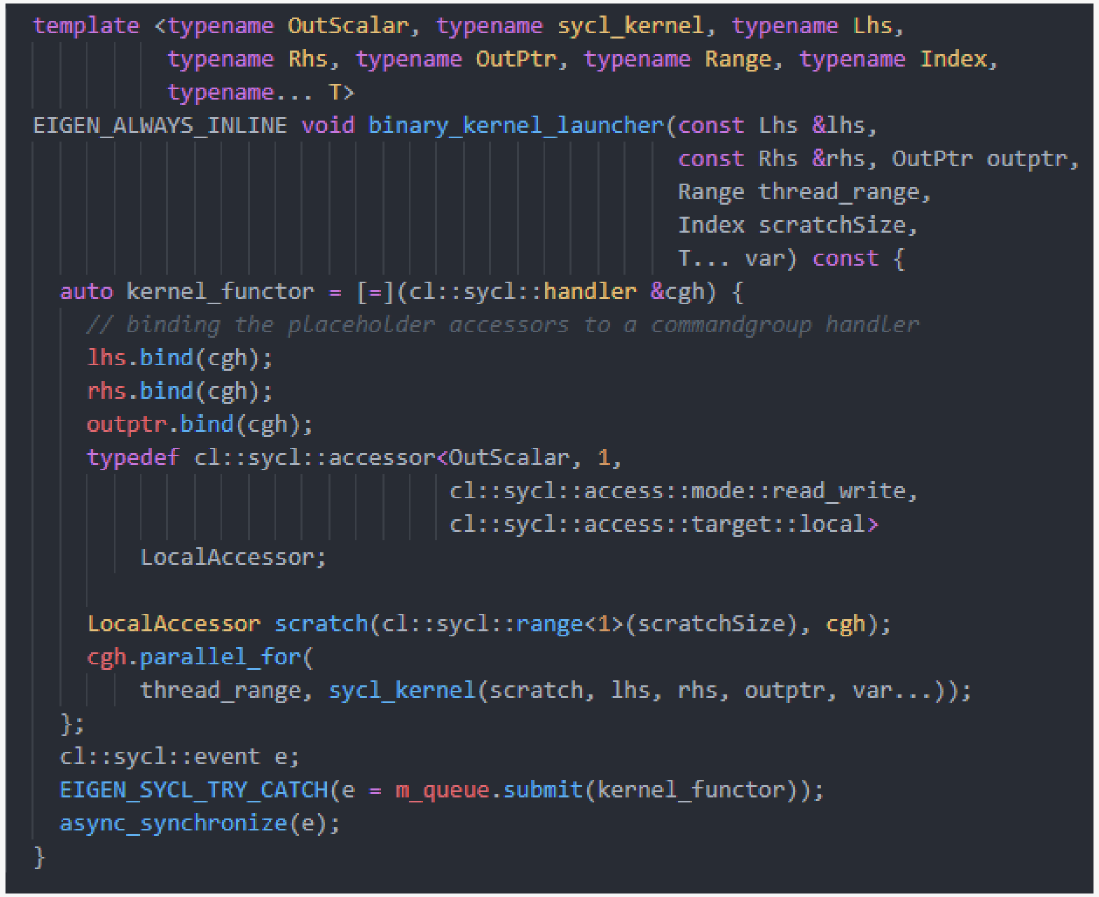


SYCL Features used for AI Graph

- SYCL programming model
  - C++ Template Meta programming
  - std::enable_if
  - If constexpr
  - Constexpr variable/functions
- SYCL runtime optimization
  - Scheduler DAG
  - Map a queue to multiple Internal low-level queue
- SYCL memory model
  - Place-holder accessor
  - On Chip Memory
  - USM-buffer interop
- Cross-platform performance portability
  - Reconfigurable SYCL Code
  - SYCL interoperability
    - OpenCL
    - CUDA

SYCL Programming Model

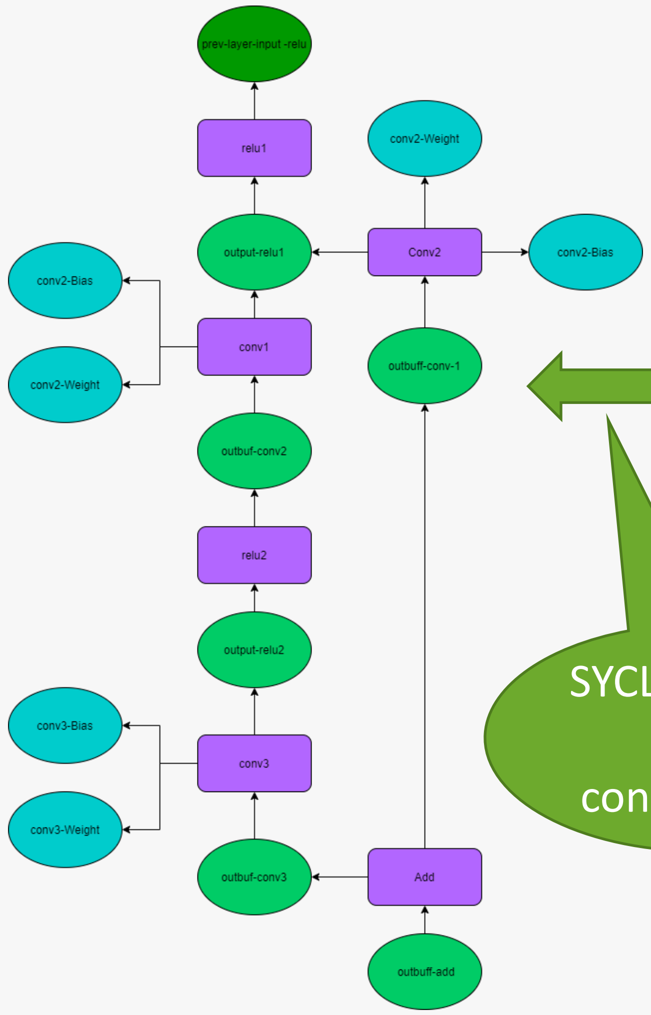
- C++ feature support is a must for Eigen framework integration.
- Eigen's kernels follow a heavily C++-template-based expression tree model
  - SYCL can dispatch device kernels from C++ applications, similar to CUDA, etc.
  - OpenCL 1.2 does not support C++
  - OpenCL 2.1 does support C++ templates inside the kernel
- Eigen uses the single-source programming model for both CUDA and CPU.
  - SYCL supports single-source programming style
  - Use the same existing template code for both host and device
  - OpenCL needs to re-implement the backend and maintaining will be difficult



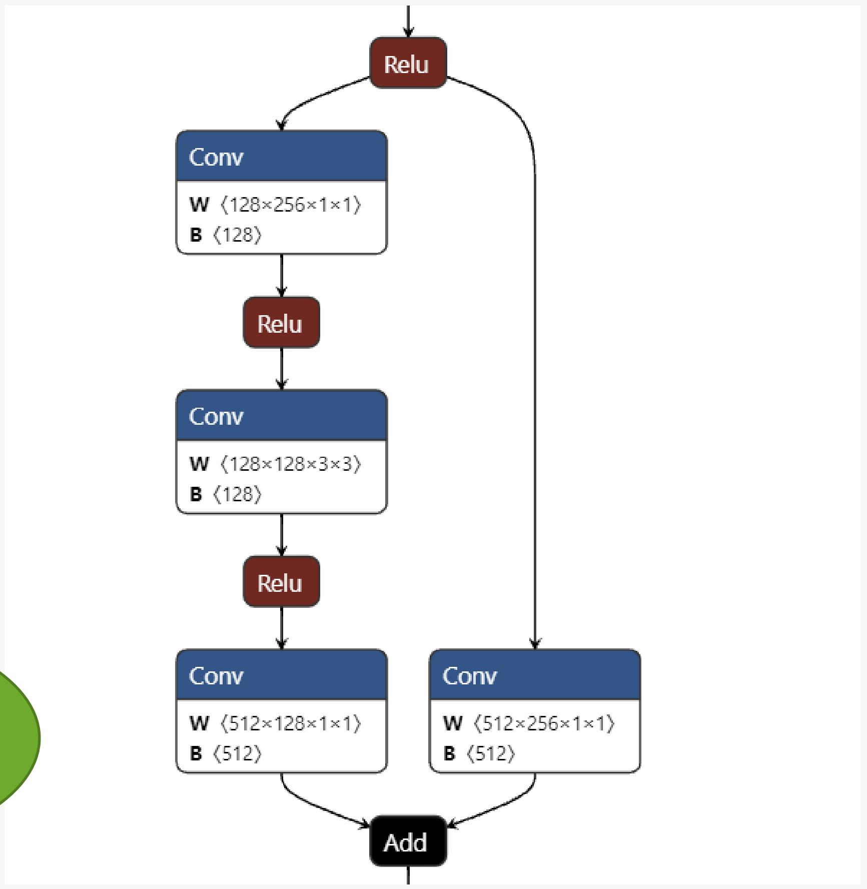
```
template <typename OutScalar, typename sycl_kernel, typename Lhs,
          typename Rhs, typename OutPtr, typename Range, typename Index,
          typename... T>
EIGEN_ALWAYS_INLINE void binary_kernel_launcher(const Lhs &lhs,
                                                const Rhs &rhs, OutPtr outptr,
                                                Range thread_range,
                                                Index scratchSize,
                                                T... var) const {
  auto kernel_functor = [=](cl::sycl::handler &cgh) {
    // binding the placeholder accessors to a commandgroup handler
    lhs.bind(cgh);
    rhs.bind(cgh);
    outptr.bind(cgh);
    typedef cl::sycl::accessor<OutScalar, 1,
                               cl::sycl::access::mode::read_write,
                               cl::sycl::access::target::local>
        LocalAccessor;

    LocalAccessor scratch(cl::sycl::range<1>(scratchSize), cgh);
    cgh.parallel_for(
        thread_range, sycl_kernel(scratch, lhs, rhs, outptr, var...));
  };
  cl::sycl::event e;
  EIGEN_SYCL_TRY_CATCH(e = m_queue.submit(kernel_functor));
  async_synchronize(e);
}
```

SYCL Runtime Optimization- Scheduler

- SYCL Data flow Graph
  - Buffer Memory only
  - Concurrent Execution of Conv2 and COnv1
- Frameworks
  - ONNX
  - TensorFlow
  - EIGEN
  - SYCL-BLAS
  - SYCL-DNN

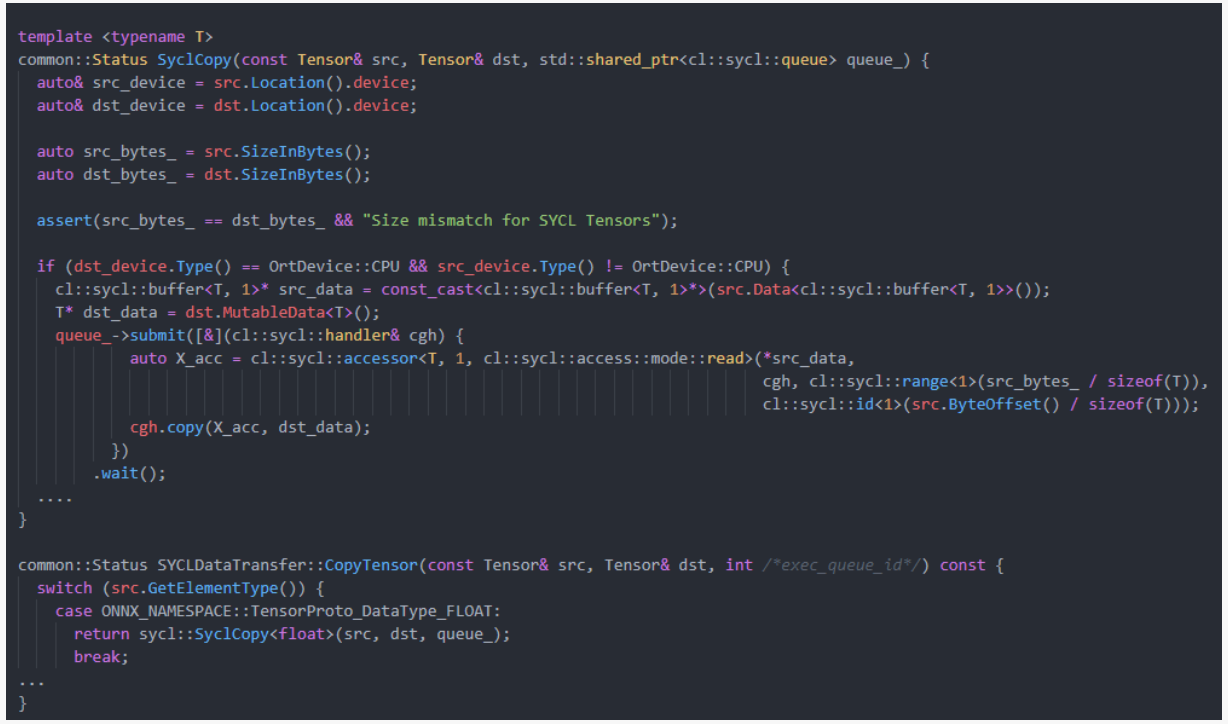


SYCL Runtime DAG construction



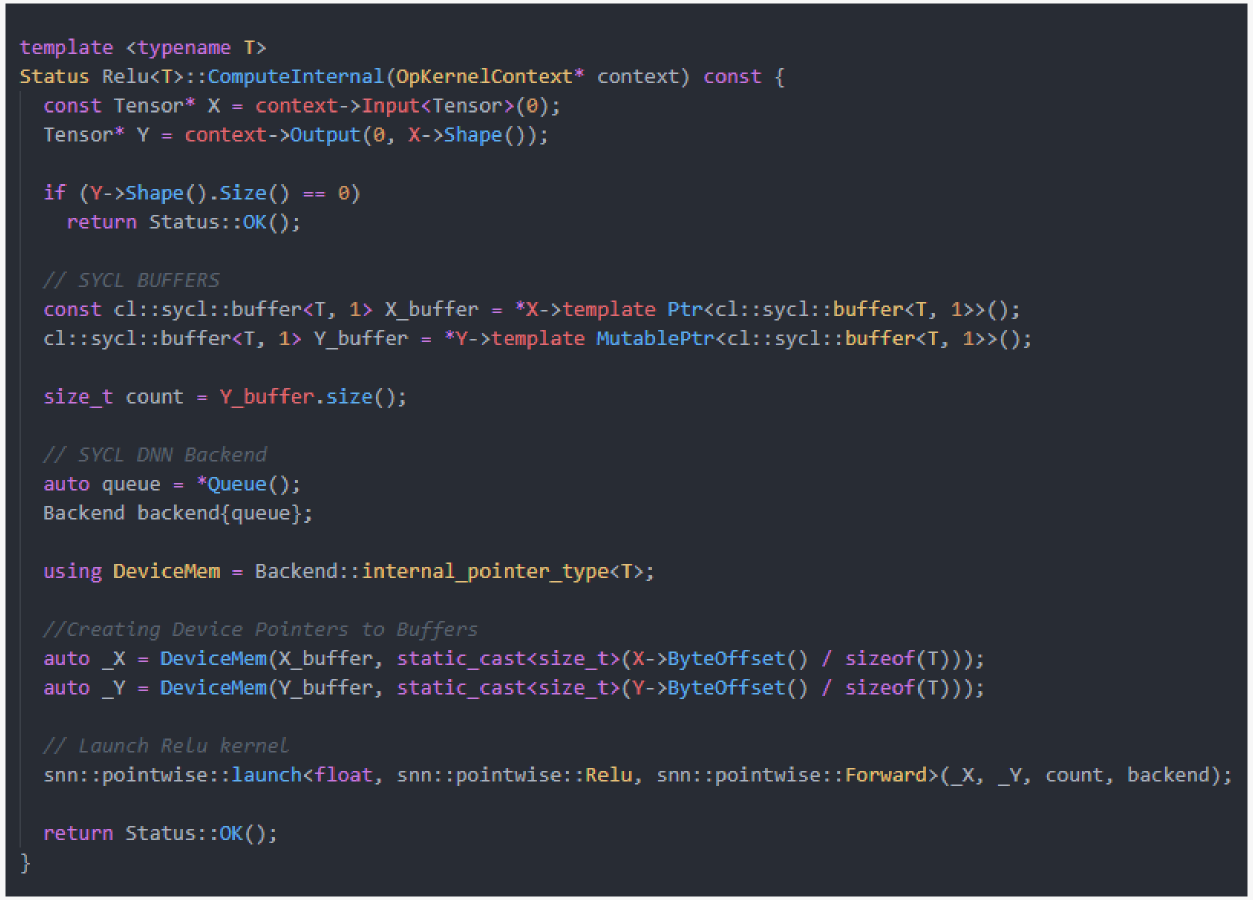
SYCL Runtime Optimization- Internal Queues

- A SYCL queue can map to multiple low-level queues
  - Seamless map from the user point of view
  - ComputeCpp
  - DPC++ CUDA backend (in Progress)



```cpp
template <typename T>
common::Status SyclCopy(const Tensor& src, Tensor& dst, std::shared_ptr<cl::sycl::queue> queue_) {
  auto& src_device = src.Location().device;
  auto& dst_device = dst.Location().device;

  auto src_bytes_ = src.SizeInBytes();
  auto dst_bytes_ = dst.SizeInBytes();

  assert(src_bytes_ == dst_bytes_ && "Size mismatch for SYCL Tensors");

  if (dst_device.Type() == OrtDevice::CPU && src_device.Type() != OrtDevice::CPU) {
    cl::sycl::buffer<T, 1>* src_data = const_cast<cl::sycl::buffer<T, 1>*>(src.Data<cl::sycl::buffer<T, 1>>());
    T* dst_data = dst.MutableData<T>();
    queue_->submit([&](cl::sycl::handler& cgh) {
            auto X_acc = cl::sycl::accessor<T, 1, cl::sycl::access::mode::read>(*src_data,
                                                                                  cgh, cl::sycl::range<1>(src_bytes_ / sizeof(T)),
                                                                                  cl::sycl::id<1>(src.ByteOffset() / sizeof(T)));
            cgh.copy(X_acc, dst_data);
          })
        .wait();
  ....
}

common::Status SYCLDataTransfer::CopyTensor(const Tensor& src, Tensor& dst, int /*exec_queue_id*/) const {
  switch (src.GetElementType()) {
    case ONNX_NAMESPACE::TensorProto_DataType_FLOAT:
      return sycl::SyclCopy<float>(src, dst, queue_);
      break;
...
}
```

Data Transfer Operation

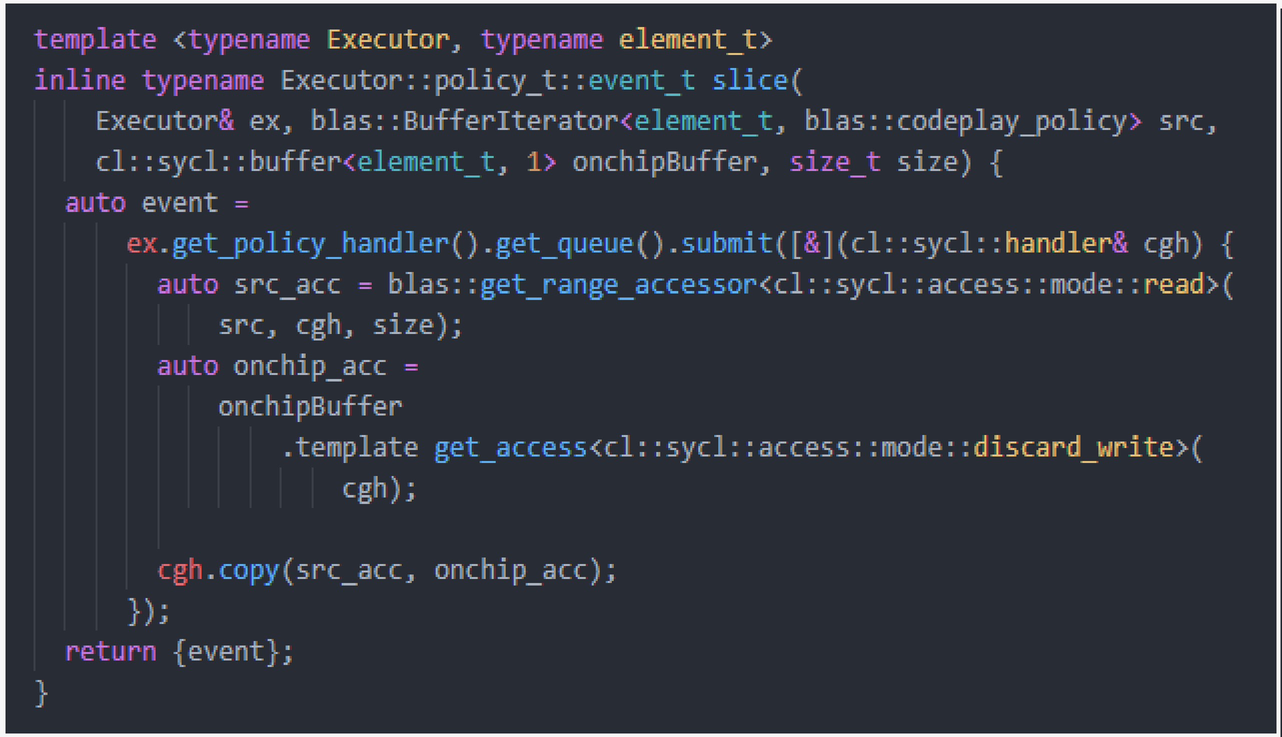


```cpp
template <typename T>
Status Relu<T>::ComputeInternal(OpKernelContext* context) const {
  const Tensor* X = context->Input<Tensor>(0);
  Tensor* Y = context->Output(0, X->Shape());

  if (Y->Shape().Size() == 0)
    return Status::OK();

  // SYCL BUFFERS
  const cl::sycl::buffer<T, 1> X_buffer = *X->template Ptr<cl::sycl::buffer<T, 1>>();
  cl::sycl::buffer<T, 1> Y_buffer = *Y->template MutablePtr<cl::sycl::buffer<T, 1>>();

  size_t count = Y_buffer.size();

  // SYCL DNN Backend
  auto queue = *Queue();
  Backend backend{queue};

  using DeviceMem = Backend::internal_pointer_type<T>;

  //Creating Device Pointers to Buffers
  auto _X = DeviceMem(X_buffer, static_cast<size_t>(X->ByteOffset() / sizeof(T)));
  auto _Y = DeviceMem(Y_buffer, static_cast<size_t>(Y->ByteOffset() / sizeof(T)));

  // Launch Relu kernel
  snn::pointwise::launch<float, snn::pointwise::Relu, snn::pointwise::Forward>(_X, _Y, count, backend);

  return Status::OK();
}
```

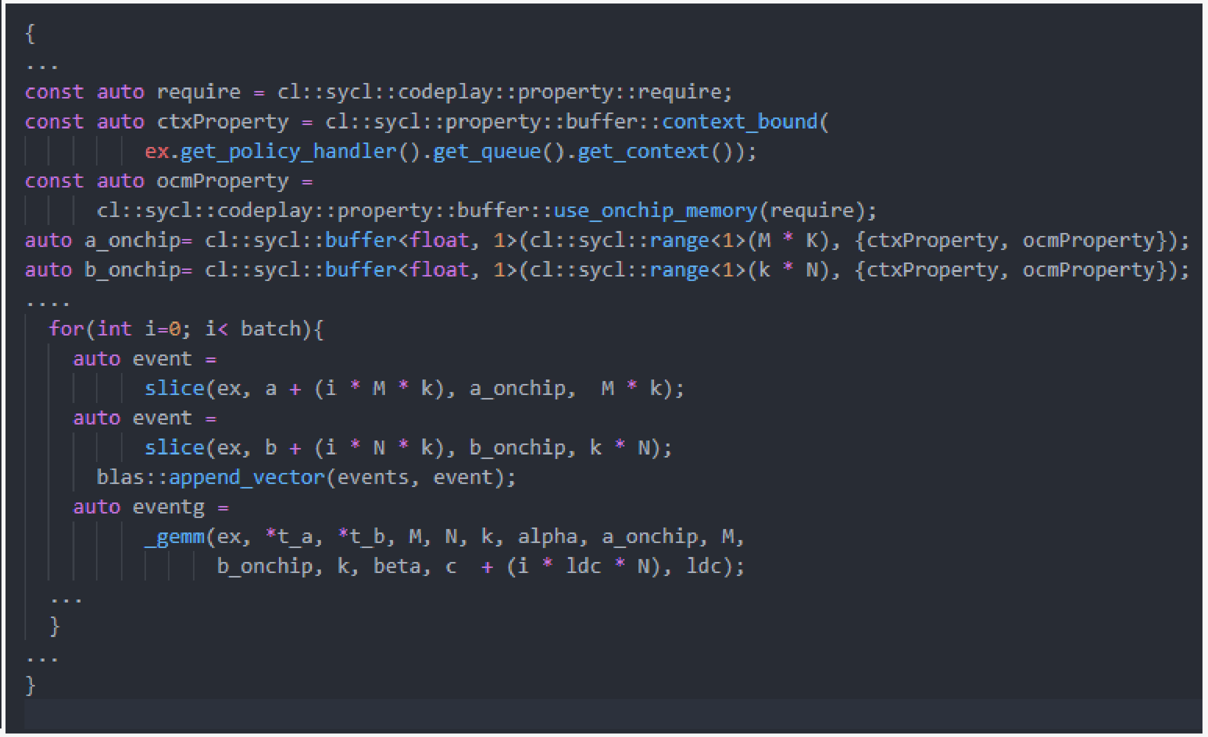
Launching Kernel Operations

SYCL Memory Model-OnChip Buffer

- Codeplay extension: Same SYCL copy function trigger the DMA transfer from an embedded device



```cpp
template <typename Executor, typename element_t>
inline typename Executor::policy_t::event_t slice(
    Executor& ex, blas::BufferIterator<element_t, blas::codeplay_policy> src,
    cl::sycl::buffer<element_t, 1> onchipBuffer, size_t size) {
  auto event =
      ex.get_policy_handler().get_queue().submit([&](cl::sycl::handler& cgh) {
        auto src_acc = blas::get_range_accessor<cl::sycl::access::mode::read>(
            src, cgh, size);
        auto onchip_acc =
            onchipBuffer
                .template get_access<cl::sycl::access::mode::discard_write>(
                    cgh);

        cgh.copy(src_acc, onchip_acc);
      });
  return {event};
}
```



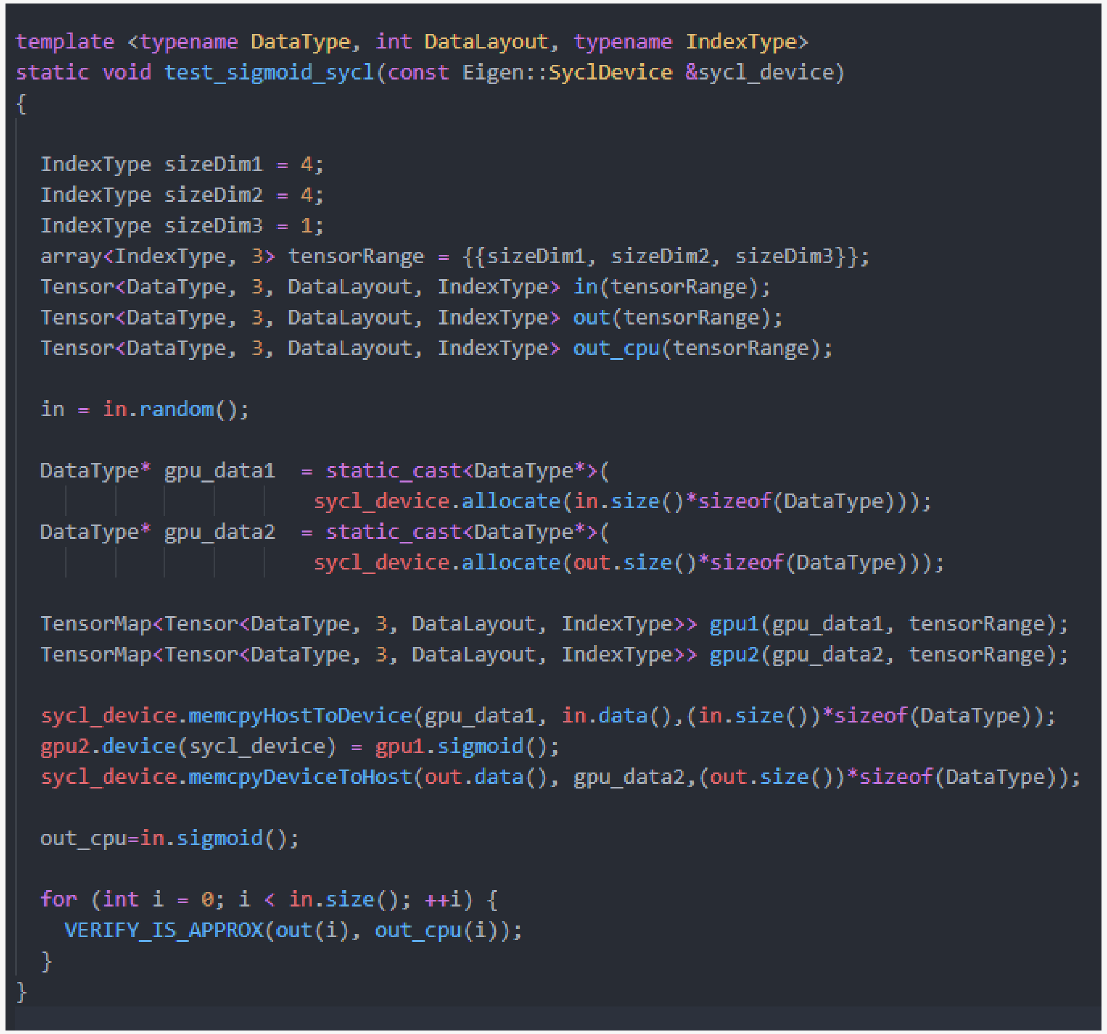
```cpp
{
...
const auto require = cl::sycl::codeplay::property::require;
const auto ctxProperty = cl::sycl::property::buffer::context_bound(
          ex.get_policy_handler().get_queue().get_context());
const auto ocmProperty =
      cl::sycl::codeplay::property::buffer::use_onchip_memory(require);
auto a_onchip= cl::sycl::buffer<float, 1>(cl::sycl::range<1>(M * K), {ctxProperty, ocmProperty});
auto b_onchip= cl::sycl::buffer<float, 1>(cl::sycl::range<1>(k * N), {ctxProperty, ocmProperty});
....
  for(int i=0; i< batch){
    auto event =
          slice(ex, a + (i * M * k), a_onchip,  M * k);
    auto event =
          slice(ex, b + (i * N * k), b_onchip, k * N);
      blas::append_vector(events, event);
    auto eventg =
          _gemm(ex, *t_a, *t_b, M, N, k, alpha, a_onchip, M,
                b_onchip, k, beta, c  + (i * ldc * N), ldc);
  ...
  }
...
}
```

SYCL Memory Model-USM Buffer Interop

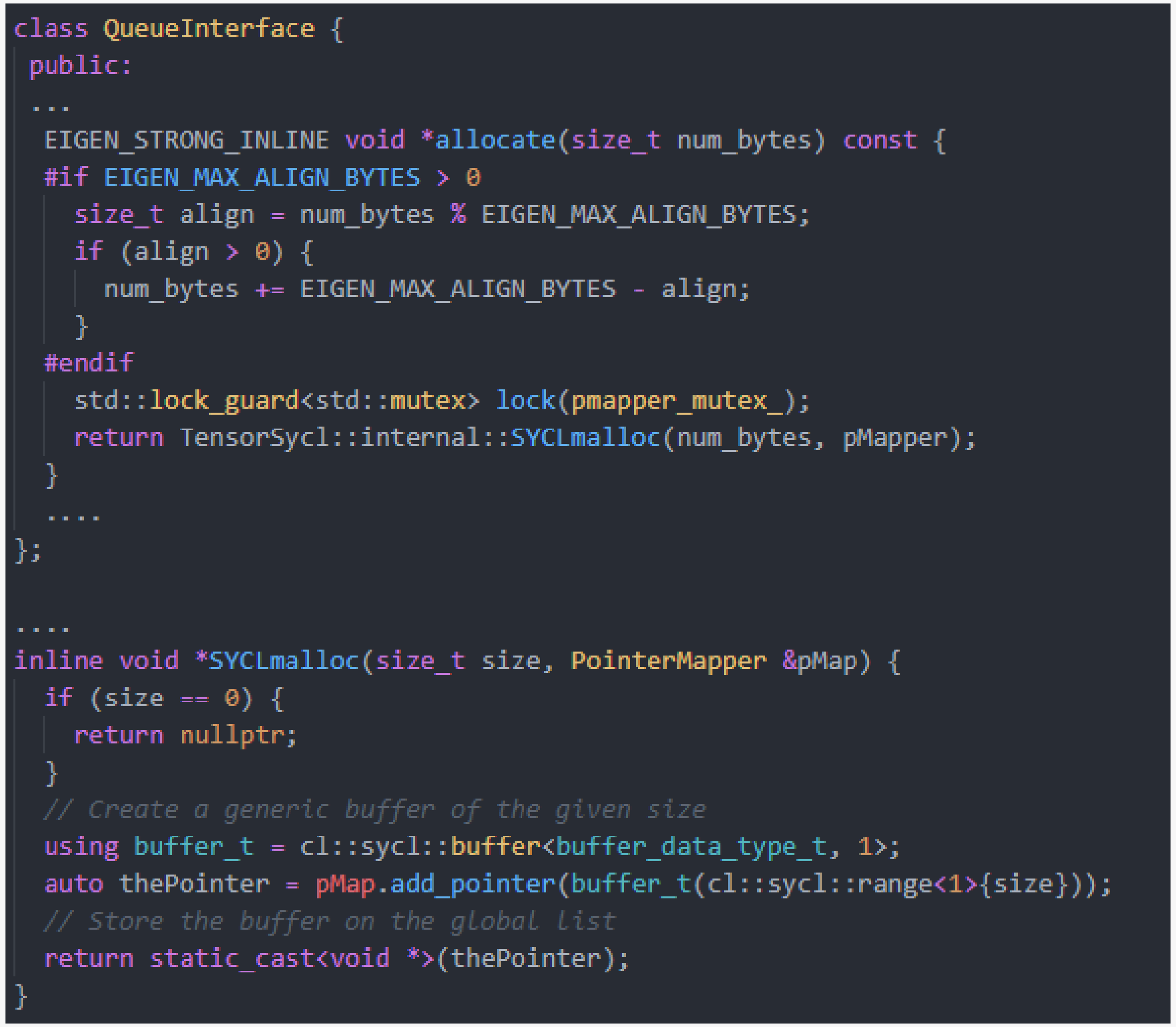
- USM -> C-style (CUDA-style) pointer
  - Pros:
    - Can be void*
    - Can be nullptr
    - Integrate easily with existing frameworks (TensorFlow/Eigen/ONNX/etc)
    - Support arithmetic operation on host
  - Cons:
    - User is responsible for Data Flow Dependency
- Buffer-> SYCL container
  - Cons:
    - Does not support Void*
    - Hard to integrate with existing framework
    - Cannot be nullptr
    - Cannot support arithmetic operation on Host
  - Pros:
    - SYCL runtime will take care of Data Flow Dependency
    - Better fit for embedded systems with custom allocator/hierarchy

- USM Buffer interop
  - Can be void*
  - Can be nullptr
  - Integrate easily with existing frameworks (TensorFlow/Eigen/ONNX/etc)
  - SYCL runtime will take care of Data Flow Dependency
  - Better fit for embedded systems with custom allocator/hierarchy

SYCL Memory Model-USM Buffer Interop



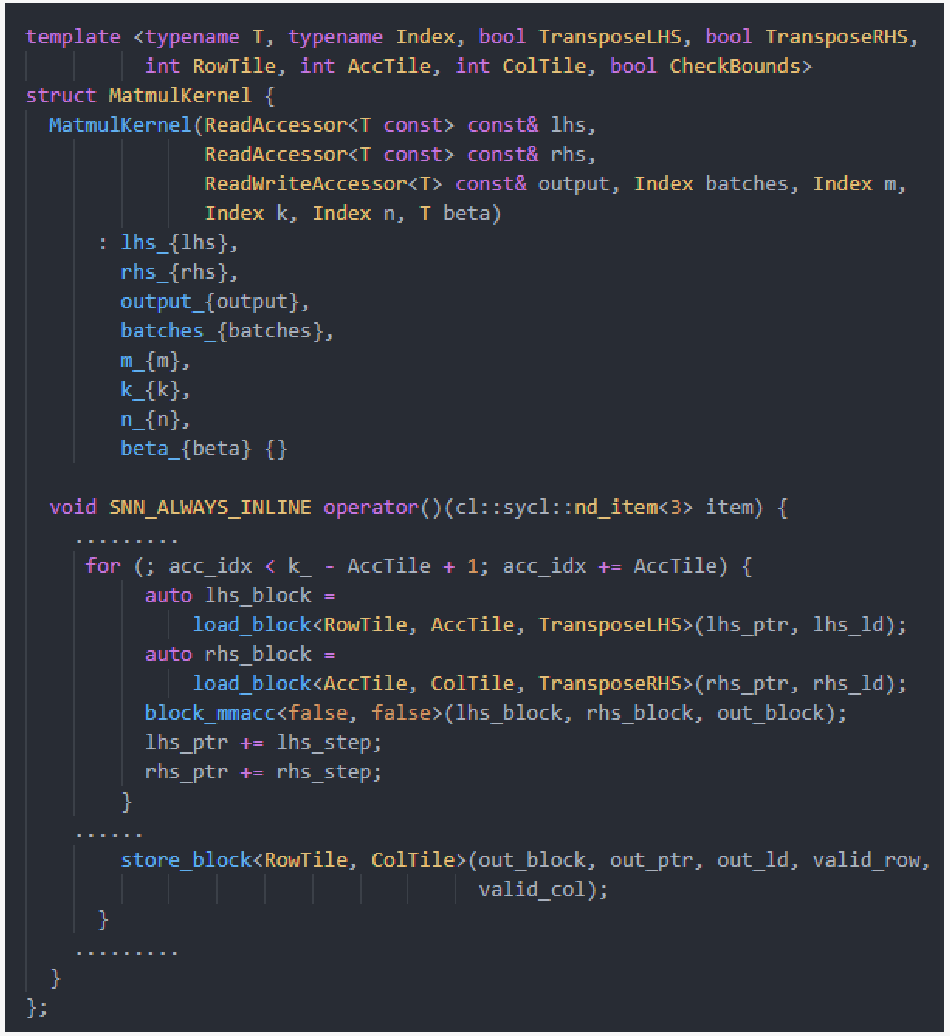
```
template <typename DataType, int DataLayout, typename IndexType>
static void test_sigmoid_sycl(const Eigen::SyclDevice &sycl_device)
{

  IndexType sizeDim1 = 4;
  IndexType sizeDim2 = 4;
  IndexType sizeDim3 = 1;
  array<IndexType, 3> tensorRange = {{sizeDim1, sizeDim2, sizeDim3}};
  Tensor<DataType, 3, DataLayout, IndexType> in(tensorRange);
  Tensor<DataType, 3, DataLayout, IndexType> out(tensorRange);
  Tensor<DataType, 3, DataLayout, IndexType> out_cpu(tensorRange);

  in = in.random();

  DataType* gpu_data1  = static_cast<DataType*>(
                        sycl_device.allocate(in.size()*sizeof(DataType)));
  DataType* gpu_data2  = static_cast<DataType*>(
                        sycl_device.allocate(out.size()*sizeof(DataType)));

  TensorMap<Tensor<DataType, 3, DataLayout, IndexType>> gpu1(gpu_data1, tensorRange);
  TensorMap<Tensor<DataType, 3, DataLayout, IndexType>> gpu2(gpu_data2, tensorRange);

  sycl_device.memcpyHostToDevice(gpu_data1, in.data(),(in.size())*sizeof(DataType));
  gpu2.device(sycl_device) = gpu1.sigmoid();
  sycl_device.memcpyDeviceToHost(out.data(), gpu_data2,(out.size())*sizeof(DataType));

  out_cpu=in.sigmoid();

  for (int i = 0; i < in.size(); ++i) {
    VERIFY_IS_APPROX(out(i), out_cpu(i));
  }
}
```



```
class QueueInterface {
 public:
 ...
  EIGEN_STRONG_INLINE void *allocate(size_t num_bytes) const {
  #if EIGEN_MAX_ALIGN_BYTES > 0
    size_t align = num_bytes % EIGEN_MAX_ALIGN_BYTES;
    if (align > 0) {
      num_bytes += EIGEN_MAX_ALIGN_BYTES - align;
    }
  #endif
    std::lock_guard<std::mutex> lock(pmapper_mutex_);
    return TensorSycl::internal::SYCLmalloc(num_bytes, pMapper);
  }
  ....
};

....
inline void *SYCLmalloc(size_t size, PointerMapper &pMap) {
  if (size == 0) {
    return nullptr;
  }
  // Create a generic buffer of the given size
  using buffer_t = cl::sycl::buffer<buffer_data_type_t, 1>;
  auto thePointer = pMap.add_pointer(buffer_t(cl::sycl::range<1>{size}));
  // Store the buffer on the global list
  return static_cast<void *>(thePointer);
}
```

SYCL as Cross-Platform Performance Portable Language

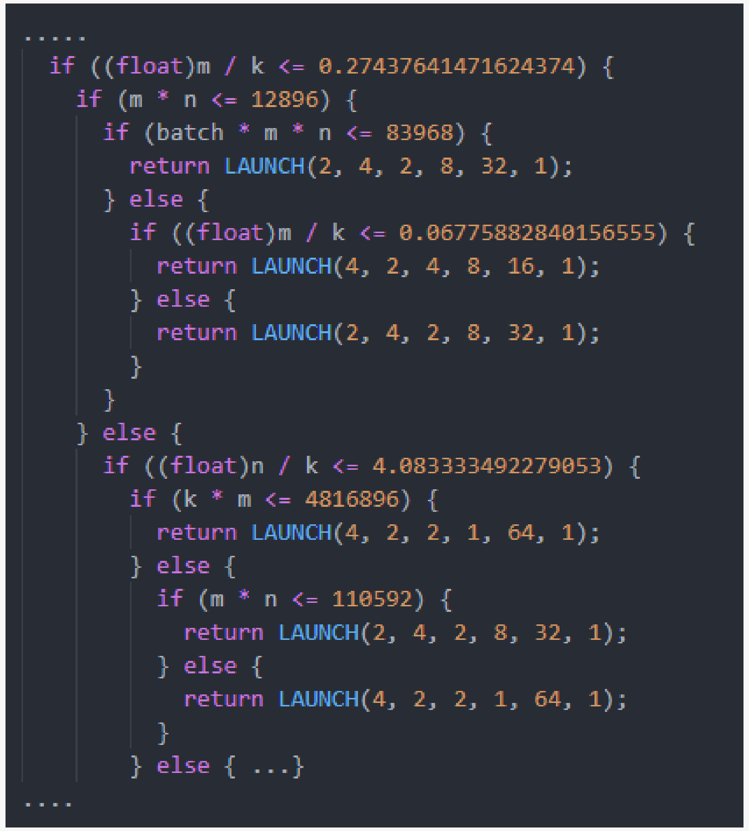
- Reconfigurable Kernels
  - Configurable parametric Tile
    - Register Usage
    - Device type
    - Cache line size
    - Item processed per thread
    - Subgroup size
  - Building blocks
    - Load
    - Compute
    - Store



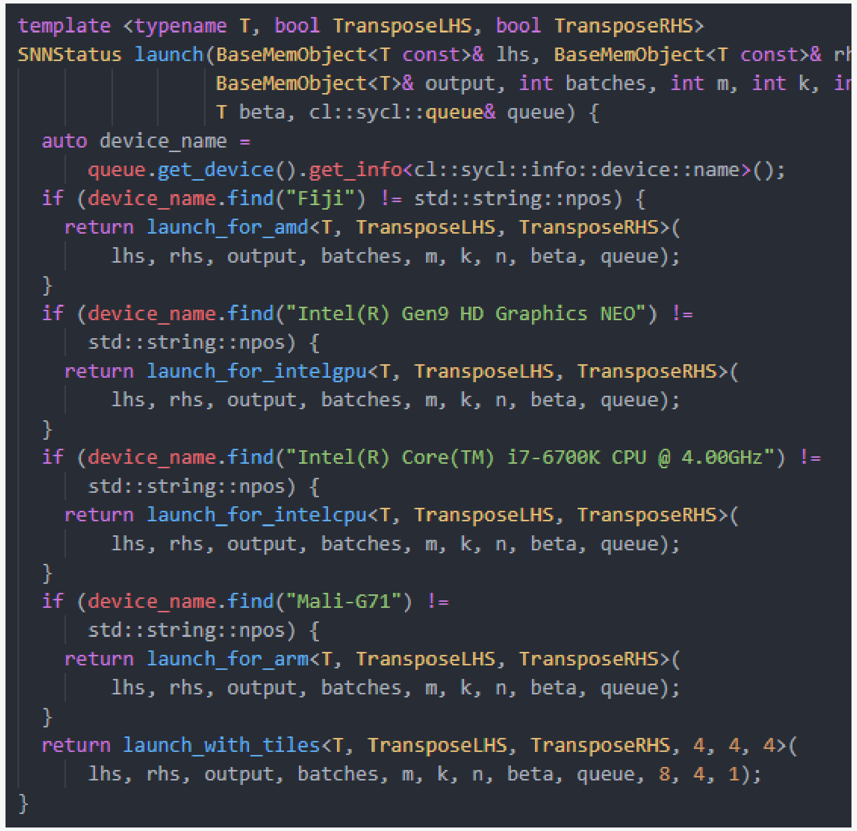
```
template <typename T, typename Index, bool TransposeLHS, bool TransposeRHS,
          int RowTile, int AccTile, int ColTile, bool CheckBounds>
struct MatmulKernel {
  MatmulKernel(ReadAccessor<T const> const& lhs,
               ReadAccessor<T const> const& rhs,
               ReadWriteAccessor<T> const& output, Index batches, Index m,
               Index k, Index n, T beta)
      : lhs_{lhs},
        rhs_{rhs},
        output_{output},
        batches_{batches},
        m_{m},
        k_{k},
        n_{n},
        beta_{beta} {}

  void SNN_ALWAYS_INLINE operator()(cl::sycl::nd_item<3> item) {
    .........
     for (; acc_idx < k_ - AccTile + 1; acc_idx += AccTile) {
          auto lhs_block =
              load_block<RowTile, AccTile, TransposeLHS>(lhs_ptr, lhs_ld);
          auto rhs_block =
              load_block<AccTile, ColTile, TransposeRHS>(rhs_ptr, rhs_ld);
          block_mmacc<false, false>(lhs_block, rhs_block, out_block);
          lhs_ptr += lhs_step;
          rhs_ptr += rhs_step;
        }
    ......
        store_block<RowTile, ColTile>(out_block, out_ptr, out_ld, valid_row,
                                      valid_col);
      }
    .........
  }
};
```

Auto Tuning Algorithm

- Per backend Optimization:
  - Kernel Selection: Clustering Method
  - Kernel Deployment: Classifier Algorithm



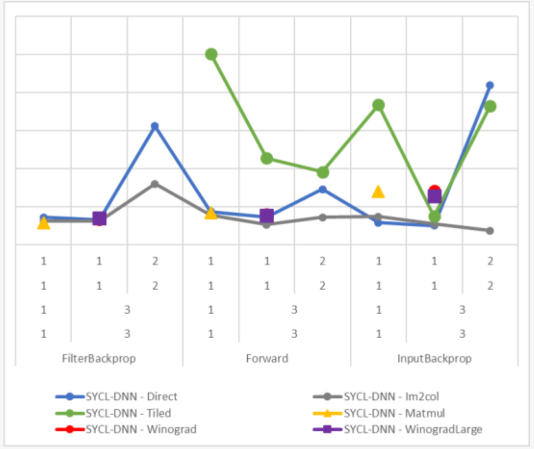
```
.....
  if ((float)m / k <= 0.27437641471624374) {
    if (m * n <= 12896) {
      if (batch * m * n <= 83968) {
        return LAUNCH(2, 4, 2, 8, 32, 1);
      } else {
        if ((float)m / k <= 0.06775882840156555) {
          return LAUNCH(4, 2, 4, 8, 16, 1);
        } else {
          return LAUNCH(2, 4, 2, 8, 32, 1);
        }
      }
    } else {
      if ((float)n / k <= 4.083333492279053) {
        if (k * m <= 4816896) {
          return LAUNCH(4, 2, 2, 1, 64, 1);
        } else {
          if (m * n <= 110592) {
            return LAUNCH(2, 4, 2, 8, 32, 1);
          } else {
            return LAUNCH(4, 2, 2, 1, 64, 1);
          }
        } else { ...}
....
```

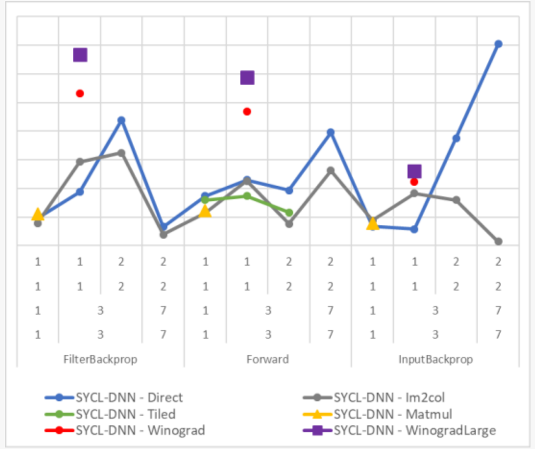


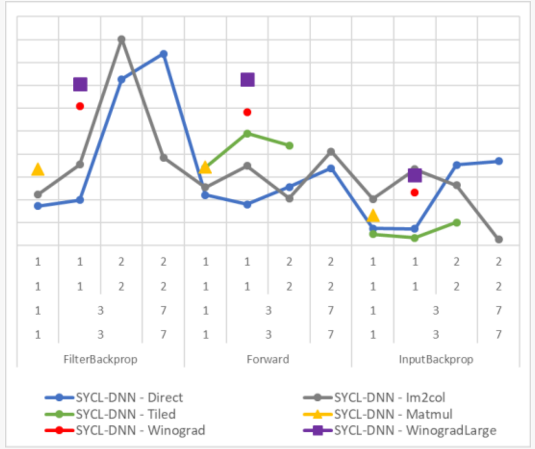
```
template <typename T, bool TransposeLHS, bool TransposeRHS>
SNNStatus launch(BaseMemObject<T const>& lhs, BaseMemObject<T const>& rh
                 BaseMemObject<T>& output, int batches, int m, int k, in
                 T beta, cl::sycl::queue& queue) {
  auto device_name =
      queue.get_device().get_info<cl::sycl::info::device::name>();
  if (device_name.find("Fiji") != std::string::npos) {
    return launch_for_amd<T, TransposeLHS, TransposeRHS>(
        lhs, rhs, output, batches, m, k, n, beta, queue);
  }
  if (device_name.find("Intel(R) Gen9 HD Graphics NEO") !=
      std::string::npos) {
    return launch_for_intelgpu<T, TransposeLHS, TransposeRHS>(
        lhs, rhs, output, batches, m, k, n, beta, queue);
  }
  if (device_name.find("Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz") !=
      std::string::npos) {
    return launch_for_intelcpu<T, TransposeLHS, TransposeRHS>(
        lhs, rhs, output, batches, m, k, n, beta, queue);
  }
  if (device_name.find("Mali-G71") !=
      std::string::npos) {
    return launch_for_arm<T, TransposeLHS, TransposeRHS>(
        lhs, rhs, output, batches, m, k, n, beta, queue);
  }
  return launch_with_tiles<T, TransposeLHS, TransposeRHS, 4, 4, 4>(
      lhs, rhs, output, batches, m, k, n, beta, queue, 8, 4, 1);
}
```

Auto-tuning per-processor

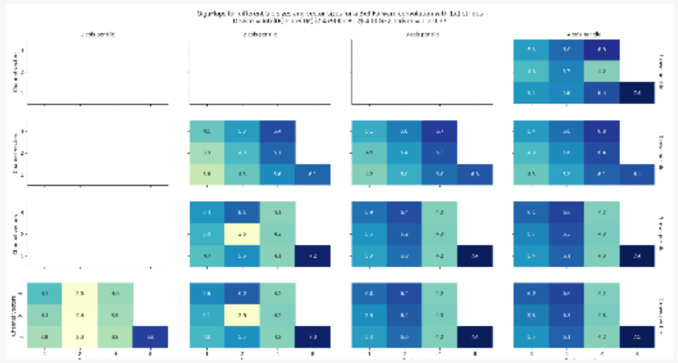
SYCL-DNN runs 5 algorithms across a range of common convolution sizes and shapes on each processor it supports and "learns" the best algorithm for each size and shape

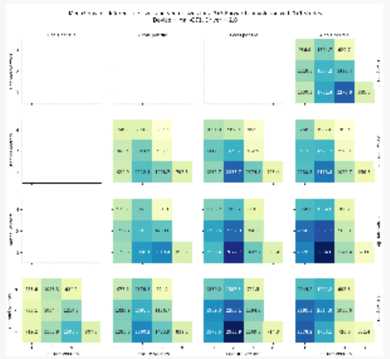


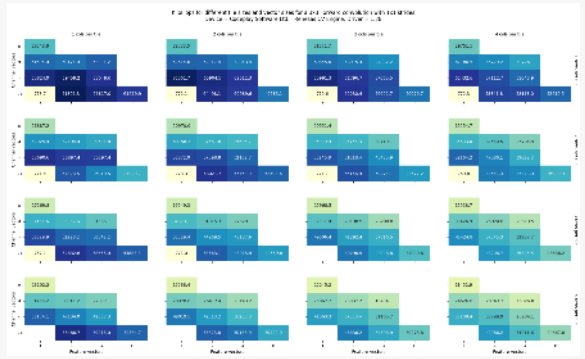




SYCL-DNN runs a range of different tile sizes and shapes on each algorithm and on each processor to "learn" the best tile sizes & shapes for each convolution



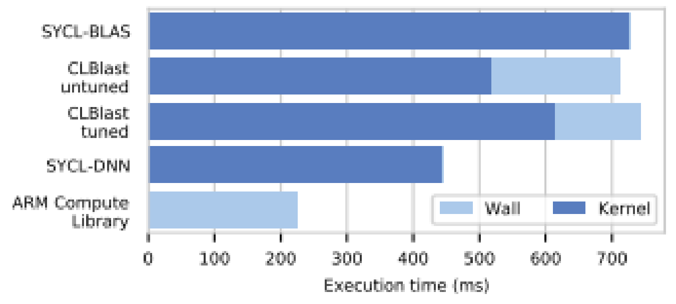




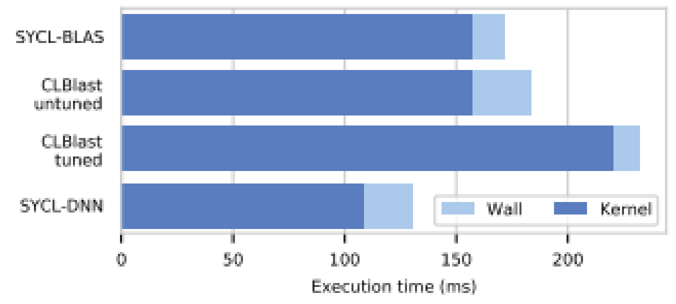
codeplay®

Auto Tuning Algorithm

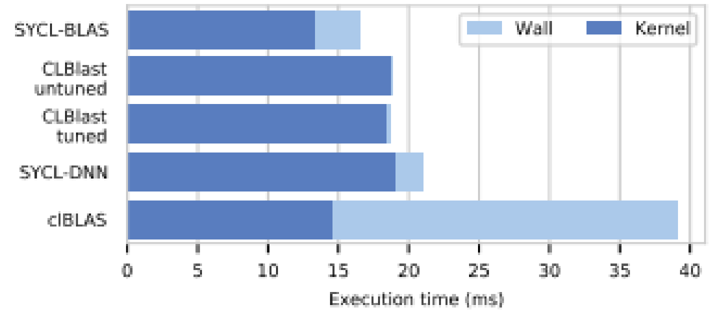
- VGG-16:
  - Inference Model
  - Time in millisecond
  - SYCL-DNN with Different Backend



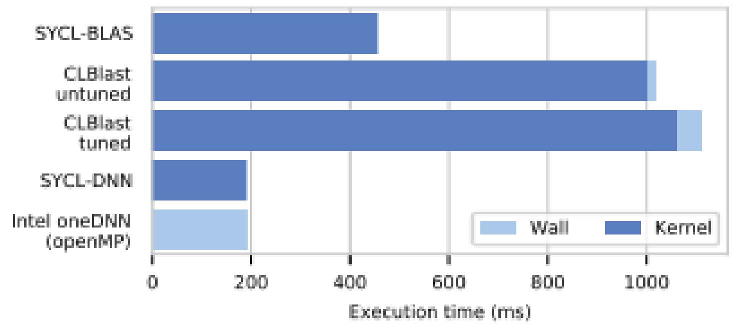
(d) ARM Mali G71



(c) Intel HD Graphics 530

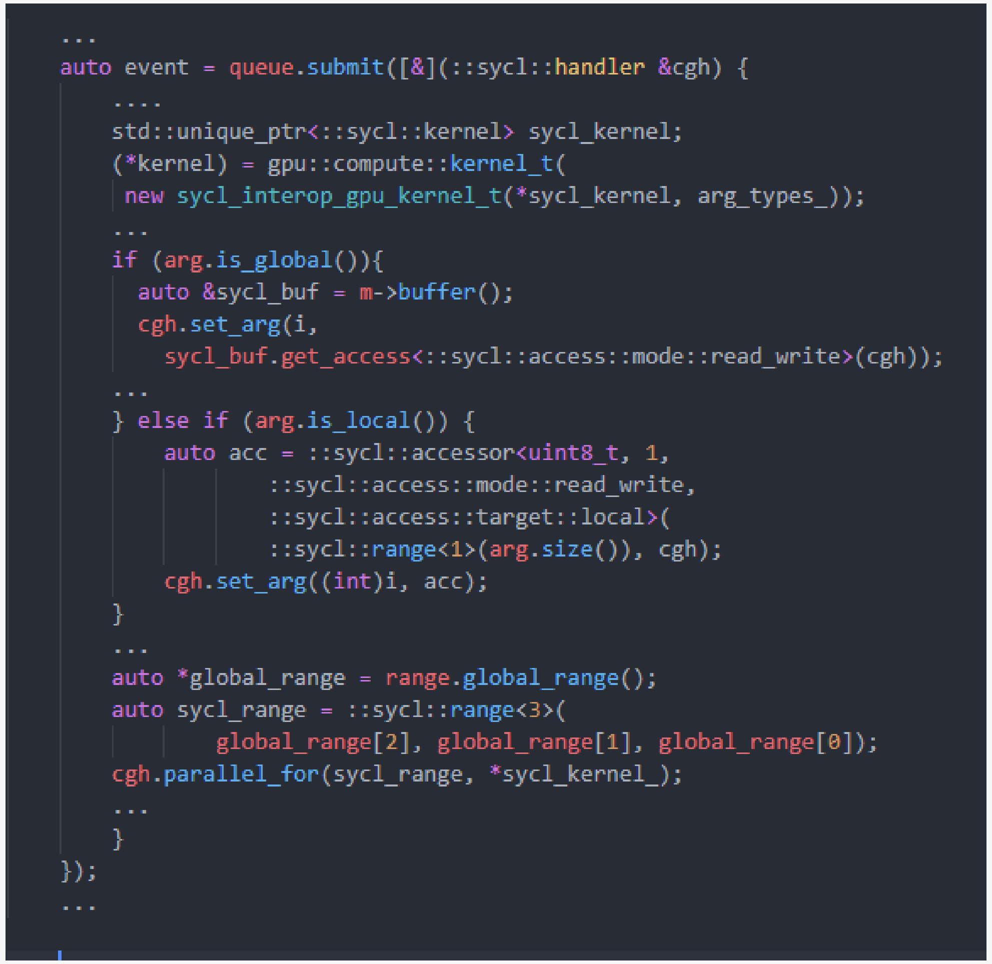


(a) AMD R9 Nano



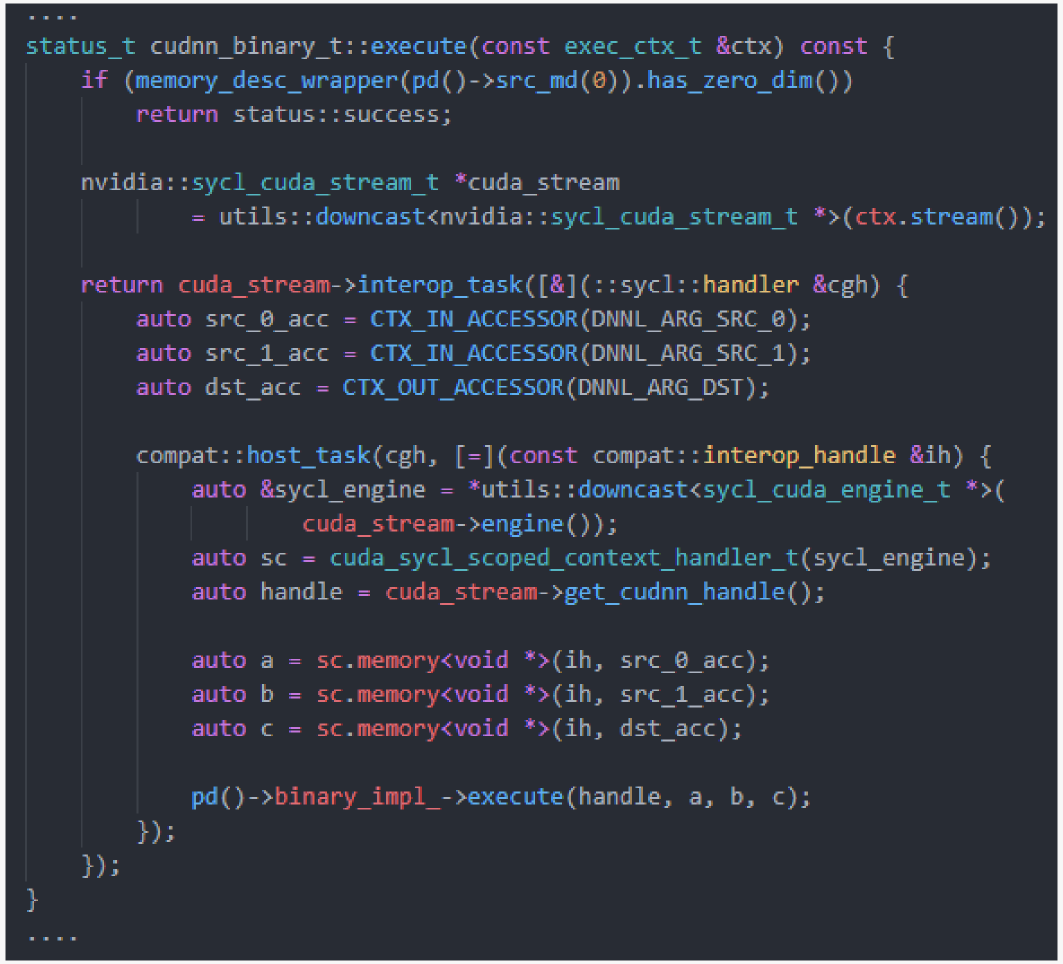
(b) Intel i7-6700K CPU

Cross-platform performance Portability via SYCL Interoperability



```
...
auto event = queue.submit([&](::sycl::handler &cgh) {
    ....
    std::unique_ptr<::sycl::kernel> sycl_kernel;
    (*kernel) = gpu::compute::kernel_t(
     new sycl_interop_gpu_kernel_t(*sycl_kernel, arg_types_));
    ...
    if (arg.is_global()){
      auto &sycl_buf = m->buffer();
      cgh.set_arg(i,
        sycl_buf.get_access<::sycl::access::mode::read_write>(cgh));
    ...
    } else if (arg.is_local()) {
        auto acc = ::sycl::accessor<uint8_t, 1,
                ::sycl::access::mode::read_write,
                ::sycl::access::target::local>(
                ::sycl::range<1>(arg.size()), cgh);
        cgh.set_arg((int)i, acc);
    }
    ...
    auto *global_range = range.global_range();
    auto sycl_range = ::sycl::range<3>(
        global_range[2], global_range[1], global_range[0]);
    cgh.parallel_for(sycl_range, *sycl_kernel_);
    ...
    }
});
...
```

OneDNN Interoperability with OpenCL



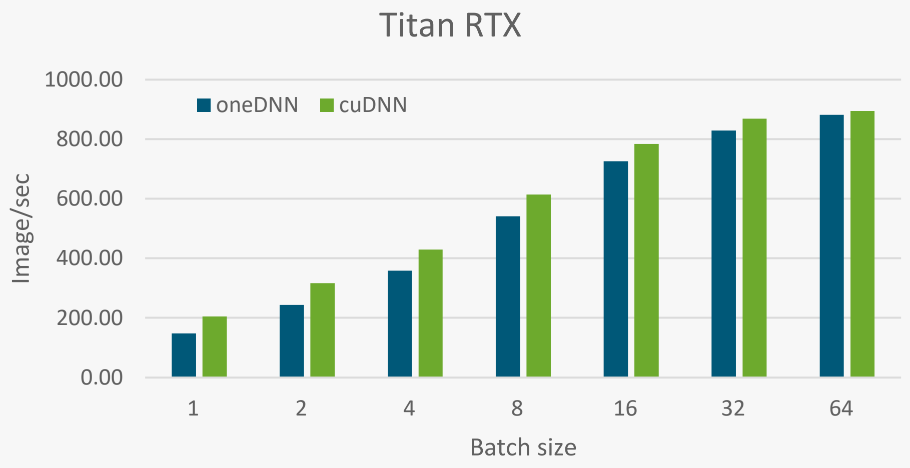
```
....
status_t cudnn_binary_t::execute(const exec_ctx_t &ctx) const {
    if (memory_desc_wrapper(pd()->src_md(0)).has_zero_dim())
        return status::success;

    nvidia::sycl_cuda_stream_t *cuda_stream
            = utils::downcast<nvidia::sycl_cuda_stream_t *>(ctx.stream());

    return cuda_stream->interop_task([&](::sycl::handler &cgh) {
        auto src_0_acc = CTX_IN_ACCESSOR(DNNL_ARG_SRC_0);
        auto src_1_acc = CTX_IN_ACCESSOR(DNNL_ARG_SRC_1);
        auto dst_acc = CTX_OUT_ACCESSOR(DNNL_ARG_DST);

        compat::host_task(cgh, [=](const compat::interop_handle &ih) {
            auto &sycl_engine = *utils::downcast<sycl_cuda_engine_t *>(
                    cuda_stream->engine());
            auto sc = cuda_sycl_scoped_context_handler_t(sycl_engine);
            auto handle = cuda_stream->get_cudnn_handle();

            auto a = sc.memory<void *>(ih, src_0_acc);
            auto b = sc.memory<void *>(ih, src_1_acc);
            auto c = sc.memory<void *>(ih, dst_acc);

            pd()->binary_impl_->execute(handle, a, b, c);
        });
    });
}
....
```

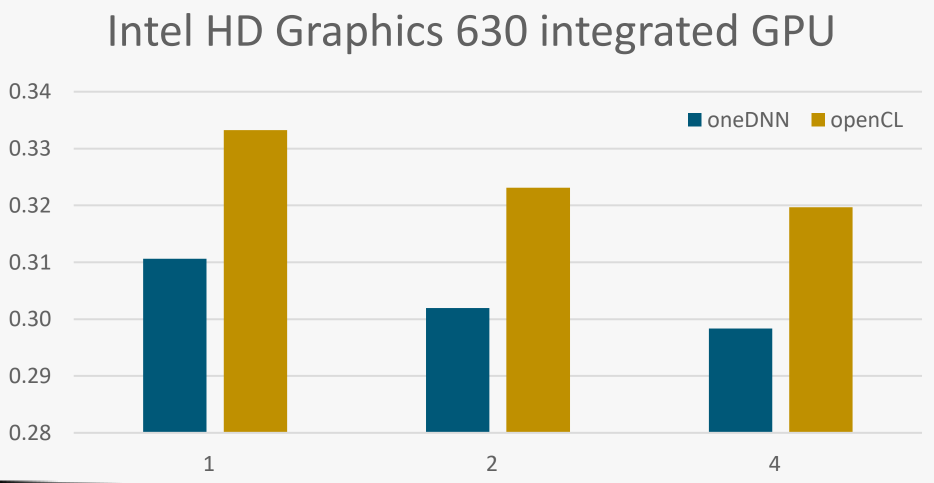
OneDNN Interoperability with CUDA

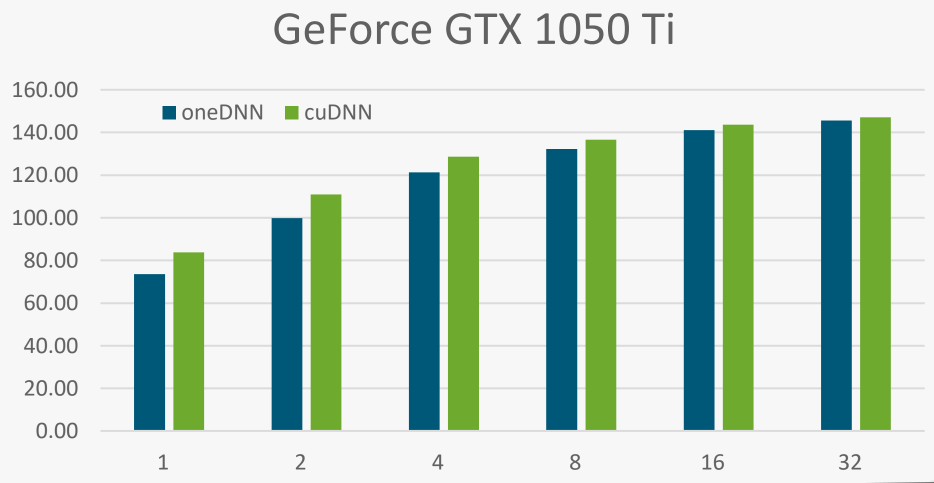
codeplay

OneDNN Cuda backend: RESNET-50

- Titan RTX Overhead (1%-27%)
- GeForce GTX 1050ti (1%-13%)
- Intel HD Graphics 630 integrated GPU (~6%)
- Constant difference of 1.8 ms in all batch sizes







AI Graph to SYCL Summary

- While ONNX model can support
  - Unified Open standard portable format across various AI platform
- SYCL integration to AI can bring
  - Unified optimisation Scheme
  - Generic Inference Engine
  - Improve the code maintainability
  - Support Cross-platform performance portability

- Resources
- Eigen: https://gitlab.com/libeigen/eigen/
- SYCL-BLAS: https://github.com/codeplaysoftware/sycl-blas
- SYCL-DNN: https://github.com/codeplaysoftware/sycl-dnn
- oneMKL: https://github.com/oneapi-src/oneMKL
- oneDNN: https://github.com/oneapi-src/oneDNN

From SYCL to hardware

Alastair Murray (VP of Product Engineering)

Outline

- Desirable abilities when creating an accelerator
- RISC-V in a simplified AI hardware context
- Supporting an ecosystem on an accelerator
- RISC-V example: 1-2 line kernels on vector hardware
- RISC-V example: tightly coupled memory
- RISC-V example SYCL extension: scratchpad memory

Desirable abilities when creating an accelerator

Create high performance software

- Parallel execution models
- High-quality compilers
- Efficient language runtimes
- Pre-written libraries
- Domain-specific optimizations
- Tuning, or auto-tuning

Access existing software ecosystem

- Creating a full-fledged AI stack from scratch is unrealistic for most
- Developers grow ecosystem to support their needs, on your hardware

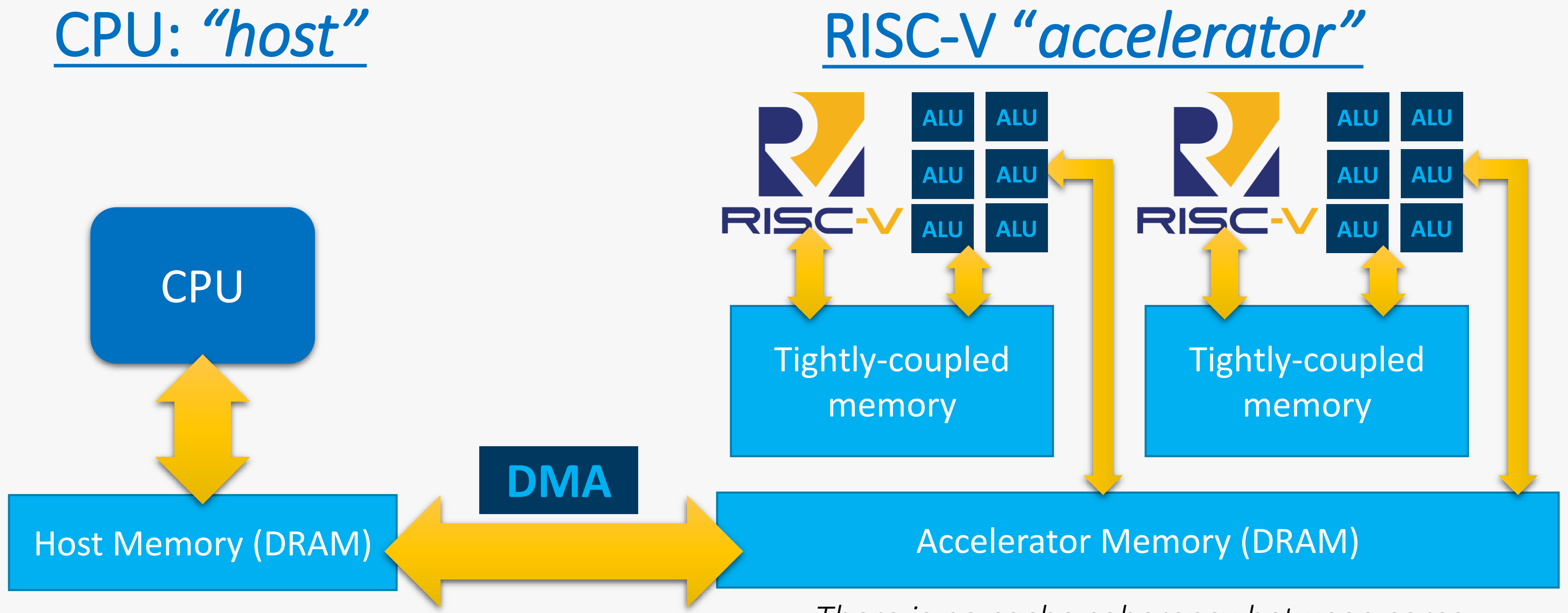
Expose all hardware to developers

- Map execution model to hardware via compiler and schedulers
- Language features to let developers explicitly control hardware
- Vendor specific extensions to standards

Experiment with architecture changes

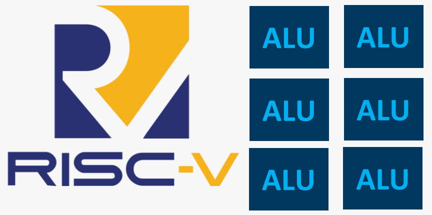
- Quickly prototype software changes to evaluate hardware ideas
- Evaluation on realistic use-cases

Simplified AI hardware

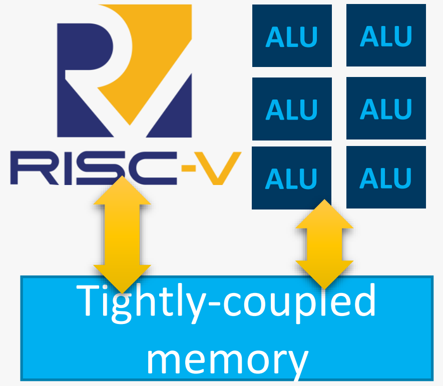


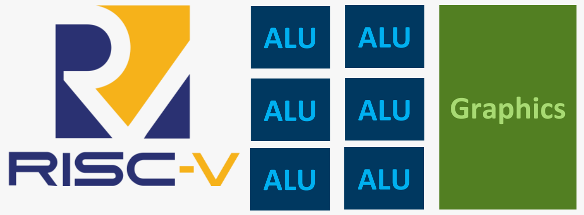
There is no cache coherency between cores

Why a RISC-V example?

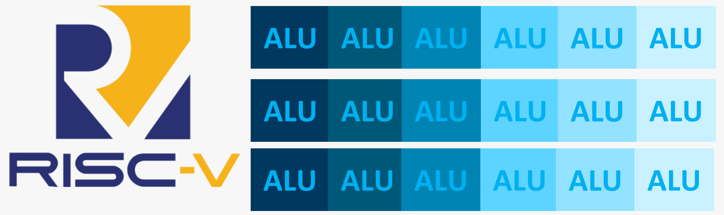


Multi-core CPU

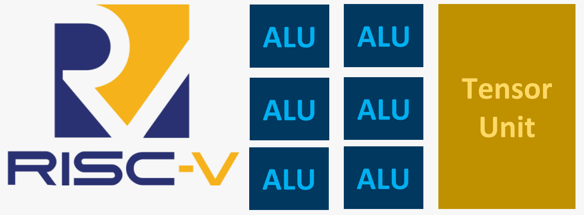

Large TCM



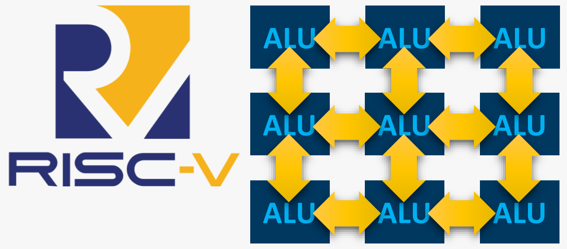
GPU



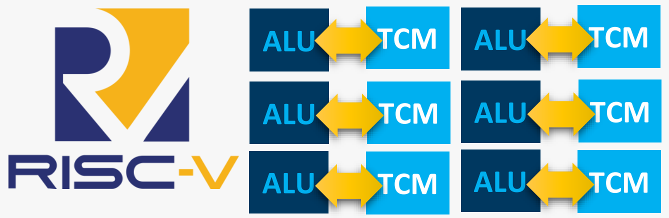
Vector processor



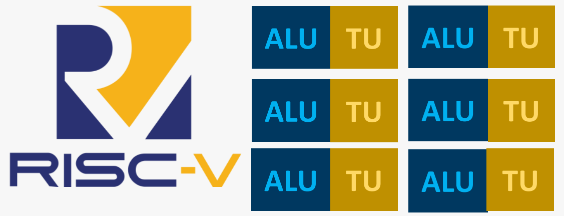
Tensor Unit



Data-flow Architecture



Per-core TCM

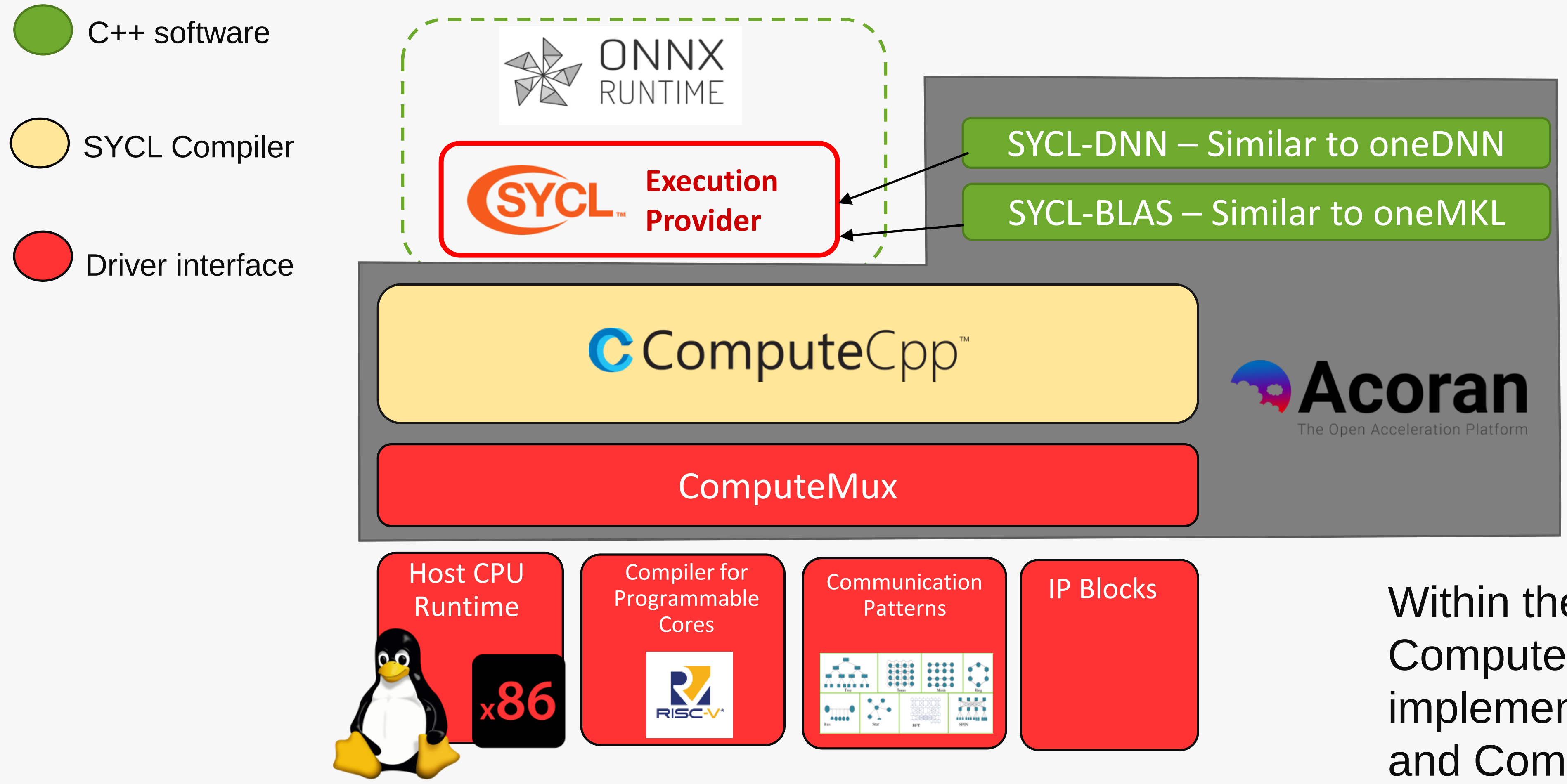


Per-core Tensor Instructions

Access existing software ecosystem: SYCL

- Building on open-standards, such as SYCL, means a pre-existing body of software is available.
- Developers keener to extend ecosystem themselves because their work is not locked to a vendor API.
- Good solutions exist to help to implement standards

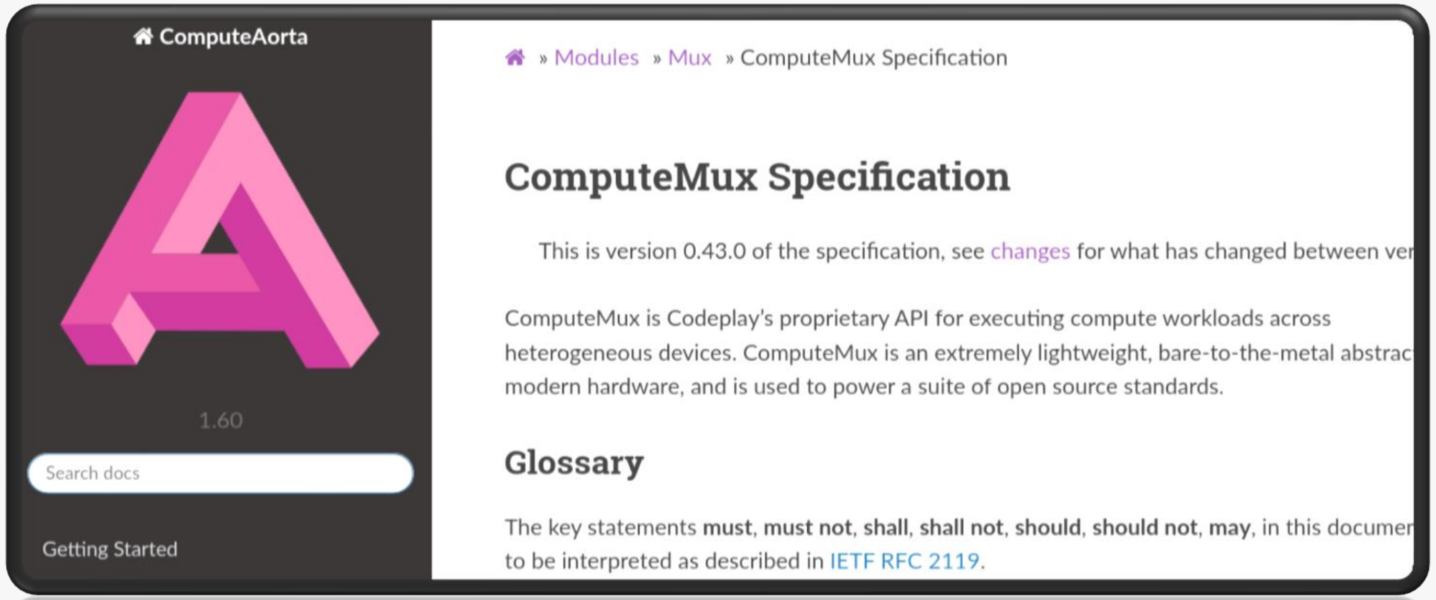
ONNX Runtime to RISC-V



Within the Acoran platform ComputeCpp is a SYCL implementation out-of-the-box and ComputeMux provides part of a “Driver Development Kit”.

ComputeMux is a specification

- Build heterogeneous standards on top of the interface
  - SYCL, OpenCL, Vulkan Compute
- Implement the specification however suits the hardware
- Runtime specification
  - How to map runtime API to drivers
- Compiler specification
  - How to map language to hardware
- For RISC-V we have a pre-written reference implementation



ComputeMux Specification

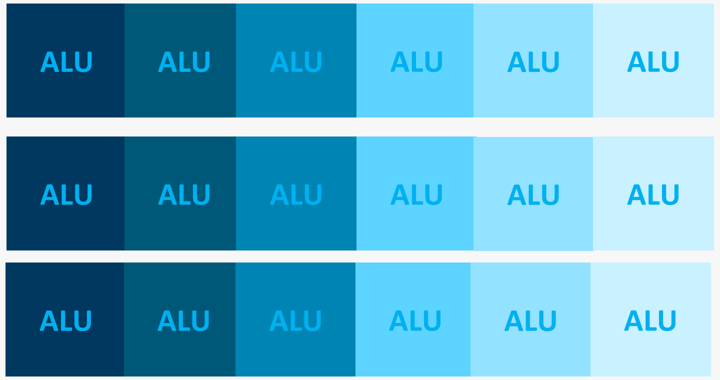
This is version 0.43.0 of the specification, see changes for what has changed between ver

ComputeMux is Codeplay's proprietary API for executing compute workloads across heterogeneous devices. ComputeMux is an extremely lightweight, bare-to-the-metal abstrac modern hardware, and is used to power a suite of open source standards.

Glossary

The key statements **must**, **must not**, **shall**, **shall not**, **should**, **should not**, **may**, in this documer to be interpreted as described in IETF RFC 2119.

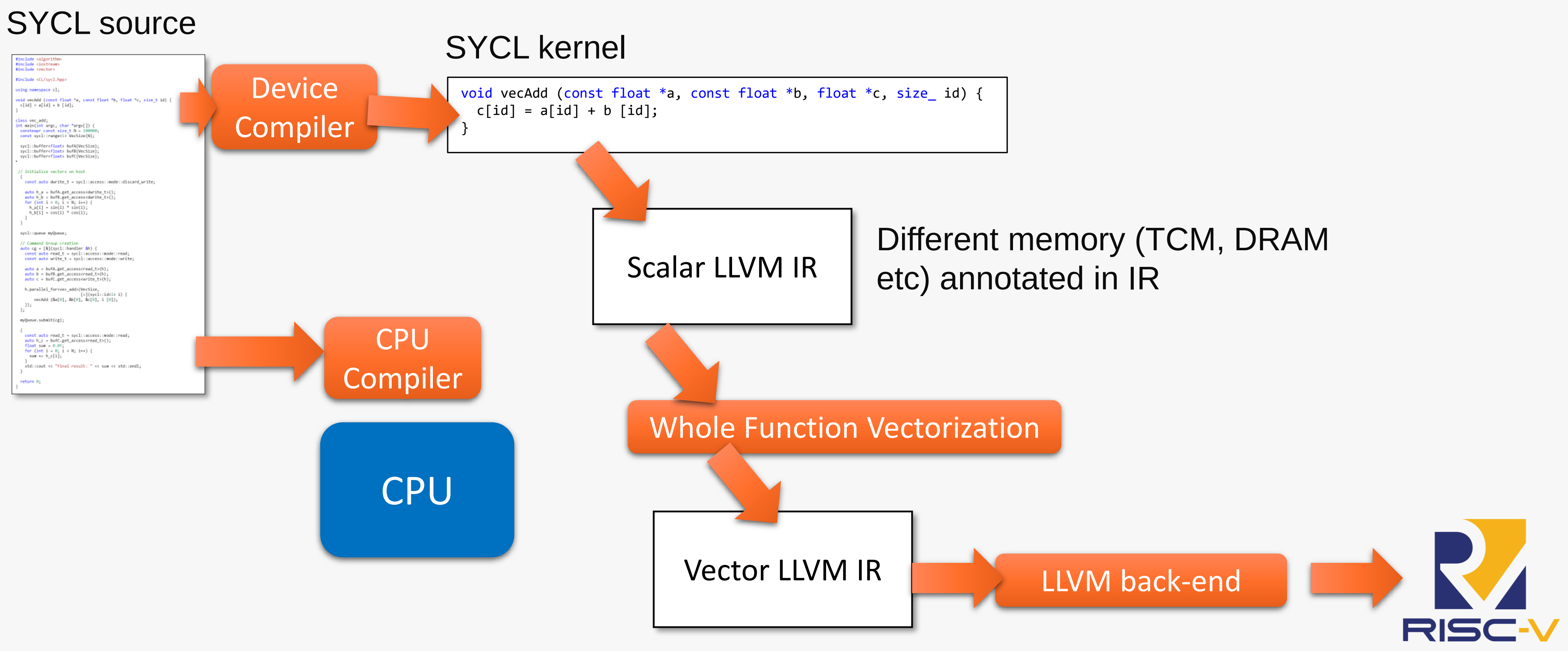
Expose all hardware to developers: Vectors



Vector processor

- RISC-V has a vector extension: RVV
- RVV is scalable vectors, i.e. vector width unknown at compile time
- Map SYCL data-parallel model to vector hardware via the compiler

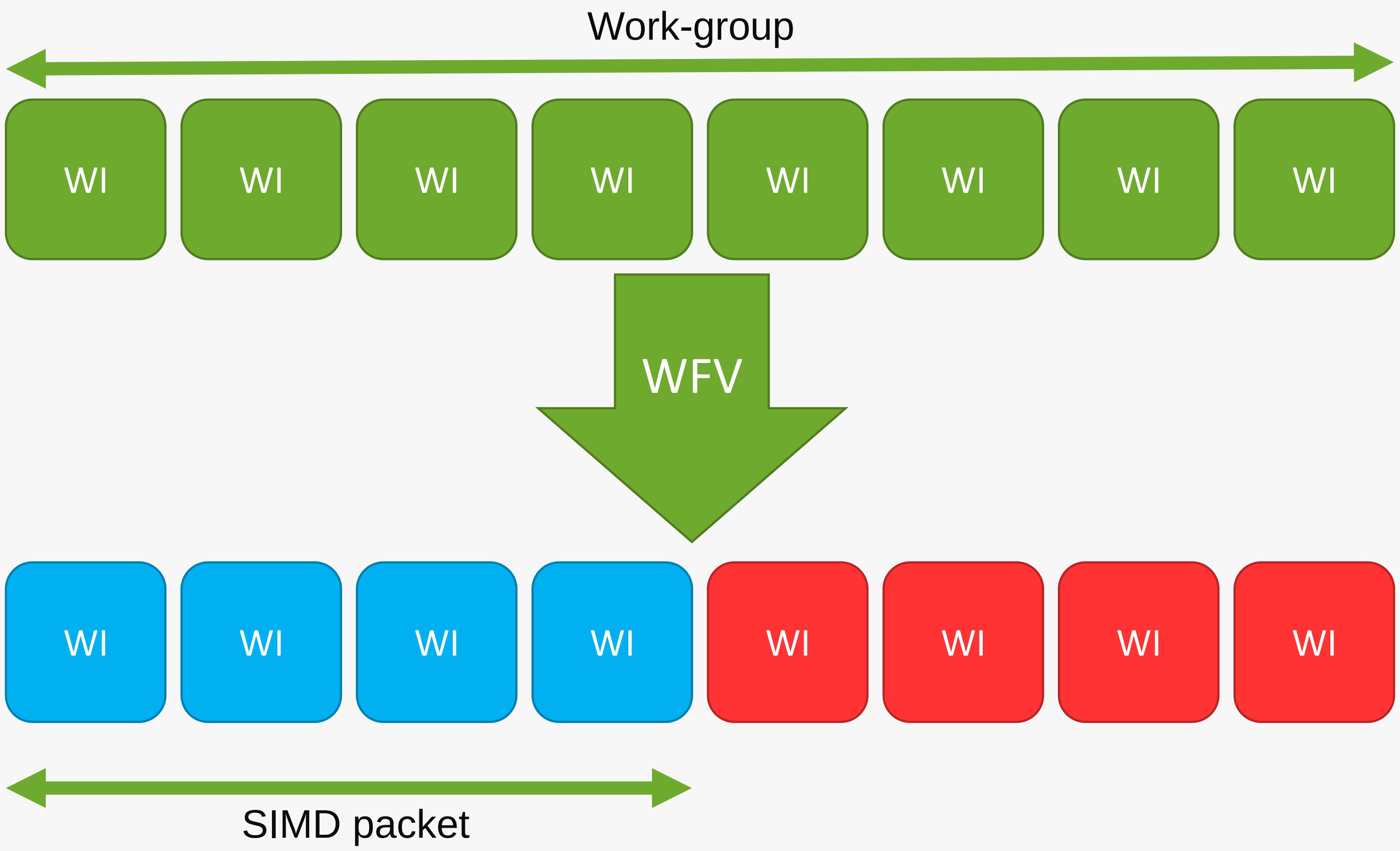
Compiling SYCL kernel to RISC-V Vectors



```
#include <algorithm>
#include <iostream>
#include <vector>

#include <CL/sycl.hpp>

using namespace cl;

void vecAdd (const float *a, const float *b, float *c, size_t id) {
  c[id] = a[id] + b [id];
}

class vec_add;
int main(int argc, char *argv[]) {
  constexpr const size_t N = 100000;
  const sycl::range<1> VecSize{N};

  sycl::buffer<float> bufA{VecSize};
  sycl::buffer<float> bufB{VecSize};
  sycl::buffer<float> bufC{VecSize};
.

 // Initialize vectors on host
  {
    const auto dwrite_t = sycl::access::mode::discard_write;

    auto h_a = bufA.get_access<dwrite_t>();
    auto h_b = bufB.get_access<dwrite_t>();
    for (int i = 0; i < N; i++) {
      h_a[i] = sin(i) * sin(i);
      h_b[i] = cos(i) * cos(i);
    }
  }

  sycl::queue myQueue;

  // Command Group creation
  auto cg = [&](sycl::handler &h) {
    const auto read_t = sycl::access::mode::read;
    const auto write_t = sycl::access::mode::write;

    auto a = bufA.get_access<read_t>(h);
    auto b = bufB.get_access<read_t>(h);
    auto c = bufC.get_access<write_t>(h);

    h.parallel_for<vec_add>(VecSize,
                            [=](sycl::id<1> i) {
        vecAdd (&a[0], &b[0], &c[0], i [0]);
    });
  };

  myQueue.submit(cg);

  {
    const auto read_t = sycl::access::mode::read;
    auto h_c = bufC.get_access<read_t>();
    float sum = 0.0f;
    for (int i = 0; i < N; i++) {
      sum += h_c[i];
    }
    std::cout << "Final result: " << sum << std::endl;
  }

  return 0;
}
```

```
void vecAdd (const float *a, const float *b, float *c, size_ id) {
  c[id] = a[id] + b [id];
}
```

Whole Function Vectorization



Vector-add on RISC-V Vectors

```
void vecAdd (const float *a, const float *b, float *c, size_t id) {
  c[id] = a[id] + b[id];
}
```

This is the scalar version of the kernel

```
.LBB0_9:
        flw     ft0, 0(a4)
        flw     ft1, 0(a5)
        fadd.s  ft0, ft0, ft1
        fsw     ft0, 0(a1)
        addi    a3, a3, -1
        addi    a1, a1, 4
        addi    a5, a5, 4
        addi    a4, a4, 4
        bnez    a3, .LBB0_9
```

The scalar version is doing 1 iteration of vecAdd per loop iteration

The vector version is doing 32 iterations of vecAdd per loop iteration

This is the vector version of the same kernel

```
.LBB0_8:
        vsetvli zero, a1, e32, m8, ta, mu
        vle32.v v8, (s0)
        vle32.v v16, (s1)
        vfadd.vv        v8, v8, v16
        vse32.v v8, (a0)
        addi    a3, a3, 32
        addi    a0, a0, 128
        addi    s1, s1, 128
        addi    s0, s0, 128
        bltu    a3, s2, .LBB0_8
```

RISC-V Vectors with vector selection

```c
void vecAdd (const float *a, const float *b, float *c, size_t id) {
  float v = a [id] + b [id];
  if (v < 0.0f) {
    v = 0.0f;
  }
  c [id] = v;
}
```

This is the scalar version of the kernel

```
.LBB0_12:
        flw     ft1, 0(a4)
        flw     ft2, 0(a5)
        fadd.s  ft1, ft1, ft2
        flt.s   a2, ft1, ft0
        fmv.s   ft2, ft0
        bnez    a2, .LBB0_11
        fmv.s   ft2, ft1
.LBB0_11:
        fsw     ft2, 0(a1)
        addi    a3, a3, -1
        addi    a1, a1, 4
        addi    a5, a5, 4
        addi    a4, a4, 4
        beqz    a3, .LBB0_9
        j       .LBB0_12
```

The scalar version uses a branch to handle the `if` conditional statement

- If we put `if` statements in our code, then we create *branch* instructions in the assembly
- When we *vectorize* this code, then what happens to the branches?
- We can't do a vector-branch in RVV
- Instead, we use *masks* or *predication*

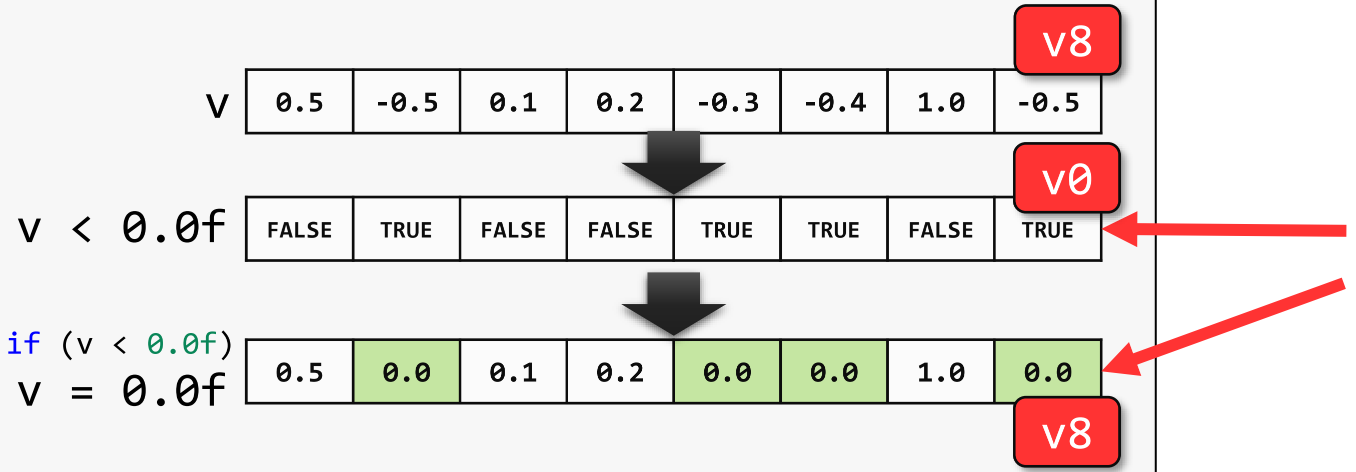
RISC-V Vectors with vector selection

```c
void vecAdd (const float *a, const float *b, float *c, size_t id) {
  float v = a [id] + b [id];
  if (v < 0.0f) {
    v = 0.0f;
  }
  c [id] = v;
}
```

This is the vector version of the same kernel

```
.LBB0_8:
        vsetvli     zero, a3, e32, m8, ta, mu
        vle32.v     v8, (s0)
        vle32.v     v16, (s1)
        vfadd.vv    v8, v8, v16
        vmflt.vf    v0, v8, ft0
        vmerge.vim  v8, v8, 0, v0
        vse32.v  v8, (a1)
        addi     a0, a0, 32
        addi     a1, a1, 128
        addi     s1, s1, 128
        addi     s0, s0, 128
        bltu     a0, t6, .LBB0_8
```

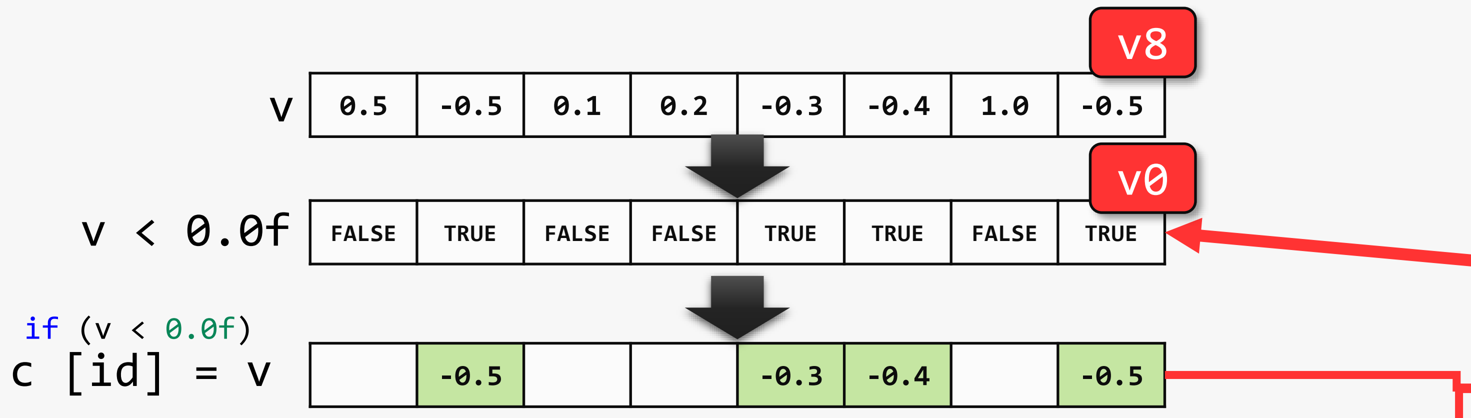
The vector version uses a vector compare and merge ('select') to handle N iterations of the **if** conditional statement in two instructions



Handling conditionals with load/store

```
void vecAdd (const float *a, const float *b, float *c, size_t id) {
  float v = a [id] + b [id];
  if (v < 0.0f) {
    c [id] = v;
  }
}
```

In this `if` statement, we are writing values to memory. How do we predicate a store to memory?
-> We use *masked stores*



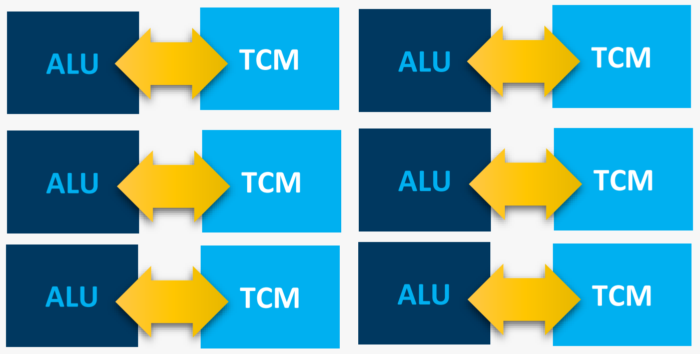
The *masked store* doesn't change (store to) the elements of `c[id]` where the conditional is false

This is a RISC-V Vector masked store instruction

This is the RISC-V assembly

```
.LBB0_11:
        vsetvli  zero, a1, e32, m8, ta, mu
        vle32.v  v8, (s0)
        vle32.v  v16, (s1)
        vfadd.vv v8, v8, v16
        vmflt.vf v0, v8, ft0
        vmv.x.s  a0, v0
        beqz     a0, .LBB0_10
        vse32.v  v8, (a5), v0.t
.LBB0_10:
        addi     a3, a3, 32
        addi     a5, a5, 128
        addi     s1, s1, 128
        addi     s0, s0, 128
        bgeu     a3, s3, .LBB0_8
        j        .LBB0_11
```

Exposing hardware to developers: DMA



Per-core TCM

- Tightly couple memory per-code maps directly to SYCL “local” memory
- SYCL provides builtin methods to manually control DMA to/from local memory

Asynchronous DMA

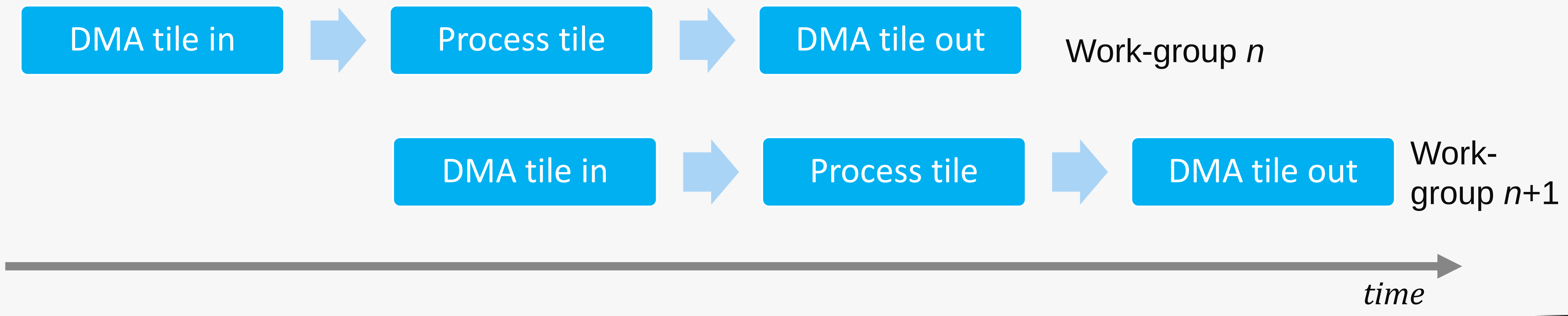
DMA tile in

Process tile

DMA tile out

```
h.parallel_for<tiled_vec_add>(sycl::nd_range<1> (VecSize, tile_size),
                       [=](sycl::nd_item<1> i) {
  auto event1 = i.async_work_group_copy (tile1.get_pointer(), a.get_pointer() + i.get_global_id(0), tile_size[0]);
  auto event2 = i.async_work_group_copy (tile2.get_pointer(), b.get_pointer() + i.get_global_id(0), tile_size[0]);
  i.wait_for (event1, event2);

  vecAdd (&tile1[0], &tile2[0], i.get_global_id (0), i.get_local_id (0));
  i.barrier();

  auto event3 = i.async_work_group_copy (c.get_pointer() + i.get_global_id(0), tile1.get_pointer(), tile_size[0]);
  i.wait_for (event3);});
}
```

We can run two work-groups at once in a pipeline



Abstracting out optimization approach

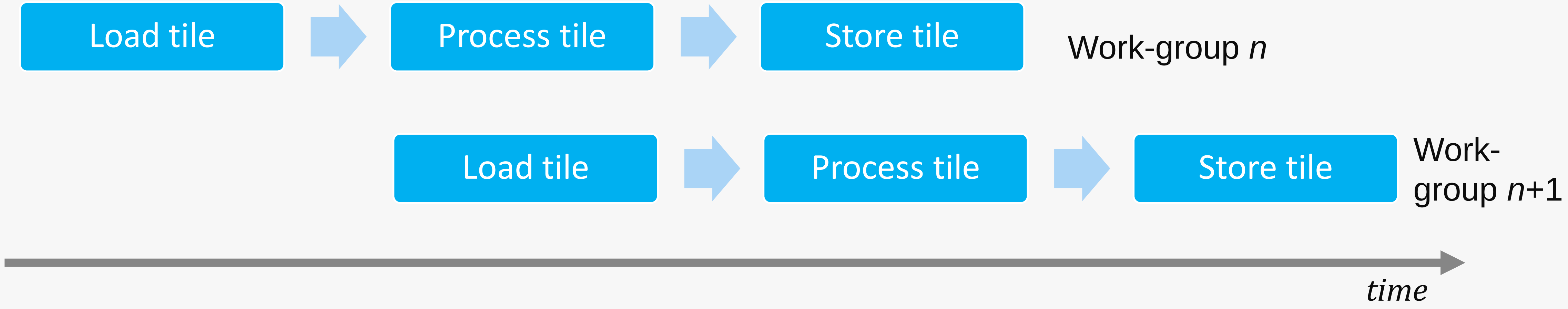
Load tile

Process tile

Store tile

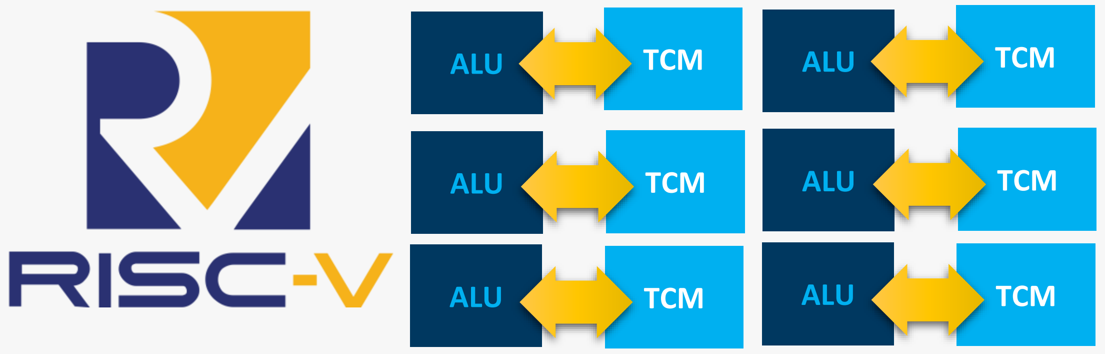
```
h.parallel_for<tiled_vec_add>(sycl::nd_range<1> (VecSize, tile_size),
                              [=](sycl::nd_item<1> i) {
  load_tile(i, tile1, a, tile_size);
  load_tile(i, tile2, b, tile_size);

  vecAdd (&tile1[0], &tile2[0], i.get_global_id (0), i.get_local_id (0));
  i.barrier();

  store_tile(i, c, tile1, tile_size);
}
```

We can run two work-groups at once in a pipeline



Experimenting with Architecture Changes

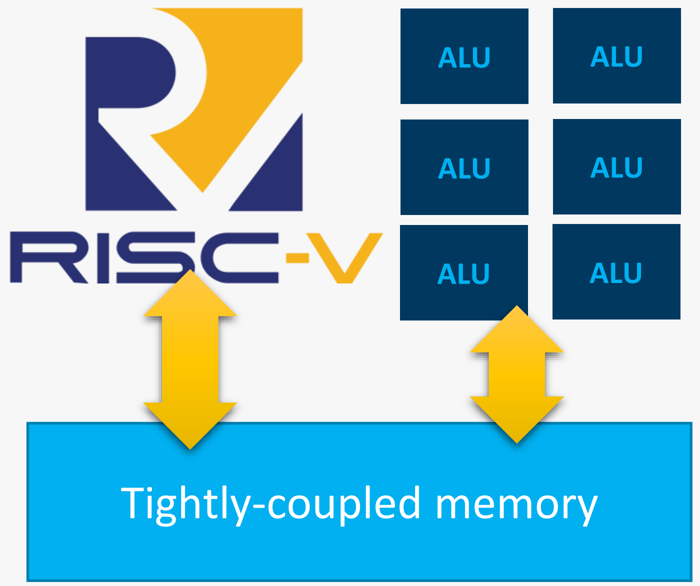
Initial architecture



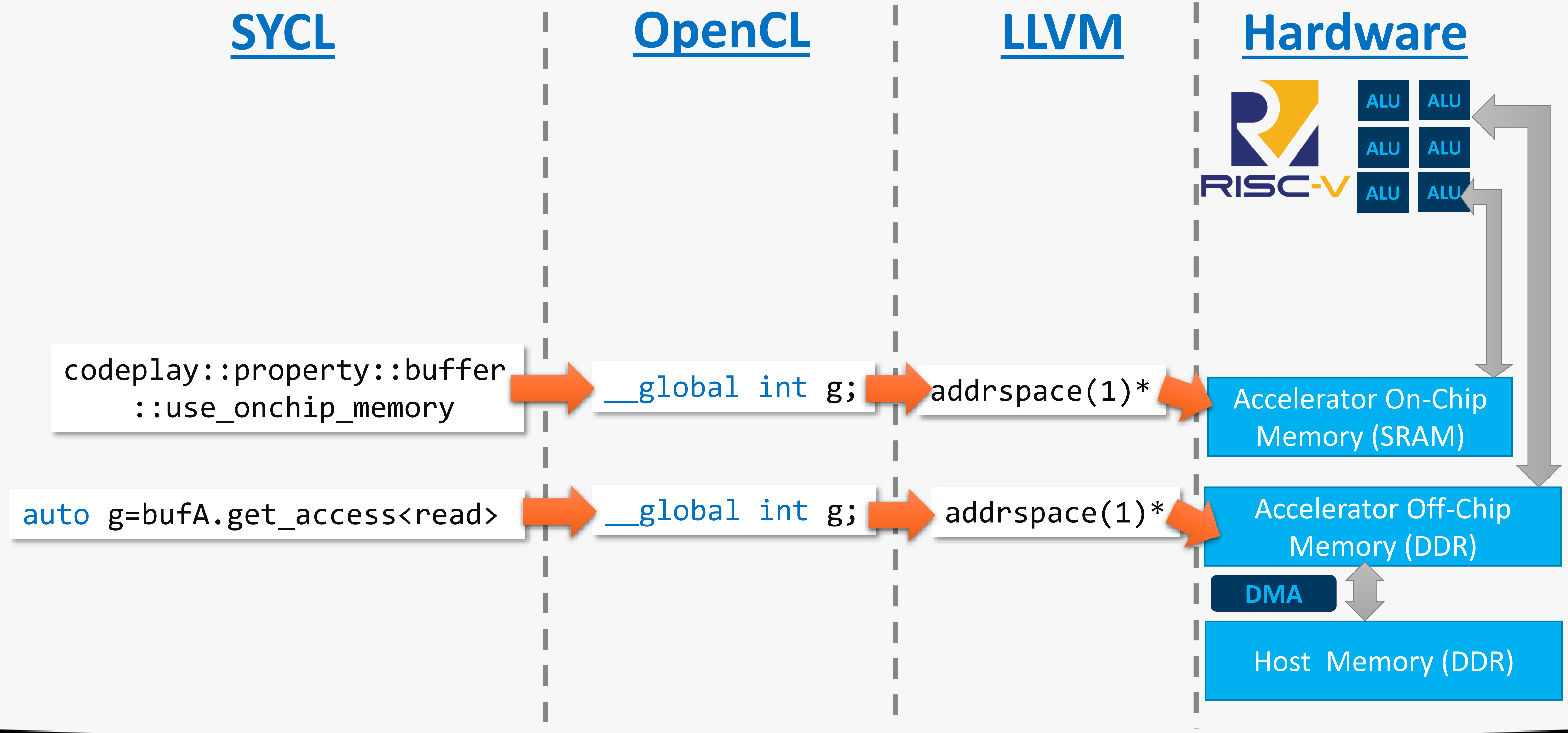
Per-core TCM

Experimental alternative

Large TCM



Memory mapping to LLVM address spaces


SYCL
OpenCL
LLVM
Hardware
codeplay::property::buffer ::use_onchip_memory
__global int g;
addrspace(1)*
Accelerator On-Chip Memory (SRAM)
auto g=bufA.get_access<read>
__global int g;
addrspace(1)*
Accelerator Off-Chip Memory (DDR)
DMA
Host Memory (DDR)
RISC-V
ALU
ALU
ALU
ALU
ALU
ALU

Desirable abilities when creating an accelerator

Create high performance software

- Mehdi covered existing libraries and paths for tuning them

Access existing software ecosystem

- Mehdi covered one existing AI stack
- It is possible to provide developers the tools to let them adapt the ecosystem to their own needs

Expose all hardware to developers

- SYCL parallel execution model can be natively mapped to parallel hardware
- SYCL can provide methods and vendor extensions to exploit hardware features

Experiment with architecture changes

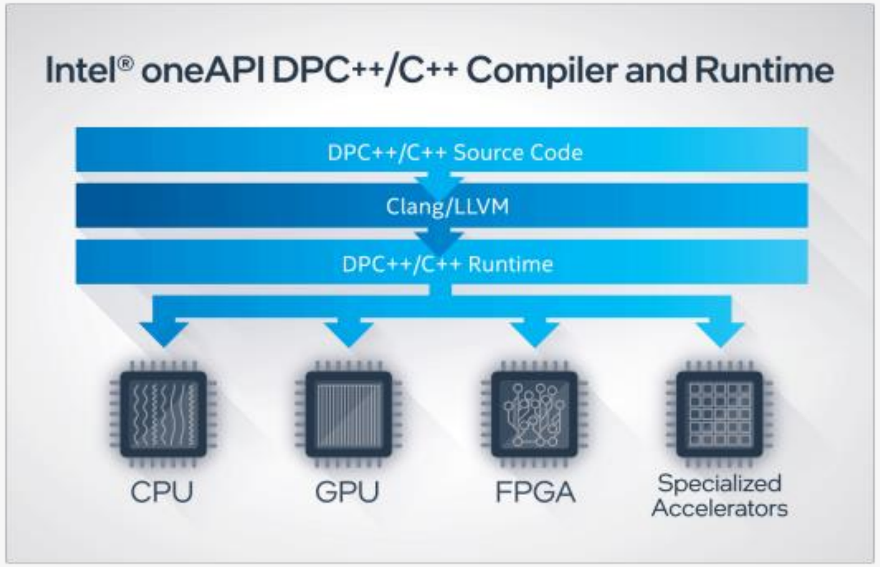
- Extensions to standards such as SYCL enable quick porting and evaluation of existing software

SYCL & oneAPI in Exascale HPC

Gordon Brown (Principal PO oneAPI & Automotive)

SYCL, DPC++ and oneAPI

- Data Parallel C++ is an open alternative to single-architecture proprietary languages
- DPC++ is an open-source implementation of SYCL with extensions
- It is part of the oneAPI programming model that includes definitions of standard library interfaces, for common operations such as math



Codeplay and SYCL

- Part of the SYCL community from the beginning
- Our team has helped to shape the SYCL standard
- Implemented the first conformant SYCL product
- Maintainers of DPC++ CUDA and HIP backends

KHRONOS GROUP
CONNECTING SOFTWARE TO SILICON

SYCL™

Open Source Contributions

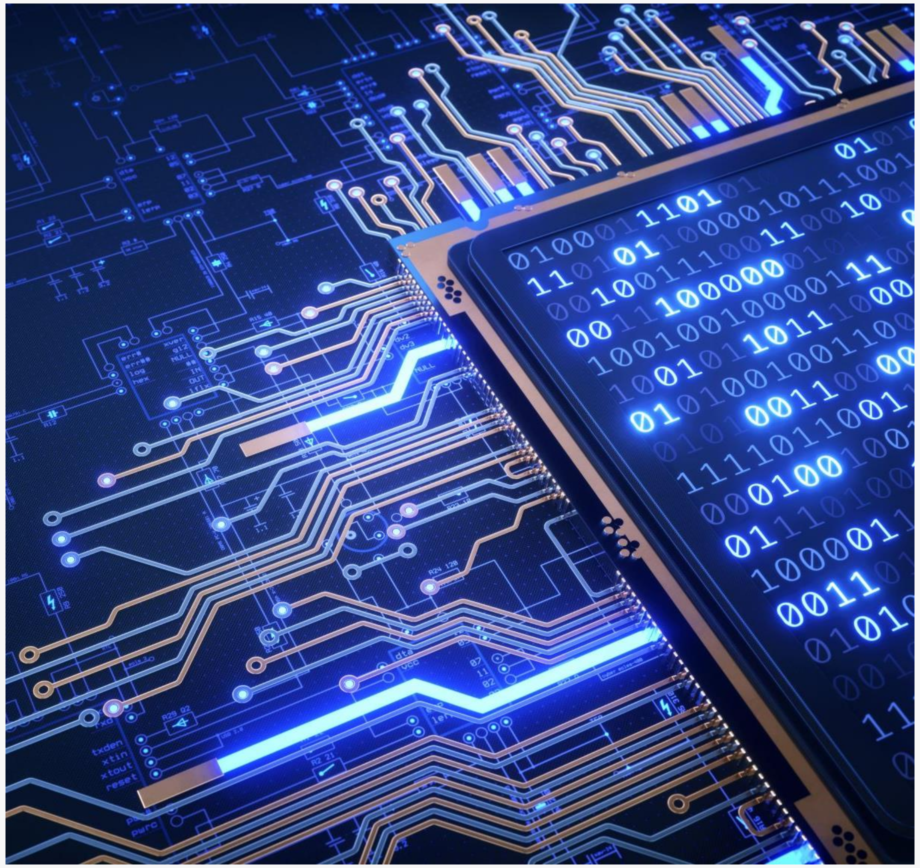
SYCL-DNN SYCL-BLAS SYCL-ML oneAPI oneDNN oneMKL

Eigen TensorFlow

ComputeCpp™ DPC++

Pre-exascale and Exascale Supercomputers

- What is a “pre-exascale” supercomputer?
  - A system capable of calculating close to the power of an exascale supercomputer
  - Exceeding 10^15 floating point operations per second > 1 petaFLOPS
- What is an “Exascale” supercomputer?
  - A system capable of calculating at least 10^18 floating point operations per second = 1 exaFLOPS



Perlmutter Supercomputer

- New pre-exascale supercomputer at Lawrence Berkeley National Laboratory
- Named after Saul Perlmutter, astrophysicist at Berkeley Lab and 2011 Nobel Laureate
- HPE Cray system with CPU-only and GPU-accelerated nodes
- 6000+ NVIDIA® A100 GPUs

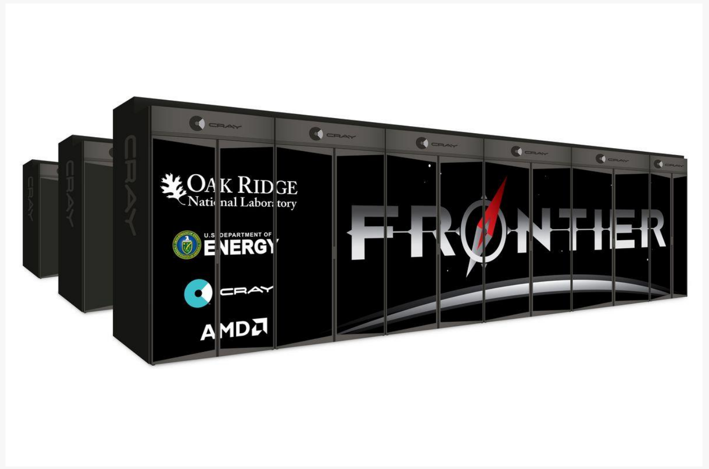
Frontier Supercomputer

- New exascale supercomputer being developed by Oak Ridge National Laboratory
- Cray system with AMD Epyc CPUs and Radeon Instinc GPUs
- Aimed for delivery end of 2021

Frontier Supercomputer

- New exascale supercomputer being developed by Argonne National Laboratory
- Cray system with Intel CPUs and GPUs
- Aimed for delivery in 2022

Performance Portability



- SYCL code will run without modifications on all three supercomputers
- Researchers collaborate on a single codebase knowing they can target all machines

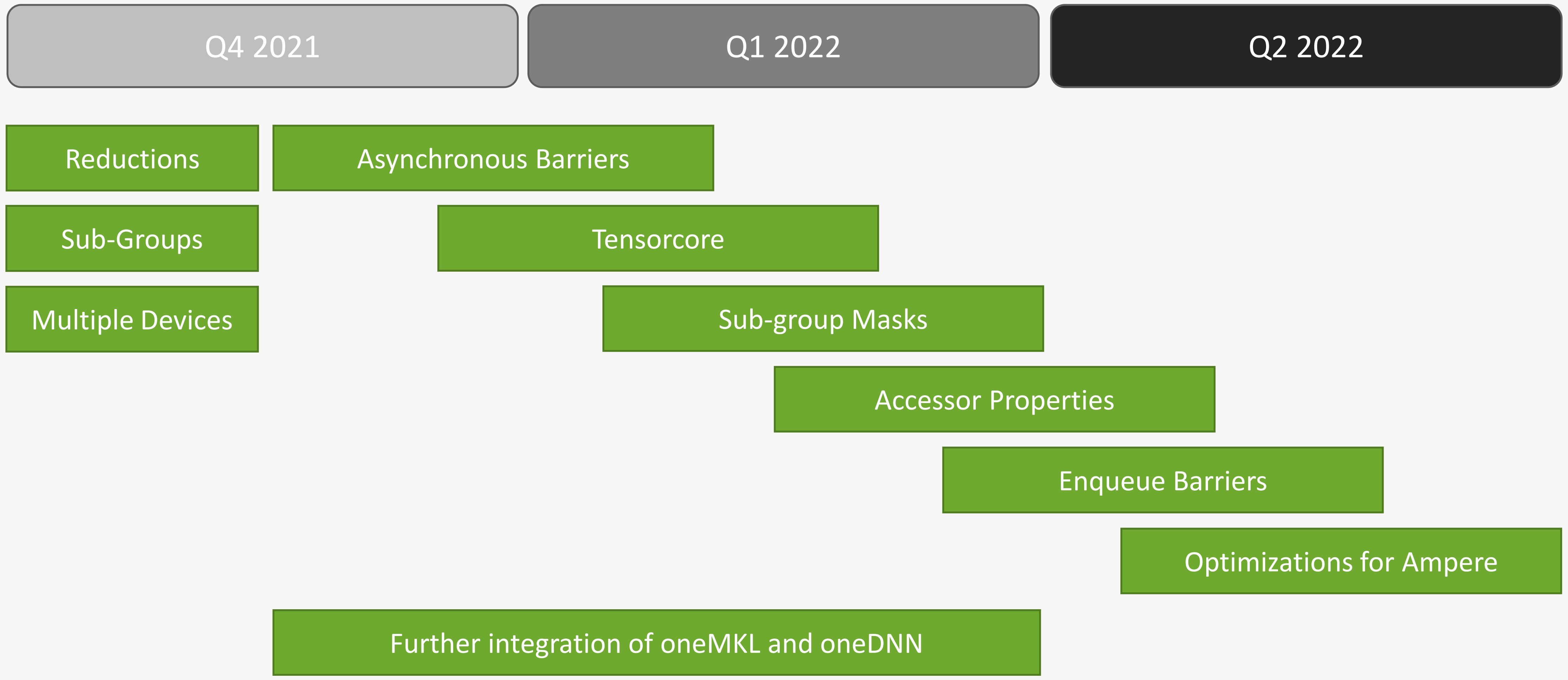
DPC++ for CUDA®

- Partnership between Codeplay, Berkeley Lab, and ANL
- Goals:
  - Expand DPC++ for CUDA to support for SYCL 2020 features
  - Expose NVIDIA Ampere features in DPC++ for CUDA
  - Optimize DPC++ for CUDA for NVIDIA Ampere hardware

BERKELEY LAB

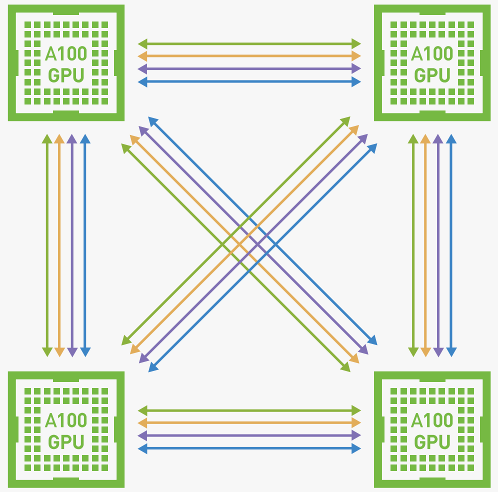
Argonne NATIONAL LABORATORY

oneAPI for CUDA Roadmap



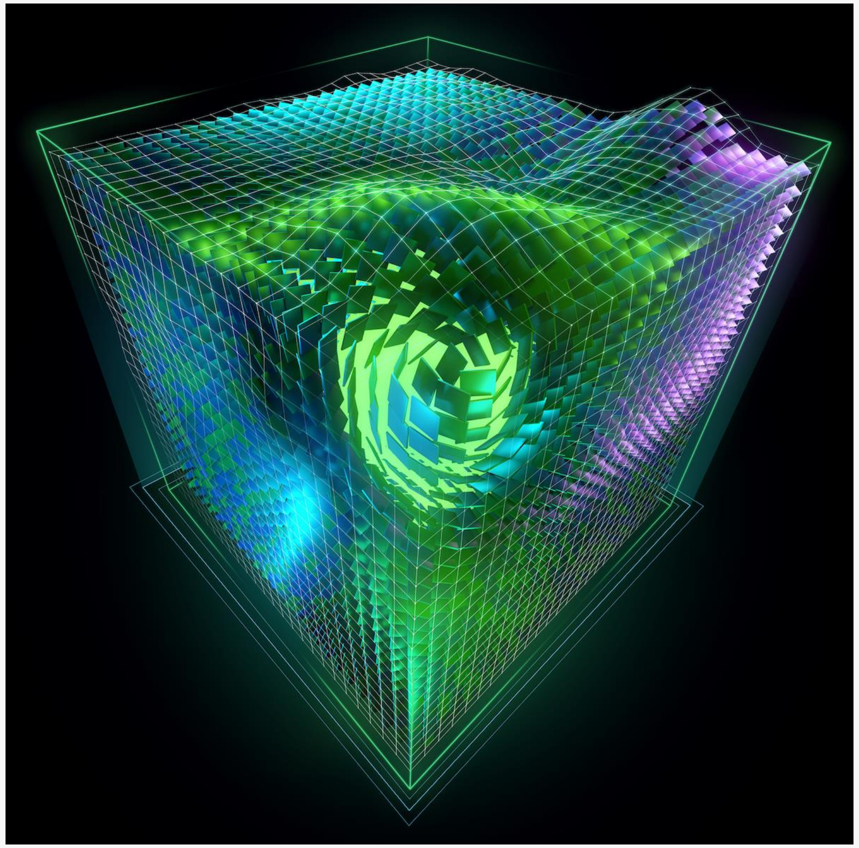
Peer-to-peer Copy

- DPC++ support for direct peer-to-peer copy
- More efficient communication between multiple GPUs
- Supporting both PCIe and NVLink
- Implementation is WIP



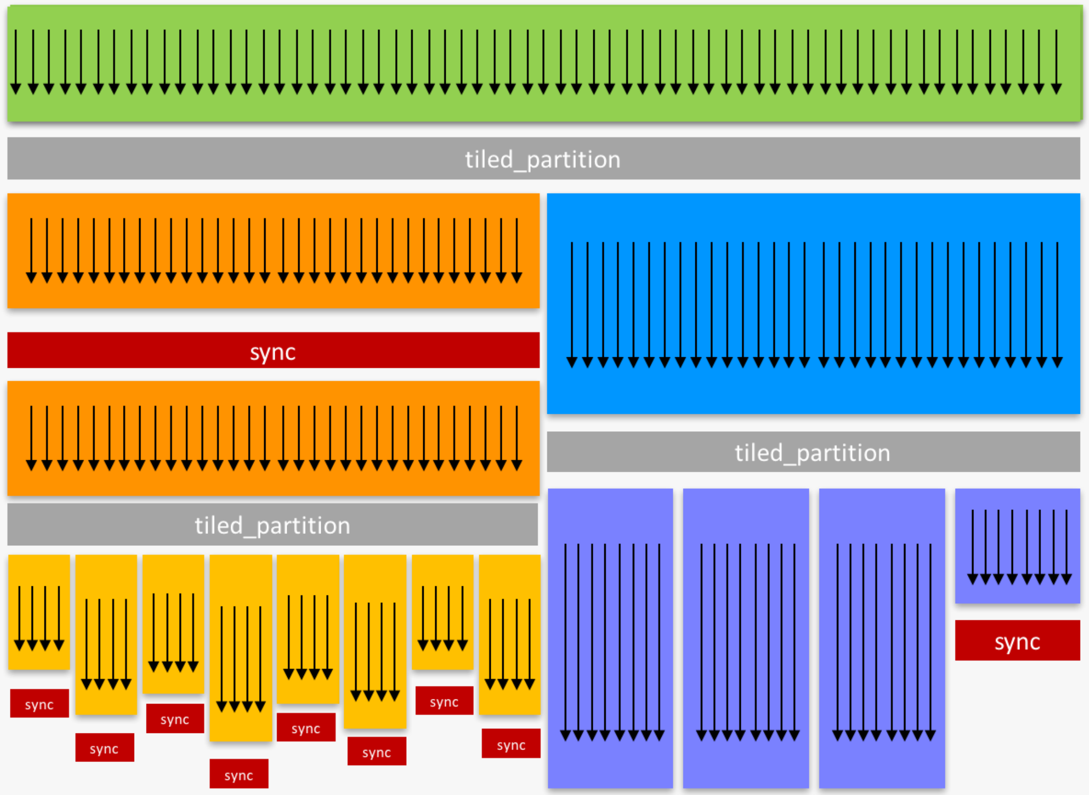
Extended Memory Model

- DPC++ support for extended atomics
- Extended atomic memory orders
- Extended atomic memory scopes
- Implementation is WIP



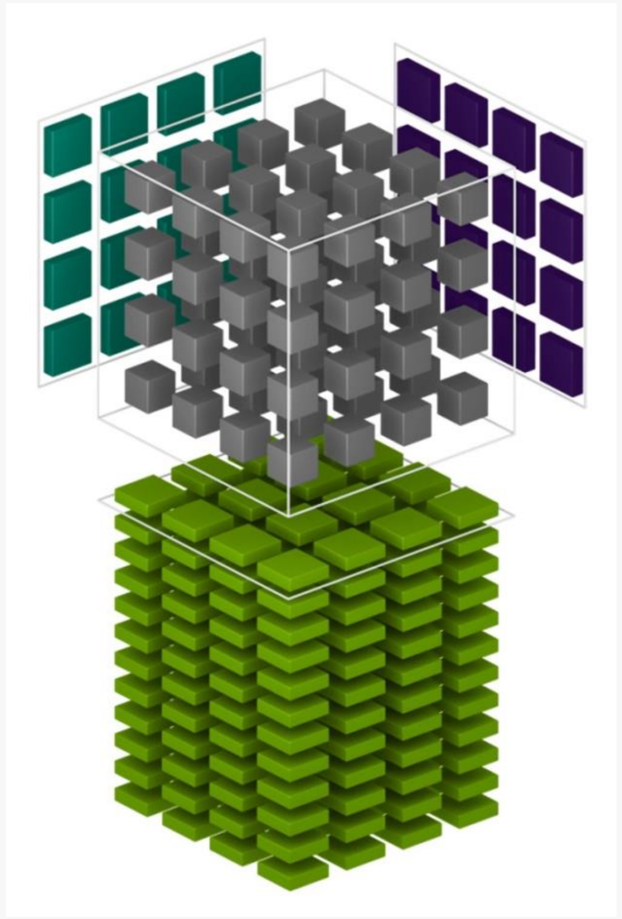
Group Collectives

- DPC++ extensions for additional SYCL group collective functions
- Providing support for CUDA collaborative group types
- Extending group functions to support these group types
- Extending group functions to support work-item masking
- Implementation is WIP



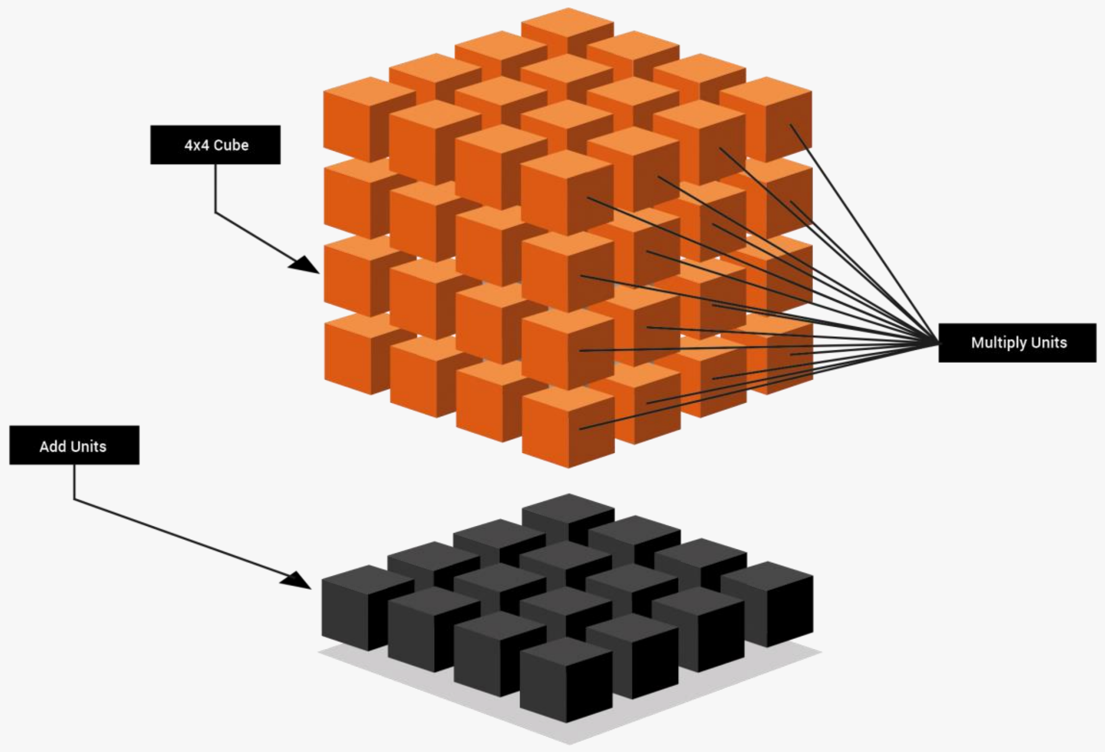
Tensorcore Instructions

- DPC++ joint_matrix extension extended to include Nvidia tensorcores for CUDA backend
- NVPTX MMA and WMMA instructions provide efficient matrix multiply-add operations
- These instructions support tf32 and bf16 data formats
- Implementation is WIP



Further Planned Features

- Future planned SYCL 2020 features / extensions:
  - Asynchronous barriers
  - Enqueue barriers
  - Accessor properties
  - Sub-group masks
  - Optimizations for Ampere
- Further oneMKL and oneDNN integration



DPC++ for HIP

- Partnership between Codeplay, ANL and Oak Ridge
- Goals:
  - Introduce support for AMD GPUs to DPC++ via the AMDGCN LLVM backend and HIP API

DPC++ HIP Backend

- New DPC++ backend for AMD GPUs
  - AMDGCN LLVM backend
  - PI HIP plugin
- Experimental support targeting HPC benchmarks
- Supports both AMD and Nvidia platforms
- Check it out:
  - https://intel.github.io/llvm-docs/GetStartedGuide.html#build-dpc-toolchain-with-support-for-hip-amd

```
git clone https://github.com/intel/llvm.git
cd llvm
python ./buildbot/configure.py --hip –t release
cd build
ninja install
```

```
clang++ -fsycl –fsycl-targets=amdgcn-amd-amdhsa sycl-app.cpp -o sycl-app-hip
```

Where to find out more?

Visit our website for set up instructions and learning materials

www.codeplay.com/oneapiforcuda/

We're Hiring!
codeplay.com/careers/

codeplay®
Enabling AI to be Open, Safe & Accessible to All

Thank You!

@codeplaysoft

info@codeplay.com

www
codeplay.com