clang
reinventing the compiler

Alp Toker

http://www.nuanti.com
Overview

• What can we achieve going beyond compilation?
• Why are we compelled to invent a better wheel?
• How can we make everyday life better for coders?
• Could the compiler itself become an instrument for wider social change?
Clang in a Nutshell

- clang driver
- clang-cl driver
- clang frontend
- LLVM backend
- lld linker
 clang frontend
“lowering”

Lex
tokenization and preprocessing

Parse
semantic analysis

Sema
semantic analysis

Analyzer
static analysis

AST
syntax tree

CodeGen
to LLVM IR

Analysis ARCMigrate AST ASTMatchers Basic CodeGen Driver Edit Format Frontend FrontendTool Headers Index Lex Parse Rewrite Sema Serialization StaticAnalyzer Tooling
Tooling
build-your-own

Clindex
indexing engine

libclang
stable C API

Rewrite
& other utilities

Plugins
& frontend passes
Why invent a better wheel?
MSVC support is coming. But why are we even doing this?
DEVELOPERS
DEVELOPERS
DEVELOPERS
DEVELOPERS
Microsoft Visual C++

a kind of geeky Rosetta stone

Objective C

ISO C

ISO C++

OpenMP

OpenCL
3.5: MSVC Compatibility

• More significant than just Windows support

• Unusual parsing

• Name mangling

• Built-in types

• Delayed template parsing
clang-cl.exe

- A cl.exe drop-in replacement driver
- Visual Studio integration
How?
The Clang Parser

- Hand-written recursive-descent parser.
- A single unified parser for C/C++/ObjC
- (Mostly) decoupled from the AST representation
Clang Semantic Analysis

- Sema: The brains of the operation.
- Builds the AST and computes types, linkage etc.
- Some problems here too.
What next?
The road to Faster Compilation

• **In-process** execution currently under investigation

• Multi-TU compilation supporting **modules**

• **Cached resources** across invocations

• Use **MCJIT** for constexpr compile-time evaluation?
Spot the problem in this code...

```cpp
bool ProcessingFailed =
for(unsigned CommandIndex = 0;
    CommandIndex < CommandLine.size();
    ++CommandIndex) {
    DEBUG({
        llvm::
    });
}
```

// FIXME: chdir is thread hostile; on the other hand, creating the same
// behavior via 'chdir' is complex: 'chdir' resolves the path once, thus
// guaranteeing that all subsequent relative path operations works
// unchanged on the original 'chdir' resulted in. This makes a difference
// on some network filesystems, where symlinks might be switched
// during runtime of the tool. Fixing this depends on having a file system
// abstraction that allows openat() style interactions.

// FIXME: We need a callback mechanism for the tool writer to output a
// customized message for each file. Fixing this depends on having a file system
// abstraction that allows openat() style interactions.
Time for Compiler Accessibility?

• Vision and motion impaired users code too.

• Hierarchical documents lend themselves to **universal access**:
  
  • The AST is a **natural representation** here to get started.
  
  • *Code completion* machinery can help **select inputs** and refactoring will enable **edits** out of scope.

  • **Diagnostics** can be annotated for voice output.

• We have all the technology today, yet **no accessibility story to speak of**.
Clang & the Linux Kernel

- **clang -m16**: Code generation to support the x86 boot loader appropriate for a CPU running in 16-bit mode.

- Integrated ASM parser support imminent for all `.S files`

- Users & developers joining the LLVM community to fulfil their needs.
The Clang AST

• Abstract Syntax Tree represented as a C++ class hierarchy

• Uses LLVM's casting system, not RTTI

• Informal representation, some problems here:
  • Objective-C duplication. Function/Method, Interface/Record/Class…
  • Some semantic analysis still “performed” by AST
  • Type system omits linkage & other details, time to address this?
LLVM and Clang are defending your Software Freedom. Here’s how...
the freedom to innovate

(Freedom #1)
LLVM Community 2.0

• Do we approach *controversial issues* effectively?

• What makes a patch acceptable?

• Do we *welcome new contributors* or is there an initiation by fire?

• How should we handle *non-code contributions*?
Clang Zeitgeist 2014

30 active developers
1.5m LoC
500 commits per month
Some introspective questions...

• Where does our *infrastructure* come from?

• Is there a framework to deal with emergencies and *existential threats* to our project?

• Do we have *transparent oversight*?

• How about…
Planet Clang
http://planet.clang.org

LLVM Weekly
http://llvmweekly.org
Optimizer pragmas & attributes

• A desire to offer *hands-on control* over the LLVM code generation and optimizer.

• Vectorization attributes

• optnone — or more granularity?
TOGETHER, WE CAN REINVENT THE COMPILER.

Alp Toker

alp@nuanti.com