LLVM + ARM = ?

Status of ARM platform support in LLVM and more

Anton Korobeynikov
anton@korobeynikov.info

Outline

1. LLVM Compiler Infrastructure
2. Status:
    1. Clang
    2. Compiler-RT
    3. Backend
3. Track, Use & Contribute
4. Interoperability Problems

LLVM: What is it?

Language independent optimizer and code generator

- Many optimizations, many targets
- Modern and library-based design

LLVM: What is it?

Language independent optimizer and code generator

- Many optimizations, many targets
- Modern and library-based design

Clang C/C++/ObjectiveC frontend

- Designed for speed, reusability, compatibility with GCC (not only!) extensions
- Good also as indexing, analysis, refactoring tool

LLVM: other subprojects

1. MC: operations on “machine code”
   - Assemblers, disassemblers, direct object code emission

LLVM: other subprojects

1. MC: operations on "machine code"
   - Assemblers, disassemblers, direct object code emission
2. LLDB: low level debugger
   - Command line debugger
   - Reuses Clang parser, some JIT bits, MC disassemblers

LLVM: other subprojects

1. MC: operations on "machine code"
   - Assemblers, disassemblers, direct object code emission
2. LLDB: low level debugger
   - Command line debugger
   - Reuses Clang parser, some JIT bits, MC disassemblers
3. libc++: C++ standard runtime library
   - Full support for C++11
   - Designed for performance

Clang: status as of 2.9

1. Production quality C, ObjC, C++, ObjC++ compiler on:
   - X86 (-32 and -64)
   - (some) ARM cores

Clang: status as of 2.9

1. Production quality C, ObjC, C++, ObjC++ compiler on:
   - X86 (-32 and -64)
   - (some) ARM cores
2. Can bootstrap; build Boost, Mozilla, Qt, ...

Clang: status as of 2.9

1. Production quality C, ObjC, C++, ObjC++ compiler on:
   - X86 (-32 and -64)
   - (some) ARM cores
2. Can bootstrap; build Boost, Mozilla, Qt, ...
3. Builds working base FreeBSD system

ARM-specific extensions

1. ‘pcs’ function attribute is fully supported

ARM-specific extensions

1. ‘pcs’ function attribute is fully supported
2. All (Soft, Soft-FP, Hard) FP ABIs are supported

ARM-specific extensions

1. ‘pcs’ function attribute is fully supported
2. All (Soft, Soft-FP, Hard) FP ABIs are supported
3. NEON builtins are fully implemented (with some extensions)

New stuff in 3.0

1. Significant progress in C++11 support
2. Various new stuff in libc++

New stuff in 3.0

1. Significant progress in C++11 support
2. Various new stuff in libc++
3. Improvements in semantic analysis: new warnings here and there

New stuff in 3.0

1. Significant progress in C++11 support
2. Various new stuff in libc++
3. Improvements in semantic analysis: new warnings here and there
4. Improvements in static analyzer

New stuff in 3.0

1. Significant progress in C++11 support
2. Various new stuff in libc++
3. Improvements in semantic analysis: new warnings here and there
4. Improvements in static analyzer
5. Thread safety annotations

ARM extensions in 3.0

1. Homogeneous aggregates in hard FP ABI

ARM extensions in 3.0

1. Homogeneous aggregates in hard FP ABI
2. Generic half FP support

ARM extensions in 3.0

1. Homogeneous aggregates in hard FP ABI
2. Generic half FP support:
   - Native ops for OpenCL & similar
   - Storage-only type for everything else

Clang: compatibility

1. C99 by default (different inline semantics)

Clang: compatibility

1. C99 by default (different inline semantics)
2. More “strict” inline assembler

Clang: compatibility

1. C99 by default (different inline semantics)
2. More “strict” inline assembler
3. C++ VLAs are not supported

Clang: compatibility

1. C99 by default (different inline semantics)
2. More “strict” inline assembler
3. C++ VLAs are not supported
4. Much standard (than gcc) in C++ templates

Check clang.llvm.org/compatibility.html

Missed things in clang

1. ‘interrupt’ attribute is not supported
2. Everything assumes little-endian byte order
3. Nothing like a ‘universal driver’ is implemented

Universal Driver: intro

1. Clang is inherently a cross compiler:
    - All backends / targets are linked in
    - In theory target switch should be easy

Universal Driver: intro

1. Clang is inherently a cross compiler:
   - All backends / targets are linked in
   - In theory target switch should be easy
2. But: compiler is only a part of the whole compilation chain

Universal Driver: intro

1. Clang is inherently a cross compiler:
   - All backends / targets are linked in
   - In theory target switch should be easy
2. But: compiler is only a part of the whole compilation chain
3. The Problem: provide paths to libraries, headers, linker, assembler, ...

Universal Driver: problem

Multiple solutions:

- -V, -B, -b gcc cmdline switches
- -m32, -m64, -mthumb, ...

Universal Driver: problem

Multiple solutions:

- -V, -B, -b gcc cmdline switches
- -m32, -m64, -mthumb, ...
- multilibs

Universal Driver: problem

Multiple solutions:

- -V, -B, -b gcc cmdline switches
- -m32, -m64, -mthumb, ...
- multilibs
- Apple's driver-driver

Universal Driver: problem

Multiple solutions:

- -V, -B, -b gcc cmdline switches
- -m32, -m64, -mthumb, ...
- multilibs
- Apple's driver-driver
- ....

These all are approximations!

Universal Driver: model

Idea:

- Single entry point to compiler regardless of target and mode

Universal Driver: model

Idea:

- Single entry point to compiler regardless of target and mode
- User specifies just a “configuration”:

```
clang --config=arm-cortex-a9-baremetal foo.c
clang --config=cortex-m4-my-toaster morning-food.c
```

Universal Driver: model

Idea:

- Single entry point to compiler regardless of target and mode
- User specifies just a “configuration”
- Configuration defines all stuff like paths, includes, libs, default cmdlines, etc..
- Configuration might be system-wide or user-wide

Universal Driver: model

Idea:

- Single entry point to compiler regardless of target and mode
- User specifies just a “configuration”
- Configuration defines all stuff like paths, includes, libs, default cmdlines, etc..
- Configuration might be system-wide or user-wide

Large and open-ended project!

Compiler-RT

1. Low-level compiler support library:
   - Routines for e.g. 64 bit arithmetic on 32 bit targets
   - Optimized versions for common stuff
2. Same as libgcc for gcc

Compiler-RT: done

1. Surely we have all "common" libgcc functions:
   - 64 bit arithmetics
   - Soft floating point ops routines
   - Multiplication, division, fp - int conversion ...

Compiler-RT: done

1. Surely we have all "common" libgcc functions:
   - 64 bit arithmetics
   - Soft floating point ops routines
   - Multiplication, division, fp - int conversion ...
2. (Some) EABI functions

Compiler-RT: todo

1. “Unusual” EABI functions

Compiler-RT: todo

1. “Unusual” EABI functions
2. Exception handling runtime

Compiler-RT: todo

1. “Unusual” EABI functions
2. Exception handling runtime

Time to open source / relicense EHEGI ? :)

Compiler-RT: todo

1. “Unusual” EABI functions
2. Exception handling runtime
3. Something else?

LLVM backend

1. Modern design
   - Some parts are quite ‘unique’
2. 3 types of IR:
   - SDAGs
   - MachineInstr (MI)
   - MachineCode (MCInst)

LLVM backend: status

1. Pretty stable, can codegen many apps

LLVM backend: status

1. Pretty stable, can codegen many apps
2. Supports ARMv5+, Thumb, Thumb2
3. Supports VFPv3, NEON

LLVM backend: status

1. Pretty stable, can codegen many apps
2. Supports ARMv5+, Thumb, Thumb2
3. Supports VFPv3, NEON
4. Scheduling for Cortex-A8, -A9

LLVM backend: status

1. Pretty stable, can codegen many apps
2. Supports ARMv5+, Thumb, Thumb2
3. Supports VFPv3, NEON
4. Scheduling for Cortex-A8, -A9
5. Emit calls to EABI functions + libgcc

LLVM backend: status

1. Pretty stable, can codegen many apps
2. Supports ARMv5+, Thumb, Thumb2
3. Supports VFPv3, NEON
4. Scheduling for Cortex-A8, -A9
5. Emit calls to EABI functions + libgcc
6. ARM JIT is broken

Performance Comparison

- Some benchmarks are biased towards different compilers and vice versa

Performance Comparison

- Some benchmarks are biased towards different compilers and vice versa
- We still have problems with mid-level optimizers

Performance Comparison

- Some benchmarks are biased towards different compilers and vice versa
- We still have problems with mid-level optimizers:
  - nullstone ratio is 83% out of 100%

Performance Comparison

- Some benchmarks are biased towards different compilers and vice versa
- We still have problems with mid-level optimizers:
  - nullstone ratio is 83% out of 100%
  - 2 tests there show ratio < 20%

Performance Comparison

Generic facts:

- In average we're quite good comparing with gcc and other compilers.

Performance Comparison

Generic facts:

- In average we're quite good comparing with gcc and other compilers.
- In some cases nice 30% speedup is seen

Performance Comparison

Generic facts:

- In average we're quite good comparing with gcc and other compilers.
- In some cases nice 30% speedup is seen
- Some cases produce 30-40% slower code than gcc: optimizer / codegen bugs

Performance Comparison

Generic facts:

- In average we're quite good comparing with gcc and other compilers.
- In some cases nice 30% speedup is seen
- Some cases produce 30-40% slower code than gcc: optimizer / codegen bugs
- LTO really helps! Can yield 50-60% speedup

LLVM backend: new in 3.0

1. Improvements in NEON codegen

LLVM backend: new in 3.0

1. Improvements in NEON codegen
2. Atomic stuff

LLVM backend: new in 3.0

1. Improvements in NEON codegen
2. Atomic stuff
3. Calls to EABI functions, not to libgcc ones

LLVM backend: new in 3.0

1. Improvements in NEON codegen
2. Atomic stuff
3. Calls to EABI functions, not to libgcc ones
4. Completely new regalloc

LLVM backend: new in 3.0

1. Improvements in NEON codegen
2. Atomic stuff
3. Calls to EABI functions, not to libgcc ones
4. Completely new regalloc
5. Better inline asm handling (constraints)
6. Co-processor intrinsics

MC: status

1. ‘integrated-as’ enabled by default for X86 (on ELF and Darwin)

MC: status

1. ‘integrated-as’ enabled by default for X86 (on ELF and Darwin)

- ARM binary code emission is incomplete

MC: status

1. ‘integrated-as’ enabled by default for X86 (on ELF and Darwin)
   - ARM binary code emission is incomplete
2. Does not support bunch of directives
   - e.g. everything EH-related: .fnstart, .save, ...

MC: new in 3.0

1. Complete ARM assembler & disassembler support

MC: new in 3.0

1. Complete ARM assembler & disassembler support:
   - --integrated-as should (almost) work

MC: new in 3.0

1. Complete ARM assembler & disassembler support:
   - --integrated-as should (almost) work
2. MCJIT works up to some margin on arm-darwin:
   - Expression evaluation in LLDB

MC: still missed

1. MCJIT on non-MachO systems
2. (some) TLS
3. Direct object code emission: ELF on ARM
4. Switching between ARM / Thumb in one compilation unit

Your help is needed!

1. MCJIT
2. ELF direct object code emission on ARM
3. ARM EHABI + runtime library
4. Different modes & components
5. Verification of codegen & stuff
6. Codegen for size

LLVM: Overall view

- LLVM: 600k modern C++ LOC
- clang: 450k modern C++ LOC

LLVM: Overall view

- LLVM: 600k modern C++ LOC
- clang: 450k modern C++ LOC
- ~200 committers, ~90 active right now

LLVM: Overall view

- LLVM: 600k modern C++ LOC
- clang: 450k modern C++ LOC
- ~200 committers, ~90 active right now
- liberal license, no single copyright holder (e.g. FSF in gcc case)

Usage models

Track LLVM releases

Track top-of-the-tree

Usage models

Track releases

- Pros:
  - Large distance between necessary code updates
  - Releases are usually well-tested

Usage models

Track releases

- Pros:
  - Large distance between necessary code updates
  - Releases are usually well-tested
- Cons:
  - Large distance between necessary code updates
  - ! Big changes required
  - Need to backport fixes to private branch
  - ! In most cases nontrivial and time-consuming

Usage models

Track releases

- Pros:
  - Large distance between necessary code updates
  - Releases are usually well-tested
- Cons:
  - Large distance between necessary code updates
  - ! Big changes required
  - Need to backport fixes to private branch
  - ! In most cases nontrivial and time-consuming

LLVM releases are time-based, not feature-based

Usage models

Track top-of-the-tree

- Pros:
  - Fast mainline source updates
  - Always work with fresh mainline source
  - No need for bugfixes backport

Usage models

Track top-of-the-tree

- Pros:
  - Fast mainline source updates
  - Always work with fresh mainline source
  - No need for bugfixes backport
- Cons:
  - Fast mainline source updates
  - Frequent code updates necessary due to mainline change
  - Possibility to “catch” mainline bug

Usage models

Track top-of-the-tree

- Pros:
  - Fast mainline source updates
  - Always work with fresh mainline source
  - No need for bugfixes backport
- Cons:
  - Fast mainline source updates
  - Frequent code updates necessary due to mainline change
  - Possibility to “catch” mainline bug

One can make own releases when necessary

Ways of contributing

Why contribute at all?

- Make someone else support your code: reduce maintenance costs
- Provide tests for “interesting” cases and make sure mainline is bug-free on them
- Add a possibility for the code extension / fixing by the community

How to contribute

Patch submission

Commit-after-review model:

- Submit patch to mailing list
- Iterate until accepted
- In the end someone will commit the patch

How to contribute

Patch submission

Commit-before-review model:

- Code owners
- Significant contributions to specific field
- Trivial stuff

Standard Rules & Tricks

Patch submission

- Make small incremental checkins: much easier to review and show the actual progress
- Try to discuss huge changes in the ML beforehand
- Track what the other parties do: sometimes it's possible to split (or even eliminate!) tasks
- Make sure there are no layering violations across the libraries

Vendor-specific stuff

How to get your extension accepted?

- Think whether it's possible to make the extension target-neutral (ex: naked functions)
- Make sure extension is good factored and won't interfere with other code
- Provide exhaustive testsuite, so noone will break your code
- If possible: discuss the changes in ML beforehand

http://clang.llvm.org/get_involved.html

Working with ToT

How to track mainline sources?

git & git-svn:

- Pull code into your working copy
- Much easier branching & rebasing
- Allows to pull different versions of mainline
- Public git mirror (with svn metadata) is available

Interoperability Problems

1. Documentation:
    - ARMv6+ Architecture Reference Manuals are closed

Interoperability Problems

1. Documentation:
    - ARMv6+ Architecture Reference Manuals are closed
    - Few information about “internals”, e.g. pipeline stages, bypasses, functional units reservation, etc.

Interoperability Problems

1. Documentation:
   - ARMv6+ Architecture Reference Manuals are closed
   - Few information about “internals”, e.g. pipeline stages, bypasses, functional units reservation, etc.
2. Information about new cores is closed

Interoperability Problems

1. Documentation:
   - ARMv6+ Architecture Reference Manuals are closed
   - Few information about “internals”, e.g. pipeline stages, bypasses, functional units reservation, etc.
2. Information about new cores is closed

   ... but appears in gcc earlier than on arm.com

Interoperability Problems

1. Documentation:
   - ARMv6+ Architecture Reference Manuals are closed
   - Few information about “internals”, e.g. pipeline stages, bypasses, functional units reservation, etc.
2. Information about new cores is closed

   ... but appears in gcc earlier than on arm.com
3. Slow / no responses to e-mails

Q & A