



LLVM/CLANG

Watch out GCC – there’s a new compiler suite in town, and it wants your crown.

MIKE SAUNDERS

Q Uh-oh – compilers. There’ll be a lot of acronyms and initialisms in this article, then...

A Actually, no. The technology behind LLVM and Clang is tremendously complex, but here we’re more interested in its practical applications – how it’s going to make life easier for developers, and make our applications faster and more reliable.

Q So what’s going on with this strangely named software?

A Right now, GCC (the GNU Compiler Collection) is responsible for compiling virtually all of the code in a typical GNU/Linux distribution. That is, it takes the human-readable source code for programs and turns it into executable files that the CPU can understand. GCC is arguably the biggest success story of the GNU project – it’s used everywhere, from supercomputers to embedded devices, and it can generate code for a huge range of CPU types.

“Some developers claim that the politics of the GPL scare away potential contributors.”

Q Great! So GCC rules – shall we just go to the pub now?

A Hang on a second. GCC is highly regarded, but it has flaws. The codebase is very complicated, and some developers claim it’s so messy that very few people can add new features to it, hindering progress. In addition, it’s hard to split up individual parts of the compiler, making it hard to integrate with modern IDEs (integrated development environments). Finally, the licence is an issue for some developers, who’d rather avoid the GPL and use something more BSD-like.

Q So this LLVM/Clang thing solves all these problems?

A Well, it’s a start. LLVM has been around since 2000, and Clang since 2007. Developing a new compiler is a gargantuan task, especially given all the languages, language features and CPU architectures in use today.

While GCC is still the *de facto* standard compiler in the FOSS world, LLVM/Clang is nipping at its heels, thanks to support from notable companies such as Apple. One testament to the maturity of LLVM/Clang is that it’s the default compiler in FreeBSD 10, replacing GCC. FreeBSD is well regarded as a conservative and highly reliable open source Unix flavour, so its adoption of LLVM/Clang was a major event in the compiler’s history.

Q You keep talking about LLVM/Clang – what’s with the funny name? Is it like GNU/Linux?

A No; it refers to two technologies that are parts of the compiler. In the olden days, compilers would simply read source code and generate equivalent CPU instructions on the fly, perhaps performing a bit of optimisation along the way. As time went on, it made more sense to split the part of the compiler that parses the source code away from the part that generates the CPU instructions. This makes a lot of sense, as it becomes easier to support more programming languages and CPU types. For instance, someone could add Fortran processing capability to the compiler without having to know about ARM and x86 CPU instructions. Or someone heavily versed in those CPU instructions can make optimisations without having to think about the high-level languages.

In order to separate these parts of the compiler and add a level of abstraction between them, you need an intermediate language. And that’s exactly what makes LLVM/Clang work. Essentially, Clang is a front-end for C-like languages (C, C++, Objective C), so it parses C code, includes header files, handles macros and so forth, and then generates some intermediate code. This code looks a bit like a mixture between assembly language

and a high-level language:

```
%result = mul i32 %X, 8
```

Here, the 32-bit integer variable `%X` is multiplied by 8, and the result is stored in the `%result` variable. This is not specific to any kind of CPU type, but is sufficiently low-level for LLVM to process. (That's where the name comes from – Low Level Virtual Machine.) So Clang knows C, and LLVM knows this intermediate language. LLVM then performs optimisations and tricks with the intermediate code, before generating CPU instructions for the chip of your choice.

Q **Wow! That sounds rather clever... But are other programming languages supported?**

A Yes – quite a few. You can chuck out Clang and replace it with another language front-end that generates the intermediate language. LLVM doesn't care. Right now there are front-ends for Ada, D, Fortran and other languages, and many of these front-ends were taken from the GCC codebase. And because LLVM is released under a BSD-like licence (so you can do what you want with it, providing you give credit to the original developers), it's making its way into various other compiler suites, IDEs and projects: www.llvm.org/ProjectsWithLLVM.

Q **Hrm, I'm not sure I like the BSD-style licence, whereby anyone can make closed source programs with LLVM inside. Wouldn't the GPL be better?**

A We're big GPL fans at Linux Voice HQ, so we know what you're saying. And Richard Stallman isn't especially happy about LLVM either:

"The existence of LLVM is a terrible setback for our community precisely because it is not copylefted and can be used as the basis for nonfree compilers – so that all contribution to LLVM directly helps proprietary software as much as it helps us." (Full message at <http://gcc.gnu.org/ml/gcc/2014-01/msg00247.html>)

Still, there are other sides to the argument. Some developers claim that the GPL scares away potential contributors due to all the politics involved, whereas a BSD-licensed compiler makes it simpler for

companies to add patches. It's a massive debate with a million points to be made, and GPL vs BSD arguments will rage for decades to come.

Q **Hokey dokey. Anyway, earlier you talked about LLVM/Clang making our applications faster and more reliable. How does that work?**

A One of the goals of Clang is to produce more informative error messages than those spat out by GCC, making it easier for developers to spot and fix bugs. Ideally, this should result in better code with fewer bugs.

<http://clang.llvm.org/diagnostics.html>

shows what the Clang team is doing in this direction. It should be noted that GCC is making progress in this area too, with colourised output in GCC 4.9 as an example. Competition is good!

Right now, LLVM and GCC are pretty much neck-and-neck when it comes to the speed of their produced code. GCC has a tiny edge in some benchmarks, but given that LLVM is a much younger project, it's impressive that it has reached the same level so quickly.

Because LLVM's codebase is simpler and easier to navigate than GCC's, the hope is that it'll be easier for new developers to add optimisations, and the more people that can get involved with the compiler's internals, the better. So while LLVM doesn't magically make applications faster now, if its development continues more rapidly than GCC's, it might outperform the GNU compiler substantially over time.



Every good FOSS project needs a mascot, and LLVM's robot is rather darn cool, we must say.

Q **Can LLVM/Clang compile everything that GCC can?**

A Not quite. GCC provides various extensions for C and C++, and because it has been the *de facto* standard compiler for free Unix systems for decades, many developers use these extensions. Clang's support is constantly expanding, though. In addition, there are efforts underway to make the Linux kernel compilable by LLVM/Clang, (see <http://linuxfoundation.org>). Right now it's possible to compile the kernel, but some external patches are required.

Q **OK, so how do I get it installed and test my code?**

A Few distributions include LLVM/Clang by default, although almost all of the big-name distros have the compiler in their package repositories – look for **llvm** and **clang**. Interestingly, some Debian developers are trying to build a version of the distro entirely with LLVM/Clang; of the 40,000+ packages that Debian includes, only 11.6% of them can't be compiled right now. As LLVM/Clang improves and becomes more compatible with GCC, this number will go down, and in a few years we might see mainstream distros compiled entirely with the newer compiler.

GCC is still an awesome compiler, and LLVM/Clang's presence has rejuvenated its development. With both teams scrambling to make faster, more informative and more reliable compilers, the future looks very rosy indeed. ☑