WIKIPEDIA

LLVM

LLVM is a set of compiler and toolchain technologies,[4] which can be used to develop a front end for any programming language and a back end for any instruction set architecture. LLVM is designed around a language-independent intermediate representation (IR) that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.[5]

LLVM is written in C++ and is designed for compile-time, link-time, run-time, and "idle-time" optimization. Originally implemented for C and C++, the language-agnostic design of LLVM has since spawned a wide variety of front ends: languages with compilers that use LLVM include ActionScript, Ada, C#,[6][7][8] Common Lisp, PicoLisp, Crystal, CUDA, D, Delphi, Dylan, Forth,[9] Fortran, Graphical G,[10] Halide, Haskell, Java bytecode, Julia, Kotlin, Lua, Objective-C, OpenCL,[11] PostgreSQL's SQL and PLpgSQL,[12] Ruby,[13] Rust, Scala,[14] Swift, XC,[15] Xojo and Zig.

LLVM

The LLVM logo, a stylized wyvern[1]

Original author(s)	Vikram Adve, Chris Lattner
Developer(s)	LLVM Developer Group
Initial release	2003
Stable release	13.0.0 / September 30, 2021[2]
Preview release	13.0.0-rc4 / September 24, 2021[2]
Repository	github.com/llvm/llvm-project (https://github.com/llvm/llvm-project)
Written in	C++
Operating system	Cross-platform
Type	Compiler
License	UIUC (BSD-style) Apache License 2.0 with LLVM Exceptions (v9.0.0 or later)[3]
Website	www.llvm.org

Contents

History

The LLVM project started in 2000 at the University of Illinois at Urbana–Champaign, under the direction of Vikram Adve and Chris Lattner. The name *LLVM* was originally an initialism for *Low Level Virtual Machine*. LLVM was originally developed as a research infrastructure to investigate dynamic compilation techniques for static and dynamic programming languages. LLVM was released under the University of Illinois/NCSA Open Source License,[3] a permissive free software licence. In 2005, Apple Inc. hired Lattner and formed a team to work on the LLVM system for various uses within Apple's development systems.[16] LLVM has been an integral part of Apple's Xcode development tools for macOS and iOS since Xcode 4.[17]

(https://www.llvm.org)

The LLVM abbreviation has officially been removed to avoid confusion, as LLVM has evolved into an umbrella project that has little relationship to what most current developers think of as (more specifically) process virtual machines.[18] Now, LLVM is a brand that applies to the LLVM umbrella project, the LLVM intermediate representation (IR), the LLVM debugger, the LLVM implementation of the C++ Standard Library (with full support of C++11 and C++14[19]), etc. LLVM is administered by the LLVM Foundation. Its president is compiler engineer Tanya Lattner.[20]

"For designing and implementing LLVM", the Association for Computing Machinery presented Vikram Adve, Chris Lattner, and Evan Cheng with the 2012 ACM Software System Award.[21]

Since v9.0.0, it was relicensed to the Apache License 2.0 with LLVM Exceptions.[3]

Features

LLVM can provide the middle layers of a complete compiler system, taking intermediate representation (IR) code from a compiler and emitting an optimized IR. This new IR can then be converted and linked into machine-dependent assembly language code for a target platform. LLVM can accept the IR from the GNU Compiler Collection (GCC) toolchain, allowing it to be used with a wide array of existing compiler front-ends written for that project.

LLVM can also generate relocatable machine code at compile-time or link-time or even binary machine code at run-time.

LLVM supports a language-independent instruction set and type system.[5] Each instruction is in static single assignment form (SSA), meaning that each variable (called a typed register) is assigned once and then frozen. This helps simplify the analysis of dependencies among variables. LLVM allows code to be compiled statically, as it is under the traditional GCC system, or left for late-compiling from the IR to machine code via just-in-time compilation (JIT), similar to Java. The type system consists of basic types such as integer or floating-point numbers and five derived types: pointers, arrays, vectors, structures, and functions. A type construct in a concrete language can be represented by combining these basic types in LLVM. For example, a class in C++ can be represented by a mix of structures, functions and arrays of function pointers.

The LLVM JIT compiler can optimize unneeded static branches out of a program at runtime, and thus is useful for partial evaluation in cases where a program has many options, most of which can easily be determined unneeded in a specific environment. This feature is used in the OpenGL pipeline of Mac OS X Leopard (v10.5) to provide support for missing hardware features.[22]

Graphics code within the OpenGL stack can be left in intermediate representation and then compiled when run on the target machine. On systems with high-end graphics processing units (GPUs), the resulting code remains quite thin, passing the instructions on to the GPU with minimal changes. On systems with low-end GPUs, LLVM will compile optional procedures that run on the local central processing unit (CPU) that emulate instructions that the GPU cannot run internally. LLVM improved performance on low-end machines using Intel GMA chipsets. A similar system was developed under the Gallium3D LLVMpipe, and incorporated into the GNOME shell to allow it to run without a proper 3D hardware driver loaded.[23]

For run-time performance of the compiled programs, GCC formerly outperformed LLVM by 10% on average in 2011.[24][25] Newer results in 2013 indicate that LLVM has now caught up with GCC in this area, and is now compiling binaries of approximately equal performance.[26]

Components

LLVM has become an umbrella project containing multiple components.

Front ends

LLVM was originally written to be a replacement for the existing code generator in the GCC stack,[27] and many of the GCC front ends have been modified to work with it, resulting in the now-defunct LLVM-GCC suite. The modifications generally involve a GIMPLE-to-LLVM IR step so that LLVM optimizers and codegen can be used instead of GCC's GIMPLE system. Apple was a significant user of LLVM-GCC through Xcode 4.x (2013).[28][29] This use of the GCC frontend was considered mostly a temporary measure, but with the advent of Clang and advantages of LLVM and Clang's modern and modular codebase (as well as compilation speed), is mostly obsolete.

LLVM currently supports compiling of Ada, C, C++, D, Delphi, Fortran, Haskell, Julia, Objective-C, Rust, and Swift using various front ends.

Widespread interest in LLVM has led to several efforts to develop new front ends for a variety of languages. The one that has received the most attention is Clang, a new compiler supporting C, C++, and Objective-C. Primarily supported by Apple, Clang is aimed at replacing the C/Objective-C compiler in the GCC system with a system that is more easily integrated with integrated development environments (IDEs) and has wider support for multithreading. Support for OpenMP directives has been included in Clang since release 3.8.[30]

The Utrecht Haskell compiler can generate code for LLVM. Though the generator is in the early stages of development, in many cases it has been more efficient than the C code generator.[31] There is a Glasgow Haskell Compiler (GHC) backend using LLVM that achieves a 30% speed-up of the compiled code relative to native code compiling via GHC or C code generation followed by compiling, missing only one of the many optimizing techniques implemented by the GHC.[32]

Many other components are in various stages of development, including, but not limited to, the Rust compiler, a Java bytecode front end, a Common Intermediate Language (CIL) front end, the MacRuby implementation of Ruby 1.9, various front ends for Standard ML, and a new graph coloring register allocator.

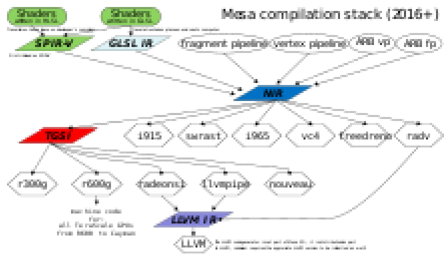
Intermediate representation

The core of LLVM is the intermediate representation (IR), a low-level programming language similar to assembly. IR is a strongly typed reduced instruction set computing (RISC) instruction set which abstracts away most details of the target. For example, the calling convention is abstracted through *call* and *ret* instructions with explicit arguments. Also, instead of a fixed set of registers, IR uses an infinite set of temporaries of the form %0, %1, etc. LLVM supports three equivalent forms of IR: a human-readable assembly format, an in-memory format suitable for frontends, and a dense bitcode format for serializing. A simple "Hello, world!" program in the IR format:[33]

```
@.str = internal constant [14 x i8] c"hello,
world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str,
i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 )
nounwind
    ret i32 0
}
```



LLVM IR is used e.g., by radeonsi and by llvmpipe. Both are part of Mesa 3D.

The many different conventions used and features provided by different targets mean that LLVM cannot truly produce a target-independent IR and retarget it without breaking some established rules. Examples of target dependence beyond what is explicitly mentioned in the documentation can be found in a 2011 proposal for "wordcode", a fully target-independent variant of LLVM IR intended for online distribution.[34] A more practical example is PNaCl.[35]

Back ends

At version 3.4, LLVM supports many instruction sets, including ARM, Qualcomm Hexagon, MIPS, Nvidia Parallel Thread Execution (PTX; called *NVPTX* in LLVM documentation), PowerPC, AMD TeraScale,[36] AMD Graphics Core Next (GCN), SPARC, z/Architecture (called *SystemZ* in LLVM documentation), x86, x86-64, and XCore. Some features are not available on some platforms. Most features are present for x86, x86-64, z/Architecture, ARM, and PowerPC.[37] RISC-V is supported as of version 7. In the past LLVM also supported fully or partially other backends, including C backend, Cell SPU, mblaze (MicroBlaze),[38] AMD R600, DEC/Compaq Alpha (Alpha AXP)[39] and Nios2,[40] but most of this hardware is mostly obsolete, and LLVM developers decided the support and maintenance costs were no longer justified.

LLVM also supports WebAssembly as a target, enabling compiled programs to execute in WebAssembly-enabled environments such as Google Chrome / Chromium, Firefox, Microsoft Edge, Apple Safari or WAVM. LLVM-compliant WebAssembly compilers typically support mostly unmodified source code written in C, C++, D, Rust, Nim, Kotlin and several other languages.

The LLVM machine code (MC) subproject is LLVM's framework for translating machine instructions between textual forms and machine code. Formerly, LLVM relied on the system assembler, or one provided by a toolchain, to translate assembly into machine code. LLVM MC's integrated assembler supports most LLVM targets, including x86, x86-64, ARM, and ARM64. For some targets, including the various MIPS instruction sets, integrated assembly support is usable but still in the beta stage.

Linker

The lld subproject is an attempt to develop a built-in, platform-independent linker for LLVM.[41] lld aims to remove dependence on a third-party linker. As of May 2017, lld supports ELF, PE/COFF, Mach-O, and WebAssembly[42] in descending order of completeness. lld is faster than both flavors of GNU ld.[41]

Unlike the GNU linkers, lld has built-in support for link-time optimization. This allows for faster code generation as it bypasses the use of a linker plugin, but on the other hand prohibits interoperability with other flavors of LTO.[43]

C++ Standard Library

The LLVM project includes an implementation of the C++ Standard Library called libc++, dual-licensed under the MIT License and the UIUC license.[44]

Since v9.0.0, it was relicensed to the Apache License 2.0 with LLVM Exceptions.[3]

Polly

This implements a suite of cache-locality optimizations as well as auto-parallelism and vectorization using a polyhedral model.[45]

Debugger

Derivatives

Due to its permissive license, many vendors release their own tuned forks of LLVM. This is officially recognized by LLVM's documentation, which suggests against using version numbers in feature checks for this reason.[46] Some of the vendors include:

- AMD's AMD Optimizing C/C++ Compiler is based on LLVM, Clang, and Flang.
- Apple maintains an open-source fork for Xcode.[47]
- ARM maintains a fork of LLVM 9 as the "Arm Compiler".
- Intel has adopted LLVM for their next generation Intel C++ Compiler.[48]
- The Los Alamos National Laboratory has a parallel-computing fork of LLVM 8 called "Kitsune".[49]
- Since 2013, Sony has been using LLVM's primary front-end Clang compiler in the software development kit (SDK) of its PlayStation 4 console.[50]
- Nvidia uses LLVM in the implementation of its NVVM CUDA Compiler.[51] The NVVM compiler is distinct from the "NVPTX" backend mentioned in the Backends section, although both generate PTX code for Nvidia GPUs.
- IBM is adopting LLVM in its C/C++ and Fortran compilers.[52]

See also

- HHVM
- C--
- Amsterdam Compiler Kit (ACK)
- LLDB (debugger)
- GNU lightning
- GNU Compiler Collection (GCC)
- Pure

- OpenCL
- Emscripten
- TenDRA Distribution Format
- Architecture Neutral Distribution Format (ANDF)
- Comparison of application virtual machines
- SPIR-V
- University of Illinois at Urbana Champaign discoveries & innovations

Literature

- Chris Lattner - *The Architecture of Open Source Applications - Chapter 11 LLVM* (http://www.aosabook.org/en/llvm.html), ISBN 978-1257638017, released 2012 under CC BY 3.0 (Open Access).[53]
- LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation (http://llvm.org/pubs/2004-01-30-CGO-LLVM.pdf), a published paper by Chris Lattner, Vikram Adve

References

1. "LLVM Logo" (https://llvm.org/Logo.html). *The LLVM Compiler Infrastructure Project*.
2. "Releases" (https://github.com/llvm/llvm-project/releases). *Github*. July 9, 2021. Retrieved September 1, 2021.
3. "LICENSE.TXT" (https://releases.llvm.org/9.0.0/LICENSE.TXT). llvm.org. Retrieved September 24, 2019.
4. "The LLVM Compiler Infrastructure Project" (http://llvm.org/). Retrieved March 11, 2016.
5. "LLVM Language Reference Manual" (http://llvm.org/docs/LangRef.html). Retrieved June 9, 2019.
6. "Announcing LLILC - A new LLVM-based Compiler for .NET" (https://dotnetfoundation.org/blog/2015/04/14/announcing-llilc-llvm-for-dotnet). *dotnetfoundation.org*. Retrieved September 12, 2020.
7. *Mono LLVM* (http://www.mono-project.com/Mono_LLVM), retrieved March 10, 2013
8. Chris Lattner (2011). "LLVM" (http://www.aosabook.org/en/llvm.html). In Amy Brown; Greg Wilson (eds.). *The Architecture of Open Source Applications* (http://www.aosabook.org/).
9. https://github.com/reschivon/movForth
10. William Wong (May 23, 2017). "What's the Difference Between LabVIEW 2017 and LabVIEW NXG?" (http://www.electronicdesign.com/test-measurement/what-s-difference-between-labview-2017-and-labview-nxg). *Electronic Design*.
11. Michael Larabel (April 11, 2018). "Khronos Officially Announces Its LLVM/SPIR-V Translator" (https://www.phoronix.com/scan.php?page=news_item&px=SPIRV-LLVM-Translator). phoronix.com.
12. "32.1. What is JIT compilation?" (https://www.postgresql.org/docs/11/jit-reason.html). *PostgreSQL Documentation*. November 12, 2020. Retrieved January 25, 2021.
13. "Features" (http://www.rubymotion.com/tour/features/). *RubyMotion*. Scratchwork Development LLC. Retrieved June 17, 2017. "RubyMotion transforms the Ruby source code of your project into ... machine code using a[n] ... ahead-of-time (AOT) compiler, based on LLVM."
14. Reedy, Geoff (September 24, 2012). "Compiling Scala to LLVM" (http://www.infoq.com/presentations/Scala-LLVM). St. Louis, Missouri, United States. Retrieved February 19, 2013.

15. "xCORE: Multicore theory, hardware and programming (2014-12-01)" (https://www.xmos.ai/file/xcore-multicore-theory-hardware-and-programming/), *Developer Tool xTIMEcomposer*, XMOS, retrieved March 27, 2021
16. Adam Treat (February 19, 2005), *mkspecs and patches for LLVM compile of Qt4* (https://web.archive.org/web/20111004073001/http://lists.trolltech.com/qt4-preview-feedback/2005-02/msg00691.html), archived from the original (http://lists.trolltech.com/qt4-preview-feedback/2005-02/msg00691.html) on October 4, 2011, retrieved January 27, 2012
17. "Developer Tools Overview" (https://web.archive.org/web/20110423095129/https://developer.apple.com/technologies/tools/). *Apple Developer*. Apple. Archived from the original (https://developer.apple.com/technologies/tools/) on April 23, 2011.
18. Lattner, Chris (December 21, 2011). "The name of LLVM" (http://lists.llvm.org/pipermail/llvm-dev/2011-December/046445.html). *llvm-dev* (Mailing list). Retrieved March 2, 2016. ""LLVM" is officially no longer an acronym. The acronym it once expanded too was confusing, and inappropriate almost from day 1. :) As LLVM has grown to encompass other subprojects, it became even less useful and meaningless."
19. " "libc++" C++ Standard Library" (http://libcxx.llvm.org/).
20. Chris Lattner (April 3, 2014). "The LLVM Foundation" (http://blog.llvm.org/2014/04/the-llvm-foundation.html). *LLVM Project Blog*.
21. "ACM Software System Award" (https://awards.acm.org/software-system/award-winners?year=2012&award=149&region=&submit=Submit&isSpecialCategory=). ACM.
22. Chris Lattner (August 15, 2006). "A cool use of LLVM at Apple: the OpenGL stack" (http://lists.llvm.org/pipermail/llvm-dev/2006-August/006497.html). *llvm-dev* (Mailing list). Retrieved March 1, 2016.
23. Michael Larabel, "GNOME Shell Works Without GPU Driver Support" (https://www.phoronix.com/scan.php?page=news_item&px=MTAxMjI), *phoronix*, November 6, 2011
24. V. Makarov. "SPEC2000: Comparison of LLVM-2.9 and GCC4.6.1 on x86" (https://vmakarov.fedorapeople.org/spec/2011/llvmgcc32.html). Retrieved October 3, 2011.
25. V. Makarov. "SPEC2000: Comparison of LLVM-2.9 and GCC4.6.1 on x86_64" (https://vmakarov.fedorapeople.org/spec/2011/llvmgcc64.html). Retrieved October 3, 2011.
26. Michael Larabel (December 27, 2012). "LLVM/Clang 3.2 Compiler Competing With GCC" (https://www.phoronix.com/scan.php?page=article&item=llvm_clang32_final). Retrieved March 31, 2013.
27. Lattner, Chris; Vikram Adve (May 2003). *Architecture For a Next-Generation GCC* (http://llvm.org/pubs/2003-05-01-GCCSummit2003.html). First Annual GCC Developers' Summit. Retrieved September 6, 2009.
28. "LLVM Compiler Overview" (https://developer.apple.com/library/archive/documentation/CompilerTools/Conceptual/LLVMCompilerOverview/index.html). *developer.apple.com*.
29. "Xcode 5 Release Notes" (https://developer.apple.com/library/archive/documentation/Xcode/Conceptual/RN-Xcode-Archive/Chapters/xc5_release_notes.html). *Apple Inc*.
30. "Clang 3.8 Release Notes" (http://llvm.org/releases/3.8.0/tools/clang/docs/ReleaseNotes.html#openmp-support-in-clan). Retrieved August 24, 2016.
31. "Compiling Haskell To LLVM" (http://www.cs.uu.nl/wiki/bin/view/Stc/CompilingHaskellToLLVM). Retrieved February 22, 2009.
32. "LLVM Project Blog: The Glasgow Haskell Compiler and LLVM" (http://blog.llvm.org/2010/05/glasgow-haskell-compiler-and-llvm.html). Retrieved August 13, 2010.
33. For the full documentation, refer to llvm.org/docs/LangRef.html (http://llvm.org/docs/LangRef.html).
34. Kang, Jin-Gu. "Wordcode: more target independent LLVM bitcode" (https://llvm.org/devmtg/2011-09-16/EuroLLVM2011-MoreTargetIndependentLLVMBitcode.pdf) (PDF). Retrieved December 1, 2019.

35. "PNaCl: Portable Native Client Executables" (https://web.archive.org/web/20120502135033/http://nativeclient.googlecode.com/svn/data/site/pnacl.pdf) (PDF). Archived from the original (http://nativeclient.googlecode.com/svn/data/site/pnacl.pdf) (PDF) on 2 May 2012. Retrieved 25 April 2012.
36. Stellard, Tom (March 26, 2012). "[LLVMdev] RFC: R600, a new backend for AMD GPUs" (http://lists.llvm.org/pipermail/llvm-dev/2012-March/048409.html). *llvm-dev* (Mailing list).
37. Target-specific Implementation Notes: Target Feature Matrix (http://llvm.org/docs/CodeGenerator.html#target-feature-matrix) // The LLVM Target-Independent Code Generator, LLVM site.
38. "Remove the mblaze backend from llvm" (https://github.com/llvm/llvm-project/commit/729866670b05108c399221ca3908400d94ef9783). *GitHub*. July 25, 2013. Retrieved January 26, 2020.
39. "Remove the Alpha backend" (https://github.com/llvm/llvm-project/commit/4c9fca99c9a6734bb33c34aeaf40b71c4002757e). *GitHub*. October 27, 2011. Retrieved January 26, 2020.
40. "[Nios2] Remove Nios2 backend" (https://github.com/llvm/llvm-project/commit/99fcbf67d04d488d819bffb8fda3bb9d5504b63b). *GitHub*. January 15, 2019. Retrieved January 26, 2020.
41. "lld - The LLVM Linker" (http://lld.llvm.org/). The LLVM Project. Retrieved May 10, 2017.
42. "WebAssembly lld port" (https://lld.llvm.org/WebAssembly.html).
43. "42446 – lld can't handle gcc LTO files" (https://bugs.llvm.org/show_bug.cgi?id=42446). *bugs.llvm.org*.
44. " "libc++" C++ Standard Library" (http://libcxx.llvm.org).
45. "Polly - Polyhedral optimizations for LLVM" (https://polly.llvm.org).
46. "Clang Language Extensions" (https://clang.llvm.org/docs/LanguageExtensions.html#feature-checking-macros). *Clang 12 documentation*. "Note that marketing version numbers should not be used to check for language features, as different vendors use different numbering schemes. Instead, use the Feature Checking Macros."
47. "apple/llvm-project" (https://github.com/apple/llvm-project). Apple. September 5, 2020.
48. "Intel C/C++ compilers complete adoption of LLVM" (https://www.intel.com/content/www/us/en/develop/blogs/adoption-of-llvm-complete-icx.html). *Intel*. Retrieved August 17, 2021.
49. "lanl/kitsune" (https://github.com/lanl/kitsune). Los Alamos National Laboratory. February 27, 2020.
50. *Developer Toolchain for ps4* (http://llvm.org/devmtg/2013-11/slides/Robinson-PS4Toolchain.pdf) (PDF), retrieved February 24, 2015
51. "NVVM IR Specification 1.5" (https://docs.nvidia.com/cuda/nvvm-ir-spec/index.html). "The current NVVM IR is based on LLVM 5.0"
52. "IBM C/C++ and Fortran compilers to adopt LLVM open source infrastructure" (https://developer.ibm.com/blogs/c-and-fortran-adopt-llvm-open-source/).
53. Chris Lattner (March 15, 2012). "Chapter 11" (http://www.aosabook.org/en/llvm.html). *The Architecture of Open Source Applications*. Amy Brown, Greg Wilson. ISBN 978-1257638017.

External links

- Official website (https://llvm.org)