C++ Resources for All

Richard Powell (rmpowell77@me.com), 6/15/2022

CMake

reddit

Algorithms + Data Structures = Programs

C++ now

cppreference.com

ADC audio.dev

Medium

stackoverflow

COMPILER EXPLORER

CppCast

YouTube

Visual Studio Code


MacBook Air

CMake Tutorial

https://cmake.org/cmake-tutorial/

Professional CMake: A Practical Guide

https://crascit.com/professional-cmake/

CGold

https://cgold.readthedocs.io/en/latest/index.html

Effective Modern CMake

https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1

Make sure I have cmake up to date

richardpowell — richardpowell@Richards-MacBook-Air-2 — ~ — -zsh — 130×41

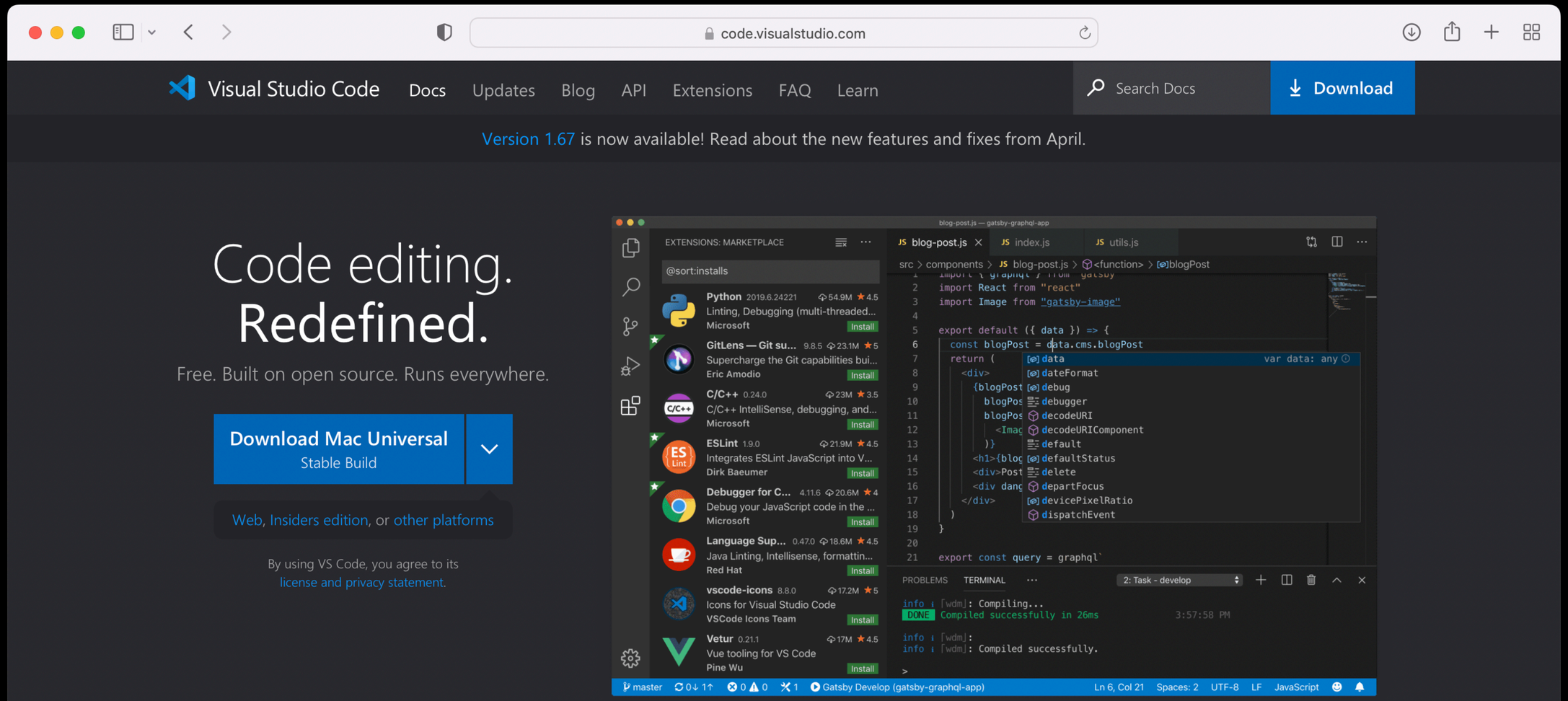
```
[➜  ~ brew install cmake
Warning: Treating cmake as a formula. For the cask, use homebrew/cask/cmake
cmake 3.21.0 is already installed but outdated (so it will be upgraded).
==> Downloading https://ghcr.io/v2/homebrew/core/cmake/manifests/3.23.1
######################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/cmake/blobs/sha256:9fe99b75d9e37e3acd653f93d429af93f94c5460a50284e6c56cf99fb4f8
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:9fe99b75d9e37e3acd653f93d429af93f94c5460a50
######################################################################## 100.0%
==> Upgrading cmake
  3.21.0 -> 3.23.1

==> Pouring cmake--3.23.1.arm64_monterey.bottle.tar.gz
==> Caveats
To install the CMake documentation, run:
  brew install cmake-docs

Emacs Lisp files have been installed to:
  /opt/homebrew/share/emacs/site-lisp/cmake
==> Summary
🍺  /opt/homebrew/Cellar/cmake/3.23.1: 3,043 files, 42.1MB
==> `brew cleanup` has not been run in the last 30 days, running now...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
```

Create the basic project files

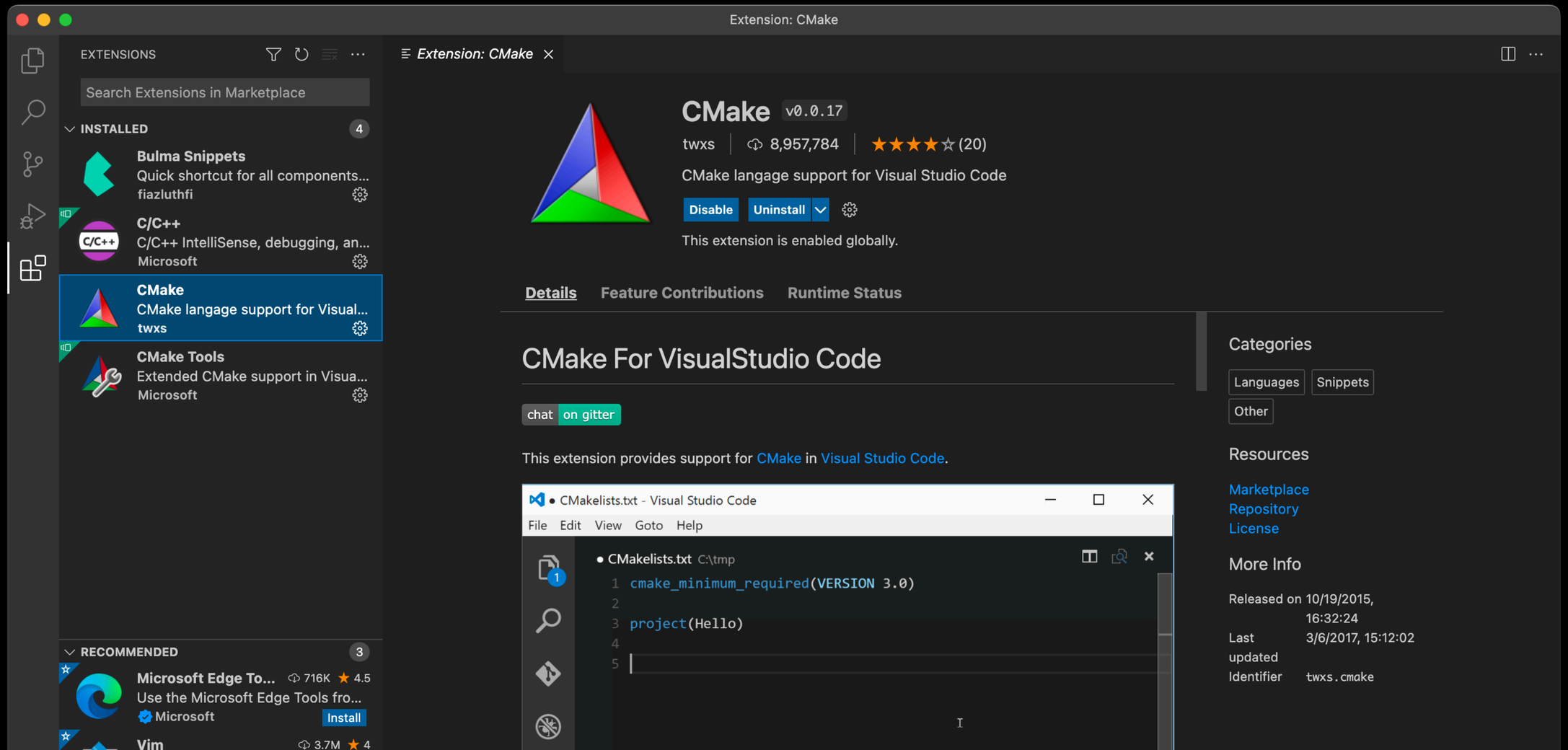
test_fft_benchmark — richardpowell@Richards-MacBook-Air-2 — ..fft_b...

```
➜  test_fft_benchmark cat CMakeLists.txt
cmake_minimum_required(VERSION 3.23)

set(CMAKE_CXX_STANDARD 20)

project(
  test_fft_benchmark
  VERSION 1.0
  LANGUAGES CXX)

add_executable(test_fft_benchmark test_fft_benchmark.cpp)

➜  test_fft_benchmark cat test_fft_benchmark.cpp
#include <iostream>

int main() {
  std::cout<<"hello world\n";
}
➜  test_fft_benchmark
```

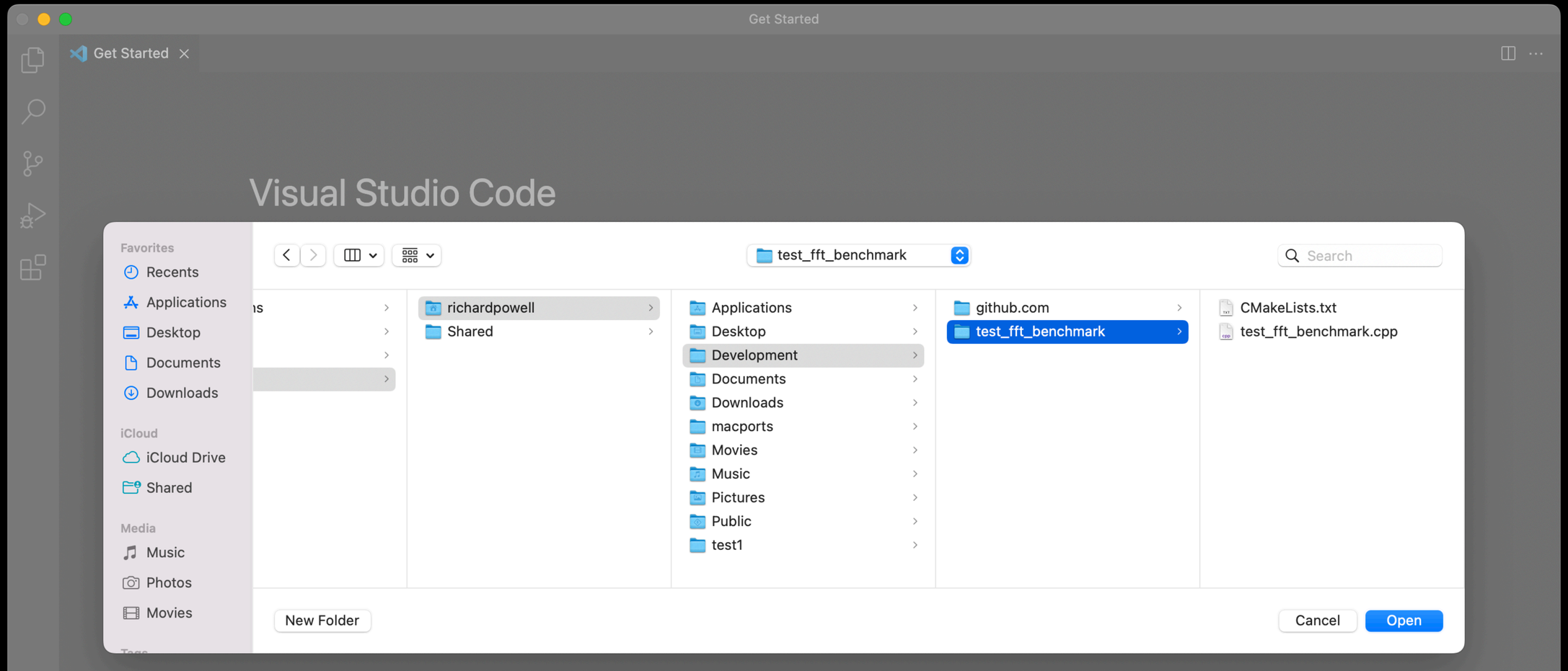
Visual Studio Code



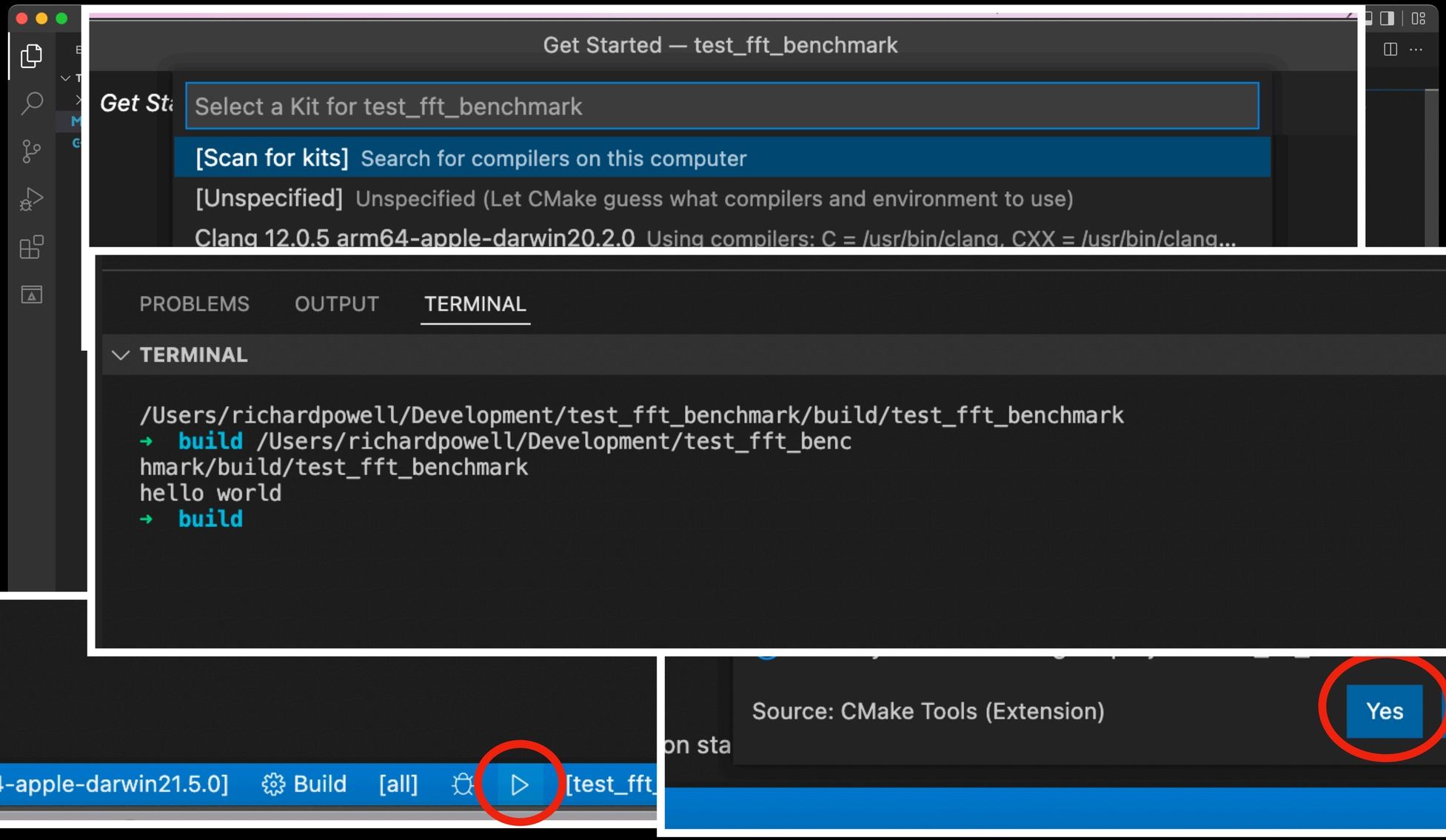
Power up VSC with Extensions



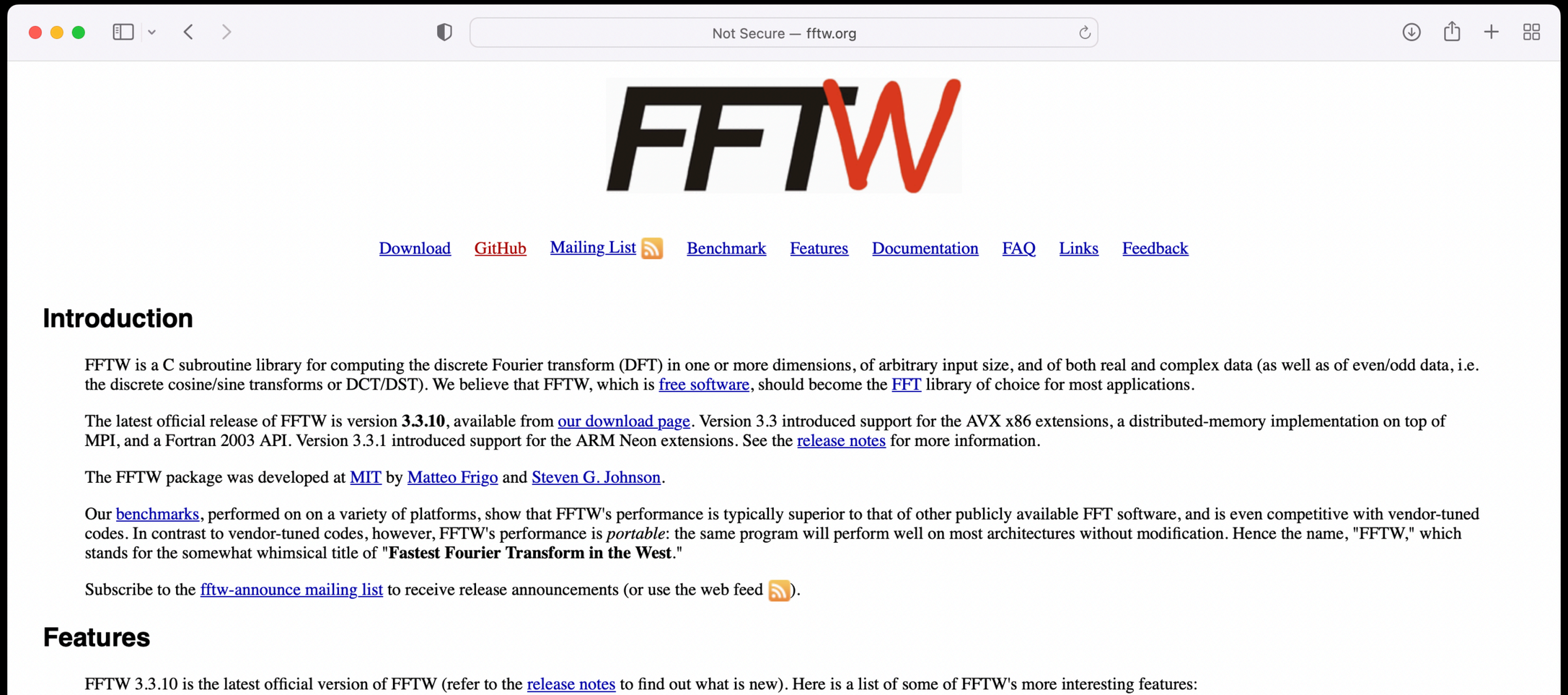
Open the directory



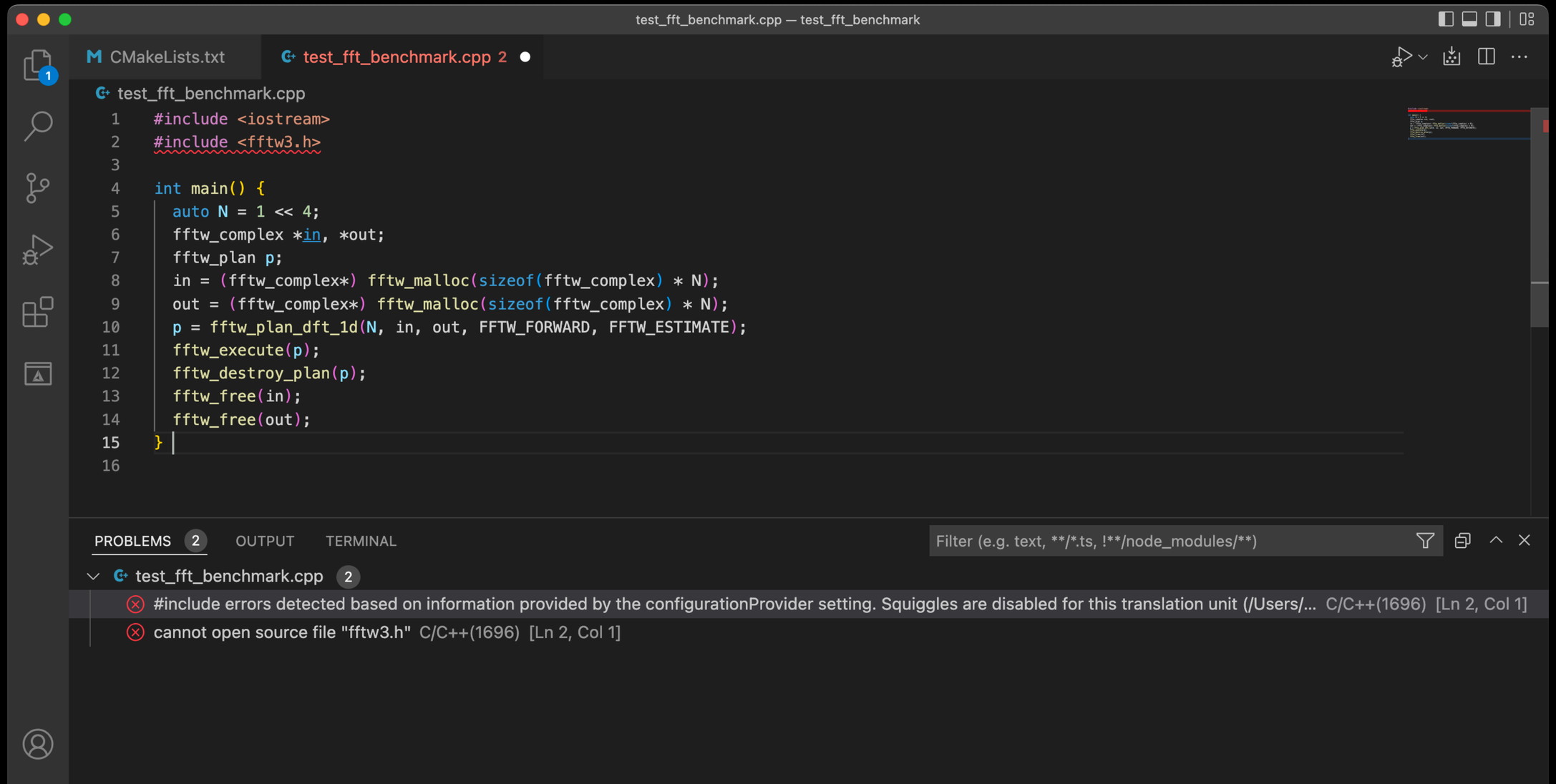
Configuring and build



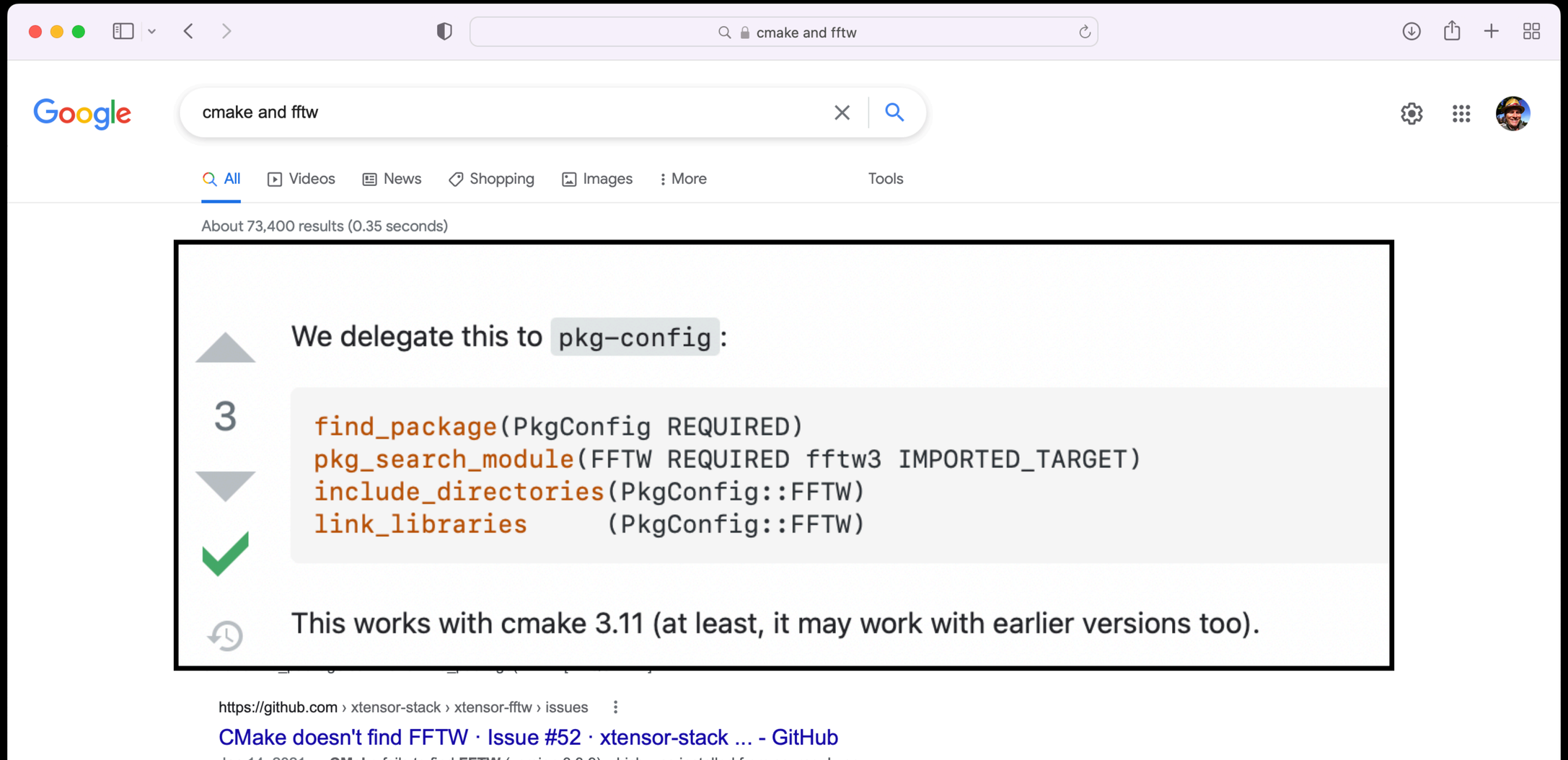
FFTW (the Fastest Fourier Transform in the west!)

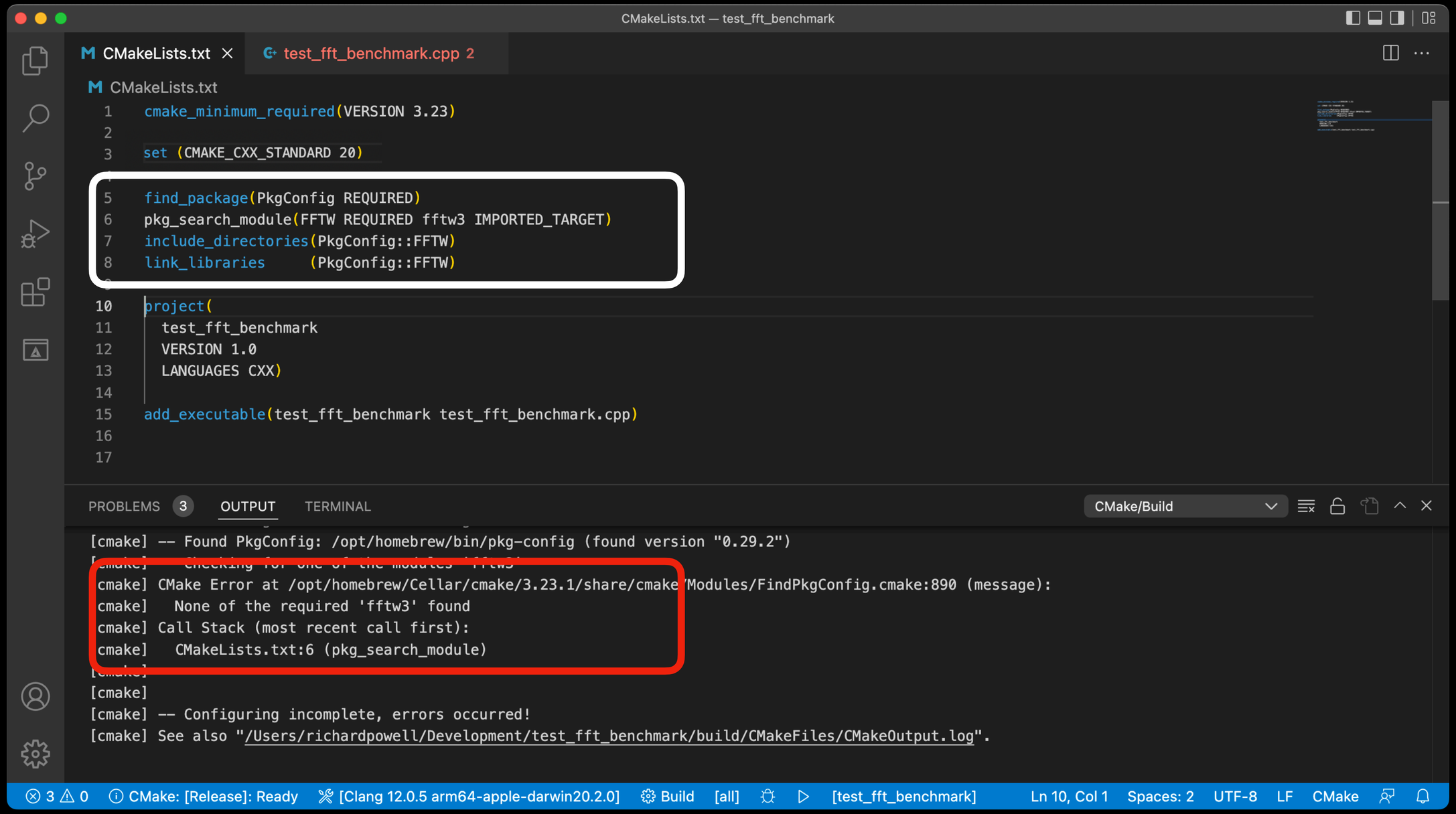


Let’s use it!


test_fft_benchmark.cpp — test_fft_benchmark
CMakeLists.txt
test_fft_benchmark.cpp 2
test_fft_benchmark.cpp
#include <iostream>
#include <fftw3.h>
int main() {
auto N = 1 << 4;
fftw_complex *in, *out;
fftw_plan p;
in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
fftw_execute(p);
fftw_destroy_plan(p);
fftw_free(in);
fftw_free(out);
}
PROBLEMS 2
OUTPUT
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
test_fft_benchmark.cpp 2
#include errors detected based on information provided by the configurationProvider setting. Squiggles are disabled for this translation unit (/Users/... C/C++(1696) [Ln 2, Col 1]
cannot open source file "fftw3.h" C/C++(1696) [Ln 2, Col 1]

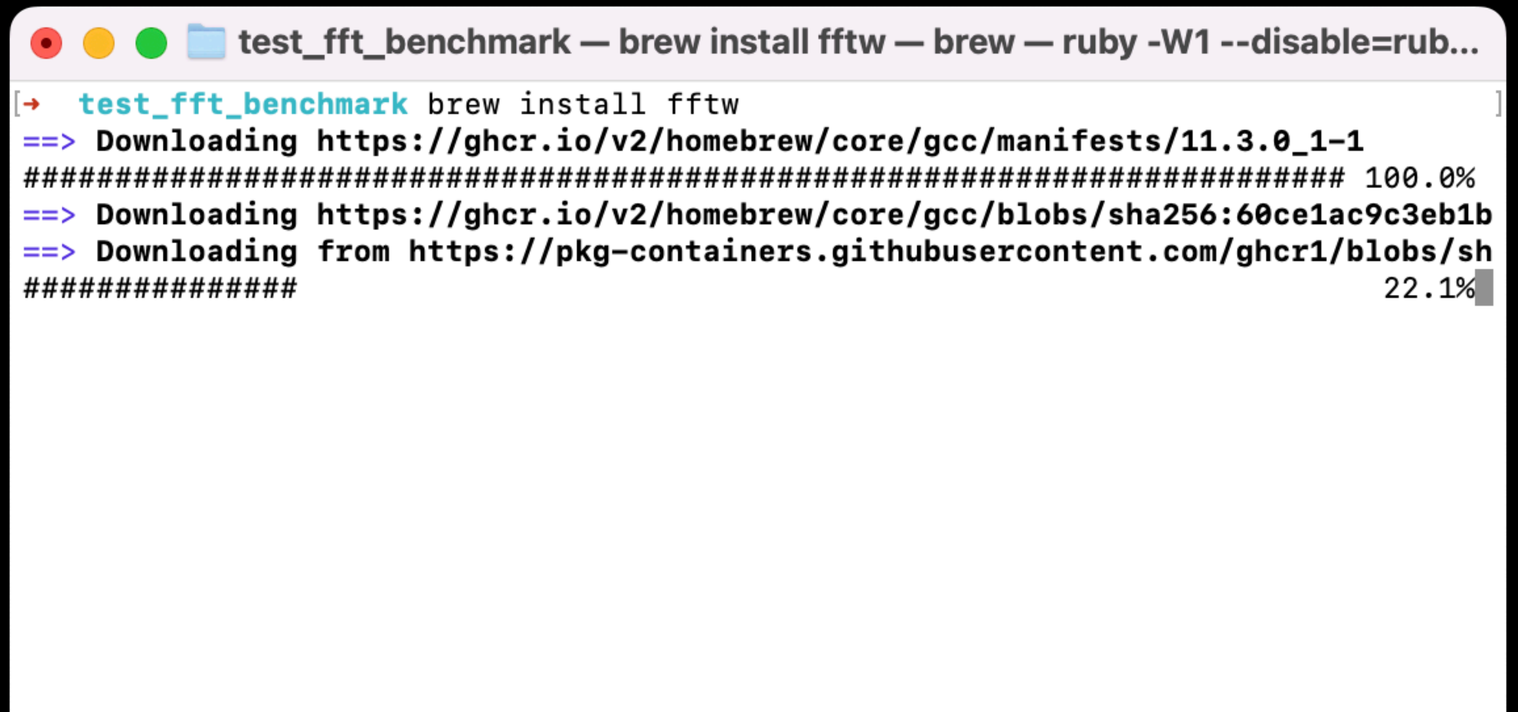
Stack Overflow to the rescue



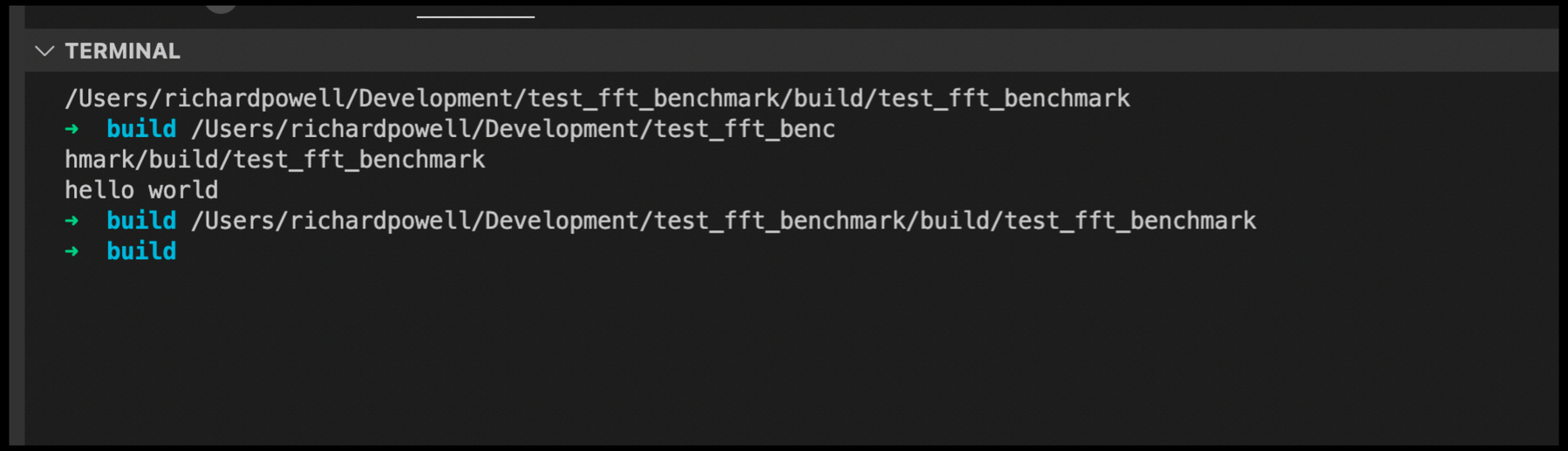


Oops, need to make sure I've got it installed…

Oops, need to make sure I've got it installed…



Well, that’s not interesting…

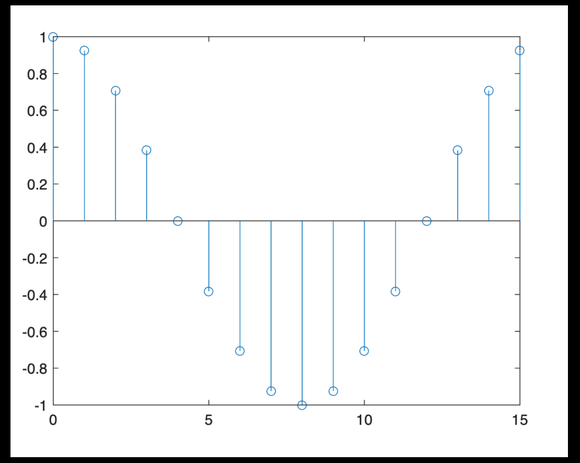


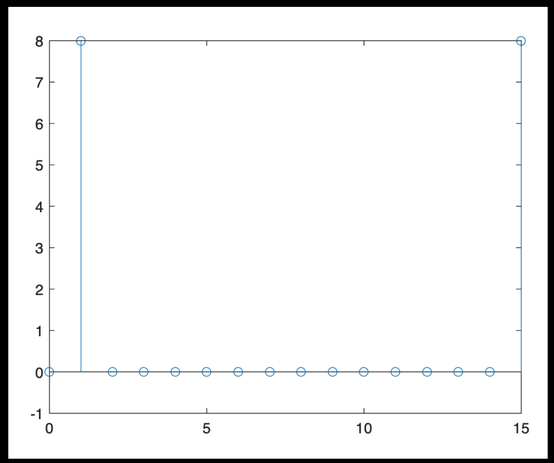
Cosine!

test_fft_benchmark.cpp > main()

```cpp
#include <iostream>
#include <numbers>
#include <algorithm>
#include <math.h>
#include <fftw3.h>

int main() {
  auto N = 1 << 4;
  fftw_complex *in, *out;
  fftw_plan p;
  in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);

  for (auto i = 0; i < N; ++i) {
    in[i][0] = cos(i * 2 * std::numbers::pi/N);
  }

  std::cout<<"Input: ";
  for (auto i = 0; i < N; ++i) {
    std::cout<<"("<<in[i][0]<<","<<in[i][1]<<")";
  }
  std::cout<<"\n";

  out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
  p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);

  fftw_execute(p);

  std::cout<<"\nOutput: ";
  for (auto i = 0; i < N; ++i) {
    std::cout<<"("<<out[i][0]<<","<<out[i][1]<<")";
  }
  std::cout<<"\n";

  fftw_destroy_plan(p);
  fftw_free(in);
  fftw_free(out);
}
```

```
PROBLEMS  OUTPUT  TERMINAL

TERMINAL
/Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
➜  build /Users/richardpowell/Development/test_fft_benc
hmark/build/test_fft_benchmark
hello world
➜  build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
➜  build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
➜  build /Users/richardpowell/Development/test_fft_benchmark/build/test_fft_benchmark
Input: (1,0)(0.92388,0)(0.707107,0)(0.382683,0)(6.12323e-17,0)(-0.382683,0)(-0.707107,0)(-0.92388,0)(-1,0)(-0.92388,0)(-0.707107,0)(-0.382683,0)(-1.83697e-16,0)(0.3
(0.707107,0)(0.92388,0)

Output: (-8.99621e-16,0)(8,-1.08026e-15)(4.39601e-17,4.71028e-16)(-9.05066e-16,-6.90891e-17)(9.95799e-17,-8.88178e-16)(3.66477e-16,6.90891e-17)(2.00969e-16,4.71028e
.59171e-17)(2.10602e-16,0)(0,-3.76406e-17)(2.00969e-16,-4.71028e-16)(3.58201e-16,-6.90891e-17)(9.95799e-17,8.88178e-16)(-9.13342e-16,6.90891e-17)(4.39601e-17,-4.710
8,1.07198e-15)
➜  build
```





Hey, this is a C++ talk!

```cpp
struct FFT {
  using complex = std::complex<double>;
  FFT(int log2n)
    : log2n(log2n)
    , N(1 << log2n)
    , in(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
    , out(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
    , p(fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE)) {

  }

  ~FFT() {
    fftw_destroy_plan(p);
    fftw_free(in);
    fftw_free(out);
  }

  void fill_input(std::vector<complex> const& input) {
    std::copy(input.begin(), input.end(), reinterpret_cast<complex*>(in));
  }

  std::vector<complex> fetch_output() const {
    auto output = reinterpret_cast<complex*>(out);
    return { output, output + N };
  }

  void execute() {
    fftw_execute(p);
  }

  int log2n{};
  int N{};
  fftw_complex *in{};
  fftw_complex *out{};
  fftw_plan p{};
};
```

```cpp
int main() {
  auto fft = FFT(4);

  auto in = std::vector<FFT::complex>(fft.N);
  for (auto i = 0; i < fft.N; ++i) {
    in[i] = cos(i * 2 * std::numbers::pi/fft.N);
  }

  std::cout<<"Input: ";
  std::copy(in.begin(), in.end(), std::ostream_iterator<FFT::complex>(std::cout, ", "));
  std::cout<<"\n";

  fft.fill_input(in);
  fft.execute();
  auto out = fft.fetch_output();

  std::cout<<"\nOutput: ";
  std::copy(out.begin(), out.end(), std::ostream_iterator<FFT::complex>(std::cout, ", "));
  std::cout<<"\n";

}
```

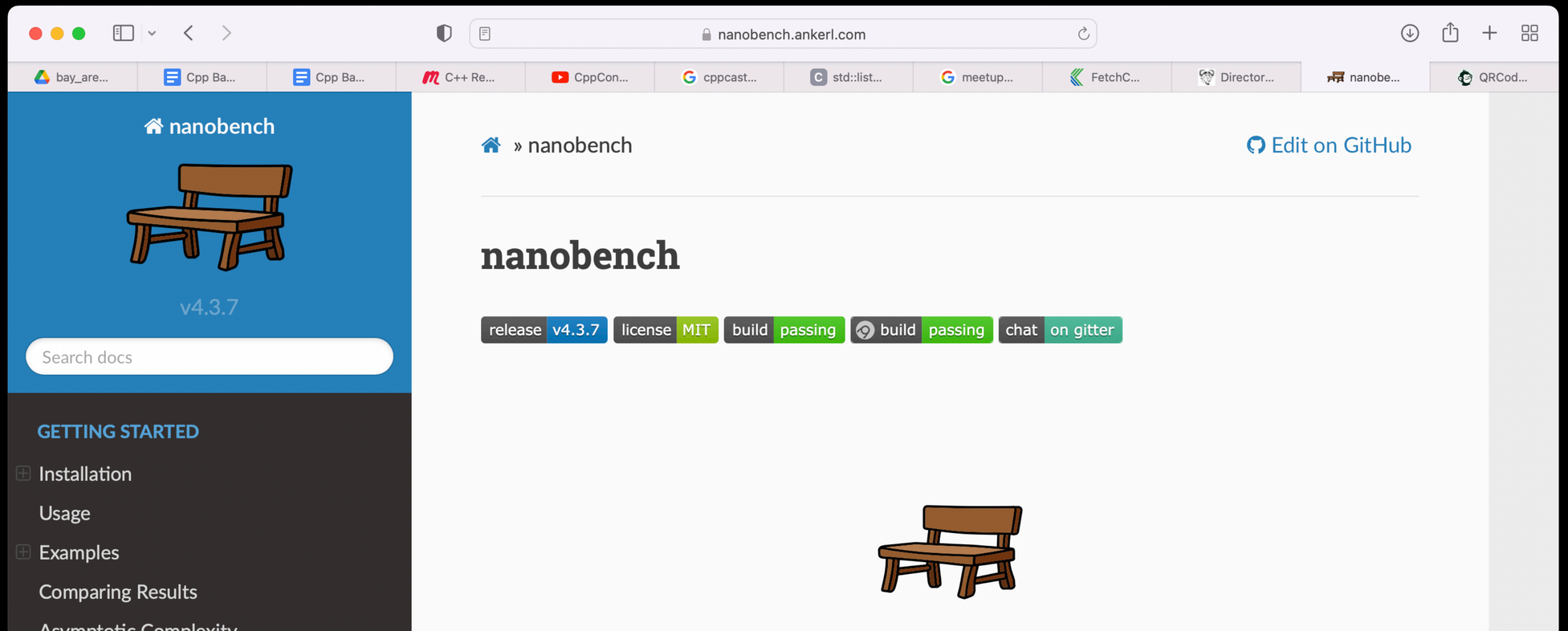
Now some benchmarks

Posted by u/SubstantialRange 7 months ago

59

Nanobench: Fast, Accurate, Single-Header Microbenchmarking Functionality For C++

github.com/martin...



Integrate with nanobench

CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.23)
2
3   set (CMAKE_CXX_STANDARD 20)
4
5   find_package(PkgConfig REQUIRED)
6   pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7   include_directories(PkgConfig::FFTW)
8   link_libraries     (PkgConfig::FFTW)
9
10  project(
11    test_fft_benchmark
12    VERSION 1.0
13    LANGUAGES CXX)
14
15  add_executable(test_fft_benchmark test_fft_benchmark.cpp)
16
```

```
1   cmake_minimum_required(VERSION 3.23)
2
3   set (CMAKE_CXX_STANDARD 20)
4
5   find_package(PkgConfig REQUIRED)
6   pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7   include_directories(PkgConfig::FFTW)
8   link_libraries     (PkgConfig::FFTW)
9
10  include(FetchContent)
11
12  FetchContent_Declare(
13      nanobench
14      GIT_REPOSITORY https://github.com/martinus/nanobench.git
15      GIT_TAG v4.1.0
16      GIT_SHALLOW TRUE)
17
18  FetchContent_MakeAvailable(nanobench)
19
20  project(
21    test_fft_benchmark
22    VERSION 1.0
23    LANGUAGES CXX)
24
25  add_executable(test_fft_benchmark test_fft_benchmark.cpp)
26  target_link_libraries(test_fft_benchmark PRIVATE nanobench)
```

Integrate with nanobench

```cpp
int main() {
  {…

  for (auto i : { 7, 8, 9, 10, 11, 12 }) {
    auto fft = FFT(i);
    auto in = std::vector<FFT::complex>(fft.N);
    for (auto i = 0; i < fft.N; ++i) {
      in[i] = cos(i * 2 * std::numbers::pi/fft.N);
    }
    ankerl::nanobench::Bench().run(std::string("fftw ") + std::to_string(fft.N) , [&] {
        fft.execute();
    });
  }
}
```

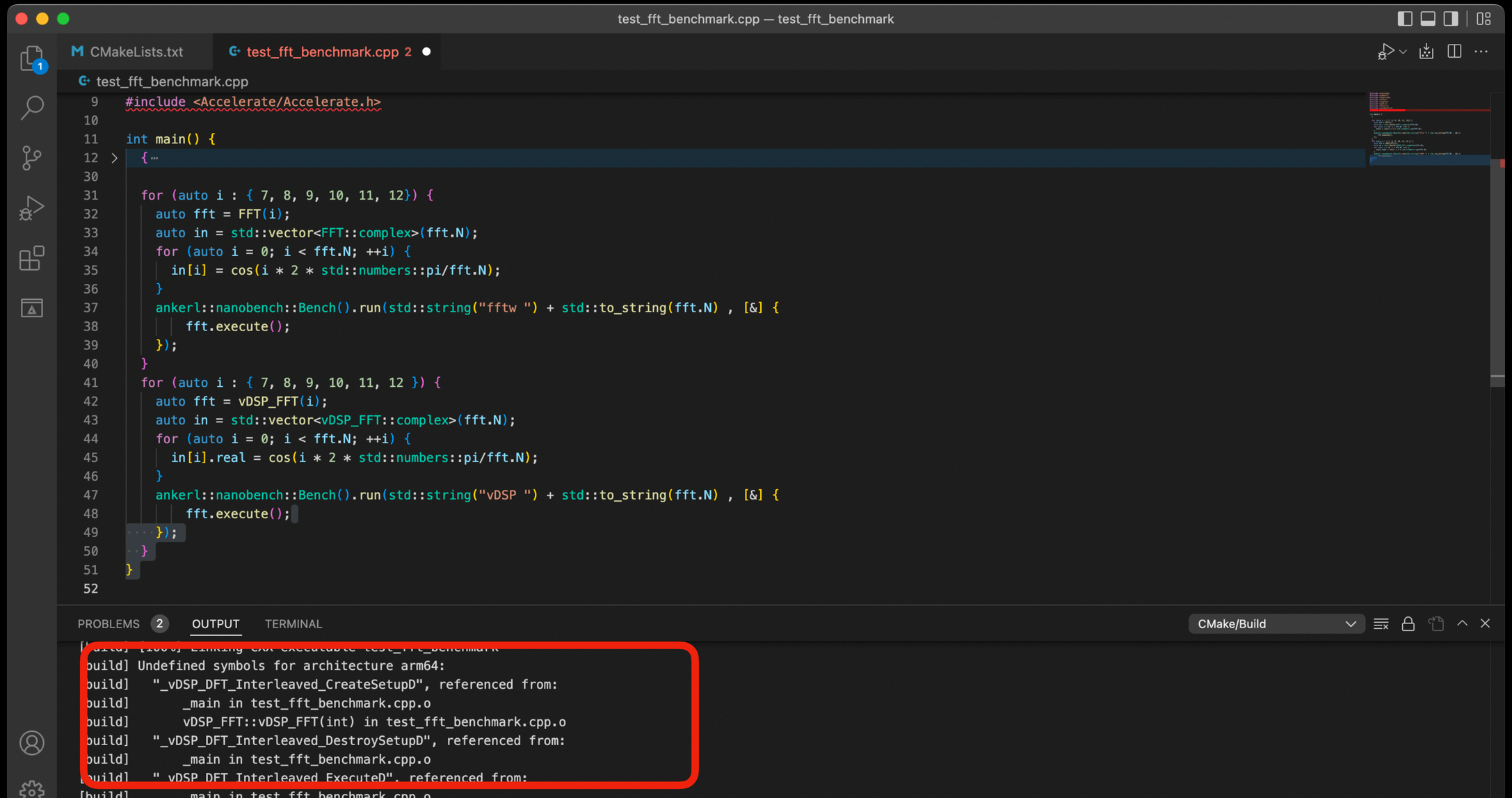
ns/op	op/s	err%	total	benchmark
245.86	4,067,377.93	0.1%	0.00	`fftw 128`
575.71	1,736,993.73	0.1%	0.00	`fftw 256`
1,304.05	766,839.38	0.1%	0.00	`fftw 512`
3,016.67	331,491.71	0.0%	0.00	`fftw 1024`
6,541.71	152,865.13	0.1%	0.00	`fftw 2048`
14,763.67	67,733.85	0.4%	0.00	`fftw 4096`

Ok, how about the other guy?

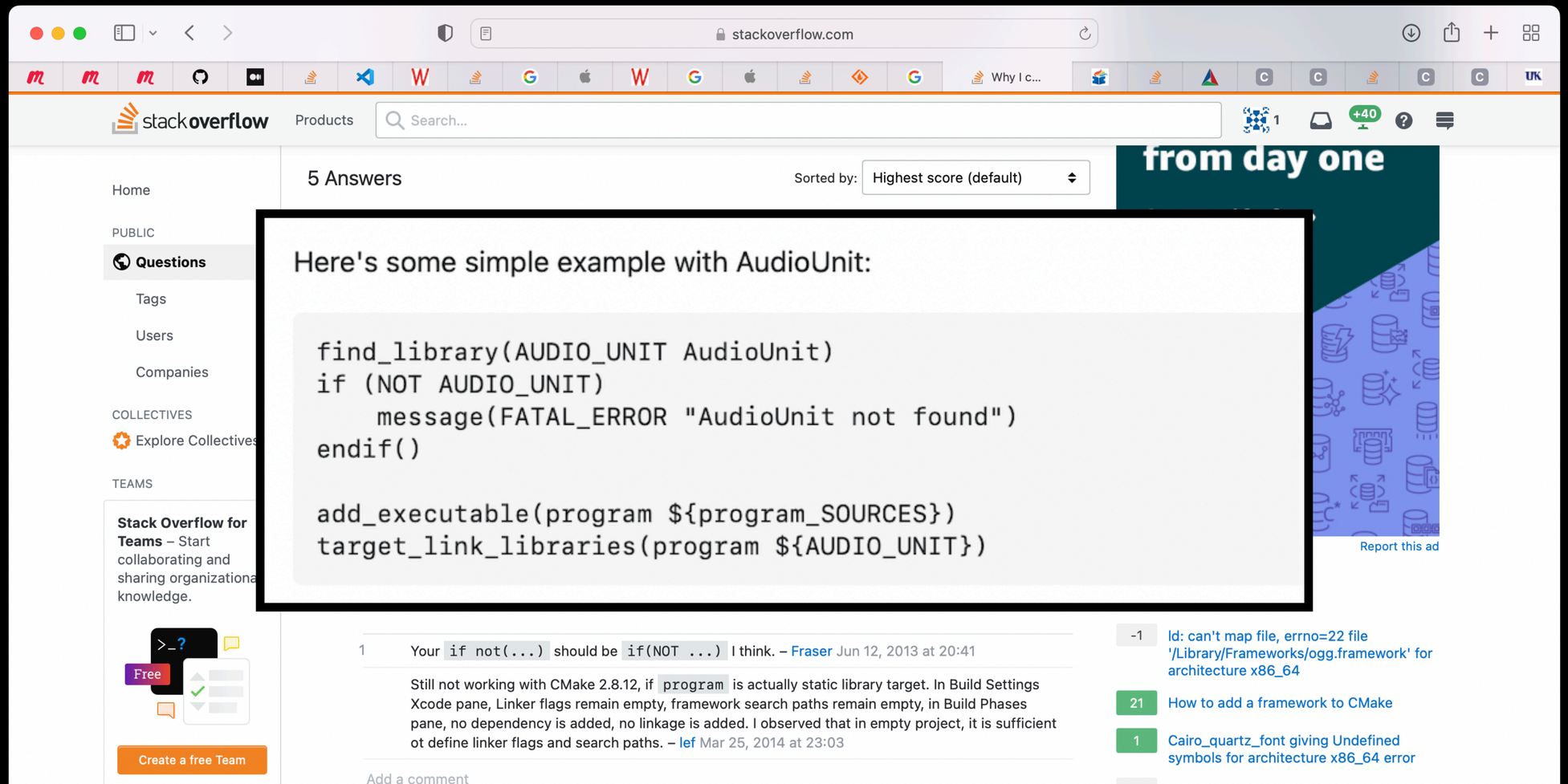
```cpp
struct FFT {
  using complex = std::complex<double>;
  FFT(int log2n)
    : log2n(log2n)
    , N(1 << log2n)
    , in(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
    , out(static_cast<fftw_complex *>(fftw_malloc(sizeof(fftw_complex) * N)))
    , p(fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE)) {

  }

  ~FFT() {
    fftw_destroy_plan(p);
    fftw_free(in);
    fftw_free(out);
  }

  void fill_input(std::vector<complex> const& input) {
    std::copy(input.begin(), input.end(), reinterpret_cast<complex*>(in));
  }

  std::vector<complex> fetch_output() const {
    auto output = reinterpret_cast<complex*>(out);
    return { output, output + N };
  }

  void execute() {
    fftw_execute(p);
  }

  int log2n{};
  int N{};
  fftw_complex *in{};
  fftw_complex *out{};
  fftw_plan p{};
};
```

```cpp
struct vDSP_FFT {
  using complex = DSPDoubleComplex;
  vDSP_FFT(int log2n)
    : log2n(log2n)
    , N(1 << log2n)
    , in(N)
    , out(N)
    , setup(vDSP_DFT_Interleaved_CreateSetupD(nullptr, N, vDSP_DFT_FORWARD, vDSP_DFT_Interleaved_ComplextoComplex))
  }

  ~vDSP_FFT() {
    vDSP_DFT_Interleaved_DestroySetupD(setup);
  }

  void fill_input(std::vector<complex> const& input) {
    std::copy(input.begin(), input.end(), in.begin());
  }

  std::vector<complex> fetch_output() const {
    return out;
  }

  void execute() {
    vDSP_DFT_Interleaved_ExecuteD(setup, in.data(), out.data());
  }

  int log2n{};
  int N{};
  std::vector<DSPDoubleComplex> in;
  std::vector<DSPDoubleComplex> out;
  vDSP_DFT_Interleaved_SetupD setup;
};
```

No workie…


test_fft_benchmark.cpp — test_fft_benchmark
CMakeLists.txt
test_fft_benchmark.cpp 2
test_fft_benchmark.cpp
#include <Accelerate/Accelerate.h>
int main() {
for (auto i : { 7, 8, 9, 10, 11, 12}) {
auto fft = FFT(i);
auto in = std::vector<FFT::complex>(fft.N);
for (auto i = 0; i < fft.N; ++i) {
in[i] = cos(i * 2 * std::numbers::pi/fft.N);
}
ankerl::nanobench::Bench().run(std::string("fftw ") + std::to_string(fft.N) , [&] {
fft.execute();
});
}
for (auto i : { 7, 8, 9, 10, 11, 12 }) {
auto fft = vDSP_FFT(i);
auto in = std::vector<vDSP_FFT::complex>(fft.N);
for (auto i = 0; i < fft.N; ++i) {
in[i].real = cos(i * 2 * std::numbers::pi/fft.N);
}
ankerl::nanobench::Bench().run(std::string("vDSP ") + std::to_string(fft.N) , [&] {
fft.execute();
});
}
}
PROBLEMS 2
OUTPUT
TERMINAL
CMake/Build
build] Undefined symbols for architecture arm64:
build] "_vDSP_DFT_Interleaved_CreateSetupD", referenced from:
build] _main in test_fft_benchmark.cpp.o
build] vDSP_FFT::vDSP_FFT(int) in test_fft_benchmark.cpp.o
build] "_vDSP_DFT_Interleaved_DestroySetupD", referenced from:
build] _main in test_fft_benchmark.cpp.o

Stack Overflow to the rescue again!



Add Accelerate to cmake

```
1  cmake_minimum_required(VERSION 3.23)
2
3  set (CMAKE_CXX_STANDARD 20)
4
5  find_package(PkgConfig REQUIRED)
6  pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7  include_directories(PkgConfig::FFTW)
8  link_libraries     (PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 FetchContent_Declare(
13     nanobench
14     GIT_REPOSITORY https://github.com/martinus/nanobench.git
15     GIT_TAG v4.1.0
16     GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 project(
21   test_fft_benchmark
22   VERSION 1.0
23   LANGUAGES CXX)
24
25 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
26 target_link_libraries(test_fft_benchmark PRIVATE nanobench)
27
28
```

```
1  cmake_minimum_required(VERSION 3.23)
2
3  set (CMAKE_CXX_STANDARD 20)
4
5  find_package(PkgConfig REQUIRED)
6  pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
7  include_directories(PkgConfig::FFTW)
8  link_libraries     (PkgConfig::FFTW)
9
10 include(FetchContent)
11
12 FetchContent_Declare(
13     nanobench
14     GIT_REPOSITORY https://github.com/martinus/nanobench.git
15     GIT_TAG v4.1.0
16     GIT_SHALLOW TRUE)
17
18 FetchContent_MakeAvailable(nanobench)
19
20 project(
21   test_fft_benchmark
22   VERSION 1.0
23   LANGUAGES CXX)
24
25 find_library(ACCELERATE_LIB Accelerate)
26 add_executable(test_fft_benchmark test_fft_benchmark.cpp)
27 target_link_libraries(test_fft_benchmark PRIVATE nanobench ${ACCELERATE
28
```

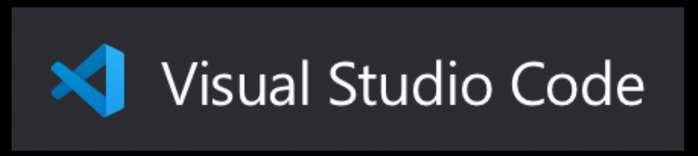
And Viola!

```
cmake_minimum_required(VERSION 3.23)

set (CMAKE_CXX_STANDARD 20)

find_package(PkgConfig REQUIRED)
pkg_search_module(FFTW REQUIRED fftw3 IMPORTED_TARGET)
include_directories(PkgConfig::FFTW)
link_libraries     (PkgConfig::FFTW)

include(FetchContent)

FetchContent_Declare(
    nanobench
    GIT_REPOSITORY https://github.com/martinus/nanobench.git
    GIT_TAG v4.1.0
    GIT_SHALLOW TRUE)

FetchContent_MakeAvailable(nanobench)

project(
  test_fft_benchmark
  VERSION 1.0
  LANGUAGES CXX)

find_library(ACCELERATE_LIB Accelerate)
add_executable(test_fft_benchmark test_fft_benchmark.cpp)
target_link_libraries(test_fft_benchmark PRIVATE nanobench ${ACCELERATE_LIB})

```

ns/op	op/s	err%	total	benchmark
246.35	4,059,256.78	0.3%	0.00	`fftw 128`
582.77	1,715,933.01	1.1%	0.00	`fftw 256`
1,383.33	722,891.57	2.3%	0.00	`fftw 512`
3,097.22	322,869.96	2.5%	0.00	`fftw 1024`
6,553.57	152,588.56	1.9%	0.00	`fftw 2048`
14,722.33	67,924.02	1.1%	0.00	`fftw 4096`
199.68	5,007,957.56	0.4%	0.00	`accelerate 128`
459.70	2,175,316.14	0.4%	0.00	`accelerate 256`
1,251.24	799,210.19	0.6%	0.00	`accelerate 512`
2,816.67	355,029.59	1.0%	0.00	`accelerate 1024`
5,785.71	172,839.51	0.2%	0.00	`accelerate 2048`
13,291.75	75,234.64	1.0%	0.00	`accelerate 4096`

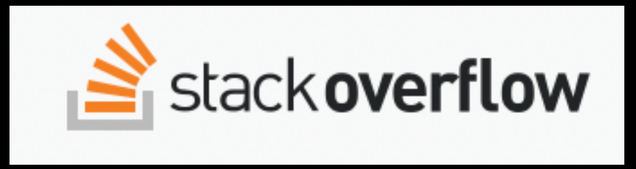
ns/op	op/s	err%	total	benchmark
246.35	4,059,256.78	0.3%	0.00	`fftw 128`
582.77	1,715,933.01	1.1%	0.00	`fftw 256`
1,383.33	722,891.57	2.3%	0.00	`fftw 512`
3,097.22	322,869.96	2.5%	0.00	`fftw 1024`
6,553.57	152,588.56	1.9%	0.00	`fftw 2048`
14,722.33	67,924.02	1.1%	0.00	`fftw 4096`
199.68	5,007,957.56	0.4%	0.00	`accelerate 128`
459.70	2,175,316.14	0.4%	0.00	`accelerate 256`
1,251.24	799,210.19	0.6%	0.00	`accelerate 512`
2,816.67	355,029.59	1.0%	0.00	`accelerate 1024`
5,785.71	172,839.51	0.2%	0.00	`accelerate 2048`
13,291.75	75,234.64	1.0%	0.00	`accelerate 4096`


CMake


Visual Studio Code


Homebrew
The Missing Package Manager for macOS (or Linux)


nanobench


stackoverflow