HKG18-215: Building AOSP master with Clang master

MinSeong Kim
Building AOSP master with Clang master

- Prebuilt clang in AOSP lag behind in time 3 to 6 months
  - **Purpose**
    - Check clang master’s performance and size impact on AOSP
    - Help clang master to soft-land on AOSP when the time comes

- **CI 1: Clang master build**
  - [https://ci.linaro.org/job/android-clang-toolchain/](https://ci.linaro.org/job/android-clang-toolchain/)
  - Build command: toolchain/llvm_android/build.py

- **CI 2: AOSP master build using clang from CI 1**
  - [https://ci.linaro.org/job/android-master-clang/](https://ci.linaro.org/job/android-master-clang/)
  - Minor patches to AOSP: tuning compiler flags to bypass newly added errors/warnings
  - If bugs are found, fix/report them

- CI jobs often fail as two moving targets → requires constant maintenance

- Upon success, boot-up test on hikey using LAVA
Performance and Size comparisons

● Size
  ○ Not much: Roughly 0.1% difference in size (even only ELF files)

● Performance
  ○ Sometimes, no visible performance increase compared to prebuilt clang
  ○ Some other times,
    ■ Quadrant Pro: 12649 → 12990 (2.7% up)
    ■ Antutu 6.0: 31605 → 32661 (3.3% up)
    ■ Geekbench 3: 2597 → 2597 (0%)
    ■ Andebench Pro 2015: 3344 → 3831 (13% up)
  ○ Fortunately, haven’t seen cases with performance and size regression so far
Clang master with new techniques for AOSP

- **LLD (LLVM Linker: a drop-in replacement for GNU linkers)**
  - Current default linker for aarch64 AOSP: gold
  - Tried to replace gold to LLD for linking AOSP
  - Found a few issues (3 missing options and some compatibility issues)
  - system.img size increases by roughly 7 or 8%

- **Souper (Peep-hole optimizer on LLVM IR)**
  - Performs missed simplifications on the final LLVM IR.
  - Applied to bionic and benchmarked - no visible performance increase

- **Polly (Loop and data-locality optimizer)**
  - 3 major options to turn on its optimizations
    - Scalar, OpenMP, and Vectorizer
  - No visible performance increase (e.g. Antutu 6.3)
Building kernels, bootloaders and TEEs with Clang

Bernhard “Bero” Rosenkränzer
Building kernels with clang

- Mainline kernels are looking good… Except for some rarely used modules (exofs, gcc code analysis plugins), kernels can build with clang 6.0 on aarch64 and x86_64.
  armv7 is a little behind due to more inline assembly that clang’s internal assembler doesn’t like, but isn’t far from working.
- Relevant patches have been backported to most android-common trees that are still relevant
- Right now, building the kernel with gcc is getting a lot more testing; for production builds, it is probably still a good idea to use gcc. Latest version from TCWG is recommended over the outdated 4.x based AOSP gcc toolchain.
U-boot still contains many gcc hardcodes, from compiler naming conventions ($(CROSS_COMPILE)-gcc) to compiler flags (-ffixed-r9) to inline assembly (incorrect arguments to msr/mrs).

It’s clearly possible to fix, but will take some more time.

Are there other bootloaders we should look into? Tianocore?
Building TEEs with clang

- OP-TEE Makefiles hardcode gcc-isms, and OP-TEE includes some code (Linux 4.12, grub, …) that can’t currently be built with clang. Many other components contained in OP-TEE work great when built with clang already.
- Trusty Makefiles also hardcode gcc-isms, the lk kernel isn’t ready to be built with clang, other than that a lot of code is shared with AOSP (where it builds with clang perfectly).
- It should be possible to get both TEEs to be buildable with clang, but it will be a bit of an effort.
Thank You

Questions/Comments:
minseong.kim@linaro.org, bero@linaro.org

#HKG18

HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org