Clang

From Wikipedia, the free encyclopedia

Clang /ˈklæŋ/[3] is a compiler front end for the C, C++, Objective-C, Objective-C++, OpenMP,[4] OpenCL, and CUDA programming languages. It uses LLVM as its back end and has been part of the LLVM release cycle since LLVM 2.6.

It is designed to offer a complete replacement to the GNU Compiler Collection (GCC). It is open-source,[5] and its contributors include Apple, Microsoft, Google, ARM, Sony, Intel and AMD. Its source code is available under the University of Illinois/NCSA License, a permissive free software licence.

The Clang project includes the Clang front end and the Clang static analyzer and several code analysis tools.[6]

Clang

Original author(s)	Chris Lattner and others
Developer(s)	Apple Inc. and others
Stable release	3.8.0[1] / 8 March 2016
Development status	Active
Written in	C++
Operating system	Unix-like
Platform	Cross-platform
Type	Compiler
License	University of Illinois/NCSA Open Source License[2]
Website	clang.llvm.org (http://clang.llvm.org)

Contents

Background

Starting in 2005, Apple has made extensive use of LLVM in a number of commercial systems,[7] including the iPhone development kit and Xcode 3.1.

One of the first uses of LLVM was an OpenGL code compiler for OS X that converts OpenGL calls into more fundamental calls for graphics processing units (GPU) that do not support certain features. This allowed Apple to support the entire OpenGL application programming interface (API) on computers using Intel Graphics Media Accelerator (GMA) chipsets, increasing performance on those machines.[8] For sufficiently capable GPUs, the code is compiled to take full advantage of the underlying hardware, but on GMA machines, LLVM compiles the same OpenGL code into subroutines to ensure it continues to work properly.

LLVM was originally intended to use GCC's front end, but GCC turned out to cause some problems for both the LLVM developers and Apple. The GCC source code is a large and somewhat cumbersome system for developers to work with; as one long-time GCC developer put it, "Trying to make the hippo dance is not really a lot of fun".[9]

Apple software makes heavy use of Objective-C, but the Objective-C front-end in GCC is a low priority for the current GCC developers. Also, GCC does not fit smoothly into Apple's IDE.[10] Finally, GCC is GPL version 3 licensed, which requires developers who distribute extensions for (or modified versions of) GCC to make their

source code available, whereas LLVM has a BSD-like license[11] which permits including the source in proprietary software.

Apple chose to develop a new compiler front end from scratch, supporting C, Objective-C and C++.[10] This "clang" project was open-sourced in July 2007.[12]

Design

Clang is intended specifically to work on top of LLVM.[11] The combination of Clang and LLVM provides the majority of a toolchain, allowing the replacement of the whole GCC stack. Because it is built with a library-based design, like the rest of LLVM, Clang is easy to embed into other applications. This is one reason why a majority of the OpenCL implementations are built with Clang and LLVM.

One of Clang's primary goals is to provide a library-based architecture [1] (http://clang.llvm.org/features.html#libraryarch), in order to allow the compiler to be more tightly tied to tools that interact with source code, such as an IDE GUI. In contrast, GCC is designed to work in a "classic" compile-link-debug cycle, and integrating it with other tools is not always easy. For instance, GCC uses a step called "fold" that is key to the overall compile process, which has the side effect of translating the code tree into a form that does not look very much like the original source code. If an error is found during or after the fold step, it can be difficult to translate that back into a single location in the original source. Additionally, vendors using the GCC stack within IDEs use separate tools to index the code, to provide features like syntax highlighting and autocomplete.

Clang is designed to retain more information during the compilation process than GCC, and preserve the overall form of the original code. The objective of this is to make it easier to map errors back into the original source. The error reports offered by Clang are also aimed to be more detailed and specific, as well as machine-readable, so IDEs can index the output of the compiler during compilation. Modular design of the compiler can offer source code indexing, syntax checking, and other features normally associated with rapid application development systems. The parse tree is also more suitable for supporting automated code refactoring, as it directly represents the original source code.

Clang is a compiler only for C and C-like languages, including C, C++, Objective-C, Objective-C++, OpenCL, and CUDA. For other languages, including Java, Fortran, and Ada, LLVM remains dependent on GCC or another compiler frontend. In many cases, Clang can be used or swapped out for GCC as needed, with no other effects on the toolchain as a whole. It supports most of the commonly used GCC options.

Performance and GCC compatibility

Clang is designed to be highly compatible with GCC.[13] Clang's command-line interface is similar to and shares many flags and options with GCC. Clang implements many GNU language extensions and enables them by default. Clang implements many GCC compiler intrinsics purely for compatibility. For example, even though Clang implements atomic intrinsics which correspond exactly with C11 atomics, it also implements GCC's __sync_* intrinsics for compatibility with GCC and libstdc++. Clang also maintains ABI compatibility with GCC-generated object code. In practice Clang can often be used as a drop-in replacement for GCC.

Clang's developers aim to reduce memory footprint and increase compilation speed compared to competing compilers, such as GCC. In October 2007, they report that Clang compiled the Carbon libraries well over twice as fast as GCC, while using about one-sixth GCC's memory and disk space.[14] However, as of 2011 this was not a typical result.[15][16] As of the middle of 2014, runtime performance has improved to the point where Clang wins more than a third of the benchmarks, with GCC winning a majority.[17]

While there are still a few tests where performance of a Clang-compiled program lags behind performance of the GCC-compiled program, by large factors (up to 5.5x),[17] it has been reported that Clang "continues to be under very active development," and a hope was expressed for further improvement.[15]

Status history

This table presents only significant steps and releases in Clang history.

Date	Highlights
11 July 2007	Clang frontend released under an open-source license
25 February 2009	Clang/LLVM able to compile a working FreeBSD kernel.[18][19]
16 March 2009	Clang/LLVM able to compile a working DragonFly BSD kernel.[20][21]
23 October 2009	Clang 1.0 released along with LLVM 2.6 for the first time.
December 2009	Code generation for C and Objective-C reach production quality (support for C++ and Objective-C++ still incomplete). Clang C++ able to parse GCC 4.2 libstdc++ and generate working code for non-trivial programs[11] and was able to compile itself[22]
2 February 2010	Clang self-hosting.[23]
20 February 2010	The source code of HelenOS was modified to successfully compile with Clang, and passed all kernel and user space regression tests on IA-32.[24]
20 May 2010	The latest version of Clang successfully built the Boost C++ libraries, and passed nearly all tests.[25]
10 June 2010	Clang/LLVM became an integral part of FreeBSD (The default compiler is still GCC)[26]
25 October 2010	Clang/LLVM able to compile a working modified Linux Kernel.[27]
January 2011	Preliminary work completed to support the draft C++0x standard, with a few of the draft's new features supported in the development version of Clang.[28][29]
10 February 2011	Clang able to compile a working HotSpot Java Virtual Machine[15]
19 January 2012	Clang becomes an optional component in NetBSD's cross-platform build system (GCC is still default)[30]
29 February 2012	Clang 3.0 able to rebuild 91.2% of the Debian archive[31]
29 February 2012	Clang becomes the default compiler in MINIX 3[32]
12 May 2012	Clang/LLVM is announced to replace GCC in FreeBSD[33]
5 November 2012	Clang becomes the default compiler in FreeBSD 10.x[34]
18 February 2013	Clang/LLVM able to compile a working modified Android Linux Kernel for Nexus 7.[35][36]
19 April 2013	Clang is C++11 feature complete.[37]
6 November 2013	Clang is C++14 feature complete.[38]
11 September 2014	Clang 3.5 is able to rebuild 94.3% of the Debian archive. The percentage of failures has dropped by 1.2% per release since January 2013, primary due to increased compatibility with GCC flags.[39]

See also

- LLDB

- Portable C Compiler

References

1. http://llvm.org/releases/
2. *LLVM License*, retrieved 2014-07-24
3. Christopher, Eric (3 July 2008). "simply wonder pronunciation of Clang". *LLVMdev* (Mailing list). Retrieved 2015-09-22.
4. "OpenMP Support". *LLVM Project Blog*. Retrieved 28 March 2016.
5. *Clang "Getting started" instructions*, Clang.llvm.org, retrieved 2012-09-18
6. "Clang Static Analyzer". LLVM. Retrieved 3 September 2009.
7. Treat, Adam (19 February 2005). "mkspecs and patches for LLVM compile of Qt4". *Qt4-preview-feedback* (Mailing list).
8. Lattner, Chris (25 May 2007). *LLVM for OpenGL and other stuff* (Slides). LLVM Developers' Meeting.
9. Zadeck, Kenneth (19 November 2005). "Re: LLVM/GCC Integration Proposal". *GCC development* (Mailing list).
10. Naroff, Steve (25 May 2007). *New LLVM C Front-end* (Slides). LLVM Developers' Meeting.
11. Clang team, *clang: a C language family frontend for LLVM (http://clang.llvm.org/)*
12. Lattner, Chris (11 July 2007). "New LLVM C front-end: "clang" ". *cfe-dev* (Mailing list).
13. *Clang - Features and Goals: GCC Compatibility*, 15 April 2013
14. *Clang - Features and Goals: Fast compiles and Low Memory Use*, October 2007
15. Simonis, Volker (10 February 2011). "Compiling the HotSpot VM with Clang". Archived from the original on 18 February 2011. Retrieved 13 February 2011. "*While the overall GCC compatibility is excellent and the compile times are impressive, the performance of the generated code is still lacking behind a recent GCC version.*"
16. "Benchmarking LLVM & Clang Against GCC 4.5". Phoronix. 21 April 2010. Retrieved 13 February 2011. "*Binaries from LLVM-GCC and Clang both struggled to compete with GCC 4.5.0 in the timed HMMer benchmark of a Pfam database search. LLVM-GCC and Clang were about 23% slower(...)Though LLVM / Clang isn't the performance champion at this point, both components continue to be under very active development and there will hopefully be more news to report in the coming months*"
17. "GCC 4.9 VS. LLVM CLANG 3.5 LINUX COMPILER BENCHMARKS". OpenBenchmarking.org. 14 April 2014. Retrieved 25 June 2014.
18. Divacky, Roman. "[ANNOUNCE] clang/llvm can compile booting FreeBSD kernel on i386/amd64".
19. *Building FreeBSD with Clang*, Wiki.freebsd.org, 2012-08-24, retrieved 2012-09-18
20. Hornung, Alex. "llvm/clang once more".
21. *Clang*, DragonFlyBSD, retrieved 2012-09-18
22. "Clang can compile LLVM and Clang". LLVM Project Blog.
23. "Clang Successfully Self-Hosts". LLVM Project Blog.
24. "HelenOS mainline changeset head,294".
25. Gregor, Doug. "Clang++ Builds Boost!". LLVM Project Blog.
26. Davis, Brad. "FreeBSD Status Reports April - June, 2010".
27. *Clang builds a working Linux Kernel (Boots to RL5 with SMP, networking and X, self hosts)*, Lists.cs.uiuc.edu, retrieved 2012-09-18
28. Gregor, Douglas (26 January 2011). "New C++0x feature support in Clang" (Mailing list). Retrieved 29 January 2011.
29. "C++ and C++'0x Support in Clang". LLVM.
30. Sonnenberger, Jörg (2012-01-19). "Status of NetBSD and LLVM". Retrieved 2014-02-26.
31. Ledru, Sylvestre. "Rebuild of the Debian archive with clang".
32. "Official Minix 3 website: News".
33. Gerzo, Daniel (12 May 2012). "FreeBSD Quarterly Status Report January-March, 2012" (Mailing list). Retrieved 14 May 2012.
34. Davis, Brooks (5 November 2012). "HEADS UP: Clang now the default on x86" (Mailing list). Retrieved 7 November 2012.
35. Webster, Behan (18 February 2013). "LLVMLinux: Compiling Android with LLVM" (PDF). Retrieved 11 May 2013.
36. Tinti, Vinicius (17 March 2013). "LLVMLinux: Nexus 7". Retrieved 11 May 2013.
37. Du Toit, Stefanus. "Clang is C++11 feature complete as of *just now*!".
38. "[llvm-project] Revision 194194".
39. Ledru, Sylvestre. "Rebuild of Debian using Clang 3.5.0".

External links

- Official website (http://clang.llvm.org)
- LLVMdev: New LLVM C front-end: "clang" (http://lists.cs.uiuc.edu/pipermail/llvmdev/2007-July/009817.html), announcement (11 July 2007)
- Presentation: Ted Kremenek - Finding Bugs with the Clang Static Analyzer (http://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer_Hi.m4v), Slides (http://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer.pdf)
- Presentation: Steve Naroff - Clang Internals (http://llvm.org/devmtg/2008-08/Naroff_Clang_Hi.m4v), Slides (http://llvm.org/devmtg/2008-08/Naroff_Clang.pdf)
- 2009 DevMtg Clang presentation (http://llvm.org/devmtg/2009-10/StateOfClang.pdf)

Categories: Apple Inc. software | C compilers | C++ compilers | Free compilers and interpreters | Software using the NCSA license | Static program analysis tools

---