Modern CMake
Date: September 19, 2018
Presented by: Bill Hoffman (Kitware)

Bill Hoffman, Robert Maynard, Zack Galbreath, Chuck Atkins and Jamie Snape contributed to the answers.

Q: Where can we get the video after the presentation?

A: https://www.exascaleproject.org/event/cmake/. A link will also be emailed to all those that registered.

Q:Is it possible to create a release using cmake that an end-user can build without having cmake on their system?

A: No, CMake is always required. The only requirement for CMake is however a C++ compiler. If you need the user to compile the project they will need to have CMake on the system. If you want to provide binaries/libraries they can be provided without any requirement on having CMake installed.

Q: Can you describe the differences between CMake and Bazel. I've used CMake. But I am not familiar with Bazel. From what I heard it is like CMake, but I am not sure how much their functionalities meet

A: Sure.

CMake: Supports a wide variety of platforms and compilers. CMake has a very active open source community accepting of patches large and small. Has a backwards compatibility system built into it with policies. Has a system for finding system installed software and software built with CMake. Has a config system that allows even non CMake based systems to specify targets to CMake. Qt is a good example of this. Supports man IDE's and workflows.

Bazel: Bazel has full support for Linux and Mac only with limited Windows support. It has sandboxing. All inputs and outputs must be declared. Limited number of compilers are supported. No built-in functionality for finding or using most system libraries. Better caching of build artifacts (compared to CMake with Make/Ninja). Caching of test results. Support for remote (distributed) caching of build artifacts and test results. Limited IDE

support (IntelliJ tools and Xcode). Better Java/Go/Python support. Poor interfacing with other build systems. Very unstable, no backwards compatibility at all. Very limited open-source community. Difficult to get patches merged.
In summary, if you are building on platforms that Google is not interested in or have problems that Google is not interested in they may never be addressed by Bazel. CMake on the other hand is very interested in the HPC community, Fortran, and getting many platforms working well.

Q: Can you recommend any good resources to get started with CMake? I find it quite hard to find good tutorials.

A: [Robert Maynard] I think that Craig Scott book is excellent if you are happy with a ebook. https://crascit.com/professional-cmake/

Other good community tutorials can be found at:

[Jamie Finney] Two good starter resources I used:

- https://rix0r.nl/blog/2015/08/13/cmake-guide/
- https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1

I liked CLion tutorial at https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html and integration.

If you want more hands on tutorials, kitware does offer training courses

Q: Where can I find concrete CMake "recipes" for moderately complex tasks that would show me the best practices that I should be using? The documentation is quite thorough, but very reference-like. I can usually find the functionality that I think I need in the main CMake documentation, but the best way to use it and how it interacts with the rest of the system are things that I have to end up searching the web.

A: See answer to previous question.

Q: Should the angle brackets around the INSTALL prefix be {} ?

A: Was this for the <INSTALL>/directory example? If so that means that the variable is expanded as a generator expression, and not CMake configuration time. Basically you

can think of generator expressions as deferring evaluation of something until execution. You can find more information on the concept of generator expressions at:

- https://cmake.org/cmake/help/latest/manual/cmake-buildsystem.7.html#build-specification-with-generator-expressions
- https://cmake.org/cmake/help/latest/manual/cmake-generator-expressions.7.html

Q: I have 40-core system. How do I configure in parallel? (My builds are already parallel)

A: Currently CMake is single threaded only, so we aren't able to configure in parallel.
[Q: what kind of configure times are you currently seeing?]
A: I configure for a minute or more and build in seconds on 36 cores (Skylake). I was thinking to use cmake -E <magic_command> to make that faster (but I might be heading the wrong direction)
A: No CMake language and command constructs would need work to handle parallel configurations.

Q: Is there a nice built-in facility for regression tests that does the output comparison automatically?

A: [Robert Maynard] As far as I remember CMake out of the box only offers regex / and error code handling to determine if a test fails. I will have to ask Bill once he is done if it offers other ways. I know that VTK handles image regression comparisons as part of the test driver and not as part of CMake. There is an undocumented feature in ctest that is used to get image differences to CDash. You can read about that here https://stackoverflow.com/questions/50719666/upload-image-diff-using-ctest-and-cdash/50760569#50760569.

Q: Is there a difference in functionality between include(CTest) and enable_testing()?

A: enable_testing is a subset of include(CTest). So include(CTest) brings in more things like the BUILD_TESTING option. It also sets up the extra targets for CDash like Nightly, Experimental, Coverage builds (if i remember correctly)

Q: Our project can be linked with a few different math libraries (blas, openblas, mkl, essl, etc), are there built-in ways to test for libraries/compile options? I've done a set of if()s but that's bulky. Just looking for suggestions/best practices.

A: Are you looking for a way to detect which math library the user has selected. Or are you looking to test some options with that library. For the second you can look at the CheckCXX modules (https://cmake.org/cmake/help/v3.12/manual/cmake-modules.7.html) Maybe something like: https://cmake.org/cmake/help/v3.12/module/CheckCXXSymbolExists.html

> Follow-up Q: A technique that I'd like to do is create a list of libraries that work with the project and then check to see if I can link against those, using the first one that works (unless specified by the user). That is a technique that seems to have worked when doing an autoconf style script in the past and I'd like to replicate that if it's possible.
>
> Follow-up A: You can do this with iterating a list of libraries (using foreach) and using try_compile to see which ones works.

Q: when can we expect next cdash release?

A: October 31, 2018

Q: How do I let other projects know what I need as deps for headers, libs, flags. Etc.? For example, I found "librare.so" in an obscure location and it's now required to link against it. How does the user that links my library know what to do without repeating my effort to find "librare.so"?

A: You could create your own export target for it. Obviously that would only work on the computer where you built your software, so might not work with CPack.

Q: How is Cmake compared to other build tools such as SCons and GNU Autotools?

A: This is a good article on that topic: https://lwn.net/Articles/188693/

Q: Have you noticed CMake builds being significantly larger than the autotools? In HDF5 the difference is enormous. Not sure if that's inherent to CMake or a problem with our build scripts.

Autotools: 230 MB
CMake 3.3 GB

^^^ Size of build directory. Same build options. Not a debug/release difference. Both are debug.

A: Large in which manner? Binary size / build directory? I would recommend checking the build options of cmake with a build line with 'make VERBOSE=1'. That order of magnitude smells like a debug build instead of a release build
Send an email to: robert.maynard@kitware.com, and bill.hoffman@kitware.com . Cmake shouldn't have any influence on the library size, so some logic in HDF5 must be causing different command line invocations of the compiler

-- Will do. Thanks!

[Chuck Atkins] It's also possible (I'd say even likely) that the autotools build is stripping the installed binaries while the CMake build leaves them unstripped. Try to use "make install/strip" and see if the resulting installed files are much smaller.

Q: Any good books about CMake? "Mastering CMAKE" seems dated and doesn't cover "modern CMAKE".

A: I think that Craig Scott book is excellent if you are happy with a ebook. https://crascit.com/professional-cmake/

Q: What is the difference between compile features and compile options?

A: in general compile features is what we name c++ language level features that are from different c++ levels (11, 14, 17, …). Compile options are for general compile flags to the compiler.

Q: Question from The HDF Group: Are you developing any tools to help with cross-compilation on HPC systems? Thank you!

A: We are working with the SPACK team for better cmake integration into HPC builds. However, this might not answer your question. Can you give more specific details as to the HPC systems you are building for and what your issues are?

Q: I currently work on a project that cannot yet use Cmake as the build system, but we love using CDash. I currently do it by spoofing the Test.xml file with a script, which is very hacky. Is there a better way to do this?

A: We are working with SPACK to make this work better. You can use CTest for this, but if you don't want to do that we are working on a CDash API that should make this easier in the future. If you are interested in this we would be interested in helping to move this type of feature forward. Please contact bill.hoffman@kitware.com and we can see how we can help.

Q: Any advice for dealing with antiquated CMake build systems? For example I have a multi-physics project that I am working on with a requirement to build on Windows, and Linux. Some of the software dependencies include SEACAS-Exodus and CGNS (and their dependencies). It seems that CGNS won't compile with MSVS with Fortran bindings enabled using Intel IVF/ifort and MSVC. (Also it took some searching to get a version of HDF5 that CGNS would compile and link against.) Then SEACAS-Exodus uses a very unix centric bash script to handle their dependencies, which includes NetCDF4 built against HDF5. A super build was defined to reduce tight coupling, but it seems that the CGNS CMake build system is not robustly cross platform (at least for Fortran support) and then shared dependencies and use of custom written find modules in SEACAS-Exodus further complicate compilation on windows. If the upstream dependencies used modern CMake with packageConfig.cmake files then this likely would not have been an issue (or less of an issue). Any tips for dealing with this dependency hell and upstream software writing and packaging custom find modules?

A: A superbuild would be the way to go. You can use that to constrain the problem and know where depends will be installed into your build tree or built. When you find packages that are not robust for the platform you need, in the long run it is better to fix them and then contribute upstream. There is no magic CMake bullet to fix this. However, Kitware can be contracted to help clean up the build if you have the funding for that. The approach we would take would be to fix things up and get those fixes upstream if possible or maintain patches if not.

Q: What is the best way to deal with differences between CMake versions? (Cmake 3.0 does not support what Cmake 3.12 supports. For example, C99 detection and handling of -std=c99)

A: In general the goal is that CMake is backwards compatible so CMake 3.12 will be able to behave like CMake 3.0. This is done through the cmake_minimum_required command which needs to be at the top of your project (basically the first line in your CMakeLists.txt before the first project() command). If you need to rely on things like C99 detection you can specify a minimum cmake version. If you want to gracefully degrade support that is possible, and you can find the cmake version with

`CMAKE_VERSION`:
https://cmake.org/cmake/help/v3.8/variable/CMAKE_VERSION.html

Q: if I have a CMake build system now, how soon do I have to update it from older CMake to New CMake?

A: Depends on what you need to build. Some projects will have a minimum cmake version, for which you will need to have at least that cmake version. It is possible to have several versions of cmake installed on a system if you do the installation yourself. Since CMake is backwards compatible you could keep it with old CMake for as long as you like. After time you will likely get warnings and the recommended fix is to update your code to the new style.

Q: What hardware resources are needed for parallel testing, for example for ParaView?

A: So let me clarify quickly what we mean by parallel testing. In general that simply means that CMake will run N number of tests in parallel where N is provided by the user. It provides ways to describe if tests have setup or teardown (aka fixtures, https://crascit.com/2016/10/18/test-fixtures-with-cmake-ctest/), or if tests need to be run serially, or if a test itself uses multiple cores (MPI) and should be scheduled as using M (from N).

Now paraview for the hardware capabilities to run the tests are basically the following:

- The ability to spawn a rendering window (onscreen, offscreen or egl)
- Have more than 1 core

That is about it. The CMake buildsystem will enable all the tests related to the features that are enabled in the build. if you disabled the ParaView client, you will just run the tests related to the server component.

As far as MPI tests go, it uses a pretty low MPI count and just tests locally.

When using MPI in tests you can use this test property:
https://cmake.org/cmake/help/v3.0/prop_test/PROCESSORS.html

That can allow ctest -j to not overload the machine by running tests that use more than one processor as if they only use one processor.

Do that answer your question?