Modular Codegen

Further Benefits of Explicit Modularization
Motivating Example

```c
#ifndef FOO_H
#define FOO_H
inline void foo() { ... }
#endif

#include "foo.h"
void bar() {
    foo();
}

#include "foo.h"
void baz() {
    foo();
}
```
Implicit Modules

- User writes `.modulemap` files

```
module foo {
    header "foo.h"
    export *
}
```
Implicit Modules
Build Process

Build System

bar.cpp
clang++

baz.cpp
clang++
Implicit Modules
Build Process

Build System

bar.cpp
clang++

foo.mm
clang++

baz.cpp
clang++

foo.pcm
Implicit Modules
Build Process

Build System

bar.cpp
clang++
bar.o
baz.cpp
clang++
baz.o

foo.mm
clang++
foo.pcm
Implicit Modules

- User writes .modulemap files
- Compiler finds them and implicitly builds module descriptions in a filesystem cache
- Build system agnostic

- Difficult to parallelize - build system isn’t aware of the dependencies
- Doesn’t distribute (clang doesn’t know about distribution scheme)
Explicit Modules

- Build system explicitly invokes the compiler on .modulemap files
- Passes resulting .pcm files when compiling .cpp files for use
Explicit Modules

Build Process

Build System

foo.mm

clang++

foo.pcm
Explicit Modules
Build Process

Build System

foo.mm

bar.cpp
clang++

baz.cpp
clang++

foo.pcm

bar.o

baz.o
Modules TS (Technical Specification)

- New file type (C++ with some new syntax - .cppm?)
- New import syntax
- Also needs build system support
Modular Codegen
Duplication in Object Files

Each object file contains independent definitions of:

- Uninlined 'inline' functions (& some other bits)
- Debug information descriptions of classes
Modular Objects

The module can be used as a ‘home’ for these entities so they don’t need to be carried by every user.

- bar.cpp
  - bar()

- foo.mm
  - foo.pcm
  - foo.o
    - f1()

- baz.cpp
  - baz()

- a.out
  - bar()
  - baz()
  - f1()
Risks

Unused entities may increase linker inputs.
Constraints

- Headers are compiled separately (& only once) from uses
- Dependencies must be well formed
  - Headers cannot be implemented by a different library - they form circular dependencies no longer broken by duplicated definitions at every use.
void a1() { b(); }
void a2() { ... }

void b() { a2(); }
void a1() { b(); }
void a2() { ... }

void b() { a2(); }
Diversion: ‘How Unix Linkers Work (lite)’

```
void a1() { b(); }
void a2() { ... }
void b() { a2(); }
```

Diagram:

- `void a1() { b(); }
  void a2() { ... }
  void b() { a2(); }

- `a1()`? → `a1() ✓`
- `b()`?
Diversion: ‘How Unix Linkers Work (lite)’

void a1() { b(); }
void a2() { ... }
void b() { a2(); }

a1()?
a1() ✓
b() ?

b() ✓
a2()?
void a1() { b(); }  
void a2() { ... }  

void b() { a2(); }
Clang/LLVM Codebase

- *.def files are textual/non-modular
- lib/support/regc* are non-modular
- MCTargetOptionsCommandFlags.h non-modular
- CommandFlags.h non-modular
- Target ASM Parsers depend on MC Target Description
- static namespace-scope functions in headers -> inline, non-static
- Missing #includes
- No idea what to do with abi-breaking.h
- Weird things in Hexagon (non-modular headers that are included exactly once...)
- ASTMatchers defining global variables in headers... no idea how this isn't causing link errors, maybe they've got implicit internal linkage.
Results
Object Section Sizes
-00 -fmodules-codegen -gsplit-dwarf
Object Section Sizes
-00 -fmodules-codegen -gsplit-dwarf

- .strtab
- .text
- .shstrtab
- .symtab
- .rela.text
- .rela.debug_ranges

45 more
Object Section Sizes
-00 -fmodules-codegen -gsplit-dwarf
Object Section Sizes

-00 -fmodules-codegen -gsplit-dwarf

### Baseline

<table>
<thead>
<tr>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.strtab</td>
<td>0.2 GB</td>
</tr>
<tr>
<td>.text</td>
<td>0.8 GB</td>
</tr>
<tr>
<td>.shstrtab</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>.symtab</td>
<td>0.05 GB</td>
</tr>
<tr>
<td>.rela.text</td>
<td>0.03 GB</td>
</tr>
<tr>
<td>.rela.debug_ranges</td>
<td>0.02 GB</td>
</tr>
<tr>
<td>.rodata</td>
<td>0.01 GB</td>
</tr>
<tr>
<td>.rela.debug_addr</td>
<td>0.01 GB</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.strtab</td>
<td>0.2 GB</td>
</tr>
<tr>
<td>.text</td>
<td>0.8 GB</td>
</tr>
<tr>
<td>.shstrtab</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>.symtab</td>
<td>0.05 GB</td>
</tr>
<tr>
<td>.rela.text</td>
<td>0.03 GB</td>
</tr>
<tr>
<td>.rela.debug_ranges</td>
<td>0.02 GB</td>
</tr>
<tr>
<td>.rodata</td>
<td>0.01 GB</td>
</tr>
<tr>
<td>.rela.debug_addr</td>
<td>0.01 GB</td>
</tr>
</tbody>
</table>

### Debug

<table>
<thead>
<tr>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.strtab</td>
<td>0.2 GB</td>
</tr>
<tr>
<td>.text</td>
<td>0.8 GB</td>
</tr>
<tr>
<td>.shstrtab</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>.symtab</td>
<td>0.05 GB</td>
</tr>
<tr>
<td>.rela.text</td>
<td>0.03 GB</td>
</tr>
<tr>
<td>.rela.debug_ranges</td>
<td>0.02 GB</td>
</tr>
<tr>
<td>.rodata</td>
<td>0.01 GB</td>
</tr>
<tr>
<td>.rela.debug_addr</td>
<td>0.01 GB</td>
</tr>
</tbody>
</table>

### Both

<table>
<thead>
<tr>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.strtab</td>
<td>0.2 GB</td>
</tr>
<tr>
<td>.text</td>
<td>0.8 GB</td>
</tr>
<tr>
<td>.shstrtab</td>
<td>0.1 GB</td>
</tr>
<tr>
<td>.symtab</td>
<td>0.05 GB</td>
</tr>
<tr>
<td>.rela.text</td>
<td>0.03 GB</td>
</tr>
<tr>
<td>.rela.debug_ranges</td>
<td>0.02 GB</td>
</tr>
<tr>
<td>.rodata</td>
<td>0.01 GB</td>
</tr>
<tr>
<td>.rela.debug_addr</td>
<td>0.01 GB</td>
</tr>
</tbody>
</table>

52 more sections are present.
Further Work

- Other aspects needed for Modules TS
  - Variables (implemented - could be backported to non-TS style, may not be needed)
  - ???
- Avoid homing `alwaysinline` functions (maybe other reasonable inlining heuristics to avoid homing functions unlikely to remain uninlined)
- Avoid type units when a home is likely to be unique (not an implicit template instantiation, or has a strong vtable, etc)
Thanks!

David Blaikie

Email/etc: dblaikie@gmail.com
Twitter: @dwblaikie
20%

Use this slide to show a major stat. It can help enforce the presentation’s main message or argument.
This is the most important takeaway that everyone has to remember.
Final point

A one-line description of it
“This is a super-important quote”

- From an expert