The Windsor Build and Testing Framework

Shane Peelar
University of Windsor
401 Sunset Avenue
Windsor, Ontario N9B 3P4
peelar@uwindsor.ca

Paul Preney
University of Windsor
401 Sunset Avenue
Windsor, Ontario N9B 3P4
preney@uwindsor.ca

ABSTRACT

Khronos open source components, including the ICD and Clang compiler, require significant time and effort to manually download, build, and install. Source code updates to these components require recompilation, and developers must repeat error-prone steps to build new test environments. Ideally developers should be able to use a tool that automatically obtains, builds, and installs OpenCL codes, libraries, and tools. The Windsor Build and Testing Framework (WBTF) is a tool that has been developed at the University of Windsor that does this. This paper will discuss how the WBTF works, demonstrate how it is used, will show how OpenCL C and C++ programs can be built, run, and/or used to perform various header-only, link, and/or various conformance-style tests using OpenCL reference, host-installed, or using device-installed header and libraries. Those interested in OpenCL C/C++ development, the Khronos OpenCL Clang compiler, and in writing conformance tests will be interested in this framework.

CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**; **Software maintenance tools**; *Software libraries and repositories*;

KEYWORDS

OpenCL C, OpenCL C++, software development, conformance tests

ACM Reference format:
Shane Peelar and Paul Preney. 2017. The Windsor Build and Testing Framework. In *Proceedings of May 16-18, 2017,*
, Toronto, Canada (IWOCL '17), 2 pages.
DOI: http://dx.doi.org/10.1145/3078155.3078184

1 INTRODUCTION

The Khronos Group [1] is responsible for a number of industry standards including OpenCL, SPIR-V, and various open source software tools and libraries for such [13]. Supported software includes the OpenCL ICD Loader [8], OpenCL C header files [11], OpenCL C++ Standard Library [10], Khronos Reference OpenCL C and OpenCL C++ compiler [12], and Khronos Reference LLVM Framework with SPIR-V support [9]. Unfortunately downloading, configuring, and building these requires a considerable amount of time, effort, and expertise. The purpose of the Windsor Build and Testing Framework (WBTF) is to automate:

- downloading and building of software components in a platform-independent manner,
- building header-only, host-installed, or device-installed code using specific versions of Khronos software components, and,
- testing/running of such.

We achieve platform independence by using CMake [14]. CMake is a cross-platform software utility used to build programs taking into account file modification dates, source code dependencies, auto-detection of installed tool configurations, etc. Complete with its own programming language, modern versions of CMake also support the downloading of code from the Internet and some shell scripting operations.

The following sections discuss (i) the design of our framework, (ii) how the download-and-build software tools subsystem operates, (iii) how the build-and-testing subsystem can be used to build and test programs, and (iv) additional toolchain support.

2 THE WBTF FRAMEWORK

The WBTF is comprised of two subsystems called the (1) download-and-build subsystem (DABS) and the (2) build-and-test subsystem (BATS). The DABS is used to download various software tools and libraries, CMake, and Ninja [2], etc. followed by building them using a user-installed C/C++ compiler. If the DABS script is run more than once, it will update existing downloads and if there have been any changes, as detected by CMake, then those tools will be automatically rebuilt if necessary. The result of the DABS is the installation of various OpenCL-related headers and libraries and a toolchain capable of compiling OpenCL C and OpenCL C++ code. The BATS is then able to use the installations from the DABS to build programs and/or to run conformance-style tests. The DABS can also be used on its own if a user only wants to use the DABS.

2.1 The Download-And-Build Subsystem (DABS)

The DABS requires these previously-installed tools:

- CMake version 2.8 or higher
- Git version 2.10 or higher [4]
- Python version 2.7 [7]
- a C/C++ compiler capable of compiling Clang (e.g., Clang [3], GCC [6], Visual Studio C++ (2015 or 2017) [5])

IWOCL '17,
 978-1-4503-5214-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3078155.3078184

which are necessary to download and build CMake v3.8, Ninja v1.7.2, the various aforementioned tools and libraries. This is done by running a `build.sh` (for Linux/Unix systems) or `build-win.bat` (for Windows) script in the `02-buildsystem` directory. These scripts invoke CMake to: (i) determine the version of CMake installed on the system (if too old, then it downloads and builds CMake v3.8), (ii) downloads and builds Ninja, and, (iii) downloads and builds all of the aforementioned software tools and libraries.

The downloads, builds, and installations from this process are placed, by default, as sibling directories to the top-level WBTF installation directory. Each sibling directory has the same name as the top-level directory with an added suffix: (a) `-build`: location where builds occur; (b) `-checks`: location where conformance-style tests are placed; (c) `-dloads`: location of all downloads; (d) `-installs`: location where installs are placed; (e) `-sdks`: for various SDKs, headers, and other OpenCL libraries.

Running the build script is the only task required unless one needs to override the default paths and/or add additional repositories. The latter is done by editing specific CMake files per WBTF documentation. After the DABS' installations have completed, the BATS can then be used.

2.2 The Build-And-Test Subsystem (BATS)

Since OpenCL configurations can vary widely in practice, we use these definitions for describing BATS configurations: (i) a "build system" identifies the system that code/tests are being run on; (ii) a "host system" identifies the system that will execute code/tests, e.g., "native" and "android"; (iii) a "device" identifies the targeted OpenCL device attached to the system.

Additionally we use these definitions to describe types of common OpenCL tests: (i) a "test configuration" is a CMake file that defines the parameters for the tests being performed; (ii) a "run-time test" is a host program linked against an OpenCL implementation; (iii) a "header-only test" or "compile-time test" is a C or C++ source file that #includes OpenCL headers whose resulting code after compilation is not intended to be executed (i.e., the test itself occurs during compilation and the test is only successful if compilation was successful). Compile-time checks are intended to check host/device code and/or headers for validity. Two types of compile-time tests are supported: (i) a "host test" is a test that is run using the host compiler; and (ii) a "device test" is a test that is run using an OpenCL C or OpenCL C++ offline (device) compiler.

A testing configuration is invoked by running the `runtests.sh` script in the `03-conformance-checks` directory passing in a CMake script for a testing configuration, e.g.,

```
./runtests configs/default.cmake
```

A configuration script determines which tests are invoked and the toolchain used by setting some special variables including: (i) OPENCL_VERSION which identifies the version of OpenCL being used; (ii) VENDOR which identifies the toolchain being used; and the (iii) TEST_TYPE which may be "runtime", "compiletime-host", or "compiletime-device". Just as the DABS placed its results in sibling directories, so does the BATS. The sibling BATS directory has the suffix `-results` and it captures its results in a subdirectory named as follows:

```
opencl-<OPENCL_VERSION>-<VENDOR>-<TEST_TYPE>
```

The BATS uses the tools and libraries the DABS retrieves. If additional SDKs are required, they can be added to the `-sdks` directory as new vendor implementations. The WBTF provides an example of this for the Android Mali chipset. Also the BATS is not intended to replace existing infrastructure: it is to be used as a convenient means to manage builds and conformance testing across a wide variety of hardware, OpenCL configurations, and/or infrastructures. Existing conformance testing systems should be able to plug directly into this system.

2.3 Additional Toolchain Support

Using Linux or Windows with a compiler toolchain already installed, the WBTF will work without any configuration. As there will be times when defaults are insufficient, we've tested and documented how one can add additional toolchain support to build and test OpenCL code, e.g., Android Mali support. In the BATS, this is supported via a "vendor" tag. This ability is needed to fully support both compile-time and run-time tests on the host or device (e.g., using a host cross-compiler, or, a device compiler via SSH). This ability has, as of this writing, only been tested for Android Mali targets with the rest of our BATS development using OpenCL devices installed on the host x86_64 computer, e.g., a graphics card.

3 CONCLUSIONS

The WBTF DABS makes it easy to use The Khronos Group's OpenCL C/C++ compiler toolchain and libraries. This allows interested hobbyists, researchers, and professionals to develop and explore OpenCL and SPIR-V in addition to the downloaded sources. Although still a work-in-progress, the BATS permits those who have at least one OpenCL implementation available to run available Khronos conformance tests (e.g., OpenCL C++) and to explore how OpenCL software can be built including the use of the ICD. Future work will add the ability to add software hooks to that execute when code is deployed, run, etc. and some additional support for some of the more common SDKs.

REFERENCES

[1] 2000. The Khronos Group. (2000). https://www.khronos.org
[2] 2012. Ninja: a Small Build System With a Focus On Speed. (2012). https://github.com/ninja-build/ninja.git
[3] 2017. clang: A C Language Family Frontend for LLVM. (2017). https://clang.llvm.org/
[4] 2017. Git. (2017). https://git-scm.com/
[5] Microsoft Corporation. 2017. Visual Studio C++. (2017). https://www.visualstudio.com/vs/cplusplus/
[6] Free Software Foundation. 2017. GNU Compiler Collection. (2017). https://gcc.gnu.org/
[7] Python Software Foundation. 2017. Python. (2017). https://www.python.org/
[8] The Khronos Group. 2015. OpenCL ICD Loader Repository. (2015). https://github.com/KhronosGroup/OpenCL-ICD-Loader
[9] The Khronos Group. 2016. LLVM Framework with SPIR-V Support. (2016). https://github.com/KhronosGroup/SPIRV-LLVM.git
[10] The Khronos Group. 2016. OpenCL C++ Standard Library Repository. (2016). https://github.com/KhronosGroup/libclcxx.git
[11] The Khronos Group. 2016. OpenCL Headers Repository. (2016). https://github.com/KhronosGroup/OpenCL-Headers.git
[12] The Khronos Group. 2016. SPIR Generator/Clang Compiler with OpenCL C and OpenCL C++ Support. (2016). https://github.com/KhronosGroup/SPIR.git
[13] The Khronos Group. 2017. Public GitHub Repositories. (2017). https://github.com/KhronosGroup
[14] Kitware Inc. 2000. CMake: The Cross-Platform, Open-Source Build System. (2000). https://cmake.org/