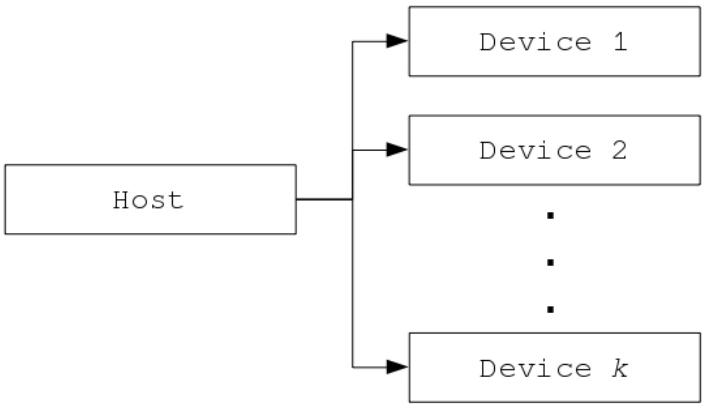
An Introduction to OpenCL C++

AJ Guillon, YetiWare Inc.

1 About OpenCL

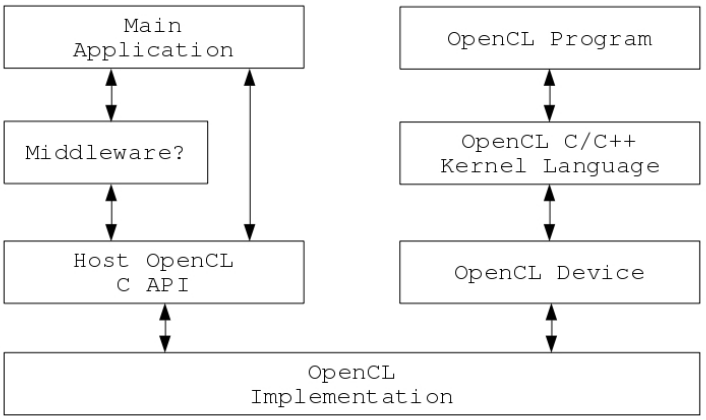
Today servers, desktops, mobile devices, and embedded systems contain many processors in addition to the CPU that runs programs. These extra processors are generally called *accelerators* and could be a GPU, FPGA, Xeon Phi, or other programmable device. There are many types of accelerators available, from many vendors, for many different environments. Khronos developed the OpenCL™ standard so that an application can offload parallel computation to accelerators in a common way, regardless of their underlying architecture or programming model. OpenCL-enabled software can obtain peak performance by using every OpenCL-enabled parallel processor and accelerator on the system together at the same time.

OpenCL provides a unique abstraction of the computing system, as depicted below.



The *host* is an abstraction that defines where an application is executed, and for most developers this is going to be the operating system and CPU. The host can access *devices* that support OpenCL, such as accelerators or the host CPU itself. OpenCL provides a common hardware abstraction for devices that includes a hardware model, memory model, and parallel execution model. Devices cannot run traditional C or C++ programs, instead OpenCL provides *kernel languages* that are used to write programs that can run on OpenCL devices.

There are several logical components of OpenCL from the point of view of the software developer, as depicted below.



The application runs on the host, and can use a standard C API to manage OpenCL devices directly or rely on middleware to do the management. Applications or middleware are responsible for discovering devices, managing device memory, and compiling and running device programs. OpenCL standardizes how the host calls device programs, and how memory is transferred between host and device.

The OpenCL kernel languages are based on traditional languages with the minimal possible changes to enable parallel processing for maximum performance and portability across devices. The kernel languages are defined through restrictions and enhancements to standard languages so that the kernel compilers can generate optimized code. Khronos designed a modified version of C for devices called OpenCL C, and now Khronos has modified C++ to create OpenCL C++.

The OpenCL standard has been developed iteratively through several versions, each adding new features culminating in OpenCL 2.1. This latest provisional release introduces a new OpenCL C++ kernel language along with SPIR-V™, a common intermediate representation (IR) for compute and graphics that is shared with the Vulkan graphics API. Developers can now write device programs using OpenCL C++, which is based on C++14, and compile these programs to a SPIR-V representation that can then be loaded for execution by the device. The motivations behind these changes will be described in this white paper.

2 Why OpenCL C++ ?

Previous versions of OpenCL have featured OpenCL C, based upon the C programming language. The provisional OpenCL 2.1 specification is the first OpenCL version to feature a kernel language based upon C++. The relative advantages and disadvantages of C and C++ are already well known, and OpenCL C++ inherits many of them. The OpenCL working group has responded to developers' requests to be able to write high-level abstractions enabled by C++, while maintaining some compatibility with OpenCL C, by providing the OpenCL C++ kernel language.

OpenCL C++ provides library developers with powerful facilities for abstraction so that high-level libraries and containers can be implemented on the device itself. An OpenCL C++ library might provide a very simple interface to developers that internally relies upon sophisticated template metaprogramming techniques to provide peak performance on the target hardware. The OpenCL working group will ensure that OpenCL C++ provides the right language features to enable innovative high-level libraries and containers.

3 What About OpenCL C?

It is not possible to provide OpenCL C as a subset of OpenCL C++ because some features and types cannot be supported in OpenCL C++. Khronos realizes that not every programmer will wish to migrate from OpenCL C to OpenCL C++ and so OpenCL C is still fully supported in OpenCL 2.1. Developers can opt to write OpenCL C-style code using OpenCL C++ if they choose.

4 How Are OpenCL C++ and SPIR-V Related?

Previous OpenCL versions required that the OpenCL implementation have a built-in OpenCL C compiler[1]. Developers called `clCreateProgramWithSource` and passed in an OpenCL C program as a string. A compiler within the implementation would then build and return a handle to the compiled program. This approach requires that every OpenCL implementation contain an OpenCL C compiler, and any bug in the OpenCL C compiler would affect the stability of the entire implementation.

This design had several issues.

[1] With the exception of the embedded profile

The first issue was that applications were forced to ship OpenCL C source code or platform specific binaries. Some application developers would prefer a program representation that requires effort and intent to disassemble. Although OpenCL implementations may support loading a program from a binary representation, the binary might not be obfuscated, and it would be device specific. This reduces application portability.

The second issue was that OpenCL implementations and drivers became rather complicated because of the imposed requirement for them to contain an OpenCL C compiler. This made it hard to test OpenCL implementations and difficult to ensure that implementations shipped in devices were correct. It can be extremely difficult, or impossible, to update OpenCL implementations in some hardware environments. Developers might be forced to restructure their OpenCL C code to work around issues in specific OpenCL versions on specific devices.

OpenCL 2.1 addresses these issues by changing how OpenCL programs are compiled and by introducing SPIR-V.

SPIR-V is a low-level representation of an OpenCL program that is designed to be easily loaded by an OpenCL implementation. It has been designed to eliminate both of the design issues mentioned. SPIR-V is not source code so a reverse engineer would have to take several intentional steps to obtain proprietary software information, and would not obtain the original source code. SPIR-V is also designed to split a compiler into two parts in such a way that OpenCL implementations are easier to test. This design enhances implementation stability.

An OpenCL C++ compiler is not a required part of an OpenCL 2.1 implementation. This means that an OpenCL implementation does not have to contain a compiler to be conformant. Instead, developers will call `clCreateProgramWithIR` and pass in a SPIR-V program that the implementation will: load, transform to device specific code, optimize, and return a handle to the program. Developers will use an external OpenCL C++ compiler that will compile OpenCL C++ to SPIR-V.

So where do you get an OpenCL C++ compiler? There are a few possibilities:

- Vendors may ship an OpenCL C++ compiler that generates SPIR-V as part of their SDK
- Khronos is considering providing an open source OpenCL C++ compiler available under a permissive license
- Independent tools vendors might develop and provide their own optimizing OpenCL C++ compilers for use across multiple OpenCL runtime implementations

SPIR-V and the compilation changes in OpenCL 2.1 have opened up many possibilities for the software ecosystem. If an OpenCL C++ compiler is available as a library with an API, then it could be linked into applications. This means that developers could ship OpenCL C++ source code and compile it in the same way that they did with previous OpenCL versions by using the compiler library.

There are additional reasons to use a separate OpenCL C++ compiler that targets SPIR-V, instead of requiring that implementations contain an OpenCL C++ compiler.

One reason is to ensure that devices can load programs relatively quickly[2]. OpenCL C++ programs that leverage template metaprogramming can take a significant amount of time to compile, and loading SPIR-V will likely be faster than instantiating complex templates at runtime.

[2] With the potential exception of FPGAs

Another reason to prefer a separate OpenCL C++ compiler is to insulate the application from any potential OpenCL C++ compiler issues. Developers can do unit testing on their SPIR-V code with the assurance that their distributed programs will be loaded in a simple manner with some optimization passes. Alternatively, developers might have to work around compiler bugs and test specific OpenCL C++ compiler versions.

Traditionally, C++ header library developers had to deal with many compilers that each supported a subset of the complete C++ language. With OpenCL C++, Khronos tests compilers for conformance to ensure they properly adhere to the specification. This means that any conformant OpenCL C++ compiler will support a common set of language features as validated by conformance testing. It is still possible for a compiler to contain bugs because conformance testing cannot guarantee correctness, and this is why SPIR-V provides an extra layer of re-assurance for the quality of shipped code.

5 What Does OpenCL C++ Provide?

OpenCL C++ is based on C++14 and supports many useful features, including:

- Classes and basic object-oriented programming[3]
- Templates, type traits, and template metaprogramming
- Operator overloading
- Function overloading
- Lambdas
- Namespaces

OpenCL C++ combines the expressive power of C++ with the parallel programming model and hardware efficiency of OpenCL. It provides a basic library in the form of classes for atomic operations, pipes, images, and other OpenCL features. The OpenCL working group is currently focused on C++ language features to enable high-level libraries to be built over time.

OpenCL C++ does not yet provide an extensive template library. The C++14 template library provides containers, iostreams, signals, and other features that rely upon capabilities that are not part of OpenCL at this time. Instead, OpenCL C++ provides the language facilities needed to build innovative parallel and heterogeneous reusable libraries that are unique to OpenCL.

OpenCL C++ does not support many features of C++14, including:

- Exceptions
- Memory allocation
- Recursion
- Function pointers
- `goto`

[3] virtual member functions are not supported, therefore runtime polymorphism is not supported

- Virtual member functions and runtime polymorphism
- C++ threading and concurrency features[4]

The C++ language has been restricted to ensure that OpenCL C++ can be aggressively optimized by the compiler while adhering to the underlying OpenCL memory and execution models. OpenCL C++ inherits many of its restrictions from OpenCL C because the underlying hardware requirements have not changed with OpenCL 2.1.

Although the restrictions imposed by OpenCL on the C++ language may seem limiting, a significant degree of abstraction is possible[5]. OpenCL C++ is the result of extensive discussions amongst software professionals, hardware architects, and compiler teams within Khronos. It is a C++ language that has been distilled to a core set of features that should work well on a variety of processors. Software developers can take this core feature set and build high-level abstractions with confidence.

6 A Taste of OpenCL C++

OpenCL C++ provides many opportunities for developers to create innovative high-level libraries and solutions that would have been challenging with OpenCL C. For example, it would be difficult to provide elegant new types using OpenCL C due to a lack of operator overloading and other C++ features. For this first taste of OpenCL C++, consider a matrix type with dimensions known at compile-time. The skeletal class definition is straight forward:

```
template <typename T, size_t Rows, size_t Columns>
class matrix {
public:
   matrix() { }

   /* Count rows starting from one not zero */
   T& operator()(size_t row, size_t col) { return _data[row-1][col-1]; }

   constexpr size_t num_rows() { return Rows; }

   constexpr size_t num_columns() { return Columns; }

private:
   T _data[Rows][Columns];
};
```

Using operator overloading it is easy to define matrix addition:

```
template <typename T, size_t Rows, size_t Columns>
matrix<T, Rows, Columns> operator+(const matrix<T, Rows, Columns>& x, const  ↩
    matrix<T, Rows, Columns>& y) {

   matrix<T, Rows, Columns> tmp;

   for ( size_t row = 0; row < Rows; ++row ) {
      for ( size_t column = 0; column < Columns; ++column ) {
         tmp(row, column) = x(row, column) + y(row, column);
      }
```

[4] OpenCL concurrency features are supported instead

[5] OpenCL C++ is a provisional standard and some of these restrictions may be removed

```
    }

    return tmp;
}
```

Finally, an OpenCL kernel can be written that will add two collections of matrices together:

```
matrix<float, 2, 2> float4_to_matrix(float4* in) {

    matrix<float, 2, 2> m;

    float4 tmp = *in;

    m(1,1) = tmp.s0;
    m(1,2) = tmp.s1;
    m(2,1) = tmp.s2;
    m(2,2) = tmp.s3;

    return m;
}

float4 matrix_to_float4(const matrix<float, 2, 2>& m)

    float4 vec;

    vec.s0 = m(1,1) ;
    vec.s1 = m(1,2) ;
    vec.s2 = m(2,1) ;
    vec.s3 = m(2,2) ;

    return vec;
}




/*
 * Matrices are read and written as 4-element float values.
 */
kernel void add_matrices(float4* in1, float4* in2, float4* result) {

    size_t idx = get_global_id(0);

    matrix<float, 2, 2> m1 = float4_to_matrix( in1[idx] );
    matrix<float, 2, 2> m2 = float4_to_matrix( in2[idx] );

    result[idx] = matrix_to_float4(m1 + m2);


}
```

Note that this example is not actually different from traditional C++ code except for the kernel itself.

7 What Does a *Provisional* Release Mean?

OpenCL 2.1 and the OpenCL C++ kernel language specifications have been provisionally released, enabling industry feedback and comments on the standard and to report potential errors or omissions. Based upon the feedback received, the final release could differ from the provisional standard. If you see something you like, or do not like, you are encouraged to tell the working group so that your feedback can be considered. You have a unique opportunity to tell the working group how you would like to see the OpenCL standard evolve. The OpenCL working group is committed to the new C++-based kernel language and welcomes appropriate feedback from the community.

8 How Can I Give Feedback?

The industry can provide feedback to Khronos related to the provisional OpenCL 2.1 and OpenCL C++ kernel language by posting a message to the official feedback forum thread[6]. This provides a public record of all public feedback received for consideration. This forum thread is closely monitored by the working group and all feedback received will be considered. Unfortunately, we are quite busy working on OpenCL behind the scenes so we might not be able to thank you personally. Rest assured, your feedback does contribute to the future direction of OpenCL, and it will be discussed within the working group.

If you find that posting to a forum limits your ability to express yourself, or you have a lot of feedback to provide, you are welcome to post a feedback thread with an attached PDF.

Please note that we cannot accept detailed designs or other contributions from non-members due to intellectual property issues that might arise. We recommend that you suggest big picture ideas or requirements, or perhaps show a problem you want to solve, and we will discuss its impact on the specification within the working group.

You are welcome to show counter-examples that break the specification. If you provide a fragment of code that is technically permitted by the specification but is somehow undesirable or contradictory, the working group can internally discuss its impact and improve the specification.

The OpenCL working group participates in BOFs and panel discussions at various events throughout the year such as GDC, Supercomputing, IWOCL, and more. We invite you to join the discussion at these events. See http://www.khronos.org/news/events for a list of upcoming events.

Finally, if your company's business interests would be served by having a voice at the OpenCL working group – any company is always welcome to join.

[6] https://www.khronos.org/message_boards/showthread.php/9651-Official-OpenCL-2-1-Feedback-thread

9 About

AJ Guillon is a Khronos member and contributed to OpenCL 2.1 and the OpenCL C++ kernel language. He has worked extensively with OpenCL since December 2008, when the first OpenCL 1.0 specification was released. AJ is the founder of YetiWare Inc, a software company developing innovative parallel platforms built on top of OpenCL. He is available to share his experience and knowledge of the OpenCL standard through consulting and training, or public speaking events. AJ is a masters swimmer, water polo player, and enjoys rock climbing when time permits. He lives in Toronto, Ontario, Canada and earned an Honors Bachelor of Science from the University of Toronto, where he specialized in Computer Science.

AJ's personal blog can be found at http://www.ajguillon.com and you can contact him at aj@ajguillon.com.

The Khronos Group is an industry consortium creating open standards to enable the authoring and acceleration of parallel computing, graphics, vision, sensor processing, and dynamic media on a wide variety of platforms and devices. Khronos standards include Vulkan™, OpenGL®, OpenGL® ES, WebGL™, OpenCL™, SPIR™, SYCL™, WebCL™, OpenVX™, EGL™, OpenMAX™, OpenVG™, OpenSL ES™, StreamInput™, COLLADA™, and glTF™. All Khronos members are enabled to contribute to the development of Khronos specifications, are empowered to vote at various stages before public deployment, and are able to accelerate the delivery of their cutting-edge media platforms and applications through early access to specification drafts and conformance tests. More information is available at www.khronos.org.