Building and Packaging Modern C++

Piotr Gaczkowski

https://github.com/DoomHammer | @doomhammerng

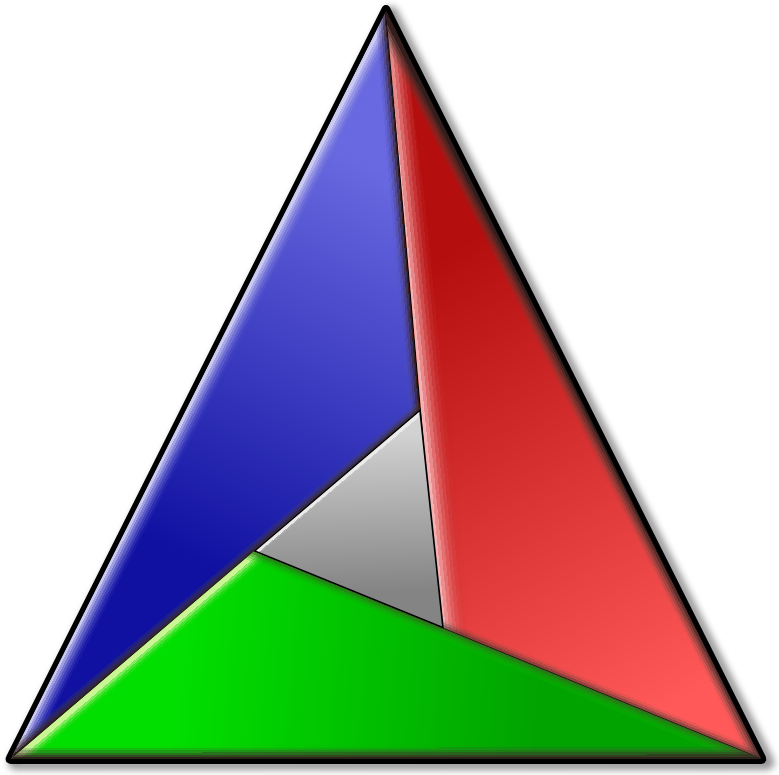
https://doomhammer.info

Adrian Ostrowski

https://github.com/aostrowski | @adr_ostrowski

habana
An Intel Company

CMake



CMake - installation

- system package managers, e. g. apt install cmake

CMake - installation

- system package managers, e. g. apt install cmake
- brew install cmake

CMake - installation

- system package managers, e. g. apt install cmake
- brew install cmake
- pip install cmake>=3.21

CMake - installation

- system package managers, e. g. apt install cmake
- brew install cmake
- pip install cmake>=3.21
- conan install -g virtualrunenv 'cmake/[>=3.21]@'

CMake - usage

```
add_executable(my_app main.cpp)
target_link_libraries(myapp PUBLIC mylib)
```

CMake - usage

```
add_executable(my_app main.cpp)
target_link_libraries(myapp PUBLIC mylib)
```

```
find_package(mydependency REQUIRED)
target_link_libraries(mylib PRIVATE mydependency::mydependency)
```

CMake - integrating other tools

- CMAKE_CXX_CPPCHECK
- CMAKE_CXX_CPPLINT
- CMAKE_CXX_CLANG_TIDY
- CMAKE_CXX_INCLUDE_WHAT_YOU_USE

CMake - integrating other tools

- CMAKE_CXX_CPPCHECK
- CMAKE_CXX_CPPLINT
- CMAKE_CXX_CLANG_TIDY
- CMAKE_CXX_INCLUDE_WHAT_YOU_USE
- CMAKE_CXX_COMPILER_LAUNCHER

Speeding up Builds

Low hanging CMake fruits

- changing your build system
- building only what's required
- using only required tooling

Ninja

Ninja

- small build system

Ninja

- small build system
- designed to be used with a build system generator

Ninja

- small build system
- designed to be used with a build system generator
- especially faster for incremental builds

Ninja

- small build system
- designed to be used with a build system generator
- especially faster for incremental builds
- used by Chrome, Android, LLVM

Ninja + CMake: generating

Several ways possible:

1. cmake -G Ninja (mature since CMake 3.3)

Ninja + CMake: generating

Several ways possible:

1. cmake -G Ninja (mature since CMake 3.3)
2. cmake -G 'Ninja Multi-Config' (CMake 3.17+)

Ninja + CMake: generating

Several ways possible:

1. cmake -G Ninja (mature since CMake 3.3)
2. cmake -G 'Ninja Multi-Config' (CMake 3.17+)
3. export CMAKE_GENERATOR=Ninja (CMake 3.15+)

Ninja + CMake: building:

Several ways possible:

1. ninja

Ninja + CMake: building:

Several ways possible:

1. ninja
2. cmake --build .

Building only what's required

Don't:

```
rmdir build; mkdir build; cd build
cmake -DYADDA=YADDA ..
make -j # or make -j all
```

Do:

```
cmake --build . --target my_app
```

Avoid unnecessary tooling

Include What You Use

- analyzes what you must include and forward declare

Include What You Use

- analyzes what you must include and forward declare
- can lead to great build speedups

Include What You Use

- analyzes what you must include and forward declare
- can lead to great build speedups
- but analysis has quite an overhead

CCache

https://ccache.dev/

CCache - features

- much faster recompilation

CCache - features

- much faster recompilation
- compression

CCache - features

- much faster recompilation
- compression
- statistics

CCache - features

- much faster recompilation
- compression
- statistics
- silent fallback in unsupported cases

CCache - features

- much faster recompilation
- compression
- statistics
- silent fallback in unsupported cases
- easy integration

CCache - features

- much faster recompilation
- compression
- statistics
- silent fallback in unsupported cases
- easy integration
- support for C++20's modules

How much does it help?

A lot!

Personal experience: builds shorter by up to 95%

How much does it help - cont'd

ccache.c

Here are the results of building ccache's own ccache.c with `-g -O2 -MD` and needed `-I` flags:

	Elapsed time	Percent	Factor
Without ccache	0.6988 s	100.00 %	1.00 x
ccache 3.7.1 prepr., first time	0.7251 s	103.77 %	0.96 x
ccache 3.7.1 prepr., second time	0.0247 s	3.53 %	28.33 x
ccache 3.7.1 direct, first time	0.7268 s	104.01 %	0.96 x
ccache 3.7.1 direct, second time	0.0048 s	0.69 %	145.39 x
ccache 3.7.1 depend, first time	0.7102 s	101.64 %	0.98 x
ccache 3.7.1 depend, second time	0.0051 s	0.73 %	137.81 x

CCache - supported environment

- works on Linux and macOS, other Unixes, and Windows
- supports GCC, Clang and NVCC
- MSVC support underway (PR #506)

CCache - installation

- Windows:
  - just use binaries from GitHub
  - scoop install ccache
- Others:
  - system package manager - usually not the latest version
  - brew install ccache
  - nix-env -i ccache
  - build from sources (CMake)

CCache - usage

- invoke manually

```
ccache <compiler> <compiler_args>
```

CCache - usage

- invoke manually

  ```
  ccache <compiler> <compiler_args>
  ```

- invoke via symbolic links masquerading the compilers

CCache - usage

- invoke manually

  ```
  ccache <compiler> <compiler_args>
  ```

- invoke via symbolic links masquerading the compilers
- integrate with build systems

CCache - masquerading compilers

To ensure CCache is used by default:

CCache - masquerading compilers

To ensure CCache is used by default:

1. Run:

```
cp ccache /usr/local/bin/
ln -s ccache /usr/local/bin/gcc
ln -s ccache /usr/local/bin/g++
ln -s ccache /usr/local/bin/cc
ln -s ccache /usr/local/bin/c++
```

CCache - masquerading compilers

To ensure CCache is used by default:

1. Run:

```
cp ccache /usr/local/bin/
ln -s ccache /usr/local/bin/gcc
ln -s ccache /usr/local/bin/g++
ln -s ccache /usr/local/bin/cc
ln -s ccache /usr/local/bin/c++
```

2. Put /usr/local/bin early in PATH

3. Call your compiler by name, e.g. g++

CCache - integrating with CMake

Available since CMake 3.4

CCache - integrating with CMake

Available since CMake 3.4

```
-DCMAKE_CXX_COMPILER_LAUNCHER=ccache
```

CCache - integrating with CMake

Available since CMake 3.4

```
-DCMAKE_CXX_COMPILER_LAUNCHER=ccache
```

```
find_program(CCACHE_PROGRAM ccache)
if(CCACHE_PROGRAM)
 set_property(GLOBAL PROPERTY RULE_LAUNCH_COMPILE "${CCACHE_PROGRAM}")
endif()
```

CCache - configuration

- many environment variables
- corresponding settings in ccache.conf

CCache - configuration, cont'd

- cache size and location
- behavior: sloppiness, preprocessing, etc.
- compiler specific, e. g. prefix_command
- read only mode
- debugging and logging

CCache - sharing cache

- possible on same machine and using a network storage

CCache - sharing cache

- possible on same machine and using a network storage
- for locations afar, consider providing their own caches

CCache - sharing cache

- possible on same machine and using a network storage
- for locations afar, consider providing their own caches
- users need to be in same group

CCache - sharing cache

- possible on same machine and using a network storage
- for locations afar, consider providing their own caches
- users need to be in same group
- in config, provide:

```
cache_size = 100G
base_dir = /home/current/user/
cache_dir = /network/storage/path
hash_dir = false
temporary_dir = /some/local/dir/like/tmp
umask = 002
```

CCache - caveats

- unable to cache results from clang-based tools

What else a developer needs?

Icecream

https://github.com/icecc/icecream

Icecream - features

- scheduler

Icecream - features

- scheduler
  - only uses free resources on machines

Icecream - features

- scheduler
  - only uses free resources on machines
  - allows good perf on heterogeneous environments

Icecream - features

- scheduler
  - only uses free resources on machines
  - allows good perf on heterogeneous environments
  - allows some machines to be off during compilation

Icecream - features

- scheduler
  - only uses free resources on machines
  - allows good perf on heterogeneous environments
  - allows some machines to be off during compilation
- remote cross compiling

Icecream - features

- scheduler
  - only uses free resources on machines
  - allows good perf on heterogeneous environments
  - allows some machines to be off during compilation
- remote cross compiling
- monitoring

How much does it help?

Benoit Girard (:BenWa)

Comment 20 • 5 years ago

We ran:
$ sudo apt-get install icecc

on about 8 desktop machines in Toronto. Now with 40 to 70 jobs we can get 4:30mins Linux builds compared to about 15-20mins on a single machine.

Monitoring - Sundae

https://github.com/JPEWdev/icecream-sundae

Monitoring - Sundae - cont'd

```
Netname: ICECREAM
Servers: Total:10 Available:10 Active:10
Total: Remote:294 Local:53
Jobs: Maximum:99 Active:62 Local:11 Pending:1
      [%%%%%%%%%%%===================================================

↓ ID NAME                   IN    CUR MAX JOBS                OUT   LOCAL ACTIVE PENDING SPEED
+ 1  Host f5bbf6bc2028c02a  35    9   10  [%======== ]        8     3     4      0       100
+ 2  Host d24b929ae3eebe9   31    9   9   [%%=======]         16    4     9      0       100
+ 3  Host 7c2cf5d1d84954fa  24    7   17  [%%%%%%=          ] 57    14    30     0       100
+ 4  Host ef46e21006e8d58e  26    5   13  [=====        ]     9     2     1      0       100
+ 5  Host 7f9d46ea0934991   29    4   5   [==== ]             22    4     1      0       100
+ 6  Host 1ba9c7dd3ee6b75f  12    2   2   [==]                37    1     3      0       100
+ 7  Host 51db20ff5f836f09  31    5   10  [%====     ]        50    15    3      0       100
+ 8  Host 3fb7c9a4e5a7ff70  33    9   11  [%========  ]       22    2     5      0       100
+ 9  Host 71d74fc65f51dc60  37    8   15  [========       ]   50    7     2      0       100
+ 10 Host 1f640d6848ebda75  36    4   7   [====   ]           23    1     4      1       100
```

Icecream - supported environments

- Linux
- macOS
- FreeBSD
- Cygwin

No native Windows :(

Icecream - installation

- developers recommend using distro's package
  - sudo apt install icecc
  - sudo apt install icecc-scheduler
  - sudo apt install icecream-sundae
- be sure to run version 1.3.1 or later

Icecream - configuration

- firewall
  - TCP: 10245, 8765, 8766
  - UDP: 8765
- other defaults should work fine
- persistent connections:
  - --scheduler-host for daemon
  - --persistent-client-connection for scheduler

Combining CCache and Icecream

- Your ccache.conf file must contain:

```
prefix_command=icecc
```

Icecream without CCache

To ensure Icecream is always used by default, put

`/usr/lib/icecc/bin`

early in your PATH.

Icecream without CCache - different approach

```
find_program(ICECC_PROGRAM icecc)
if(ICECC_PROGRAM)
  set_property(GLOBAL PROPERTY RULE_LAUNCH_COMPILE "${ICECC_PROGRAM}")
endif()
```

Icecream - caveats

- bugs in older versions
- only supports GCC and Clang
- tricky cross-compilation cases are... tricky

Noteworthy alternatives

IncrediBuild

- distributed building for Windows and Linux
- commercial
- able to support Intel compilers
- able to distribute tests
- uses CCache under the hood

https://www.incredibuild.com/

sccache

- Mozilla's ccache-like compiler cache
- built-in icecream-style distributed compilation
- supports C, C++, Rust, and NVCC
- on Windows, Linux and macOS

Not production ready yet (current version: 0.2.15)

https://github.com/mozilla/sccache

Portable build environments

Portable build environments

How to make sure everyone's playing the same toys?

VMs

VMs

- All the software preinstalled

VMs

- All the software preinstalled
- Easy distribution

VMs

- All the software preinstalled
- Easy distribution
- May be less than pleasant to use

Containers?

Containers?

- Oooh, shiny!

Containers?

- Oooh, shiny!
- Slicker than VMs!

Containers?

- Oooh, shiny!
- Slicker than VMs!
- Application containers and toolchains don't match

What else?

Nix features

- Operates in userland
- Deterministic packages and environments
- Atomic upgrades
- Rollbacks
- Build environment management
- Multiple versions of packages side-by-side on a single system
- Runs on Linux and macOS

Functional approach

- Installing or upgrading package won't break other packages
- Every package is installed in a separate directory
- It allows easy rollback
- Prevents inconsistent state

Good for multi-user environments

- Several users can install packages without superuser privileges
- Different users can have different package versions

Projects with direnv

Uses nix-shell.

Automatically sets up development environment whenever you enter a directory.

You can pin the packages version.

.envrc

```
use_nix
. env/bin/activate
```

default.nix

```
{ pkgs ? import <nixpkgs> {} }:

with pkgs; {
  gcc11Env = stdenvNoCC.mkDerivation {
    name = "gcc11-environment";
    buildInputs = [ cmake ccache gcc11 git gnumake icecream ];
  };
}
```

How Does it Compare to The Rest?

- Still not as easy as Homebrew
- Getting a working GCC compiler from Git is still tricky
- GNU Guix using GNU Scheme (LISP)
- ... if you love parentheses, you'll love GUIX!
- ... also works with direnv!

Managing Git hooks

Managing Git hooks

- There's an app for that!

Managing Git hooks

- There's an app for that!
- pre-commit

pre-commit

```yaml
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v2.5.0
    hooks:
      - id: check-added-large-files
      - id: check-byte-order-marker
      - id: check-case-conflict
      - id: check-merge-conflict
      - id: mixed-line-ending
      - id: no-commit-to-branch
        args: [--branch, master]
      - id: trailing-whitespace
```

pre-commit

```
#[...]
  - repo: https://github.com/pocc/pre-commit-hooks
    rev: v1.3.4
    hooks:
      - id: clang-format
        args: [--style=Google, -i]
        exclude: 3rd-parties/
      - id: clang-tidy
  - repo: https://github.com/iconmaster5326/cmake-format-pre-commit-hoo
    rev: v0.6.9
    hooks:
      - id: cmake-format
        exclude: 3rd-parties/
```

Packaging

Conan

Conan

- Package manager for C++

Conan

- Package manager for C++
- Written in Python

Conan

- Package manager for C++
- Written in Python
- like pip/npm/gem but with full toolchain support

Conan

- Package manager for C++
- Written in Python
- like pip/npm/gem but with full toolchain support
- uses binaries when possible

Installing Conan

- brew install ccache
- nix-env -i ccache
- pip install conan

Conan - downsides

Conan - downsides

- Binaries might be missing for your platform

Conan - downsides

- Binaries might be missing for your platform
- Sometimes resorts to system packages in a weird way

Conan - downsides

- Binaries might be missing for your platform
- Sometimes resorts to system packages in a weird way
- Still in fast-paced development, things may not be entirely stable

Conan - downsides

- Binaries might be missing for your platform
- Sometimes resorts to system packages in a weird way
- Still in fast-paced development, things may not be entirely stable
- Creating your own packages requires some skill

Conan profile

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=11
compiler.libcxx=libstdc++11
build_type=Release
[options]
[build_requires]
[env]
```

Conanfile - old style

```
[requires]
flac/1.3.3
spdlog/[>=1.4.1]

[generators]
cmake
```

CMakeLists.txt - old style

```
#[...]
conan_basic_setup(TARGETS)
#[...]
target_link_libraries(
  songcorder
  #[...]
  ${CONAN_LIBS}
  #[...]
)
```

Conanfile

```
[requires]
ms-gsl/3.1.0

[generators]
CMakeDeps
```

CMakeLists.txt

```
find_package(ms-gsl CONFIG REQUIRED)
```

CPack

CPack

- Generates sources and binary packages

CPack

- Generates sources and binary packages
- Could spit out NSIS installers and macOS dmg archives

CPack

- Generates sources and binary packages
- Could spit out NSIS installers and macOS dmg archives
- Produces Deb and RPM on supported platforms

AppImage / Flatpack

- The new way to package portable Linux apps

AppImage

```
add_custom_target(bundle
  COMMAND "${CMAKE_MAKE_PROGRAM}" DESTDIR=AppDir install
  COMMAND bash -c
    "${PSD}/tools/linuxdeploy.AppImage --appimage-extract"
  COMMAND bash -c
    "${PSD}/tools/linuxdeploy-plugin-qt.AppImage --appimage-extract"
  COMMAND bash -c
    "${CBD}/squashfs-root/usr/bin/linuxdeploy --appdir AppDir \
    --output appimage --plugin qt -d ${CSD}/songcorder.desktop \
    -i ${CSD}/src/res/songcorder.svg -e $<TARGET_FILE:songcorder>"
  COMMENT "Build Appimage"
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
  DEPENDS songcorder)
```

AppImage

```
add_custom_command(TARGET bundle
  POST_BUILD
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
  COMMAND bash -cv
    "${PROJECT_SOURCE_DIR}/tools/build-installer.py \
    --appimage Songcorder-*.AppImage -n Songcorder \
    -i ${CMAKE_SOURCE_DIR}/src/res/songcorder.png"
  COMMENT "Build installer from appimage"
  VERBATIM)
```

Hungry for more?

Check out the book

Featuring:

- More on architectural styles
- Designing quality software

Questions?

Thank you!

https://github.com/DoomHammer

https://doomhammer.info

https://github.com/aostrowski

habana
An Intel Company

https://doomhammer.info/talks/meetingcpp2021

Attributions

- *Building Site* photo by Samuel Regan-Asante on Unsplash
- *Icecream rainbow* photo by Lama Roscu on Unsplash
- Sundae image by Gerhard G. from Pixabay
- Switch photo by Isabella and Louisa Fischer on Unsplash