Developing and Shipping LLVM and Clang with CMake

The lesser of two evils

Chris Bieneman
IRC: beanz
GitHub: llvm-beanz

Agenda

- Transition from Autoconf to CMake
- Shipping LLVM & Clang
- Putting the Build System to Work
- Current State
- Opportunities for Future Improvement

Road to One True Build

Why CMake?

- Cross-platform build configuration tool
- Simple and powerful scripting language
- Supports native development and IDEs on many platforms
- Active and attentive open source community
- Easily available binaries and source packages

CMake Language

```
cmake_minimum_version(VERSION 3.6)
project(HelloWorld)

add_executable(HelloWorld HelloWorld.cpp)
if(UNIX)
  set(extra_sources Unix.cpp)
endif()
if(UNIX)
  target_compile_definitions(HelloWorld PUBLIC UNIX)
endif()                                  ${extra_sources})
```

CMake References

- http://llvm.org/docs/CMakePrimer.html
- http://llvm.org/docs/CMake.html
- http://llvm.org/docs/AdvancedBuilds.html

Bumps in the Road

- Coordinating infrastructure updates
- Nobody wanted to think about it
- Late surfacing downstream issues

What Worked?

- Community, community, community!
- Regular status reports
- Face-to-face or IRC conversations

Lots of Bugs

- Bug 12157 - llvmconfig.cmake.in make cmake installations not relocatable
- Bug 14109 - CMake build for compiler-rt should use just-built clang
- Bug 15325 - CMake build does not contain the OCaml bindings
- Bug 15493 - No option to build shared libLLVM-version.so in CMake
- Bug 18496 - [cmake] .S assembly files not compiled by cmake in libclang_rt.ARCH
- Bug 18722 - Option to use CMake with libc++ to compile clang
- Bug 19462 - Use the INSTALL(EXPORT ...) to export CMake definitions
- Bug 19465 - Cmake shared library format on osx
- Bug 21559 - Some x86_64 don't run with Cmake on FreeBSD
- Bug 21560 - Add support to cmake for using installed versions of LLVM and Clang
- Bug 21561 - Update release scripts to use CMake
- Bug 21562 - Add a CMake equivalent for make/platform/clang_darwin.mk in compiler_rt
- Bug 21569 - Can't `make install prefix=/tmp/llvm' with CMake.
- Bug 21570 - Cannot set default configuration options for CMake
- Bug 22725 - lldb build with cmake fails with "Program error: Invalid parameters entered, -h for help. "
- Bug 24154 - CMake shared files are broken in llvm-3.7-dev
- Bug 24157 - CMake built shared library does not export all public symbols
- Bug 25664 - lib/*.dylib have invalid RPATH
- Bug 25665 - cmake build system lacks a way to build libclang_rt without building libc++
- Bug 25666 - requested re-export symbol std::set_unexpected(void (*)()) is not from a dylib, but from .../exception.cpp.o
- Bug 25675 - cmake build doesn't install FileCheck, count, not, and lli-child-target
- Bug 25681 - clang --version does not report revision like it does when built with automake

Lots of New Features

- Multi-stage clang builds
- Distribution targets
- Per-tool install targets
- CMake Caches
- Ninja Pools
- LIT-based harness for profiling clang
- Cleanup and standardization
- Modules support
- Runtimes subdirectory
- CMake Target Export fixes
- ExternalProject wrappers
- LLVM test-suite support
- CMake version bumped to 3.4.3
- Documentation!
- LIT suite targets
- LTO option (supports thin and full)
- Option to build instrumented clang
- Optimized tablegen support
- Darwin Builtins support
- Compiler-RT Embedded Darwin support

Shipping LLVM and Clang

Building a Distribution

- Build clang in two stages
- Selectively choose which tools to install
- Build configuration
  - Optimization settings, vendor settings…
- test-release.sh is over 500 lines!!!

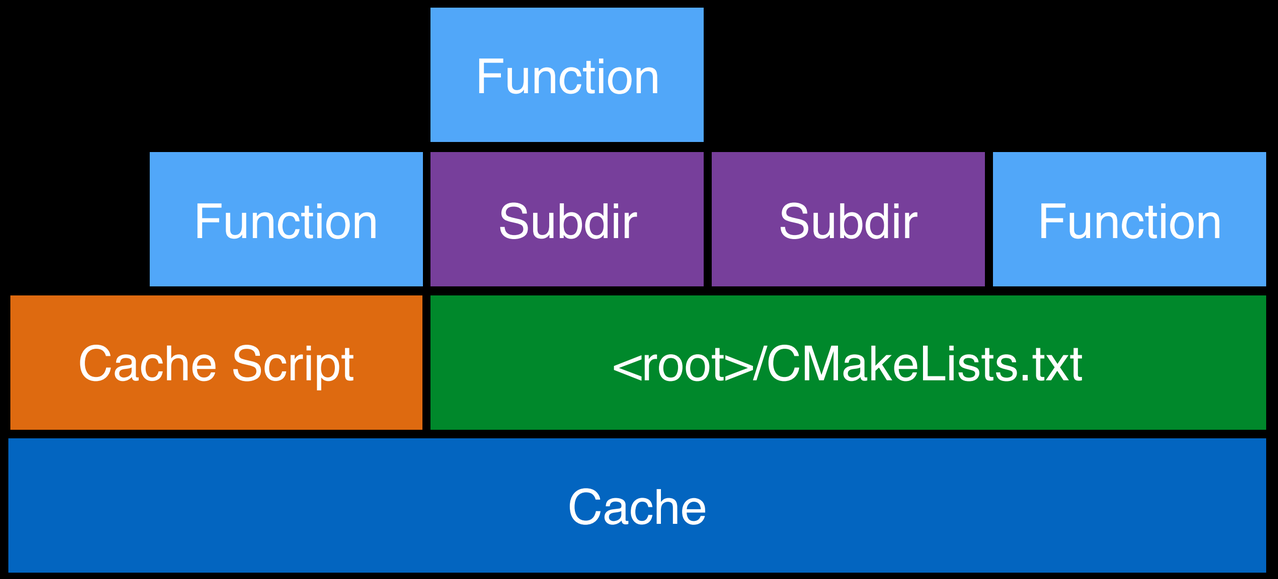
Packaging LLVM & Clang

```
> cmake -G Ninja -C ../clang/cmake/caches/DistributionExample.cmake ../llvm
> ninja stage2-distribution
> ninja stage2-check-all
> ninja stage2-install-distribution
```

- Using Ninja Generator
- DistributionExample is a CMake cache script
  - CMake scripts executed before the root CMakeLists.txt
  - Isolated scope can only modify the cache

CMake Variable Scopes

- All defined variables are effectively passed by value into child scopes on entry
- Variables can be propagated up by using the `set` command's `PARENT_SCOPE` option
- Variables can also be set in the cache using the `set` command's `CACHE` option
- Prefer using properties instead of cached variables wherever possible
- Macros, loops, and conditional statements do not have their own scope!



CMake Caches

<clang>/cmake/caches/DistributionExample.cmake

```
# This file sets up a CMakeCache for a simple distribution bootstrap build.

# Only build the native target in stage1 since it is a throwaway build.
set(LLVM_TARGETS_TO_BUILD Native CACHE STRING "")

# Optimize the stage1 compiler, but don't LTO it because that wastes time.
set(CMAKE_BUILD_TYPE Release CACHE STRING "")

# Setup vendor-specific settings.
set(PACKAGE_VENDOR LLVM.org CACHE STRING "")

# Setting up the stage2 LTO option needs to be done on the stage1 build so that
# the proper LTO library dependencies can be connected.
set(BOOTSTRAP_LLVM_ENABLE_LTO ON CACHE BOOL "")

# Expose stage2 targets through the stage1 build configuration.
set(CLANG_BOOTSTRAP_TARGETS
    check-all
    check-llvm
    check-clang
    llvm-config
    test-suite
    test-depends
    llvm-test-depends
    clang-test-depends
    distribution
    install-distribution
    clang
    CACHE STRING "")

# Setup the bootstrap build.
set(CLANG_ENABLE_BOOTSTRAP ON CACHE BOOL "")
set(CLANG_BOOTSTRAP_CMAKE_ARGS
    -C ${CMAKE_CURRENT_LIST_DIR}/DistributionExample-stage2.cmake
    CACHE STRING "")
```

CMake Caches

<clang>/cmake/caches/DistributionExample-stage2.cmake

```
# This file sets up a CMakeCache for the second stage of a simple distribution
# bootstrap build.

set(CMAKE_BUILD_TYPE RelWithDebInfo CACHE STRING "")
set(CMAKE_C_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")
set(CMAKE_CXX_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")

# setup toolchain
set(LLVM_INSTALL_TOOLCHAIN_ONLY ON CACHE BOOL "")
set(LLVM_TOOLCHAIN_TOOLS
    llvm-dsymutil
    llvm-cov
    llvm-dwarfdump
    llvm-profdata
    llvm-objdump
    llvm-nm
    llvm-size
    CACHE STRING "")

set(LLVM_DISTRIBUTION_COMPONENTS
    clang
    LTO
    clang-format
    clang-headers
    builtins
    runtimes
    ${LLVM_TOOLCHAIN_TOOLS}
    CACHE STRING "")
```

Packaging LLVM & Clang

- Targets prefixed with stage*n* are mapped from later stages
  - This enables complex multi-stage builds
- Distribution is a special target comprised components
  - Configurable using `LLVM_DISTRIBUTION_COMPONENTS` variable
  - Build only what you install

```
> cmake -G Ninja -C ../clang/cmake/caches/DistributionExample.cmake ../llvm
> ninja stage2-distribution
> ninja stage2-check-all
> ninja stage2-install-distribution
```

Packaging LLVM & Clang

```
> cmake -G Ninja -C ../clang/cmake/caches/DistributionExample.cmake ../llvm

> ninja stage2-distribution

> ninja stage2-check-all

> ninja stage2-install-distribution
```

- Installs the components built in the distribution target
- Requires all components have install targets
- Aggregate of “install-${component}” targets

Tips and Tricks

Boosting Productivity

- See Tilmann Scheller's 2015 EuroLLVM talk "Building Clang/LLVM Efficiently"
  - Use optimized host Clang (LTO + PGO)
  - Use ld64, gold, or lld
  - Build Shared Libraries (`LLVM_BUILD_SHARED=On`)

Boosting Productivity

- Use Optimized Tablegen (`LLVM_OPTIMIZED_TABLEGEN=On`)
- Build less stuff
- During iteration test what you changed
- Use Ninja
- Object caching and Distributed builds

Optimizing Clang PGO

```
> cmake -G Ninja -C <path_to_clang>/cmake/caches/PGO.cmake <source dir>
> ninja stage2
```

- PGO build stages are stage1, stage2-instrumented, and stage2 (three stages)
- Builds a compiler, an instrumented compiler, profile data, then an optimized compiler.
- “stage2-instrumented-generate-profdata” will just build the stage2-instrumented compiler and generate profdata.

Optimizing Clang PGO+LTO

```
> cmake -G Ninja -DPGO_INSTRUMENT_LTO=On -C <path_to_clang>/cmake/caches/
PGO.cmake <source dir>

> ninja stage2
```

- `PGO_INSTRUMENT_LTO` is handled by the PGO cache and sets up LTO on the instrumented and final builds

Build Less

- Only include projects you care about
- Only build backends you care about
- Every test subdirectory has a target
- Special "distribution" target is customizable

Status Report

Current Status

- No missing functionality from autoconf
- All infrastructure and downstream users have migrated
- Autoconf is gone!
- Compiler-RT cross-targeting cleanup is ongoing
- LLVM runtimes directory is taking shape
- LLDB build is also getting attention for Darwin support

Future High Value Improvements

Accurate Dependencies

- LIT test suites
  - Accurately tracking dependencies per-test suite
  - Huge reduction in iteration times
- TableGen inputs and outputs
  - Not every compile action actually depends on intrinsics_gen
  - TableGen might run too often

Improving Runtime Builds

- Continuing to expand support for runtime projects
  - Single CMake command to install a full toolchain with runtimes
- Configuring for only one target at a time
  - Doesn’t mean you can’t build for multiple targets from a single CMake command
  - Stop fighting against CMake’s cross compilation support

General Goodness

- Adding PGO test data
- Lit testing for CMake
- Migrating bot configurations into CMake cache scripts
- Remove LLVMBuild
- Investigate CMake 3.6's `CMAKE_NINJA_OUTPUT_PATH_PREFIX`

Questions?