



zapcc is a caching C++ compiler based on clang, designed to perform faster compilations. zapcc uses in-memory compilation cache in client-server architecture, remembering all compilation information between runs. zapcc is the client while zapccs is the server. Each zapcc run will reuse an existing server or if none was available will start a new one.

License

This open source release is licensed under the LLVM Release License (University of Illinois/NCSA).

Which operating systems and compilers are supported?

zapcc builds on * Linux x64 using gcc, clang or zapcc * Windows using Visual C++ or mingw-w64, targeting mingw-w64, 32 or 64 bits * Targetting Visual C++ binaries or using `zapcc-cl` is not supported * On MacOS using clang

zapcc was thoroughly tested on Linux x64 targeting Linux x64 and minimally on Windows. Rest are experimental, please share your experience.

Building

The prerequisites and build process are identical to building LLVM.

```
1 sudo apt-get install ninja-build
2 git clone https://github.com/yarnkrn/zapcc.git llvm
3 mkdir build
4 cd build
5 cmake -G Ninja -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_WARNINGS=OFF
  ../llvm
6 ninja
```

How to target mingw-w64 on Windows

You need msys2 and the mingw-builds of mingw-w64. Note there are 32- and 64- bits distributions of mingw-w64. To target x86_64:

- Download the latest MSYS2 installer WebKit and install into the default folder `C:\msys64\`
- Download one of the mingw-builds personal distributions, such as x86_64-8.1.0-release-posix-seh-rt_v6-rev0.7z and open into the folder `C:\mingw64\`

-
- Add the bin directories to the PATH, `C:\msys64\usr\bin` and `C:\mingw64\bin`
 - Make sure you have just this gcc version available on the PATH. gcc versions outside the PATH are OK.
 - Either Visual C++ or the just-installed mingw-w64 may be used to build zapcc.
 - If building using Visual C++, target `x86_64-pc-windows-gnu` must be explicitly specified, `cmake -G Ninja -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_WARNINGS=OFF -DLLVM_DEFAULT_TARGET_TRIPLE=x86_64-pc-windows-gnu ../llvm`
 - zapcc will now target mingw-w64 and ignore Visual C++, even if installed.

To target mingw-builds 32 bits, download the appropriate 32 bits distribution of mingw-builds and replace `x86_64` with `i686` in the configuration.

Running the tests

```
1 ninja check-all
```

Using

zapcc command syntax is identical to clang with the command being zapcc.

```
1 zapcc ...
```

Killing the zapccs server

```
1 pkill zapcc
```

To kill the zapccs server to free memory or replace with newly-built zapcc

FAQ

What is the typical acceleration of Zapcc?

Full builds are 2x-5x faster, see * ETL * MKVToolNix * A Performance-Based Comparison of C/C++ Compilers

Typically re-compilation of one modified source file is 10x-50x faster.

Acceleration depends on the complexity of the header files vs. the complexity of the source files. It can range from no acceleration at all for plain C projects where caching is disabled to x2-x5 for build-all of heavily templated projects, up to cases of x50 speedups in developer-mode incremental change of one source file. As a reference number, Zapcc builds the LLVM `build-all` target about x2 faster compared to building LLVM using clang.

Here are ASCII movies comparing clang and zapcc fully building WebKit and incremental building Boost.

Is Zapcc Clang compatible?

Yes, zapcc is based on heavily-modified clang code.

Is Zapcc GCC compatible?

Yes, to the extent clang is gcc compatible.

How zapcc works?

See CATC 2017 presentation and discussion at cfe-dev.

Is zapcc different from precompiled headers?

Precompiled headers requires building your project to the exact precompiled headers rules. Most projects do not bother with using precompiled headers. Even then, precompiled headers do not cache as much as zapcc. Zapcc works within your existing build.

Precompiled headers are currently ignored by Zapcc.

How zapcc is different from C++ modules?

As of C++17, modules are not standard, rarely used and do not support well legacy code and macros found in most existing C++ code, such as Boost. Modules require significant code refactoring in bottom-up approach everything or are slow. Even then, modules do not cache template instantiations and generated code that are specific to your code like zapcc does.

My project does not compile with Zapcc!

Please make sure first your project compiles successfully with Clang. If your project does not compile with Clang, Zapcc, being based on Clang, will not be able to compile any more than clang.

Are the sanitizers supported?

No.

How much memory does Zapcc use?

To avoid killing the server by using endless memory, Zapcc server has a memory limit and will automatically reset after reaching it, restarting with an empty cache and low memory usage. The memory limit is set under [MaxMemory] at bin/zapccs.config, and you can change it to optimize memory usage and the number of servers you plan to use. Usually, you should not set the -j parameter to more than the number of physical cores + 2. This is especially important for Intel CPU with hyper-threading enabled, which report twice the number of physical cores. In such cases, Zapcc may run faster with fewer servers, each using a higher memory limit.

Does it use ccache, distcc, warp or the like?

No.

Where is the zapcc code?

There are patches all around LLVM & clang. Additional zapcc-only code in

```
1 tools/zapcc
2 tools/zapccs
3 tools/clang/test/zapcc
```

When was the source last merged with LLVM trunk?

This open-source release was last merged with LLVM 325000 on 2018-02-13.