electronic design

WILLIAM WONG, Technical Editor Editor

11 Myths About OpenCL

OpenCL is a computational framework designed to take advantage of multicore platforms. Unsurprisingly, there are a few myths surrounding it.

OpenCL is an open, cross-platform, parallel-computing standard from the Khronos Group. Applications are written in a variant of C, and the latest OpenCL 2.2 brings a static subset of C++14 *(see "C++14 Adds Embedded Features" on electronicdesign.com)* into the mix.

OpenCL has wide support on both the hardware and software side. It is often a checkbox item for hardware feature sets, and developers can utilize OpenCL-based software without knowing what is involved by either the software or the hardware that it is running on. Those already writing OpenCL already know how it works and will probably not find anything new in the following myths. Likewise, those simply using OpenCL-based software will not care other than to have their software run properly on OpenCL-supported hardware. For the rest of you, please read on.

1. I just need to write C code to use OpenCL.

True, sort of. OpenCL is a language specification, a runtime API, and a software framework that includes an OpenCL compiler and matching runtime. Application code is written in the form of small kernels that are then scheduled to execute by the runtime *(see figure)*. OpenCL kernels are written in C or C++ (as of OpenCL 2.2).

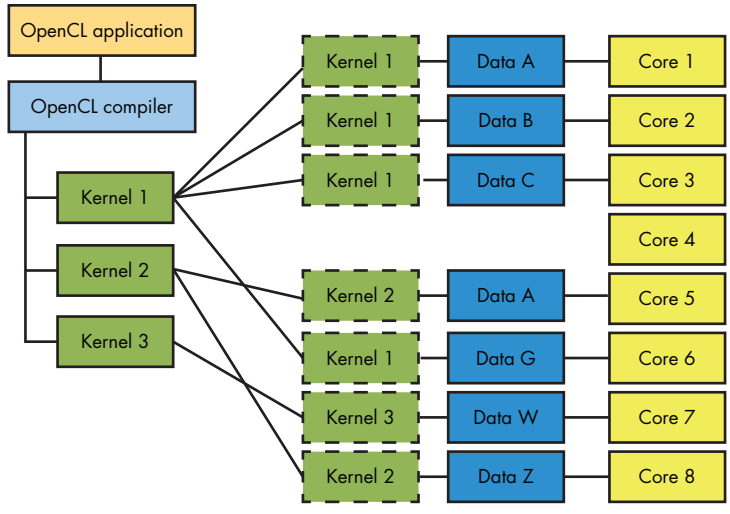
Part of the challenge in writing OpenCL code is to take advantage of the extensions and constructs for parallelizing computation, since this is now OpenCL speeds up execution of an application. This does not come free simply by using an OpenCL compiler. Even determining how functions will be implemented in a kernel will affect performance.

The runtime schedules kernels and their matching data to run on the hardware that normally consists of multiple cores, thereby potentially providing a speed-up of the application. A kernel runs until it terminates, and the resulting data can be used in future computations.

The code for a kernel may be shared among cores in a shared memory system, or it may be copied to a node as in a cluster system. Kernels are not portable across different hardware, but the source code is.

2. OpenCL only runs on GPUs.

Not true. OpenCL is a specification that's implemented by an OpenCL compiler. It can generate code for target hardware



An OpenCL program is compiled into kernels or nuggets of code, which can be applied to data. The resulting data can be passed to the next kernel that requires it. A multicore system is able to run multiple kernels at the same time. They may be the same or different, depending on the application and its current state.

that can include CPUs, GPUs, or a mix of the two. The compiler dictates what targets it will support. An OpenCL application can run on a single-core CPU, but normally multiple-core systems are the target to get more overall performance from a system. OpenCL can also run on FPGAs. This approach is a bit more static because the kernels are implemented in a FPGA, and its configuration doesn't normally change over time. Altera's SDK for OpenCL *(see "How To Put OpenCL Into An FPGA" on electronicdesign.com)* has an OpenCL compiler that generates an FPGA configuration; that includes supporting the configuration for what is essentially the OpenCL runtime. There's also software support to match that moves data between a host and the FPGA and to initiate kernel execution in the FPGA. Essentially, data is placed into the FPGA's memory where a kernel has access to it. The results can then be extracted or made available to another kernel in the same fashion, since OpenCL operates on a CPU or GPU. FPGAs have the advantage of performing many operations in parallel.

3. CUDA is just Nvidia's version of OpenCL.

False. Nvidia's CUDA is similar to OpenCL, but they are distinct. Both use the kernel approach to partitioning code, and both support C and C++ for kernel code. CUDA is an NVidia architecture that specifically targets Nvidia GPU hardware, but it can also generate code for some CPUs, such as x86 platforms providing similar functionality as OpenCL. In addition, the CUDA Toolkit includes additional libraries like cuDNN for deep learning, cuFFT, cuBLAS, and NPP (NVIDIA Performance Primitives for imaging). Nvidia device drivers for Nvidia hardware support CUDA, as well as OpenCL. Typically, developers targeting Nvidia GPUs will choose CUDA or OpenCL for their projects.

4. OpenCL will run better on a GPU than a CPU.

Yes and no. Typically, a GPU will have many more cores than even a multicore CPU. Having lots of cores can help in some applications, but any speed-up tends to be application-specific. Some applications will do better on a multicore CPU, while others will do better with GPUs. A lot depends on how various kernels are written and executed, as well as the type of operations being performed. Operations that take advantage of GPU functionality will usually run better on a GPU. Sometimes a mix of hardware may be the best alternative. It is possible to have an OpenCL system that spans both.

5. I have to program in OpenCL to take advantage of it.

False. It's possible to use applications that were written and compiled for an OpenCL platform. In this case, the platform needs to have the OpenCL runtime installed (usually as a device driver), and the application needs to be compiled for that platform. This is comparable to conventional native-code applications designed for an operating system. The applications require the matching operating system and hardware to execute.

Another way to use OpenCL is to use an OpenCL-based library and call functions in the library from an application that runs on the CPU. The OpenCL APIs are defined for a number of programming languages, including Python, Java, C, and C++.

Writing an OpenCL program is only necessary if neither of the prior scenarios is suitable.

6. OpenCL can only be used by C applications.

False. Kernel code is compiled from C or C++ source, but an OpenCL application doesn't just consist of the kernel code. It's possible to use OpenCL-based libraries and applications in conjunction with application code running on the CPU side that's written in almost any programming language. OpenCL specifies C and C++ APIs for interfacing with OpenCL applications. These APIs have been translated into other programming languages, including popular ones such as Java and Python.

7. OpenCL is hard to learn and program.

True and false. OpenCL uses C and C++, making it easy to start with if one has a background in C or C++. The trick is that OpenCL has its own conventions, techniques, and debugging methodologies that differ from standard C and C++ development. There's a large amount of information, documentation, and examples for OpenCL on the internet, as well as numerous books written on the subject. Still, there are quite a few things to learn in order to create efficient and bug-free OpenCL applications. Debuggers such as AMD's CodeXL have similar but different characteristics than conventional CPU debuggers. OpenCL also uses different tracing and profiling tools.

8. OpenCL cannot be used for stream programming.

False. Stream programming is normally associated with a data stream such as an audio or video stream. Managing the data is a little more complex than a typical OpenCL application, but it is possible.

9. OpenCL only runs on AMD and Nvidia GPUs.

False. OpenCL will run on most GPGPUs, including GPUs from ARM, Imagination Technologies, Intel, and other vendors. It will not run on all GPUs, though, and it requires a matching runtime/driver and OpenCL compiler. The GPGPUs can be integrated with a CPU, or else they can be standalone GPUs that interface with a host (usually via PCI Express).

10. OpenCL requires lots of hardware to run.

False, although more hardware tends to improve performance. OpenCL can run on single-core microcontrollers, but it makes more sense on devices with multiple cores or GPUs

that support OpenCL. OpenCL systems can scale to very large clusters that contain multiple nodes with multiple CPUs or GPUs. Distributing code and data within a cluster can be a more complex chore, but this allows developers to concentrate on the code for the kernels.

11. OpenCL is not good for embedded applications.

False. OpenCL can be very useful in embedded applications. The embedded system will need support for OpenCL, but there are a number of microcontrollers and SoCs that have sufficient resources, along with OpenCL support, to make their use worthwhile. Embedded designers actually have an advantage, since the scope of performance and requirements are known entities. They can often adjust the chips used to scale up or down to match those requirements. OpenCL can also help with power-sensitive applications, since it's often more efficient. Still, OpenCL is not a free lunch. The necessary hardware costs money and uses power. It can allow implementation of features that would not be possible without utilizing OpenCL. Deciding when and how to employ OpenCL is how developers make a difference.