



Building Clang/LLVM efficiently

Tilman Scheller
LLVM Compiler Engineer
t.scheller@samsung.com

Samsung Open Source Group
Samsung Research UK

EuroLLVM 2015
London, United Kingdom, April 13 – 14, 2015

Overview



- Introduction
- Performance
- Summary

Introduction



Introduction



- Clang/LLVM are large pieces of C++ software
- Long turnaround time harms developer productivity
- Squeeze maximum performance out of the toolchain
- Speed up the build without buying new hardware
- Focus on x86-64 Linux
- Test machine: Fedora 21 on i7-4770K CPU @ 3.50GHz, 16GB RAM, 1TB 7200RPM HDD



Ideas



- Build with Clang instead of GCC
- Only build what you really need (e.g. only build relevant backends)
- Use a faster linker: GNU gold instead of GNU ld
- Use a heavily optimized compiler binary for the build (LTO, PGO, LTO+PGO)
- For incremental debug builds: shared library build instead of static build

Performance



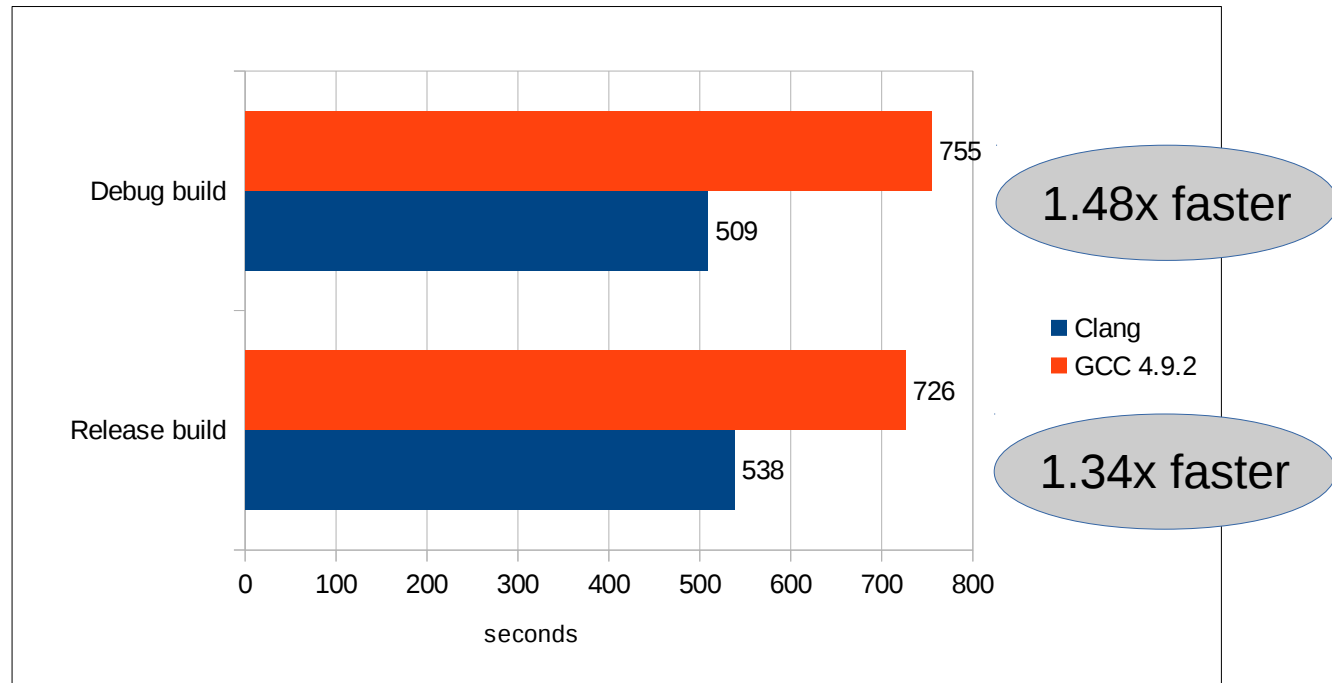


Measurement

- GCC 4.9.2 shipping with Fedora 21
- Clang trunk snapshot (r234392 from April 8, 2015)
- Standard debug/release builds of Clang (CMake build with default generator) unless otherwise noted
- Using GNU gold 2.24 for linking
- Make invoked with -j8
- Best of five runs

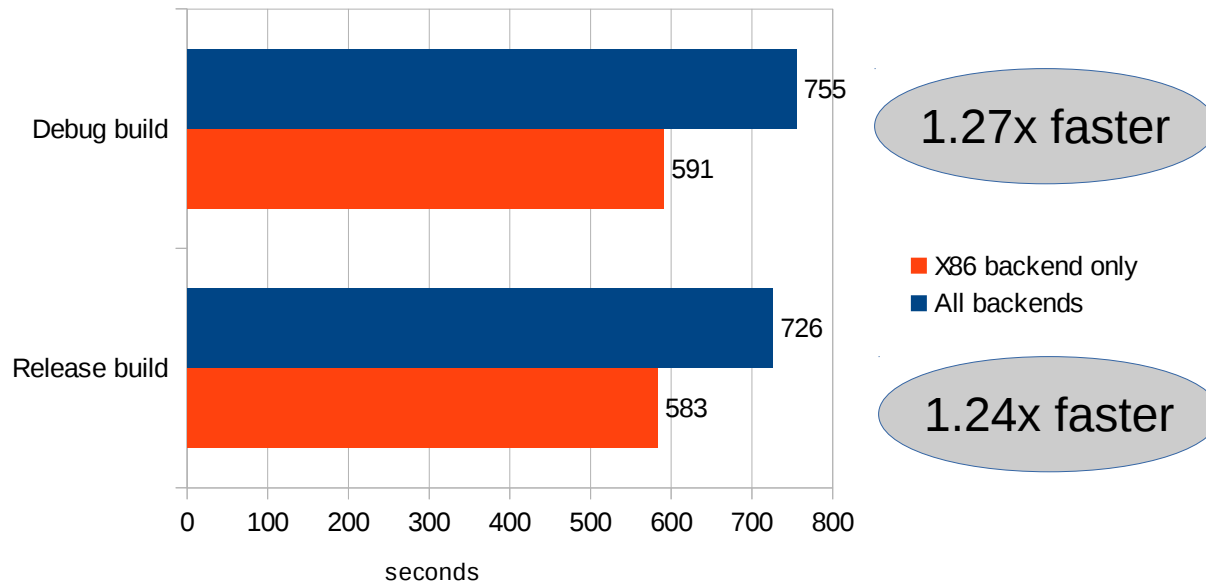
Clang vs. GCC compile time

- Both Clang and GCC are compiled with GCC 4.9.2



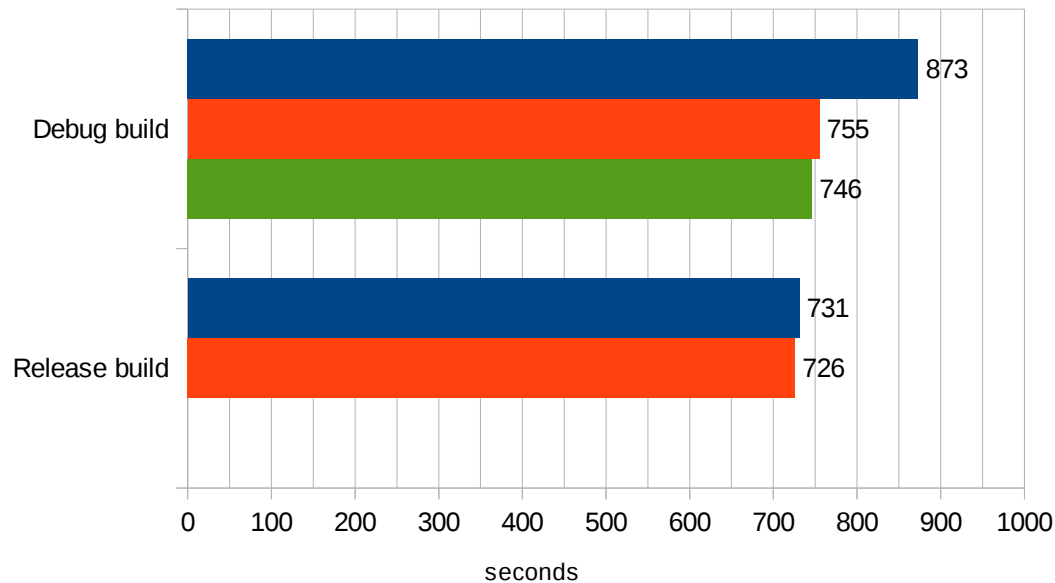
Only build a subset of Clang/LLVM

- Only build relevant backends
- Host Clang built with GCC 4.9.2



Use a faster linker

- Using GNU gold instead of GNU ld
- Building Clang with GCC 4.9.2

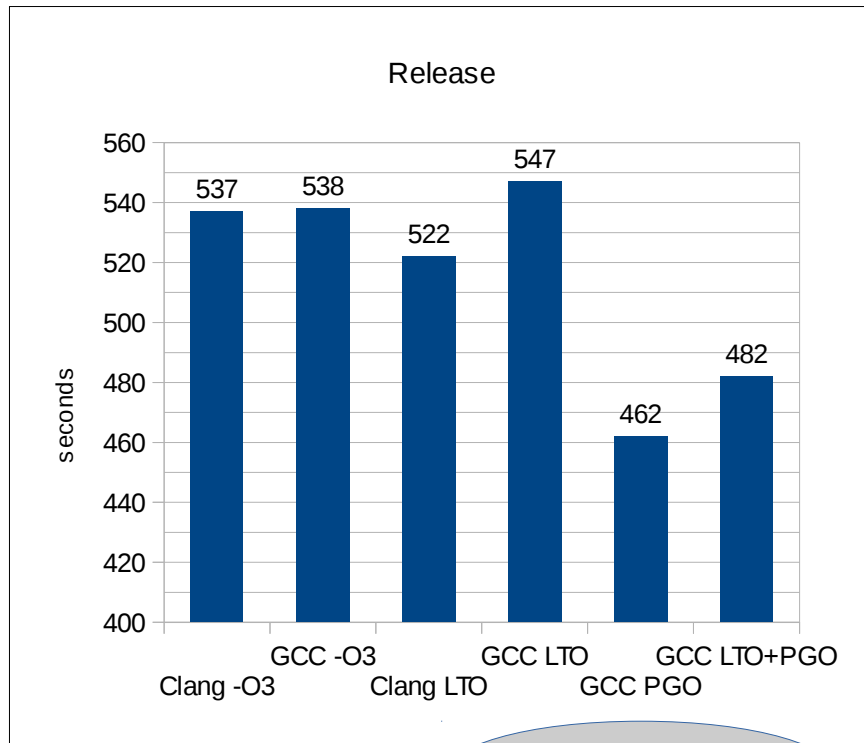


1.17x faster

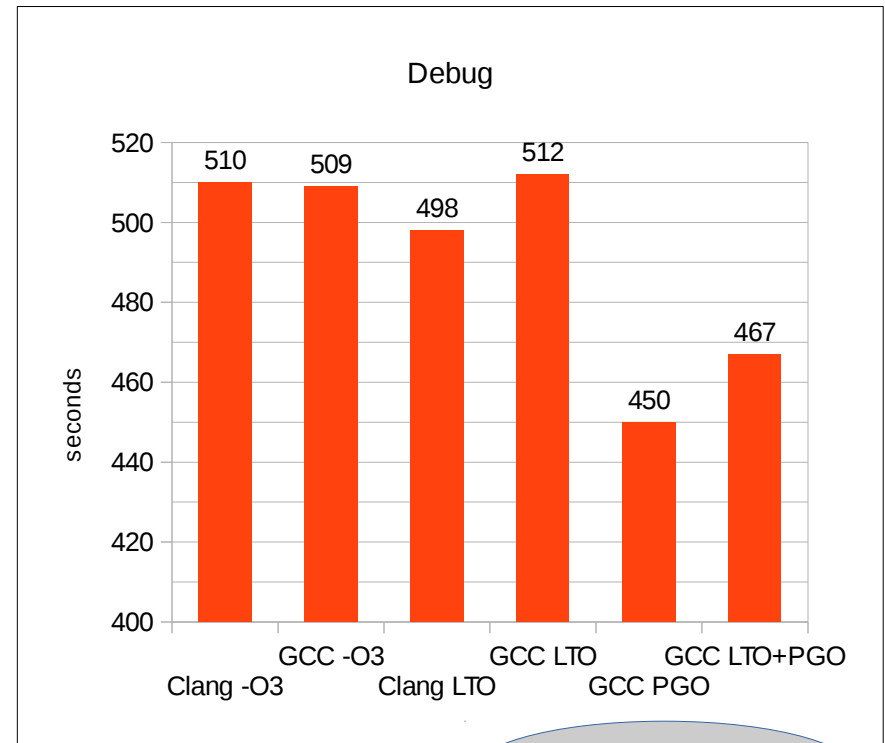
- GNU gold + split DWARF
- GNU gold
- GNU ld

Optimize host compiler binary aggressively

- Building Clang with a heavily optimized host Clang build
- Host Clang was compiled with Clang or GCC at the various different optimization levels



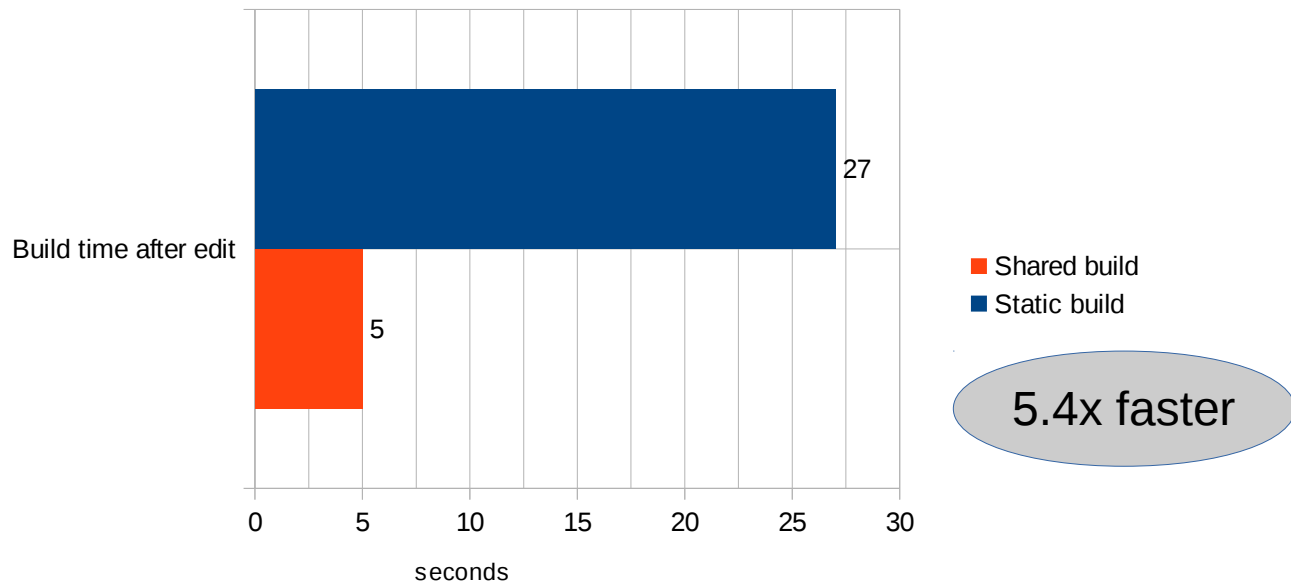
PGO 1.16x faster



PGO 1.13x faster

Speeding up incremental builds

- Shared library build rather than static build
- Debug build of Clang with GCC 4.9.2
- Incremental build after changing a single file in the ARM backend (ARMISeLowering.cpp)



Summary



Summary



- Always build with Clang if you care about compile time
- Use GNU gold rather than GNU ld
- Building Clang with GCC and PGO produces fastest binary
- Shared library builds speed up incremental debug builds tremendously
- Overall speedup: 1.58x for release builds and 1.94x for debug builds (Switching from GCC + GNU ld to a PGO build of Clang + GNU gold)



Thank you.

