triSYCL
Open Source C++17 & OpenMP-based OpenCL SYCL prototype

Ronan Keryell
Khronos OpenCL SYCL committee

05/12/2015

IWOCL 2015 SYCL Tutorial
OpenCL SYCL committee work...

- Weekly telephone meeting
- Define new ways for modern heterogeneous computing with C++
  - Single source host + kernel
  - Replace specific syntax by pure C++ abstractions
- Write SYCL specifications
- Write SYCL conformance test
- Communication & evangelism
SYCL relies on advanced C++

- Latest C++11, C++14...
- Metaprogramming
- Implicit type conversion
- ...

Difficult to know what is feasible or even correct...

- Need a prototype to experiment with the concepts
- Double-check the specification
- Test the examples

Same issue with C++ standard and GCC or Clang/LLVM
Solving the meta-problem

- SYCL specification
  - Includes header files descriptions
  - Beautified .hpp
  - Tables describing the semantics of classes and methods
- Generate Doxygen-like web pages
  ~ Generate parts of specification and documentation from a reference implementation
• Started in April 2014 as a side project
• Open Source for community purpose & dissemination
• Pure C++ implementation
  ▶ DSEL (Domain Specific Embedded Language)
  ▶ Rely on STL & Boost for zen style
  ▶ Use OpenMP 3.1 to leverage CPU parallelism
  ▶ No compiler cannot generate kernels on GPU yet
• Use Doxygen to generate
  ▶ Documentation of triSYCL implementation itself with implementation details
  ▶ SYCL API by hiding implementation details with macros & Doxygen configuration
• Python script to generate parts of LaTeX specification from Doxygen LaTeX output
Automatic generation of SYCL specification is a failure...

- Literate programming was OK with low level languages such as TeX/Web/Tangle...
- But cumbersome with modern C++ with STL+Boost library
  - STL & Boost allow to implement many SYCL methods in a terse way
  - Doxygen specification requires explicit writing of all the methods...
  - ... which do exist only implicitly in the STL & Boost implementation
- Literate programming with high level C++ \( \sim \) lot of redundancy
- Require also to have implementation (or at least declaration headers) to exist before specification

\( \sim \) Dropped the idea of generating specification from triSYCL
Outline

1. triSYCL
2. How it is implemented...
3. Future
4. Conclusion
Using triSYCL

- Get information from https://github.com/amd/triSYCL
- Developed and tested on Linux/Debian with GCC 4.9/5.0, Clang 3.6/3.7 and Boost
  - sudo apt-get install g++4.9 libboost-dev
- Download with
  - git clone git@github.com:amd/triSYCL.git (ssh access)
  - git clone https://github.com/amd/triSYCL.git
    - Branch master: the final standard
    - Branch SYCL-1.2-provisional-2: previous public version, from SC14
- Add include directory to compiler include search path
- Add -std=c++1y -fopenmp when compiling
- Look at tests directory for examples and Makefile
What is implemented

- All the small vectors `range<>`, `id<>`, `nd_range<>`, `item<>`, `nd_item<>`, `group<>`
- Parts of `buffer<>` and `accessor<>`
- Concepts of address spaces and `vec<>`
- Most of parallel constructs are implemented (`parallel_for <>...`)
  - Use OpenMP 3.1 for multicore CPU execution
- Most of command group `handler` is implemented
- No OpenCL feature is implemented
- No host implementation of OpenCL is implemented
  - No `image<>`
  - No OpenCL-like `kernel` types & functions
Outline

1. triSYCL
2. How it is implemented...
3. Future
4. Conclusion
Small vectors

- Used for `range<>`, `id<>`, `nd_range<>`, `item<>`, `nd_item<>`, `group<>`
- Require some arithmetic operations element-wise
- Use `std::array<>` for storage and basic behaviour
- Use `Boost.Operator` to add all the lacking operations
- `CL/sycl/detail/small_array.hpp`

```cpp
//template <typename BasicType, typename FinalType, std::size_t Dims>
struct small_array : std::array<BasicType, Dims>,
  // To have all the usual arithmetic operations on this type
  boost::euclidean_ring_operators<FinalType>,
  // Bitwise operations
  boost::bitwise<FinalType>,
  // Shift operations
  boost::shiftable<FinalType>,
  // Already provided by array<> lexicographically:
```
Small vectors

// boost::equality_comparable<FinalType>,
// boost::less_than_comparable<FinalType>,
// Add a display() method
detail::display_vector<FinalType> {
/// Keep other constructors
using std::array<BasicType, Dims>::array;

small_array() = default;
/** Helper macro to declare a vector operation with the given side-effect operator */
#define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op) \
  FinalType operator op(const FinalType& rhs) { \ 
    for (std::size_t i = 0; i != Dims; ++i) \ 
      (*this)[i] op rhs[i]; \ 
    return *this; \ 
  }
/// Add + like operations on the id<> and others
Small vectors

```c++
TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)
/// Add * like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)
/// Add << like operations on the id<> and others
TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)
[...]
How it is implemented...

Other Boost usage

- Boost.Log for debug messages
- Boost.MultiArray (generic N-dimensional array concept) used to implement `buffer<>` and `accessor<>`
  - Provide dynamic allocation as C99 Variable Length Array (VLA) style
  - Fortran-style arrays with triplet notation, with `[[]]` syntax
    - The viral library to attract to C++ Fortran and C99 programmers 😊
template <std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct parallel_OpenMP_for_iterate {
    parallel_OpenMP_for_iterate (Range r, ParallelForFunctor &f) {
        // Create the OpenMP threads before the for loop to avoid creating an
        // index in each iteration
        #pragma omp parallel
        {
            // Allocate an OpenMP thread-local index
            Id index;
            // Make a simple loop end condition for OpenMP
            boost::multi_array_types::index _sycl_end = r[Range::dimensionality - level];
            /* Distribute the iterations on the OpenMP threads. Some OpenMP
             * "collapse" could be useful for small iteration space, but it
             * would need some template specialization to have real contiguous
             * loop nests */
            #pragma omp for
            for (boost::multi_array_types::index _sycl_index = 0;
                 _sycl_index < _sycl_end;
                 _sycl_index++) {
                // Set the current value of the index for this dimension
                index[Range::dimensionality - level] = _sycl_index;
                // Iterate further on lower dimensions
OpenMP

```cpp
parallel_for_iterate<level - 1,
    Range,
    ParallelForFunctor,
    Id> { r, f, index };```

How it is implemented...
Some other C++11/C++14 cool stuff

- Tuple/array duality + new `std::make_index_sequence<>` to have meta-programming for-loops $\leadsto$ constructors of `cl::sycl::vec<>`
OpenCL support

- Develop the OpenCL layer
- Rely on other high-level OpenCL frameworks (Boost.Compute...) NIH
- Already refactored the code from LLVM style to Boost style
- Should be able to have OpenCL kernels through the kernel interface with OpenCL kernels as a string (non single source kernel)
GPU accelerated kernel code

- Use OpenMP 4, OpenACC, C++AMP... in current implementation
- But no OpenCL interoperability available if not provided by back-end runtime
OpenCL single source kernel and interoperability support

- **Alternative**
  - Write new Clang/LLVM phase to outline kernel code and develop OpenCL runtime back-end
  - Recycle open source C++ framework for accelerators: OpenMP 4 or C++AMP
    - Modify runtime back-end to have OpenCL interoperability
- **OpenMP 4 support in Clang/LLVM** is backed by OpenMP community (Intel, IBM, AMD...) and up-streaming already started
  - Likely the most interesting approach
  - Modify libomp5
- **Reuse LLVM MC back-ends**
  - SPIR/SPIR-V for portability, open source from Khronos/Intel/AMD
  - AMD GCN RadeonSI from Mesa/GalliumCompute/Clover/Clang/LLVM
    - Would allow `asm("...")` in SYCL code!
    - Optimized libraries directly in SYCL (linear algebra, FFT, CODEC, deep learning...)

---

**triSYCL** — Open Source C++17 & OpenMP-based OpenCL SYCL prototype

IWOCL 2015
Outline

1. triSYCL
2. How it is implemented...
3. Future
4. Conclusion
Conclusion

- SYCL ≡ best of pure modern C++ + OpenCL interoperability + task graph model
- Well accepted standard \( \sim \) different implementations
- An open source project makes dissemination and experiment easier
- triSYCL leverages many high-level tools
  - Post-modern C++, Boost, OpenMP, Clang/LLVM,...
- Open source implementation decreases entry cost...
  - ∃ Free tool to try
  - Can be used by vendors to develop their own tools
- ... and decreases exit cost too
  - Even if a vendor disappears, there is still the open source tool
- Get involved in the triSYCL development
  - Still a lot to do!
  - Also a way to influence OpenCL, SYCL and C++ standards
Table of content

1. triSYCL
   Outline
   Using triSYCL
   What is implemented

2. How it is implemented...
   Outline
   Small vectors

3. Future
   Outline
   OpenCL support
   GPU accelerated kernel code
   OpenCL single source kernel and interoperability support

4. Conclusion
   Outline
   Conclusion
   You are here!