Subject: Precompiled headers
Posted by mirek on Sun, 18 Jan 2015 10:49:42 GMT
View Forum Message <> Reply to Message

I have just finished my take on precompiled headers support in U++. Big thanks to Shire for giving me hints how to do that... (and sorry that my approach in the end is different) Here is docs excerpt:

Precompiled headers is a compiler technique trying to solve the very same problem as BLITZ. In general, we have found BLITZ faster than any precompiled header use, however BLITZ tends to have one disadvantage: by combining all files into single object file, linker has less oportunity to remove unused code. This leads to (sometimes significantly) larger executable binaries. For this reason, we do not recommend (and have off by default) BLITZ for release builds and if possible, we use precompiled headers for release builds.

Precompiled headers have a set of its own problems. Notably, Microsoft C++ precompiled headers are hard to use with multiple processes building the code (Hydra) in debug mode. Also, precompiled headers in general are very bulky files, easily surpasing 100MB, which is a problem as we need to have single precompiled header per package.

For these reasons precompiled headers support works like this:

* Precompiled headers are activated only in release mode without blitz.
* You have to set "Precompile header" flag on header files candidates for precompilation. Only single candidate per package is allowed. Note that not all headers can be with this system. Header has to have include guards and it must be possible for all files in the package to include it first before all other headers.
* Build method has to have "Allow precompiled headers" set.
* When package is build, it first checks whether using precompiled header is possible (as per rules above). Then it checks how many files are to be rebuild. If there are 3 or more files to build, U++ precompiles the header and uses it to build the package. When the package is built, U++ deletes the header to conserve the space.

U++ supports precompiled headers for MSC, GCC and CLANG. However, practical benchmarks show that with CLANG using precompiled headers actually leads to worse compilation times.

Additional note: They work, they reduce the time of release builds (except CLANG), but results are not spectactular. E.g. IDE gets build in 3:07 with GCC (in Linux) without PCH, which is reduced to 2:19 with precompiled headers.... (meanwhile, debug mode BLITZ requires just 47 seconds, 29 with CLANG).

Subject: Re: Precompiled headers
Posted by cbpporter on Mon, 26 Jan 2015 12:05:29 GMT
View Forum Message <> Reply to Message

Hmmm, interesting.

Adding PCH support for an existing project is very fiddly. I guess the fact that CPP files were including the main package header helped a lot in the transition.

I have very positive experience with PCH. I don't care about full release mode build time, but iterative development time is helped a lot by PCH. On the other hand, if you modify a header, you can go grab something to eat...

Subject: Re: Precompiled headers
Posted by chickenk on Tue, 03 Feb 2015 19:47:32 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 18 January 2015 11:49by combining all files into single object file, linker has less oportunity to remove unused code. This leads to (sometimes significantly) larger executable binaries. For this reason, we do not recommend (and have off by default) BLITZ for release builds and if possible, we use precompiled headers for release builds.
At least for GCC, your experiments would lead me to use BLITZ for release builds, with the following options:
-ffunction-sections -fdata-sections -Wl,--gc-sections
Best of BLITZ, with much more unused code removed!

Anyway thanks for working on PCH.

Subject: Re: Precompiled headers
Posted by mirek on Wed, 04 Feb 2015 07:33:00 GMT
View Forum Message <> Reply to Message

chickenk wrote on Tue, 03 February 2015 20:47mirek wrote on Sun, 18 January 2015 11:49by combining all files into single object file, linker has less oportunity to remove unused code. This leads to (sometimes significantly) larger executable binaries. For this reason, we do not recommend (and have off by default) BLITZ for release builds and if possible, we use precompiled headers for release builds.
At least for GCC, your experiments would lead me to use BLITZ for release builds, with the following options:
-ffunction-sections -fdata-sections -Wl,--gc-sections
Best of BLITZ, with much more unused code removed!

Anyway thanks for working on PCH.

Well, at the time I was testing these options, it did not make noticeable impact on code size (in short, they did not appeared to work). BTW, note that these options are part of standard installation.

You can check: build release (e.g. theide) with BLITZ and these options, then build without BLITZ, report the size...