Reaching even richer C++ in OpenCL kernels with use of libclcxx

IWOCL’22

Anastasia Stulova, Ishfaq Wardag
10-12 May 2022
Motivation

- The biggest limitation of C++ for OpenCL kernel language is the absence of the libraries support
  - This prevents accessing many C++ features

- Previous work has focused on reusing libcxx project with successful `<type_traits>` library ported to OpenCL
  - Its use is not straightforward from libcxx project as it is missing OpenCL target
  - It also does not contain OpenCL specific traits
About libclcxx

- This is a new public project hosted on GitHub
  - [https://github.com/KhronosGroup/libclcxx](https://github.com/KhronosGroup/libclcxx)
- The main goal is to gather and expose OpenCL-specific C++ libraries
- It integrates libcxx and is made to work with Clang frontend
- Type traits is the initial library feature
- Testing of new functionality is integrated and enabled in CI
- Doxygen documentation is available for the users, similar to the C++ reference pages
Type traits are vital to exploit the power of metaprogramming in C++ fully

This new type trait library imports all standard C++ type traits from libcxx

Some specializations of C++ traits have been added for OpenCL specific type system

Some custom type traits for OpenCL have been added:
- vectors, address spaces, images, half precision float

Thanks to reuse of existing type traits implementation from C++, these new traits could be added with minimal development effort
// import all type traits from C++
#pragma OPENCL EXTENSION __cl_clang_function_pointers : enable
#pragma OPENCL EXTENSION __cl_clang_variadic_functions : enable
#include <type_traits>
#pragma OPENCL EXTENSION __cl_clang_function_pointers : disable
#pragma OPENCL EXTENSION __cl_clang_variadic_functions : disable
namespace std {
/**
 * @see <a href="https://en.cppreference.com/w/cpp/types/is_signed">is_signed</a>
 */
template <> struct is_signed<int2> : public true_type {};
/**
 * @see <a href="https://en.cppreference.com/w/cpp/types/is_unsigned">is_unsigned</a>
 */
template <> struct is_unsigned<int2> : public false_type {};

/**
 * Provides the constant member variable value equal to the
 * the number of elements of a vector T if T is a vector and
 * equal to 0 otherwise.
 * @tparam T - a vector type.
 */

template <typename T> struct vector_size : public
    integral_constant<size_t, 0> {};

template <> struct vector_size<int2> : public
    integral_constant<size_t, 2> {};

// traits for vector types from int3 to int8 are omitted

template <> struct vector_size<int16> : public
    integral_constant<size_t, 16> {};
Demonstration – Address space traits

```cpp
#include <opencl_type_traits>

// Standard C++ trait

template<typename PTR_TYPE>
auto foo(PTR_TYPE ptr) {

  using pointee_type = std::remove_pointer<PTR_TYPE>::type;

  //pointee_type tmp;  /*error: pointee_type qualified with non-private address space*/

  std::remove_address_space<pointee_type>::type tmp;

  tmp = *ptr + 1;

  return tmp;
}

// foo() works generically for different pointer types and address spaces

// OpenCL-specific trait

void bar(global int * glob_int_ptr, local float * loc_float_ptr) {
  *glob_int_ptr = foo(glob_int_ptr);
  *loc_float_ptr = foo(loc_float_ptr);
}
```
Demonstration – Vector type traits

\[ r = \sum_{i=0}^{N} x_i \]

```c
__kernel void partial_reduction(__global elem_type *input,
                                __global elem_type *output, uint n) {
  auto offset = get_global_id(0);
  elem_type sum = 0;
  // every work item needs to sum up N elements
  for (auto i = offset; i < offset + n; i += vector_size ? vector_size : 1)
    sum += add_elems(&input[i]);
  output[offset] = sum;
}
```
Demonstration – Vector type traits

\[ r = \sum_{i=0}^{N} x_i \]

```c
__kernel void partial_reduction(__global elem_type *input,
                                __global elem_type *output, uint n) {
    auto offset = get_global_id(0);
    elem_type sum = 0;
    // every work item needs to sum up N elements
    for (auto i = offset; i < offset + n; i += vector_size ? vector_size : 1)
        sum += add_elems(&input[i]);
    output[offset] = sum;
}
```

```sh
c clang -I<path to libclcxx>/include -DTYPE=int reduction.clcpp
c clang -I<path to libclcxx>/include -DTYPE=char4 reduction.clcpp
```
Demonstration – Vector type traits (cont’d)

```cpp
#include <opencl_type_traits>

constexpr auto vector_size = std::vector_size<TYPE>::value;

using elem_type = std::scalar_type<TYPE>::type; // scalar_type obtains vec elem type

auto add_elems(elem_type *ptr) {
    if constexpr (vector_size == 2) {
        auto vec = vload2(0, ptr);
        return vec.s0 + vec.s1;
    } else if constexpr (vector_size == 3) {
        // analogous handling of vectors with sizes from 3 to 8 are omitted
    } else if constexpr (vector_size == 16) {
        auto vec = vload16(0, ptr);
        return vec.s0 + /* omitted other components from s1 to s14 */ vec.s15;
    }
    return *ptr; // not a vector!
}
```
Conclusions and feedback

- A new space for C++ libraries for OpenCL has been created
- First library (OpenCL type traits) is derived from the C++ implementation allowing reducing the development cycles and providing best synergy
- Developers are invited for experimenting and contributing
- Any feedback is greatly appreciated and can be submitted using libclcxx GitHub project page
Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה