codeplay

Enabling AI to be Open, Safe & Accessible to All

Using OCK as hardware toolkit for DPC++

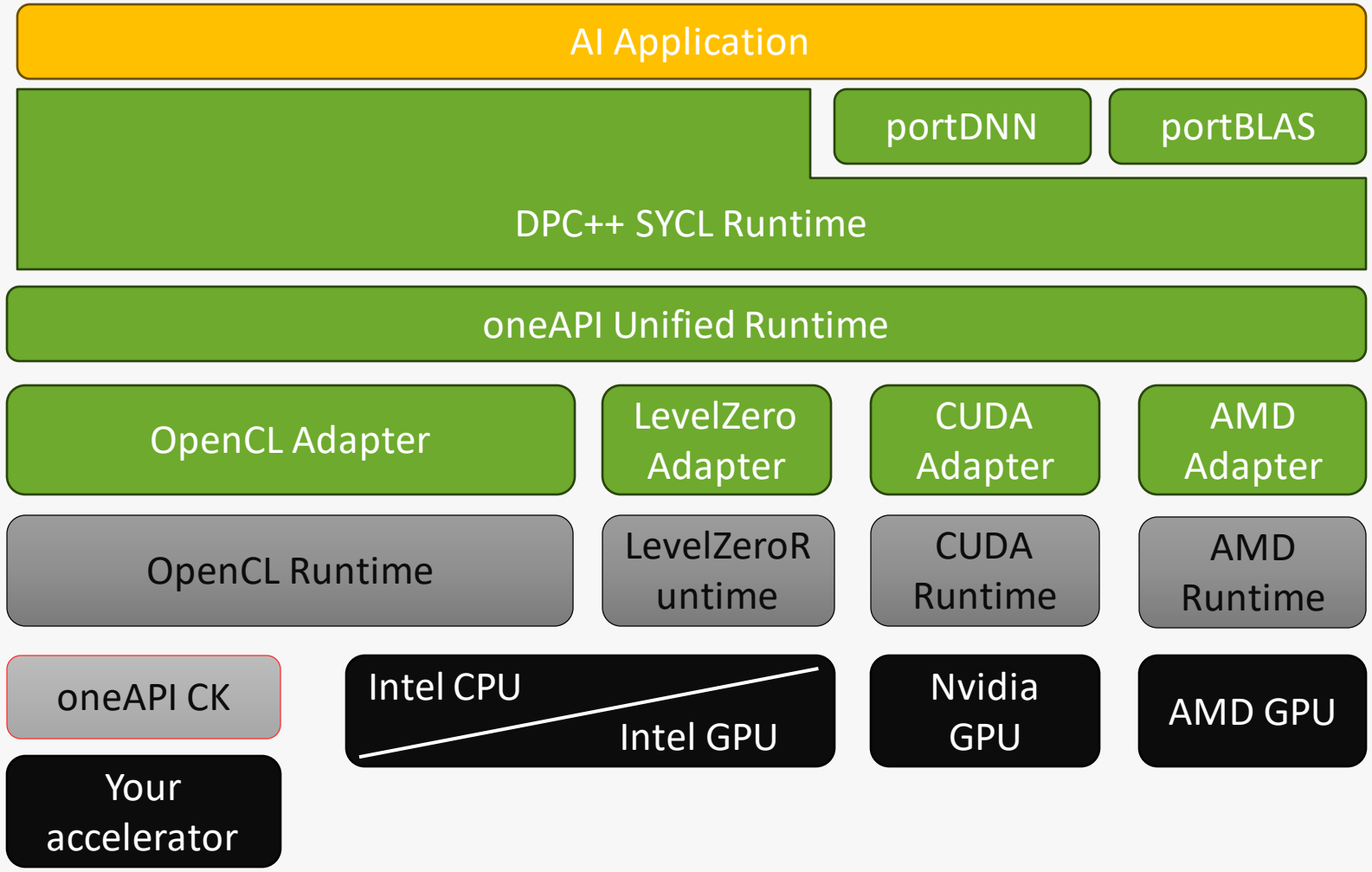
Colin Davidson PO oneAPI Construction Kit

12 February 2024

Agenda

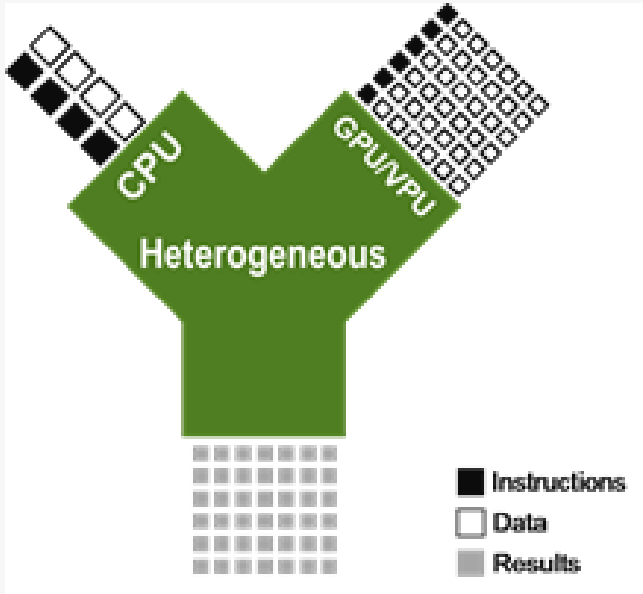
- How oneAPI Construction Kit fits into oneAPI
- Main components of oneAPI Construction Kit
- Hardware interfaces and abstraction
- Simplified toolkit (clik)
- Case study of code generation

The oneAPI Software Stack for SYCL



Offloading Model

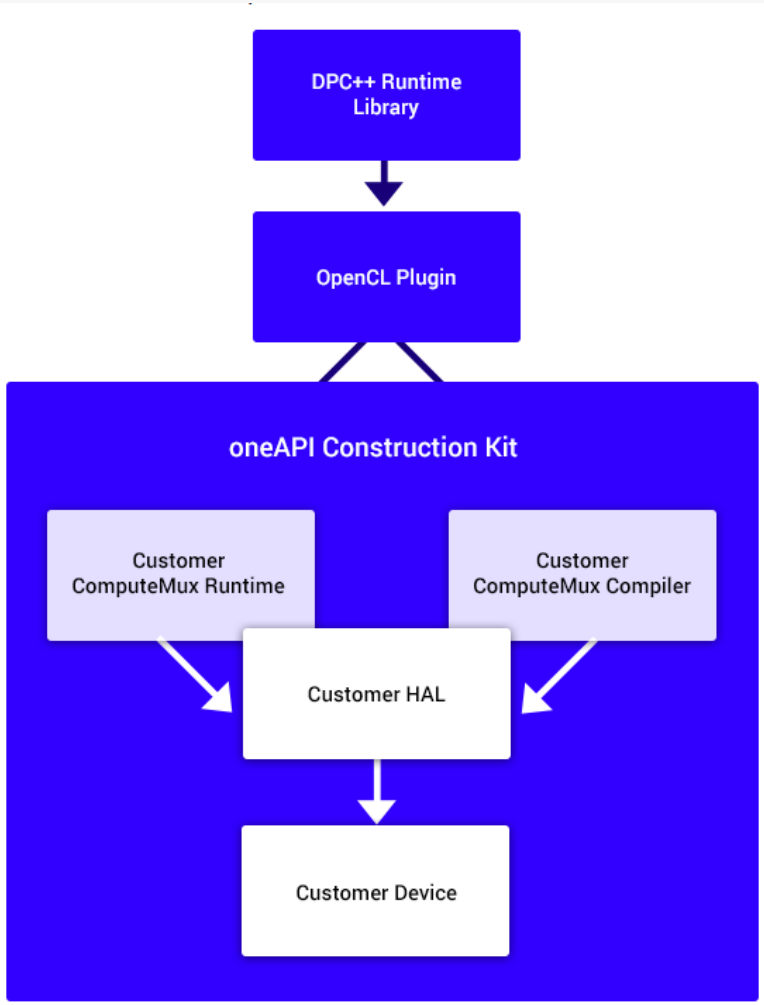
- Heterogeneous architectures require a CPU host device
- Enables offload to an accelerator



An *offloading compiler* offloads data-intensive code to the accelerator and keeps latency-sensitive code on the host CPU

oneAPI Construction Kit

- oneAPI Construction Kit enables integration of custom accelerators into the oneAPI software stack
- Only need to provide
  - Runtime component
    - e.g., data movement between host and accelerator
  - Device binary compiler
    - Sufficient to compile from SPIR-V to accelerator binary
    - Prior compilation from SYCL to SPIR-V handled by DPC++ compiler
- oneAPI Construction Kit is open-source
- Example targets "host" and RISC-V based.



ComputeMux Runtime

- Provides an interface for customers to implement their own runtime
  - Lower level than OpenCL
  - Queries
  - Device Creation/Destruction
  - Queues
  - Synchronization
  - Buffer Allocation
  - Read/Write Memory
  - Executing kernels over a range
- Flexible enough to map into different hardware
  - A GPU may map some of these features into hardware e.g. command buffers
  - Could implement all in software
- Interface is on the host, but much of this can be supported on the device as needed.
- Can run without a compiler, using prebuilt binaries
  - Assumes a specific interface of kernels in order to pass information down
    - Mainly scheduling and argument related

ComputeMux Compiler

- Customer interface to the compilation process
- Ultimately responsible for generating executable kernels that allow running work groups over a global range
  - A Work group is a group of work items running in parallel allowing for barriers for synchronization and local memory across the work group
  - The global range is split into a number of work groups
  - OCK provides a toolkit of compiler utilities to allow generating a compiler flow that works for different architectures
- Support for multiple architectures builtin
  - Arm, x86, RISC-V
- Customer provides llvm backend, so can adapt to others
- Customer will create a pass pipeline which meets the needs of their hardware in terms of the final executable
  - The final interface needs to map onto how the runtime views a kernel
  - Examples which customer can use as a basis.

ComputeMux Compiler

- Different hardware and library support will have different needs.
- We can flex to different models - main styles are:
  - whole work group in a single thread
  - work item per thread (typical GPU model)
  - OCK provides worked examples for both
  - Have supported either model in the past.
- Can add support for bespoke extensions
  - risc-v clmul included as a tutorial.
- Work group per thread requires work item loops and software barriers
  - Allows us to automatically vectorize across the workgroup using **vecz**
  - Barriers involves splitting the kernel into multiple distinct functions, each iterating over a workgroup
- SPIR-V and OpenCL builtins support
  - Maths builtins including fp16 support through **abacus**
  - Support for turning ID style builtins into something that works for the architecture
    - For example local id can get turned into **get_thread_id()** or used as part of a work item loop.
- Reference examples exists
  - Work item per thread, work group per thread, RISC-V, "host" target.

ComputeMux Runtime Simulator

- OCK includes its own reference simulator Refsi
  - RISC-V based, uses the Spike simulator as a basis
  - Multiple variants to show different aspects of hardware acceleration
  - Support RVV
  - Run as a library
  - Has its own driver interface
- OCK supply tutorials to create a fully functional OpenCL based on Refsi
- Has debug to help understand what is happening
- Easy to build or download for demo purposes.

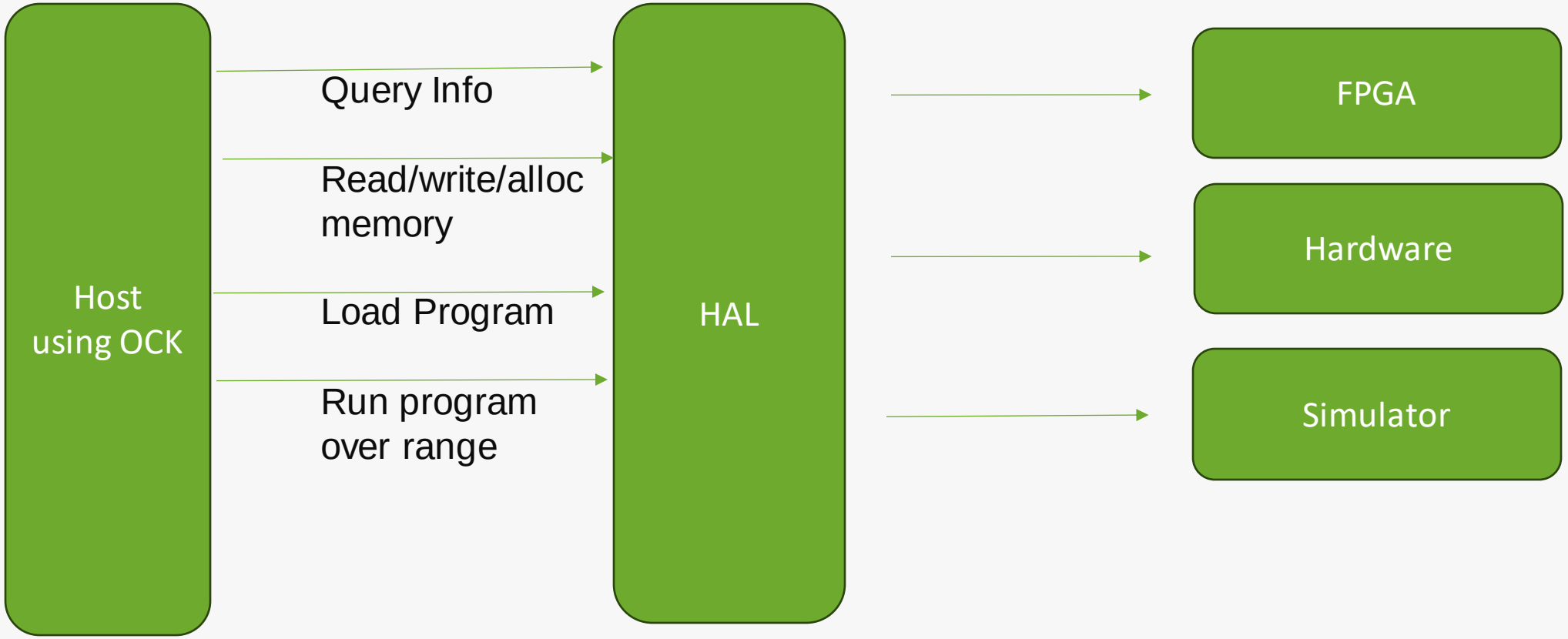
```
: core   0: 0x0000000000001000  (0x00000093)  li      ra, 0
: core   1: 0x0000000000001000  (0x00000093)  li      ra, 0
: core   0: 0x0000000000001004  (0x00000113)  li      sp, 0
: core   1: 0x0000000000001004  (0x00000113)  li      sp, 0
```

Hardware Interfaces

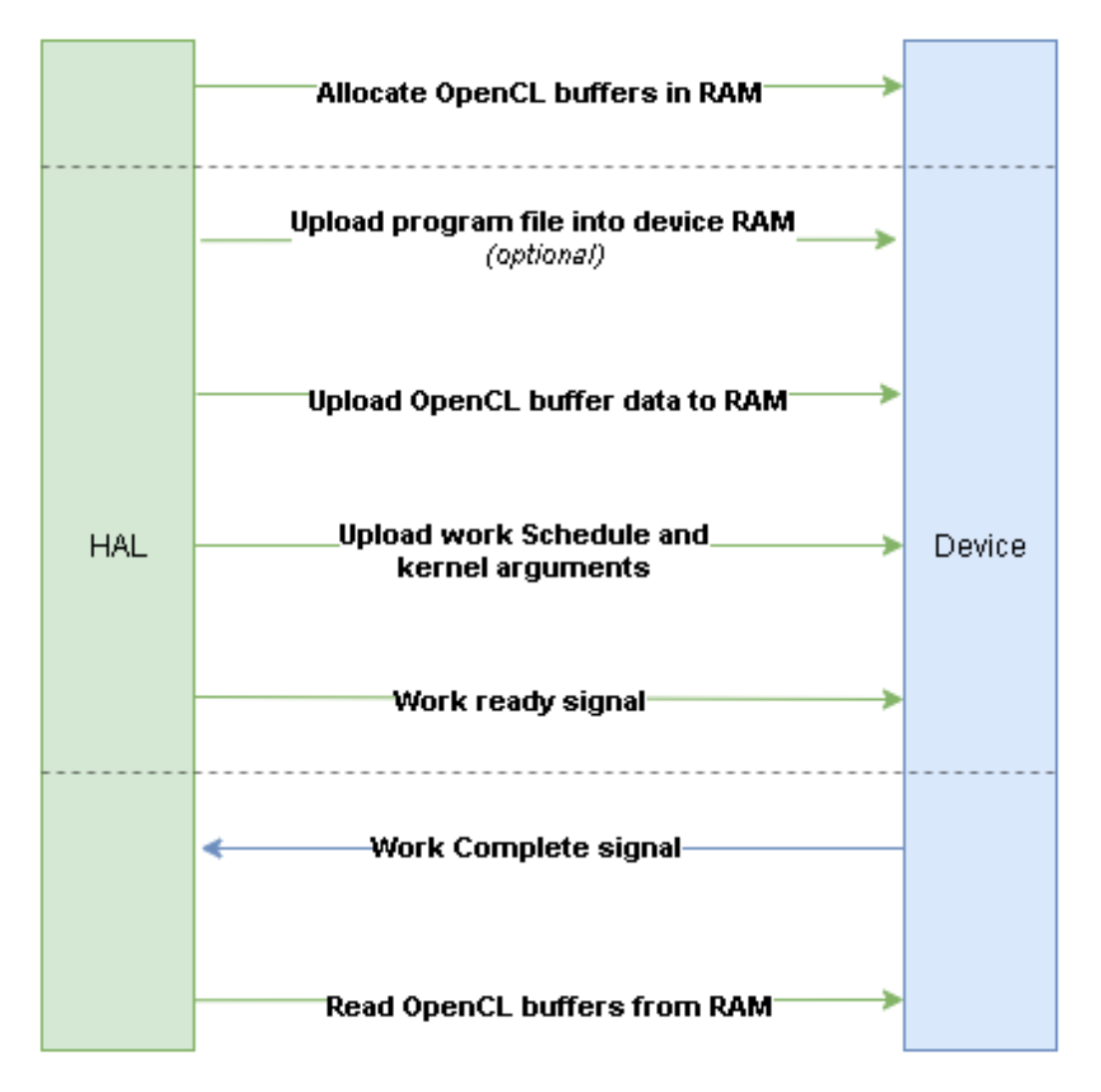
- How we interact with the accelerator is a vital part of the design
- There is some things that need to be thought of during the driver design for any hardware for remote host access:
  - How to allocate/free memory on the device
  - How to transfer data to/from device
  - How to run kernel executables on the device.
  - Even on a simulator this must be reasonably fast (e.g. debug interfaces can end up spending all their time transferring data)
- We have created a simplified HAL interface
  - Encapsulates basic methods to allow rapid support for new hardware.
  - Also encapsulates information about the device.
  - Can move towards ComputeMux Runtime as needed to improve performance.
  - Very useful for RISC-V where we can switch from simulator to hardware quickly only changing the HAL.
  - Support for a common ComputeMux Runtime which talks to the HAL.

Overview of the HAL

- HAL is our low-level interface to abstract the Hardware

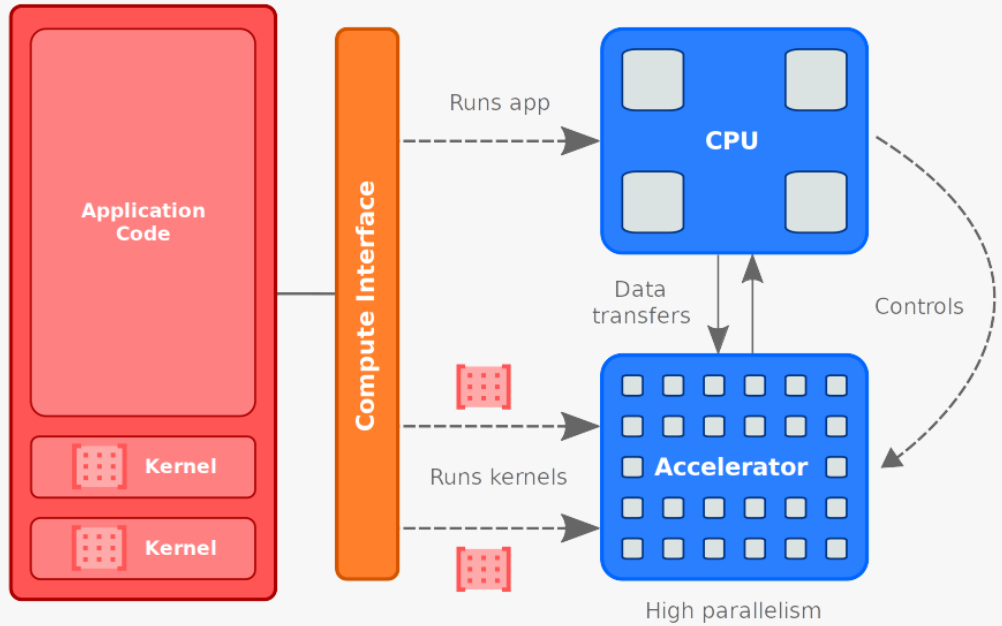


HAL and Simulator example



Simplified toolkit (clik)

- clik is a simple compute library
  - Minimum needed to run a kernel on an accelerator
  - Dynamic execution, data-parallel model
  - Supports multiple HW targets through the HAL API
- Can be used as scaffolding at the start of a project
  - Tiny code-base (~5K SLOC) with no dependencies
  - Includes a reference CPU implementation
  - Kernels compiled using existing C/C++ compiler
  - Can be used as a stepping stone to oneAPI Construction Kit
- Small test-suite for basic coverage and validation
  - Vector addition, matrix multiplication, convolution
  - Barrier and local memory



Create Target script

- In order to speed up the new target creation, we have added a script which will make a new mux target
  - Only need to provide a HAL and an LLVM backend
  - Uses JSON file to allow configurable inputs e.g. LLVM triples
  - Will tend to assume that the generated interface to the kernel is what you want.
  - Gives a starting point to allow changing it as needed.

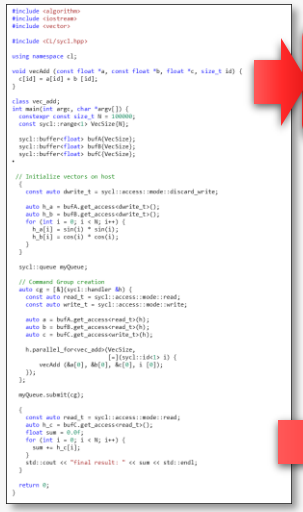
SYCL Native CPU

- There is ongoing work to support any architecture running natively on a CPU in oneAPI e.g. RISC-V
- This work is using common passes and code from OCK to provide
  - Work item loops
  - Barrier support
  - Vectorization

Case Study: SYCL on RISC-V

Compiling SYCL kernel to Native machine instruction (e.g. RISC-V Vectors)

SYCL source



Device Compiler

SYCL kernel

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

Scalar LLVM IR

Different memory (TCM, DRAM etc) annotated in IR

CPU Compiler

CPU

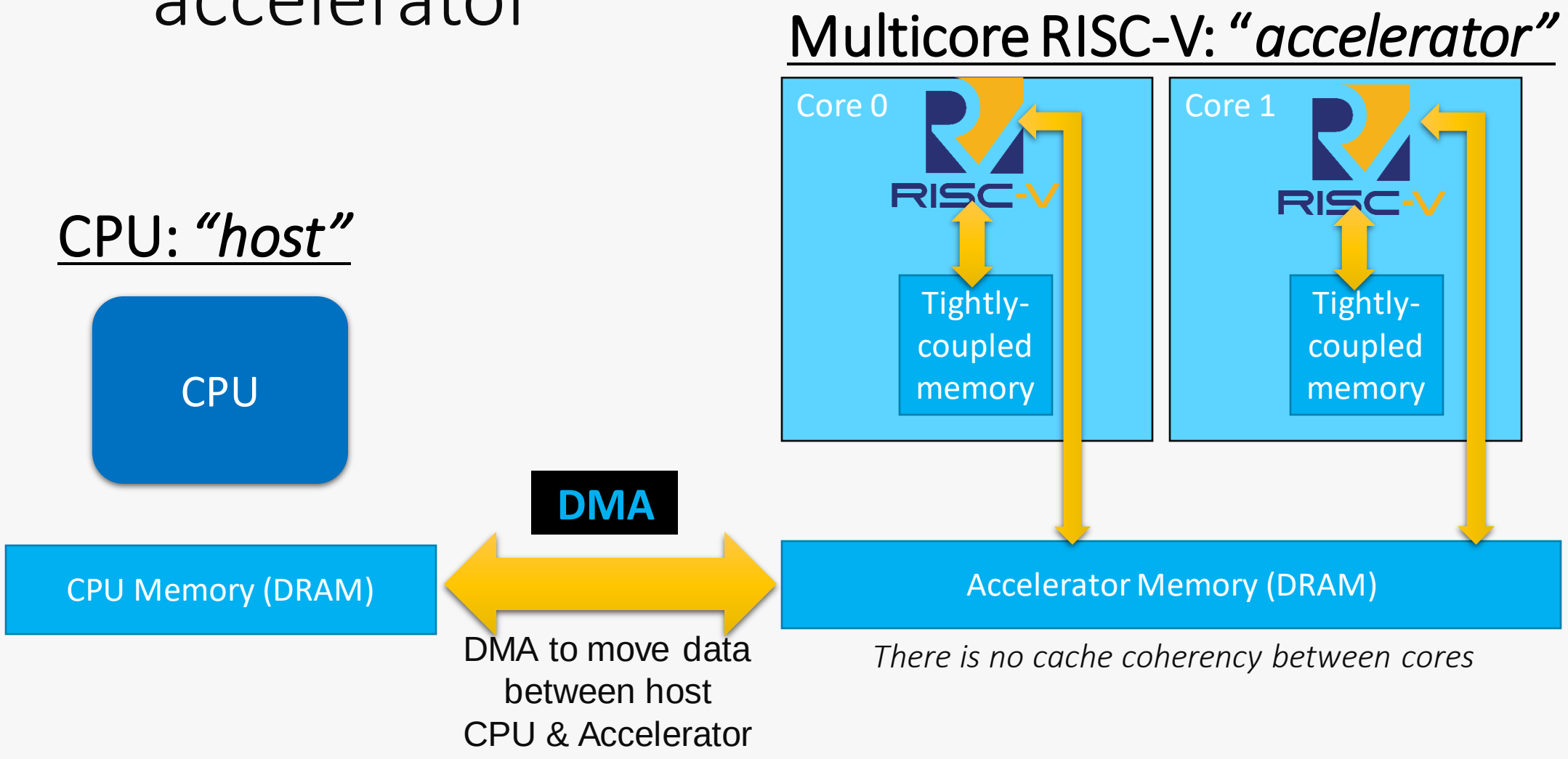
Whole Function Vectorization

Vector LLVM IR

LLVM back-end

RISC-V

Offloading to a RISC-V accelerator



Matmul: Kernel

SYCL Kernel

LLVM IR

SPIR-V

oneAPI Construction Kit
LLVM IR

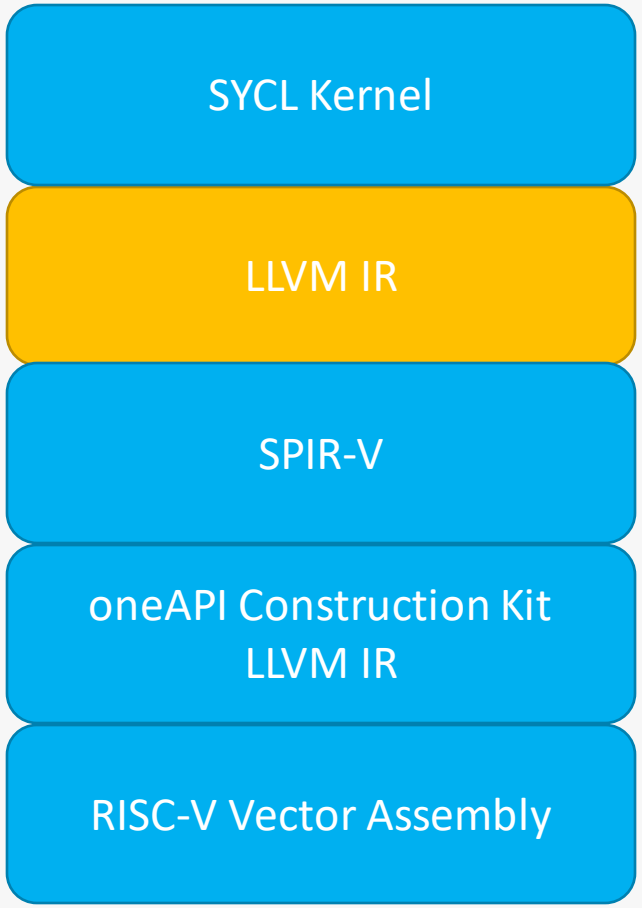
RISC-V Vector Assembly

```
cgh.parallel_for<mxm_kernel>(
    nd_range<2>{range<2>(matSize, matSize),
                range<2>(blockSize, blockSize)},
    [=](nd_item<2> it) {
    // Current block
    int blockX = it.get_group(1);
    int blockY = it.get_group(0);

    // Current local item
    int localX = it.get_local_id(1);
    int localY = it.get_local_id(0);

    // Start in the A matrix
    int a_start = matSize * blockSize * blockY;
    // End in the b matrix
    int a_end = a_start + matSize - 1;
    // Start in the b matrix
    int b_start = blockSize * blockX;

    // Result for the current C(i,j) element
    float tmp = 0.0f;
    // We go through all a, b blocks
    for (int a = a_start, b = b_start; a <= a_end;
            a += blockSize, b += (blockSize * matSize)) {
        // Copy the values in shared memory collectively
        pBA[localY * blockSize + localX] =
            pA[a + matSize * localY + localX];
        // Note the swap of X/Y to maintain contiguous access
        pBB[localX * blockSize + localY] =
            pB[b + matSize * localY + localX];
        it.barrier(access::fence_space::local_space);
        // Now each thread adds the value of its sum
        for (int k = 0; k < blockSize; k++) {
        tmp +=
            pBA[localY * blockSize + k] * pBB[localX * blockSize + k];
        }
        // The barrier ensures that all threads have written to local
        // memory before continuing
        it.barrier(access::fence_space::local_space);
    }
    auto elemIndex = it.get_global_id(0) * it.get_global_range()[1] +
                     it.get_global_id(1);
    // Each thread updates its position
    pC[elemIndex] = tmp;
    });
```

These are the 2 most inner kernel loops. The OCK compiler will *add the outer loops* to loop over all the data items to be processed by these cores

```
...
for (int a = a_start, b = b_start; a <= a_end;
                a += blockSize, b += (blockSize * matSize)) {
    for (int k = 0; k < blockSize; k++) {
    tmp +=
        pBA[localY * blockSize + k] * pBB[localX * blockSize + k];
    }
    // The barrier ensures that all threads have written to local
    // memory before continuing
    it.barrier(access::fence_space::local_space);
}
...
pC[elemIndex] = tmp;
...
```

Matmul : LLVM IR




```
; <label>:50:                                     ; preds = %55, %38
  %51 = phi float [ %39, %38 ], [ %67, %55 ]
  tail call spir_func void @_Z7barrierj(i32 1) #4
  %52 = add nsw i32 %41, %1
  %53 = add nsw i32 %40, %19
  %54 = icmp slt i32 %52, %21
  br i1 %54, label %38, label %70

; <label>:55:                                     ; preds = %55, %38
  %56 = phi float [ %67, %55 ], [ %39, %38 ]
  %57 = phi i32 [ %68, %55 ], [ 0, %38 ]
  %58 = add nsw i32 %57, %29
  %59 = sext i32 %58 to i64
  %60 = getelementptr inbounds float, float addrspace(3)* %2, i64 %59
  %61 = load float, float addrspace(3)* %60, align 4, !tbaa !11
  %62 = add nsw i32 %57, %33
  %63 = sext i32 %62 to i64
  %64 = getelementptr inbounds float, float addrspace(3)* %4, i64 %63
  %65 = load float, float addrspace(3)* %64, align 4, !tbaa !11
  %66 = fmul float %61, %65
  %67 = fadd float %56, %66
  %68 = add nuw nsw i32 %57, 1
  %69 = icmp slt i32 %68, %1
  br i1 %69, label %55, label %50
```

Matmul: SPIR-V

SYCL Kernel

LLVM IR

SPIR-V

oneAPI Construction Kit
LLVM IR

RISC-V Vector Assembly

```
2 Label 26
7 Phi 11 85 84 26 69 25
7 Phi 10 87 86 26 49 25
5 IAdd 10 88 87 58
4 SConvert 2 89 88
5 InBoundsPtrAccessChain 12 90 18 89
6 Load 11 91 90 2 4
5 IAdd 10 92 87 62
4 SConvert 2 93 92
5 InBoundsPtrAccessChain 12 94 20 93
6 Load 11 95 94 2 4
5 FMul 11 96 91 95
5 FAdd 11 84 85 96
5 IAdd 10 86 87 98
5 SLessThan 50 100 86 17
4 BranchConditional 100 26 27

2 Label 27
7 Phi 11 68 69 25 84 26
4 ControlBarrier 82 82 83
5 IAdd 10 72 73 17
5 IAdd 10 70 71 46
5 SLessThan 50 104 72 48
4 BranchConditional 104 25 28
```

Matmul: oneAPI Construction Kit LLVM IR

SYCL Kernel

LLVM IR

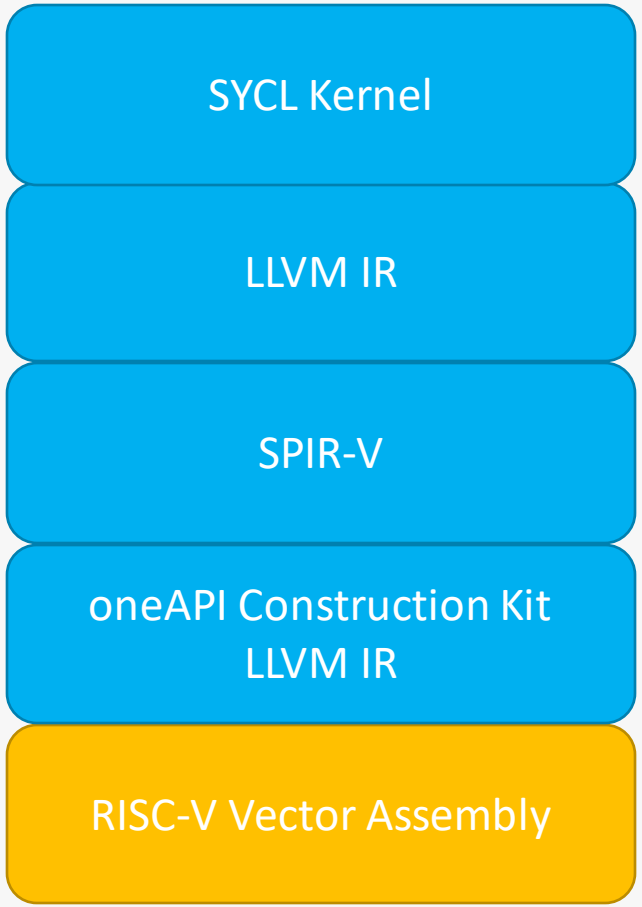
SPIR-V

oneAPI Construction Kit LLVM IR

RISC-V Vector Assembly

```
%855 = shufflevector <4 x float> %848, <4 x float> undef, <4 x i32> zeroinitializer
%mul.i.i.i37.i.i = fmul <4 x float> %855, %853
%add.i.i.i38.i.i = fadd <4 x float> %363, %mul.i.i.i37.i.i
%856 = shufflevector <4 x float> %848, <4 x float> undef, <4 x i32> <i32 1, i32 1, i32 1, i32 1>
%mul.i.i.i35.i.i = fmul <4 x float> %856, %852
%add.i.i.i36.i.i = fadd <4 x float> %mul.i.i.i35.i.i, %add.i.i.i38.i.i
%857 = shufflevector <4 x float> %848, <4 x float> undef, <4 x i32> <i32 2, i32 2, i32 2, i32 2>
%mul.i.i.i33.i.i = fmul <4 x float> %857, %851
%add.i.i.i34.i.i = fadd <4 x float> %mul.i.i.i33.i.i, %add.i.i.i36.i.i
%858 = shufflevector <4 x float> %848, <4 x float> undef, <4 x i32> <i32 3, i32 3, i32 3, i32 3>
%mul.i.i.i31.i.i = fmul <4 x float> %858, %854
%add.i.i.i32.i.i = fadd <4 x float> %mul.i.i.i31.i.i, %add.i.i.i34.i.i
%859 = shufflevector <4 x float> %849, <4 x float> undef, <4 x i32> zeroinitializer
%mul.i.i.i29.i.i = fmul <4 x float> %859, %853
%add.i.i.i30.i.i = fadd <4 x float> %362, %mul.i.i.i29.i.i
%860 = shufflevector <4 x float> %849, <4 x float> undef, <4 x i32> <i32 1, i32 1, i32 1, i32 1>
%mul.i.i.i27.i.i = fmul <4 x float> %860, %852
%add.i.i.i28.i.i = fadd <4 x float> %mul.i.i.i27.i.i, %add.i.i.i30.i.i
%861 = shufflevector <4 x float> %849, <4 x float> undef, <4 x i32> <i32 2, i32 2, i32 2, i32 2>
%mul.i.i.i25.i.i = fmul <4 x float> %861, %851
%add.i.i.i26.i.i = fadd <4 x float> %mul.i.i.i25.i.i, %add.i.i.i28.i.i
%862 = shufflevector <4 x float> %849, <4 x float> undef, <4 x i32> <i32 3, i32 3, i32 3, i32 3>
%mul.i.i.i23.i.i = fmul <4 x float> %862, %854
%add.i.i.i24.i.i = fadd <4 x float> %mul.i.i.i23.i.i, %add.i.i.i26.i.i
%863 = shufflevector <4 x float> %850, <4 x float> undef, <4 x i32> zeroinitializer
%mul.i.i.i21.i.i = fmul <4 x float> %863, %853
%add.i.i.i22.i.i = fadd <4 x float> %361, %mul.i.i.i21.i.i
%864 = shufflevector <4 x float> %850, <4 x float> undef, <4 x i32> <i32 1, i32 1, i32 1, i32 1>
%mul.i.i.i19.i.i = fmul <4 x float> %864, %852
%add.i.i.i20.i.i = fadd <4 x float> %mul.i.i.i19.i.i, %add.i.i.i22.i.i
%865 = shufflevector <4 x float> %850, <4 x float> undef, <4 x i32> <i32 2, i32 2, i32 2, i32 2>
%mul.i.i.i17.i.i = fmul <4 x float> %865, %851
%add.i.i.i18.i.i = fadd <4 x float> %mul.i.i.i17.i.i, %add.i.i.i20.i.i
%866 = shufflevector <4 x float> %850, <4 x float> undef, <4 x i32> <i32 3, i32 3, i32 3, i32 3>
%mul.i.i.i15.i.i = fmul <4 x float> %866, %854
%add.i.i.i16.i.i = fadd <4 x float> %mul.i.i.i15.i.i, %add.i.i.i18.i.i
```

Matmul: RISC-V Vector assembly




```
vsetvli a0, zero, e32, mf2, ta, mu
vfmul.vf    v24, v17, ft2
vfadd.vv    v30, v24, v30
vfmul.vf    v24, v16, ft2
vfadd.vv    v13, v24, v13
vfmul.vf    v24, v15, ft2
vfadd.vv    v22, v24, v22
vfmul.vf    v24, v14, ft2
vfadd.vv    v23, v24, v23
vsetivli    zero, 1, e32, mf2, ta, mu
vslidedown.vi   v24, v8, 2
vfmv.f.s    ft2, v24
vsetvli a0, zero, e32, mf2, ta, mu
vfmul.vf    v24, v12, ft2
vfadd.vv    v30, v24, v30
vfmul.vf    v24, v10, ft2
vfadd.vv    v13, v24, v13
vfmul.vf    v24, v9, ft2
vfadd.vv    v22, v24, v22
vfmul.vf    v24, v31, ft2
vfadd.vv    v23, v24, v23
vsetivli    zero, 1, e32, mf2, ta, mu
vslidedown.vi   v8, v8, 3
vfmv.f.s    ft2, v8
vsetvli a0, zero, e32, mf2, ta, mu
vfmul.vf    v8, v29, ft2
vfadd.vv    v2, v8, v30
vfmul.vf    v30, v28, ft2
vfadd.vv    v1, v30, v13
vfmul.vf    v30, v27, ft2
vfadd.vv    v24, v30, v22
vfmul.vf    v30, v26, ft2
vfadd.vv    v22, v30, v23
```

codeplay

Enabling AI to be Open, Safe & Accessible to All

Thank you!

@codeplaysoft

info@codeplay.com

codeplay.com