Extending DPC++ with Support for Huawei Ascend AI Chipset

Wilson Feng **Rasool Maghareh** **Amy Wang**

Huawei Heterogeneous Compiler Lab, Canada

IWOCL & SYCLcon 2021

April 2021

Agenda

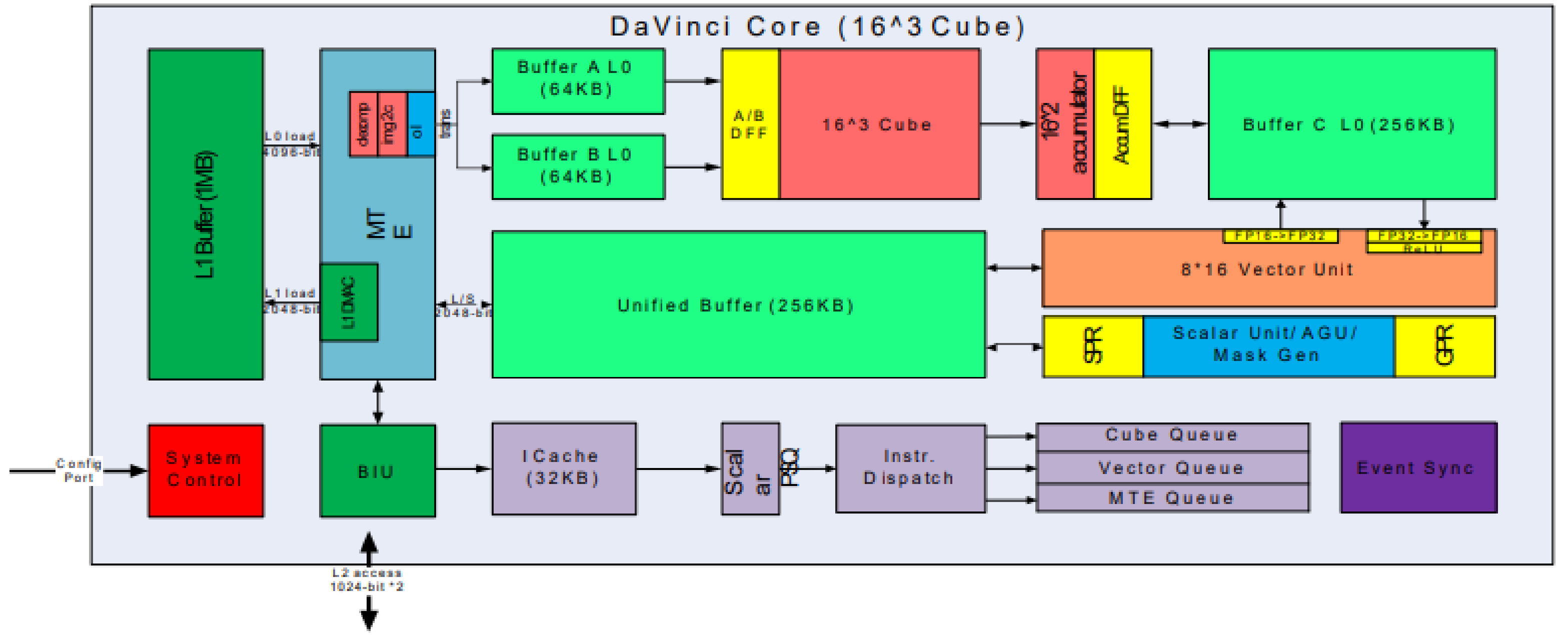
- Huawei Ascend AI Background
- Our Contribution to DPC++
  - CCE SYCL Backend
  - CCE SYCL Plugin
  - Compilation Toolchain
  - Extension to USM
  - Support for Parallel_for
- Supported Examples
- Future Work

Huawei Ascend AI Background

- Huawei’s custom SoC ASIC for AI workloads
- Host-Device programming model
  - Generic C++ Host Code
  - CCE Device Code – C/C++ based programming language for Ascend AI devices
    - Some C++ features are disabled
    - Explicit software management on different hardware pipelines (DMA transfers and synchronization)

Huawei Ascend AI Background

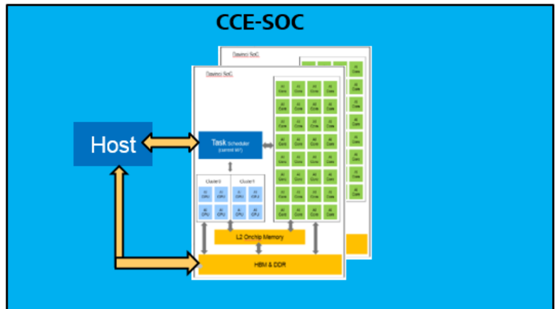
DaVinci Core



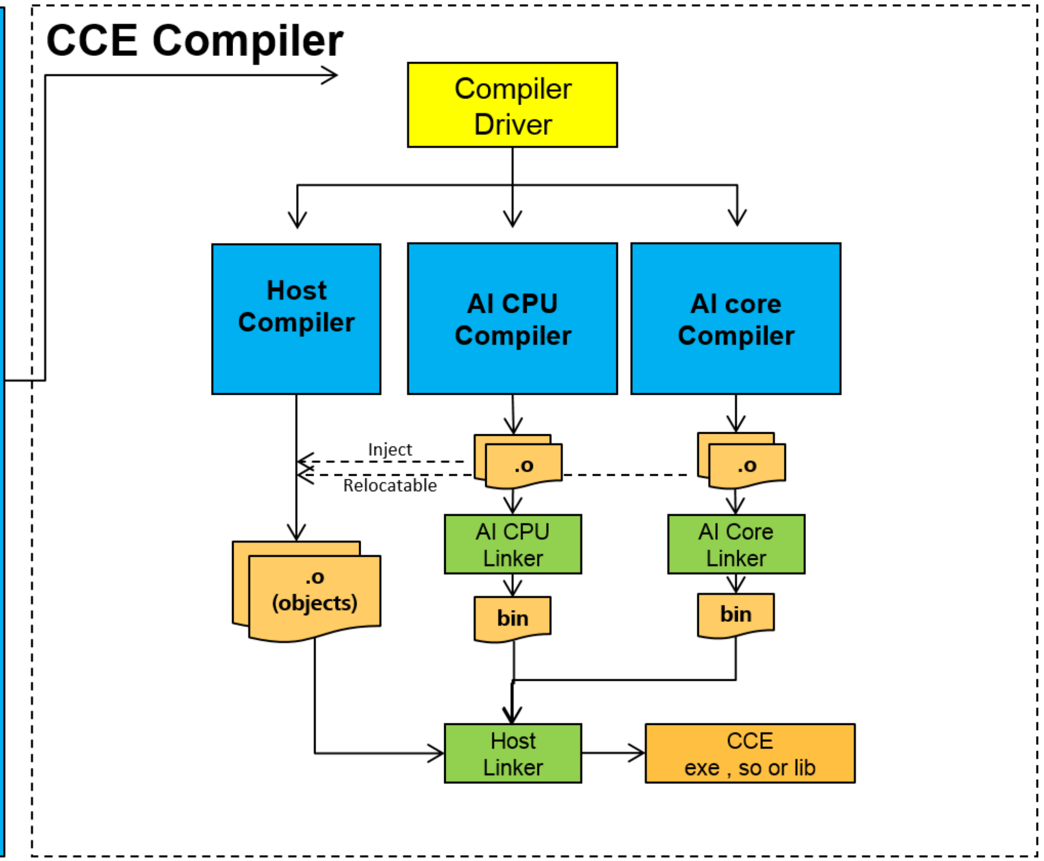
- **Cube**: 4096(16^3) FP16 MACs + 8192 INT8 MACs
- **Vector**: 2048bit INT8/FP16/FP32 vector with special functions (activation functions, NMS- Non Minimum Suppression, ROI, SORT)
- Explicit memory hierarchy design, managed by MTE

H. Liao, J. Tu, J. Xia and X. Zhou, "DaVinci: A Scalable Architecture for Neural Network Computing," 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 2019, pp. 1-44, doi: 10.1109/HOTCHIPS.2019.8875654.

CCEC Compilation

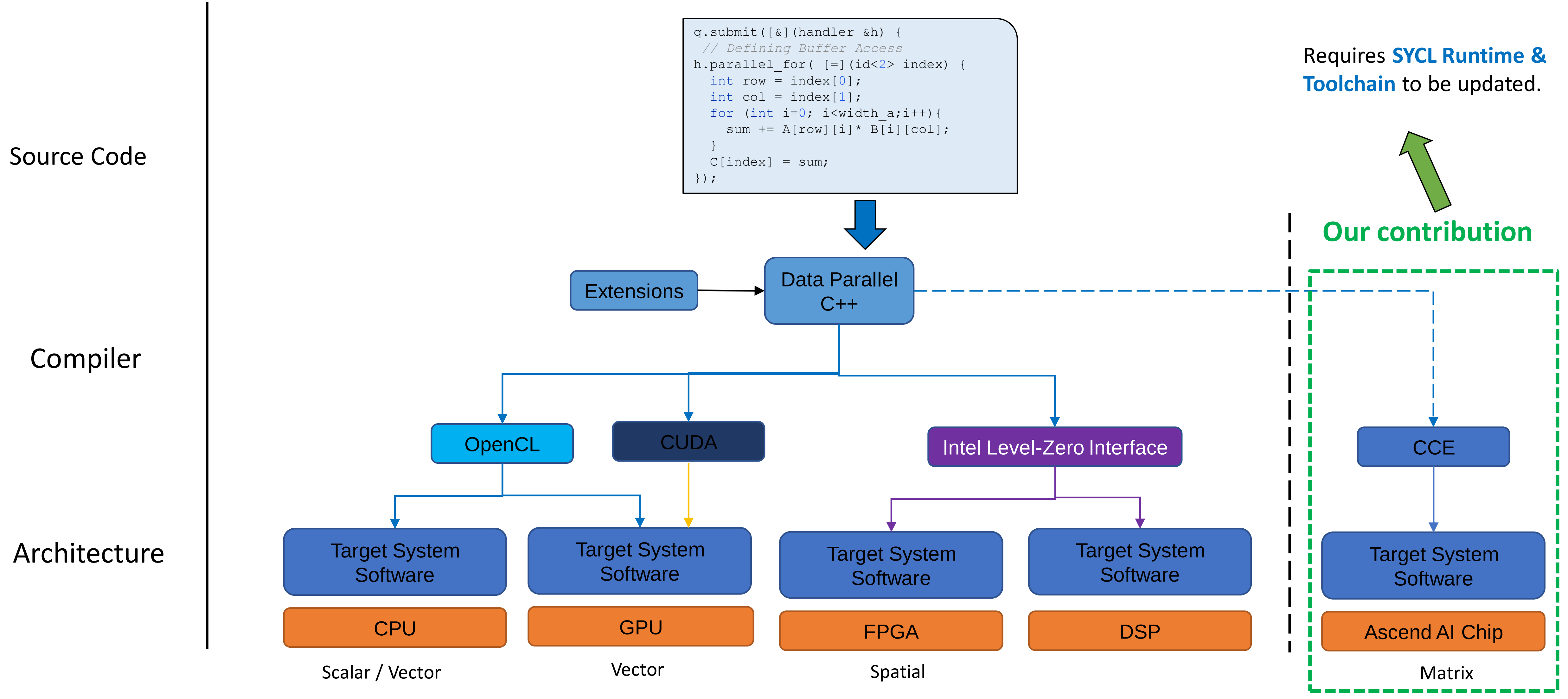


```
//axpy.cce
__global__ [aicpu] void foo1(__gm__ int*
buf) {
   ...
}
__global__ [aicore] void foo2(__gm__ int*
buf) {
  ...
}
int main(int argc, char* argv[]){
  ...
  int a[100];
  int *da = rtMalloc(…);
  rtMemcpy(da, a);
  foo0<<< nBlocks, smDesc, streamId
>>>(da);

  return 0;
}
```



- The source codes are firstly divided into three parts according to the function attribute.
- We use different compilers to compile different types of source codes, then we get the **linked** aicore and aicpu binary and relocatable host objects.
- The key step is to link device binaries and host objects together.

Our Contribution: CCE SYCL Plugin



```
q.submit([&](handler &h) {
 // Defining Buffer Access
h.parallel_for( [=](id<2> index) {
  int row = index[0];
  int col = index[1];
  for (int i=0; i<width_a;i++){
    sum += A[row][i]* B[i][col];
  }
  C[index] = sum;
});
```

Requires **SYCL Runtime & Toolchain** to be updated.

Our Contribution: CCE SYCL Backend

CCE Backend added to SYCL Runtime:

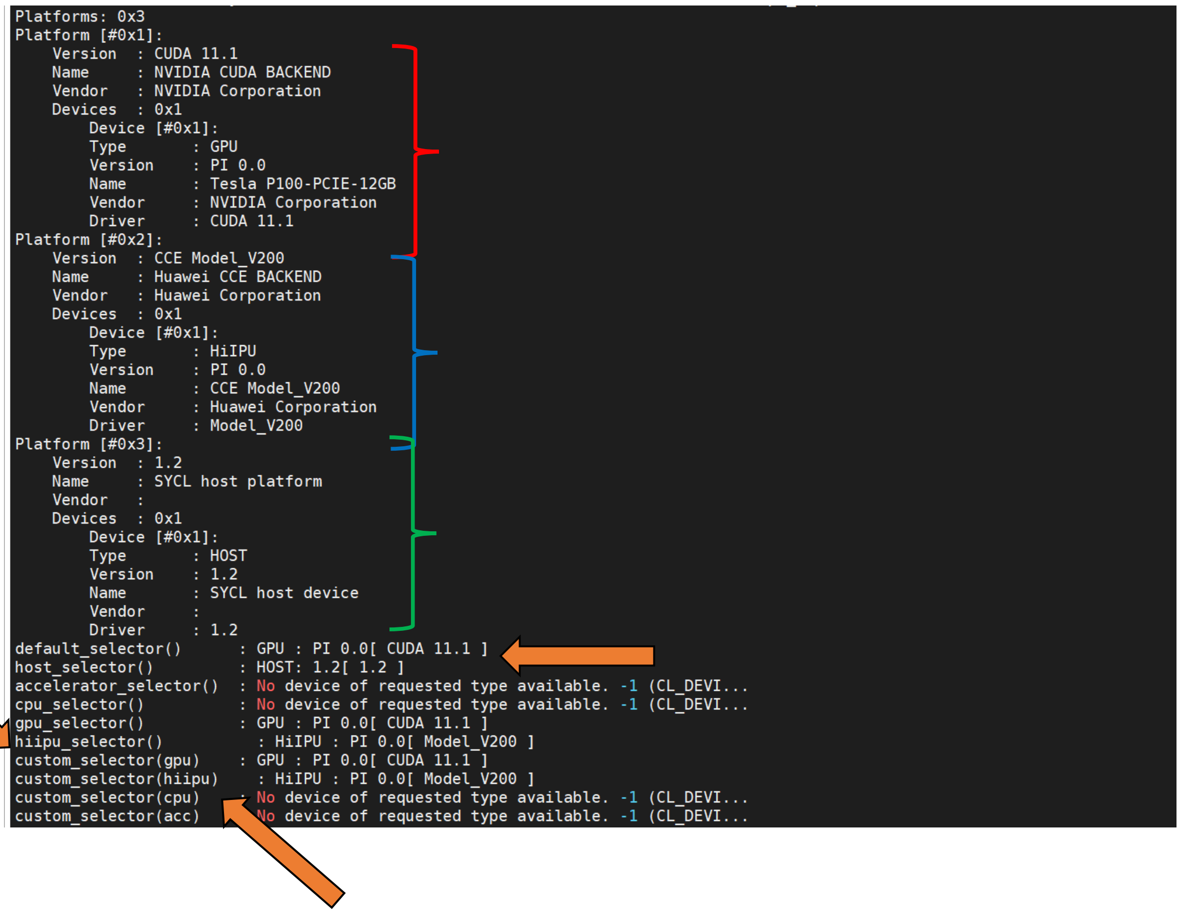
1. **Device** discovery and selection
2. **Platform** interface
3. **Context** interface
4. **Queue & Event** interfaces

Selector classes in DPC++:

- host_selector, cpu_selector, gpu_selector,…
- Added **hiipu_selector()** which selects Huawei Ascend AI Chipset

cl::sycl::queue Queue(cl::sycl::**hiipu_selector**{});

sycl-ls --verbose



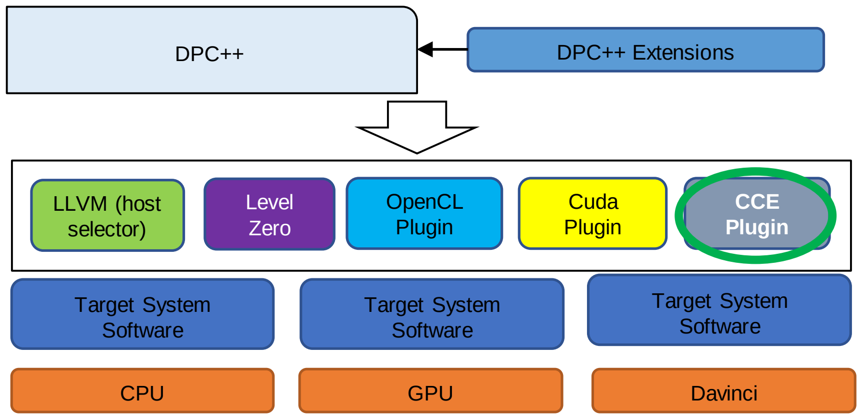
```
Platforms: 0x3
Platform [#0x1]:
    Version  : CUDA 11.1
    Name     : NVIDIA CUDA BACKEND
    Vendor   : NVIDIA Corporation
    Devices  : 0x1
        Device [#0x1]:
        Type        : GPU
        Version     : PI 0.0
        Name        : Tesla P100-PCIE-12GB
        Vendor      : NVIDIA Corporation
        Driver      : CUDA 11.1
Platform [#0x2]:
    Version  : CCE Model_V200
    Name     : Huawei CCE BACKEND
    Vendor   : Huawei Corporation
    Devices  : 0x1
        Device [#0x1]:
        Type        : HiIPU
        Version     : PI 0.0
        Name        : CCE Model_V200
        Vendor      : Huawei Corporation
        Driver      : Model_V200
Platform [#0x3]:
    Version  : 1.2
    Name     : SYCL host platform
    Vendor   :
    Devices  : 0x1
        Device [#0x1]:
        Type        : HOST
        Version     : 1.2
        Name        : SYCL host device
        Vendor      :
        Driver      : 1.2
default_selector()      : GPU : PI 0.0[ CUDA 11.1 ]
host_selector()         : HOST: 1.2[ 1.2 ]
accelerator_selector()  : No device of requested type available. -1 (CL_DEVI...
cpu_selector()          : No device of requested type available. -1 (CL_DEVI...
gpu_selector()          : GPU : PI 0.0[ CUDA 11.1 ]
hiipu_selector()          : HiIPU : PI 0.0[ Model_V200 ]
custom_selector(gpu)    : GPU : PI 0.0[ CUDA 11.1 ]
custom_selector(hiipu)    : HiIPU : PI 0.0[ Model_V200 ]
custom_selector(cpu)    : No device of requested type available. -1 (CL_DEVI...
custom_selector(acc)    : No device of requested type available. -1 (CL_DEVI...
```

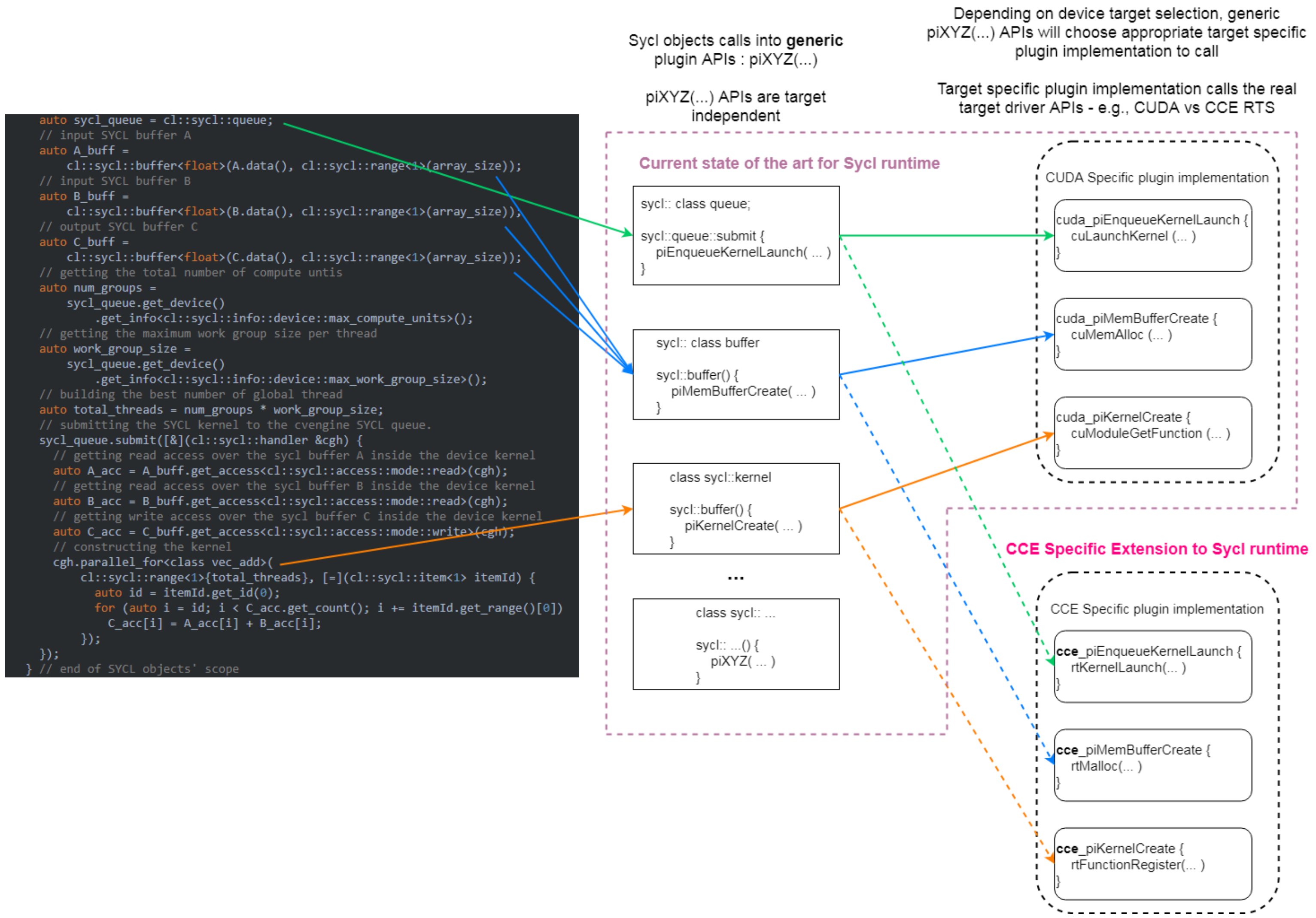
Our Contribution: CCE SYCL Plugin

CCE Runtime Plugin: The runtime plugin performs **command group scope** instructions which act as an interface between the host and device.

Current plugins: OpenCL, Level_zero, CUDA

CCE Plugin: Implementation similar to the CUDA plugin





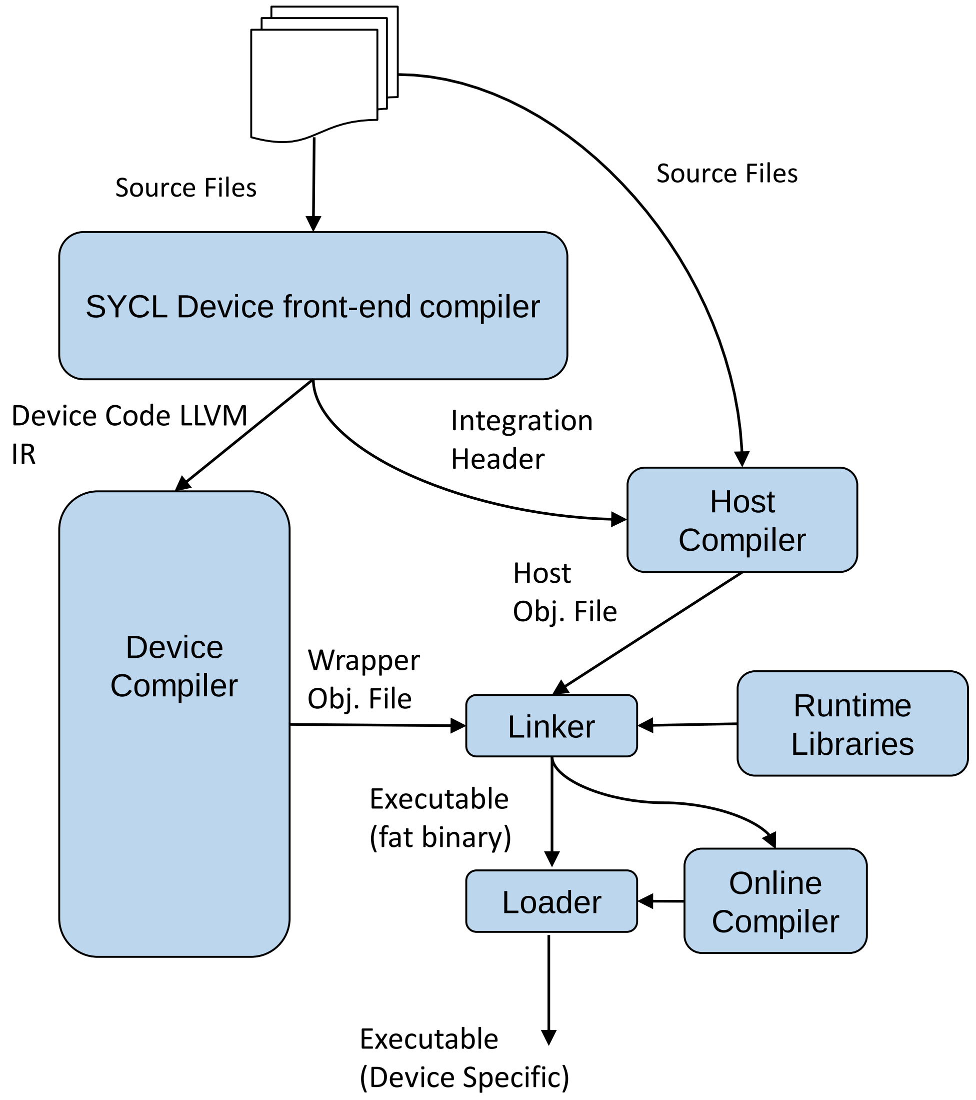
```
    auto sycl_queue = cl::sycl::queue;
    // input SYCL buffer A
    auto A_buff =
        cl::sycl::buffer<float>(A.data(), cl::sycl::range<1>(array_size));
    // input SYCL buffer B
    auto B_buff =
        cl::sycl::buffer<float>(B.data(), cl::sycl::range<1>(array_size));
    // output SYCL buffer C
    auto C_buff =
        cl::sycl::buffer<float>(C.data(), cl::sycl::range<1>(array_size));
    // getting the total number of compute untis
    auto num_groups =
        sycl_queue.get_device()
            .get_info<cl::sycl::info::device::max_compute_units>();
    // getting the maximum work group size per thread
    auto work_group_size =
        sycl_queue.get_device()
            .get_info<cl::sycl::info::device::max_work_group_size>();
    // building the best number of global thread
    auto total_threads = num_groups * work_group_size;
    // submitting the SYCL kernel to the cvengine SYCL queue.
    sycl_queue.submit([&](cl::sycl::handler &cgh) {
      // getting read access over the sycl buffer A inside the device kernel
      auto A_acc = A_buff.get_access<cl::sycl::access::mode::read>(cgh);
      // getting read access over the sycl buffer B inside the device kernel
      auto B_acc = B_buff.get_access<cl::sycl::access::mode::read>(cgh);
      // getting write access over the sycl buffer C inside the device kernel
      auto C_acc = C_buff.get_access<cl::sycl::access::mode::write>(cgh);
      // constructing the kernel
      cgh.parallel_for<class vec_add>(
          cl::sycl::range<1>{total_threads}, [=](cl::sycl::item<1> itemId) {
            auto id = itemId.get_id(0);
            for (auto i = id; i < C_acc.get_count(); i += itemId.get_range()[0])
              C_acc[i] = A_acc[i] + B_acc[i];
          });
    });
  } // end of SYCL objects' scope
```

Sycl objects calls into **generic** plugin APIs : piXYZ(...)
piXYZ(...) APIs are target independent
Depending on device target selection, generic piXYZ(...) APIs will choose appropriate target specific plugin implementation to call
Target specific plugin implementation calls the real target driver APIs - e.g., CUDA vs CCE RTS
Current state of the art for Sycl runtime
sycl:: class queue; sycl::queue::submit { piEnqueueKernelLaunch(...) }
sycl:: class buffer sycl::buffer() { piMemBufferCreate(...) }
class sycl::kernel sycl::buffer() { piKernelCreate(...) }
...
class sycl:: ... sycl:: ...() { piXYZ(...) }
CUDA Specific plugin implementation
cuda_piEnqueueKernelLaunch { cuLaunchKernel (...) }
cuda_piMemBufferCreate { cuMemAlloc (...) }
cuda_piKernelCreate { cuModuleGetFunction (...) }
CCE Specific Extension to Sycl runtime
CCE Specific plugin implementation
cce_piEnqueueKernelLaunch { rtKernelLaunch(...) }
cce_piMemBufferCreate { rtMalloc(...) }
cce_piKernelCreate { rtFunctionRegister(...) }

Background: Compilation Process in DPC++

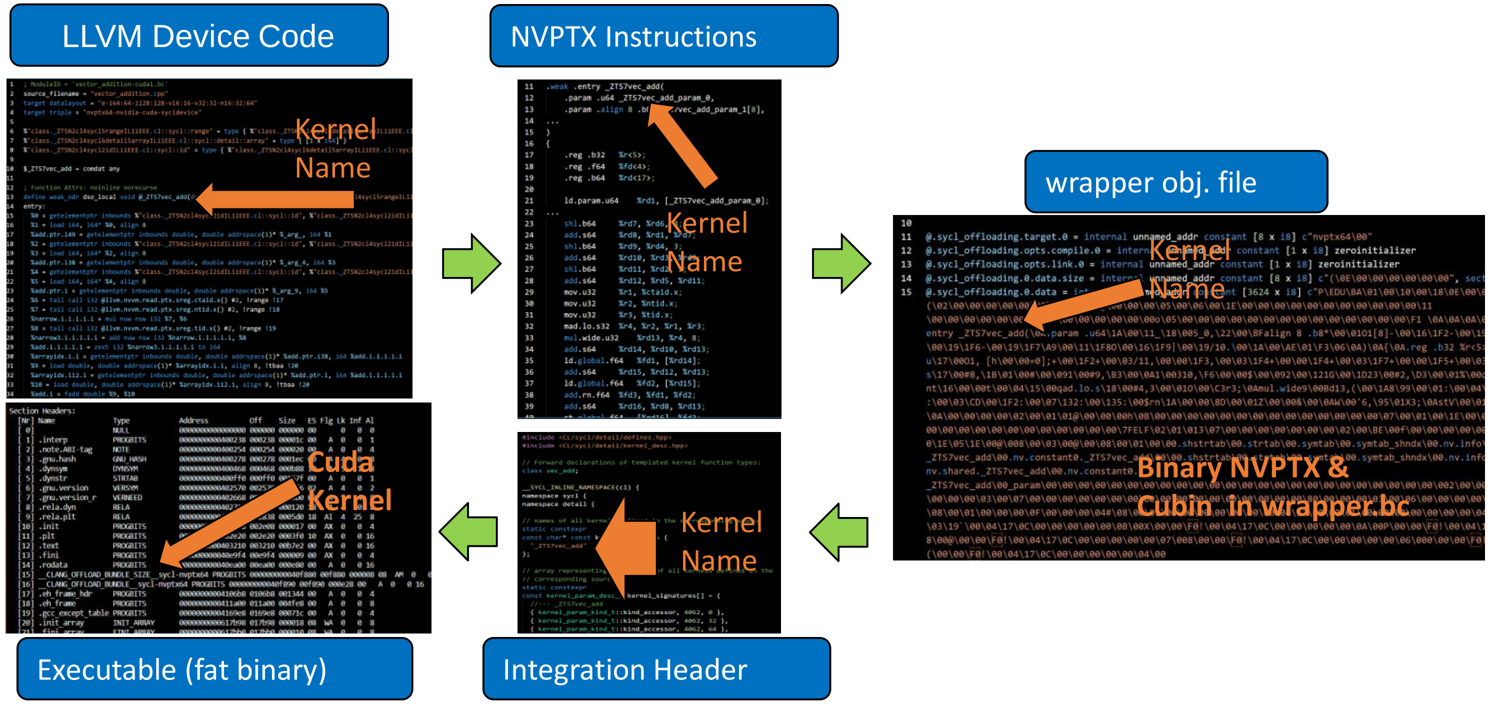
- Source codes contain both **host** and **device** code.
- DPC++ compiler **bundles** the **host codes together** and the **device code together**.
- **Device** code is compiled separately and stored in a **wrapper obj. file**.
- **Host** code is compiled with an **Integration Header** containing kernel information
- **Linker** links host and wrapper obj. files + **Runtime Libs.**
- **Loader** loads the **fat binary** for execution. It checks if a target specific executable image exists. If not, the generic image is loaded and **compiled online** to target specific image.

https://github.com/intel/llvm/tree/sycl/sycl/doc



Background: Cuda Device Code Compilation

- Device Code is compiled to NVPTX and then to Cubin
- This target-specific image is stored in wrapper obj. file
- Online compilation not required



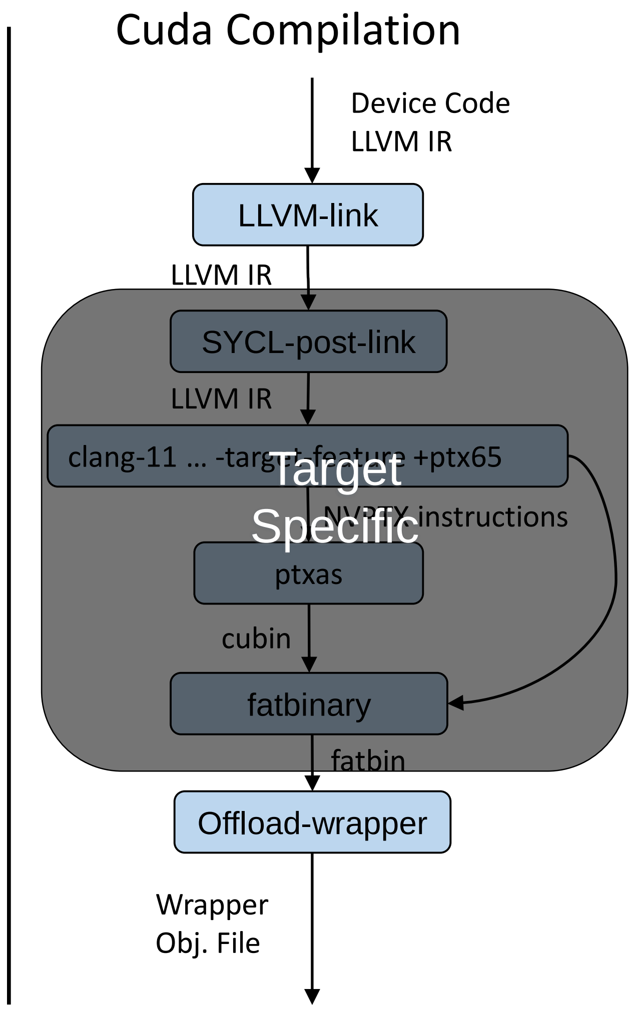
```
; ModuleID = 'vector_addition-cuda1.bc'
source_filename = "vector_addition.cpp"
target datalayout = "e-i64:64-i128:128-v16:16-v32:32-n16:32:64"
target triple = "nvptx64-nvidia-cuda-sycldevice"

%"class._ZTSN2cl4sycl5rangeILi1EEE.cl::sycl::range" = type { %"class._ZTSN2cl4sycl6detail5arrayILi1EEE.cl
%"class._ZTSN2cl4sycl6detail5arrayILi1EEE.cl::sycl::detail::array" = type { [1 x i64] }
%"class._ZTSN2cl4sycl2idILi1EEE.cl::sycl::id" = type { %"class._ZTSN2cl4sycl6detail5arrayILi1EEE.cl::sycl

$_ZTS7vec_add = comdat any

; Function Attrs: noinline norecurse
define weak_odr dso_local void @_ZTS7vec_add(d                          l4sycl5rangeILi1
entry:
  %0 = getelementptr inbounds %"class._ZTSN2cl4syc 2idILi1EEE.cl::sycl::id", %"class._ZTSN2cl4sycl2idILi1
  %1 = load i64, i64* %0, align 8
  %add.ptr.i49 = getelementptr inbounds double, double addrspace(1)* %_arg_, i64 %1
  %2 = getelementptr inbounds %"class._ZTSN2cl4sycl2idILi1EEE.cl::sycl::id", %"class._ZTSN2cl4sycl2idILi1
  %3 = load i64, i64* %2, align 8
  %add.ptr.i38 = getelementptr inbounds double, double addrspace(1)* %_arg_4, i64 %3
  %4 = getelementptr inbounds %"class._ZTSN2cl4sycl2idILi1EEE.cl::sycl::id", %"class._ZTSN2cl4sycl2idILi1
  %5 = load i64, i64* %4, align 8
  %add.ptr.i = getelementptr inbounds double, double addrspace(1)* %_arg_9, i64 %5
  %6 = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x() #2, !range !17
  %7 = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x() #2, !range !18
  %narrow.i.i.i.i.i = mul nuw nsw i32 %7, %6
  %8 = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x() #2, !range !19
  %narrow3.i.i.i.i.i = add nuw nsw i32 %narrow.i.i.i.i.i, %8
  %add.i.i.i.i.i = zext i32 %narrow3.i.i.i.i.i to i64
  %arrayidx.i.i = getelementptr inbounds double, double addrspace(1)* %add.ptr.i38, i64 %add.i.i.i.i.i
  %9 = load double, double addrspace(1)* %arrayidx.i.i, align 8, !tbaa !20
  %arrayidx.i12.i = getelementptr inbounds double, double addrspace(1)* %add.ptr.i, i64 %add.i.i.i.i.i
  %10 = load double, double addrspace(1)* %arrayidx.i12.i, align 8, !tbaa !20
  %add.i = fadd double %9, %10
```

```
.weak .entry _ZTS7vec_add(
    .param .u64 _ZTS7vec_add_param_0,
    .param .align 8 .b   /vec_add_param_1[8],
...
)
{
    .reg .b32   %r<5>;
    .reg .f64   %fd<4>;
    .reg .b64   %rd<17>;

    ld.param.u64    %rd1, [_ZTS7vec_add_param_0];
...
    shl.b64     %rd7, %rd6,
    add.s64     %rd8, %rd1, %rd7;
    shl.b64     %rd9, %rd4, 3;
    add.s64     %rd10, %rd3
    shl.b64     %rd11, %rd2,
    add.s64     %rd12, %rd5, %rd11;
    mov.u32     %r1, %ctaid.x;
    mov.u32     %r2, %ntid.x;
    mov.u32     %r3, %tid.x;
    mad.lo.s32  %r4, %r2, %r1, %r3;
    mul.wide.u32    %rd13, %r4, 8;
    add.s64     %rd14, %rd10, %rd13;
    ld.global.f64   %fd1, [%rd14];
    add.s64     %rd15, %rd12, %rd13;
    ld.global.f64   %fd2, [%rd15];
    add.rn.f64  %fd3, %fd1, %fd2;
    add.s64     %rd16, %rd8, %rd13;
```

```
@.sycl_offloading.target.0 = internal unnamed_addr constant [8 x i8] c"nvptx64\00"
@.sycl_offloading.opts.compile.0 = interna          constant [1 x i8] zeroinitializer
@.sycl_offloading.opts.link.0 = internal unnamed_addr constant [1 x i8] zeroinitializer
@.sycl_offloading.0.data.size = intern   unnamed_addr constant [8 x i8] c"(\0E\00\00\00\00\00\00", sect
@.sycl_offloading.0.data = in          ed_addr constant [3624 x i8] c"P\EDU\BA\01\00\10\00\18\0E\00\
(\02\00\00\00\00\00      \00\00\00\05\00\06\00\1E\00\00\00\00\00\00\00\00\00\00\00\11
\00\00\00\00\00\00      \00\00\00\00\00o\05\00\00\00\00\00\00\00\00\00\00\00\00\00\F1 \0A\0A\0A\
entry _ZTS7vec_add(\0A.param .u64\1A\00\11_\18\005_0,\22\00\BFalign 8 .b8*\00\0101[8]-\00\16\1F2-\00\1
\00\19\1F6-\00\19\1F7\A9\00\11\1F8O\00\16\1F9|\00\19/10.\00\1A\00\AE\01\F3\06\0A)\0A{\0A.reg .b32 %r<5>
u\17\0001, [h\00\00=0];+\00\1F2+\00\03/11,\00\00\1F3,\00\03\1F4+\00\00\1F4+\00\03\1F7+\00\00\1F5+\00\0
[illegible]
```

```
Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        0000000000400238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            0000000000400254 000254 000020 00   A  0   0  4
  [ 3] .gnu.hash         GNU_HASH        0000000000400278 000278 0001ec
  [ 4] .dynsym           DYNSYM          0000000000400468 000468 000b88
  [ 5] .dynstr           STRTAB          0000000000400ff0 000ff0         00   A  0   0  1
  [ 6] .gnu.version      VERSYM          0000000000402570 0025           02   A  4   0  2
  [ 7] .gnu.version_r    VERNEED         0000000000402668
  [ 8] .rela.dyn         RELA            000000000040227         000120
  [ 9] .rela.plt         RELA            0000000000          838 0005d0 18  AI  4  25  8
  [10] .init             PROGBITS        00000                002e08 000017 00  AX  0   0  4
  [11] .plt              PROGBITS        00              2e20 002e20 0003f0 10  AX  0   0 16
  [12] .text             PROGBITS                  0000403210 003210 00b7e2 00  AX  0   0 16
  [13] .fini             PROGBITS               000000048e9f4 00e9f4 000009 00  AX  0   0  4
  [14] .rodata           PROGBITS             000000000040ea00 00ea00 000e80 00   A  0   0 16
  [15] __CLANG_OFFLOAD_BUNDLE_SIZE__sycl-nvptx64 PROGBITS 000000000040f880 00f880 000008 08  AM  0   0
  [16] __CLANG_OFFLOAD_BUNDLE__sycl-nvptx64 PROGBITS 000000000040f890 00f890 000e28 00   A  0   0 16
  [17] .eh_frame_hdr     PROGBITS        00000000004106b8 0106b8 001344 00   A  0   0  4
  [18] .eh_frame         PROGBITS        0000000000411a00 011a00 004fe8 00   A  0   0  8
  [19] .gcc_except_table PROGBITS        00000000004169e8 0169e8 00071c 00   A  0   0  4
  [20] .init_array       INIT_ARRAY      0000000000617b98 017b98 000018 08  WA  0   0  8
```

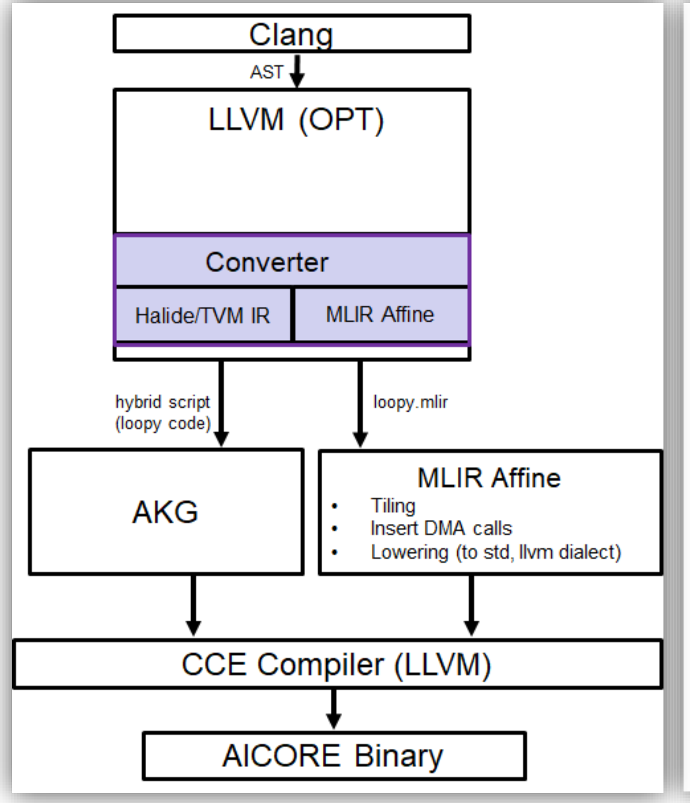
```
#include <CL/sycl/detail/defines.hpp>
#include <CL/sycl/detail/kernel_desc.hpp>

// Forward declarations of templated kernel function types:
class vec_add;

__SYCL_INLINE_NAMESPACE(cl) {
namespace sycl {
namespace detail {

// names of all kerne
static constexpr
const char* const k
  "_ZTS7vec_add"
};

// array representing          of all kernels defined in the
// corresponding sourc
static constexpr
const kernel_param_desc_  kernel_signatures[] = {
  //--- _ZTS7vec_add
  { kernel_param_kind_t::kind_accessor, 4062, 0 },
  { kernel_param_kind_t::kind_accessor, 4062, 32 },
  { kernel_param_kind_t::kind_accessor, 4062, 64 },
```

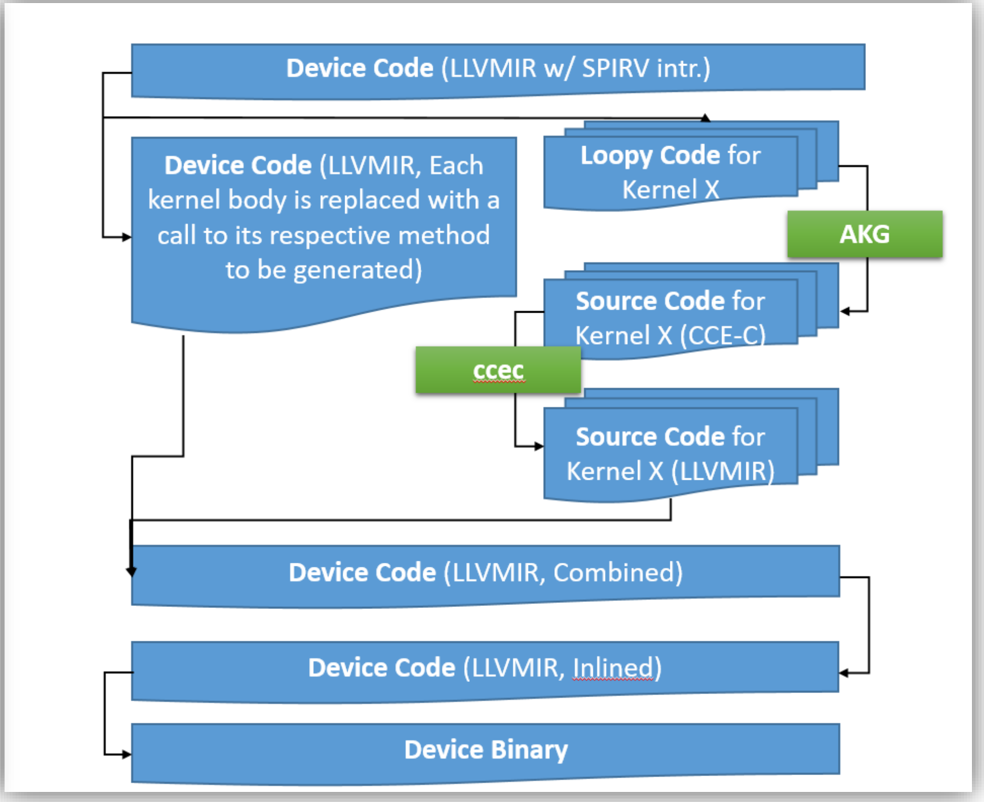


Our Contribution: CCE Compilation Toolchain

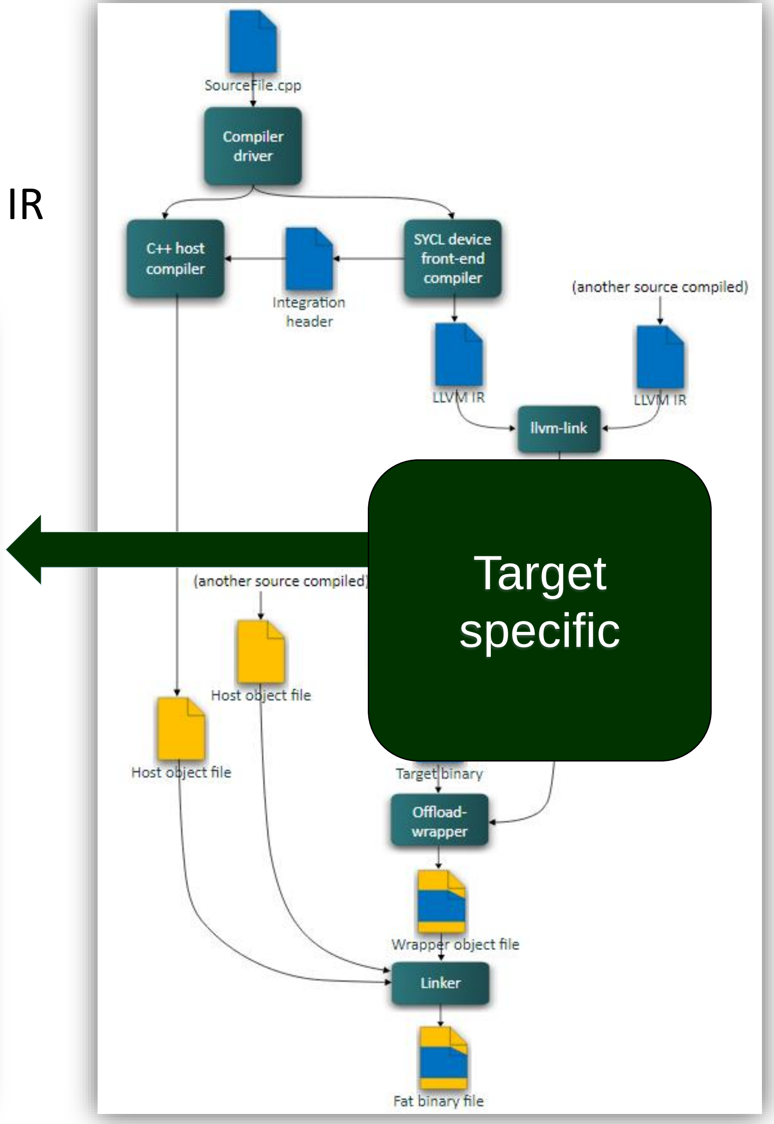
clang++ -fsycl -fsycl-targets=**hiipu64-hisilicon-cce-sycldevice** simple-hiipu.cpp -o simple-hiipu.exe

- DPC++ device kernel gets compiled into llvm IR and converted to TVM hybrid or IR for AKG. Converter is an llvm opt pass.





Device Code (LLVMIR w/ SPIRV intr.)
Device Code (LLVMIR, Each kernel body is replaced with a call to its respective method to be generated)
Loopy Code for Kernel X
AKG
Source Code for Kernel X (CCE-C)
ccec
Source Code for Kernel X (LLVMIR)
Device Code (LLVMIR, Combined)
Device Code (LLVMIR, Inlined)
Device Binary



https://github.com/intel/llvm/tree/sycl/sycl/doc

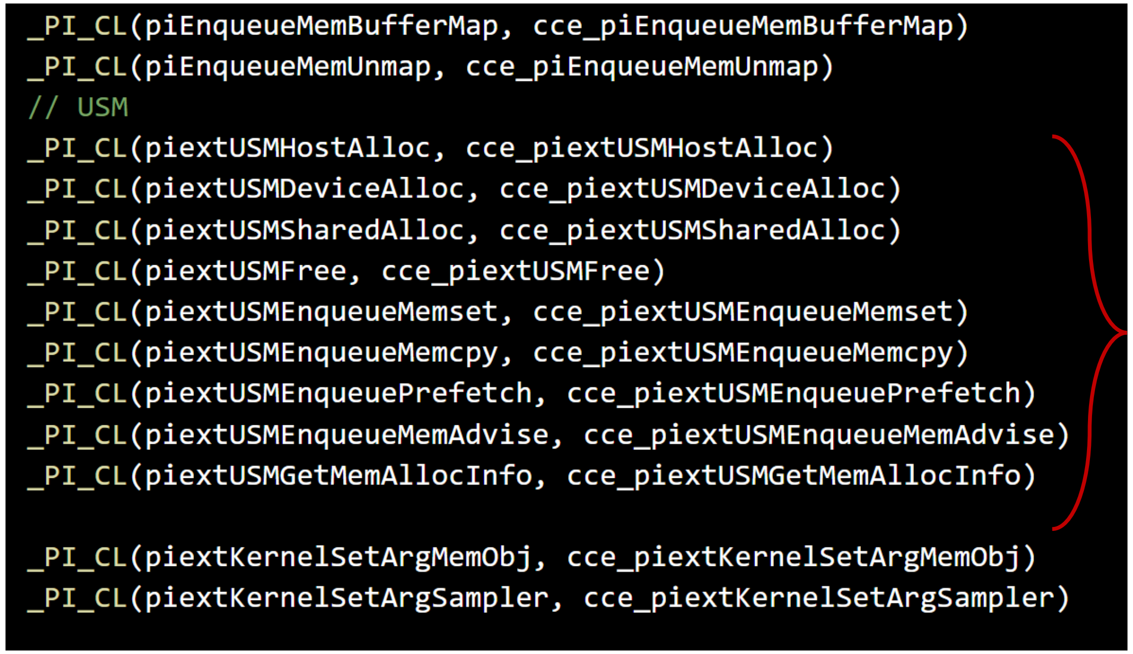
Extension to Unified Shared Memory (USM)

Implementation of Restricted USM in CCE Plugin:

1. Map **Host Allocation** to **Host Memory**
2. Map **Shared Memory Allocation** to **Managed Memory (Similar to Cuda)**
3. Map **Device Allocation** to **Device memory**

PI_CCE Plugin Api to Implement USM:

- **USM Host Allocation:** piextUSMHostAlloc
- **USM Device Allocation:** piextUSMDeviceAlloc
- **USM Shared Allocation:** piextUSMSharedAlloc
- **USM Free:** piextUSMFree
  - Frees the given USM pointer associated with the context.
- **USM Enqueue Mem. Set:** piextUSMEnqueueMemset
- **USM Enqueue Mem. Copy:** piextUSMEnqueueMemcpy
- **USM Enqueue Mem. Prefetch:** piextUSMEnqueuePrefetch
- **USM Enqueue Mem Advice:** piextUSMEnqueueMemAdvise
  - API to govern behavior of automatic migration mechanisms
- **USM Get Mem. Allocation Info.:** piextUSMGetMemAllocInfo

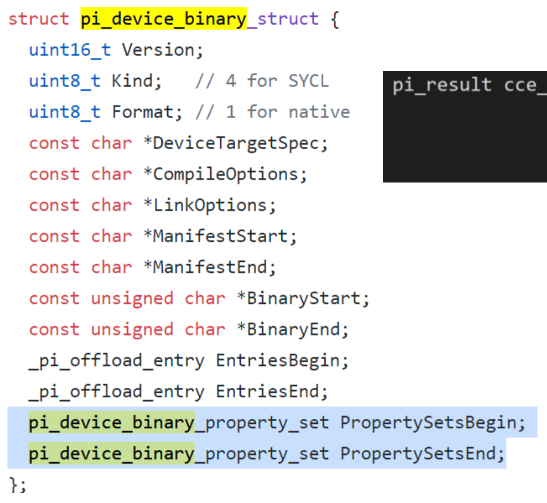


```
_PI_CL(piEnqueueMemBufferMap, cce_piEnqueueMemBufferMap)
_PI_CL(piEnqueueMemUnmap, cce_piEnqueueMemUnmap)
// USM
_PI_CL(piextUSMHostAlloc, cce_piextUSMHostAlloc)
_PI_CL(piextUSMDeviceAlloc, cce_piextUSMDeviceAlloc)
_PI_CL(piextUSMSharedAlloc, cce_piextUSMSharedAlloc)
_PI_CL(piextUSMFree, cce_piextUSMFree)
_PI_CL(piextUSMEnqueueMemset, cce_piextUSMEnqueueMemset)
_PI_CL(piextUSMEnqueueMemcpy, cce_piextUSMEnqueueMemcpy)
_PI_CL(piextUSMEnqueuePrefetch, cce_piextUSMEnqueuePrefetch)
_PI_CL(piextUSMEnqueueMemAdvise, cce_piextUSMEnqueueMemAdvise)
_PI_CL(piextUSMGetMemAllocInfo, cce_piextUSMGetMemAllocInfo)

_PI_CL(piextKernelSetArgMemObj, cce_piextKernelSetArgMemObj)
_PI_CL(piextKernelSetArgSampler, cce_piextKernelSetArgSampler)
```

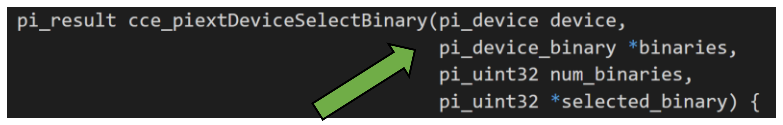
Support for Parallel-for

we have broken the boundary between the workgroups and workitems:

- AKG (an external tool) is capable of performing **automatic parallelization, vectorization**
- We created our own parallel_for C++ abstraction where we require **static number** of workgroups and workitems
- Converter would artificially create outer loops with an extent that equals to the total number of tasks as well as a loop hint to AKG to select the best parallelization factor

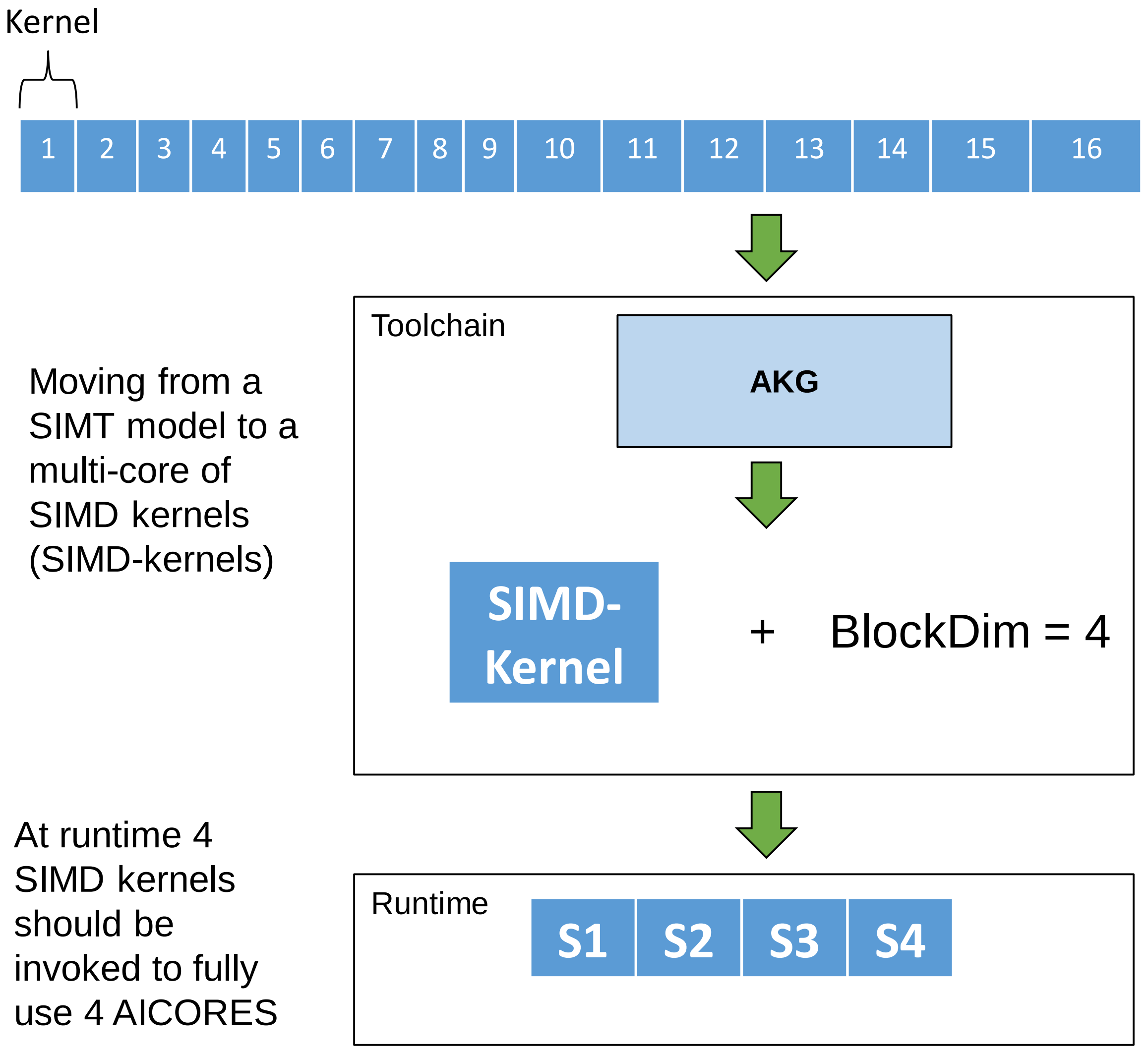


```
struct pi_device_binary_struct {
  uint16_t Version;
  uint8_t Kind;    // 4 for SYCL
  uint8_t Format; // 1 for native
  const char *DeviceTargetSpec;
  const char *CompileOptions;
  const char *LinkOptions;
  const char *ManifestStart;
  const char *ManifestEnd;
  const unsigned char *BinaryStart;
  const unsigned char *BinaryEnd;
  _pi_offload_entry EntriesBegin;
  _pi_offload_entry EntriesEnd;
  pi_device_binary_property_set PropertySetsBegin;
  pi_device_binary_property_set PropertySetsEnd;
};
```



```
pi_result cce_piextDeviceSelectBinary(pi_device device,
                                      pi_device_binary *binaries,
                                      pi_uint32 num_binaries,
                                      pi_uint32 *selected_binary) {
```

SIMT to SIMD



- Moving from a SIMT model to a multi-core of SIMD kernels (SIMD-kernels)
- At runtime 4 SIMD kernels should be invoked to fully use 4 AICORES

Supported Examples - Matmult

Size:

- **M = 384**
- **K = 256**
- **N = 512**

1. Compilation uses C100 sub-architecture of HiIPU target
2. Device selection using hiipu_selector() class
3. Current implementation uses both Vector and Cube units
4. Correctness of output checked on host side

```
$X3CPP_BUILD/bin/clang++ \
    -fsycl \
    -fsycl-targets=hiipu64_c100-hisilicon-cce-sycldevice \
    -DSYCL_DISABLE_PARALLEL_FOR_RANGE_ROUNDING \
    $PROGRAM.cpp \
    -B$AKG_HOME \
    --cce-path=$CCE_HOME
```

```
int main() {
  const size_t M = 6 * 16 * 4;
  const size_t K = 4 * 16 * 4;
  const size_t N = 8 * 16 * 4;

  std::vector<float> A(M * K);
  std::vector<float> B(K * N);
  std::vector<float> C(M * N);

  for (int i = 0; i < M * K; i++) A[i] = 2.0;
  for (int i = 0; i < K * N; i++) B[i] = 2.0;

  queue deviceQueue(hiipu_selector{});
  {

    ndbuffer<float, M, K> A_buf(A.data());
    ndbuffer<float, K, N> B_buf(B.data());
    ndbuffer<float, M, N> OUT_buf(C.data());

    deviceQueue.submit([&](handler &cgh) {
      auto A_acc = A_buf.get_access<sycl_read>(cgh);
      auto B_acc = B_buf.get_access<sycl_read>(cgh);
      auto OUT_acc = OUT_buf.get_access<sycl_write>(cgh);

      auto kern = [=](id<2> ids) {
        size_t m = ids[0];
        size_t n = ids[1];
        float sum = 0.0f;
        for (unsigned int k = 0; k < K; k++) {
          sum += A_acc[m][k] * B_acc[k][n];
        }
        OUT_acc[m][n] = sum;
      };
      static_parallel_for<class matmult, M, N>(cgh, kern);
    });
```

Future Work

- Alternative codegen support in parallel with AKG
  - MLIR path of code generation is being developed and will require another path to be added into our custom toolchain
- Polish runtime and toolchain code base for production
  - Will be looking forward to upstream our work

Questions?