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

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

Requires SYCL Runtime & Toolchain to be updated.

Our contribution

Source Code

Compiler

Architecture

- OpenCL
- CUDA
- Intel Level-Zero Interface
- Target System Software
- CPU (Scalar / Vector)
- GPU (Vector)
- FPGA (Spatial)
- DSP (Matrix)
- Ascend AI Chip

Data Parallel C++

Extensions

```
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;
    });
```

CCE
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

```cpp
cl::sycl::queue Queue(cl::sycl::hiipu_selector{});
```
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

---

**Sycl objects calls into generic plugin APIs:** pXYZ(). APIs are targeted independently.

**Current state of the art for Sycl runtime**

- sycl::queue
  - sycl::queue_submit
    - pXNYZ() APIs are targeted independently

**CUDA Specific plugin implementation**

- cuda_pqueueKernelLaunch( )
- cudaLaunchKernel( )
- cuda_pbufferCreate( )
- cudaLaunchMem( )
- cuda_pKernelCreate( )
- cudeModuleGetFunction( )

**Sycl-specific extension to Sycl runtime**

- CCE Specific plugin implementation
  - cce_pqueueKernelLaunch( )
  - cceLaunchKernel( )
  - cce_pbufferCreate( )
  - cceModuleGetFunction( )
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
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.

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`
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

- 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 - Matmul

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
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?