

Open Source Core Toolchains ARM Status Update

Matthew Gretton-Dann
March 2015

Important Note: This session is being recorded
and will be made publicly available

Open Source Core Toolchains ARM Status Update

Matthew Gretton-Dann
March 2015

Agenda

- Introduction
- GNU Toolchain
 - Progress since January 2015
 - Plans for mid 2015
- LLVM Update
 - Progress since January 2015
 - Plans for mid 2015
 - LLVM Highlights

Purpose of this Presentation

- Explain what ARM plans to work on, and what its current priorities are:
 - However, things are likely to change – so:
 - We will not achieve all this in the next six months,
 - And there will be other things we do do.
 - This is an update of the presentation given just before the end of 2014
 - If your plans include the same topics, or work in the same areas
 - Come and talk to us – we should work together
 - Preferably this conversation should happen in the appropriate upstream communities.
- If you feel that we're doing the wrong thing
 - Come and talk to us – we're happy to work out a better way forward
- If possible use the public mailing lists & bug databases to report issues
 - This is the best place to have the conversation about best ways forward.

Overview of Goals for 2015

- Support the Architecture & Cores
 - Teams are involved in development of new cores and architecture extensions
 - We will not discuss those here
 - However, we plan to upstream functionality as soon as possible after public announcements
- Support the Community
- Improve Performance:
 - Focus on Cortex-A57 performance improvement
 - Focus on a range of benchmarks, including industry standard CPU benchmarks.
 - We analyze benchmarks both:
 - for improvements we can make to the toolchains; and
 - to note any regressions and get them fixed in co-operation with the community

GNU Toolchain

Progress in early 2015

- Continuing to help prepare GCC 5 for release
- A-Profile
 - Scheduling improvements for Cortex-A57
 - Basic Cortex-A72 tuning
 - LDP / STP merging
 - Improved Neon intrinsics support along with community
 - Simplify a number of the default string routine implementations in glibc
- R/M-Profile
 - Thumb-I Prologue/Epilogues in RTL
 - Implemented aeabi_memclr/aeabi_memset/aeabi_memmov in Newlib
 - Fixed ABI support for HW vs SW floating-point

GNU Toolchain

Next steps

- Help complete GCC 5 release
- PR62173 – ivopts in GCC
 - Improves lbm in spec2006
- Cost model improvements for Cortex-A53 / Cortex-A57 (AArch64)
- float16 support in GCC
- Improve CSEL code generation
- Finish prologue / epilogue code generation improvements
- ARMv8.1-A Architecture support
- Implement floating-point conversions for M-Profile in libgcc
- Use tree matching to optimize CRC functions on Cortex-M7
- Improved fdiv & fmul instruction selection on Cortex-M series

LLVM Toolchain

Progress since January 2015

- Cortex-A72 basic support
- Support for `thread_local` in AArch64
 - Re-enables AArch64 bootstrap after LLVM started using `thread_local`.
- Issues related to building AOSP with LLVM
 - AArch32 large dynamic stack re-alignment fixed.
 - AArch64 dynamic stack re-alignment is in progress.
- AArch32 build attribute generation - now believed to be fully ABI compliant
- Improve unrolling heuristics for AArch64 – improves performance
- Improved support for ARMv7-A and ARMv7-M
 - We believe encoding/decoding and assembling/disassembling instructions in LLVM MC is now complete for ARMv7-A & ARMv7-M.

LLVM Toolchain

Currently in Progress

- ARMv8.1-A architectural support
- Issues related to building AOSP with LLVM:
 - Stack size usage optimization ("lifetime markers")
- Vectorization improvements:
 - Make use of structure load/store instructions in AArch32 and AArch64.
 - Enable vectorization of more min/max idiomatic code.
- Discussions on improving both recursive inlining and the heuristics in the top-down inliner.
- New loop rerolling pass added
 - Not yet enabled by default

LLVM Toolchain

Currently in Progress

- New float2int pass: turn floating point into fixed point arithmetic.
- PBQP register allocator: various bug fixes and optimizations.
- Aligning objects so that higher-performing mem* operations are possible
 - Thumb-I done
 - A32/T32 ongoing
- Unrolling heuristics tuned for AArch32

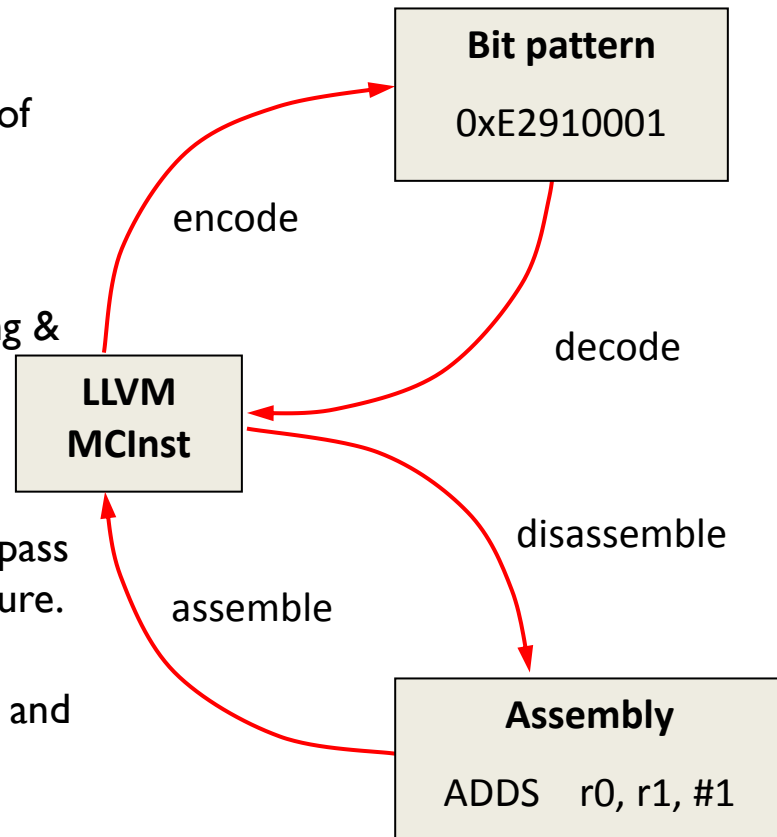
LLVM Toolchain

Next to be looked at

- Further performance tuning
 - Specifics to be decided based on benchmark analysis
- Set up a public performance-tracking AArch64 Linux BuildBot.
- Issues related to building AOSP with LLVM:
 - See https://llvm.org/bugs/show_bug.cgi?id=21420
- Improve testing for ARMv6-M and Thumb-I
 - Both in-house and public
 - Fix issues found

LLVM Toolchain – highlight 1: fully passing MC Hammer for v7

- MC Hammer test-suite performs exhaustive testing of MC-layer correctness versus a reference implementation. See Euro-LLVM 2012 presentation by Richard Barton.
- Checks correctness of encoding, decoding, assembling & disassembling functionality.
- Over the past few years – we've been gradually fixing corner case bugs.
- Late last year, we've made MC Hammer testing fully pass for the v7-A and v7-M variants of the ARM architecture. AArch64 was already passing.
- Result: This functionality in LLVM-MC can be trusted and built upon.



LLVM Toolchain – highlight 2: auto-vectorizing structure LD/ST

```
for (int i = 0; i < len; ++i)
    out[i] = in[i].r * 0.3f +
             in[i].g * 0.59f +
             in[i].b * 0.11f;
```

- LLVM's autovectorizer currently does not support vectorizing strided accesses.
- We're currently discussing the design to allow this in LLVM, ultimately leading to autovectorizing code similar to the example on this slide.

[illegible]

1d3 {v19.16b-v21.16b}, [x7]

