Here be dragons: Using clang/LLVM to build Android

Presented by:

Behan Webster

(LLVMLinux project lead)

Presentation Date: 2014.05.01

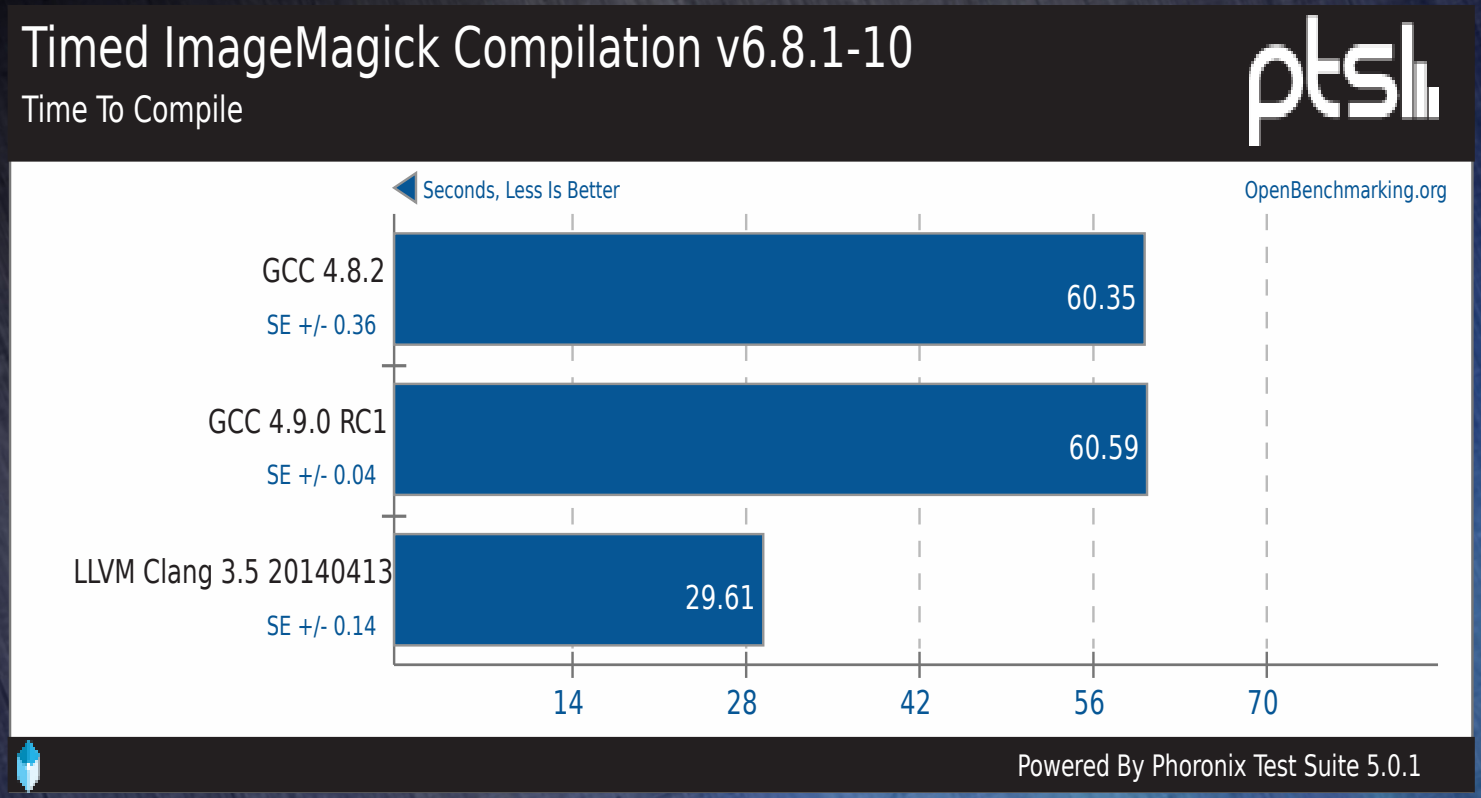
Clang/LLVM

- LLVM is a Toolchain Toolkit (libraries from which compilers and related technologies can be built)
- Clang is a C/C++ toolchain

Fast Moving Project

- In just a few years LLVM and Clang have reached and in some cases surpassed what other toolchains can do
- Written in C++ which lends itself to easy extension
- Inclusive community of developers
- Similar size and speed of resulting binaries to gcc

Compile times clang 3.5 vs gcc 4.9



http://www.phoronix.com/scan.php?page=article&item=gcc49_compiler_llvm35&num=1

One Toolchain

- LLVM is already being used in a lot of domains:
  - DSP, GPU, CPU, JIT, etc.
  - Camera, audio, video, CUDA, Renderscript, kernel, Android, applications, documentation
- Compiler extensions only need to be written once
- For companies working on a range of technologies it's convenient to only need maintain/test a single toolchain

LLVM License

- Licensed under the "UIUC" BSD-Style license
- LLVM technology can be embedded into into non-GPL software
- Allows open and proprietary extensions
  - This is attractive to some companies
- Wider development audience
- Even more full-time developers making it better

Driving Change in gcc

- Better error reporting
- Fix-it hints (guessing what the code should be)
- Macro expansion in error messages
- Color syntax highlighting in error messages (gcc v4.9)
- Address Sanitizer

Other Interesting LLVM Related Projects

- Clang is one of the Android NDK compilers
- Renderscript in Android is based on LLVM
- Official commercial compiler from ARM is based on clang/LLVM
- Clang Static Analyzer
- Energy consumption analysis of programs using LLVM
- llvmpipe (Galium3D)
- CUDA
- OpenCL (most implementations are based on LLVM)
- Code transformation tools

LLVMLinux Project Goals

- Fully build the Linux kernel for multiple architectures, using the Clang/LLVM toolchain
- Discover LLVM/Kernel issues early and find fixes quickly across both communities
- Upstream patches to the Linux Kernel and LLVM projects
- Bring together like-minded developers
- Enable the kernel community to do more in depth analysis of the kernel code

Other Avenues of Interest

- Compiling the Android kernel and AOSP with clang
- Supporting Linaro LLVM and AOSP teams
- Clang Static Analysis of the Linux kernel
- Kernel specific Checkers (GSoC)
- Building better tools based on LLVM for the kernel community

Compiling AOSP with clang

- You can build a single project within Android by setting LOCAL_CLANG=true in Android.mk
- You can use clang globally by setting LOCAL_CLANG=true in build/core/clear_vars.mk

Clang in gcc-clothing

- Bernhard Rosenkränzer wrote a wrapper to make clang look like the Android-style gcc compiler
- http://git.linaro.org/git-ro/people/bernhardrosenkranzer/clang-wrapper.git
- Prebuilt binaries are built daily at:
- http://snapshots.linaro.org/components/toolchain/llvm-clang-trunk/latest
- The wrapper makes clang behave as if it were the android version of gcc (smoothing over various differences)
- Although originally required, LOCAL_CLANG=true now replaces this

arm-linux-androideabi

- Clang internally maps:

  arm-linux-androideabi → arm-linux-gnueabi
- However android differs from gnueabi by:
  - Forces all enum sizes to 32-bits (Dalvik requirement)
  - Defaults to generating Position Independent Code (PIC)
- As a result the clang wrapper adds:

  -fno-short-enums -fPIC

Nested Functions

- Used extensively in elfutils
- Patches to fix this won't be accepted upstream
- However since elfutils has moved to GPLv3 license, the code will either have to be forked or removed from Android

Variable Length Arrays In Structs

- VLAIS isn't supported by Clang (gcc extension)

```
char vla[n];                    /* Supported, C99/C11 */

struct {
   char flexible_member[]; /* Supported, C99/C11 */
} struct_with_flexible_member;

struct {
   char vlais[n];    /* Explicitly not allowed by C99/C11 */
} variable_length_array_in_struct;
```

- Used in skia (2D Graphics Library)

Variable-Length Arrays of Non-POD Elements

- The C++ standard doesn't allow for Variable-Length Arrays of non-POD (Plain-Old-Data), but gcc does.
- frameworks/base/libs/hwui/OpenGLRenderer.cpp:

  AAVertex wLines[verticesCount];
- Turn it into AAVertex *wLines, allocate with new/delete[]

Different Symbol Visibility/Coexistence

- __kernel_sindf(double) and friends in Bionic lead to clashes because they're defined multiple times (header included by several files), causing a fatal error with clang
- The fix is to declare them static inline
- Similar problems in rotate270 and friends in Galley2's JNI code
- Also __weak_reference and __strong_reference need to be defined in asm code for clang

Different Symbol Resolution Rules

static const int digits10 = digits10<int, digits, is_signed>::value;

- Clang assumes the right-hand reference to “digits10” refers to the left-hand “static const int” definition – passing template parameters to an int
- It should be:

static const int digits10 = ::digits10<int, digits, is_signed>::value;

extern inline: Different for gnu89 and gnu99

- GNU89/GNU90 (used by gcc)
  - Function will be inlined where it is used
  - No function definition is emitted
  - A non-inlined function may also be provided
- GNU99/C99 (used by clang)
  - Function will be inlined where it is used
  - An external function is emitted
  - No other function of the same name may be provided.
- Solution? Use “static inline” instead.

Header Guard (mis)Detection

- libunwind.h and libunwind_j.h rely on having UNW_REMOTE_ONLY or UNW_LOCAL_ONLY defined

```
#ifndef UNW_REMOTE_ONLY
#define UNW_LOCAL_ONLY
#include <libunwind.h>
#include <libunwind_i.h>
[...]
#endif
```

- Since it looks just like a header guard, clang is doing the right thing warning about it - there's no way the compiler could tell this apart.
- A possible fix is using -Wno-header-guard
- The other option is moving code between the #ifndef and #define

Redefinition of recv in bionic

- recv is defined in both socket.h and recv.cpp
- clang points out this redefinition, whereas gcc doesn't
- If __BIONIC_FORTIFY is set we use the inline version, otherwise it uses the library version of recv
- Older code also uses the library version of recv
- The code in both places are currently the same, but this is really ugly
- The only way around this right now is using #ifdef magic

char* vs void*

```
dalvik/vm/compiler/Utility.cpp:412:29: error: cannot
initialize a parameter of type 'char *' with an rvalue of
type 'void *'
    __builtin___clear_cache(reinterpret_cast<void*>(start),
reinterpret_cast<void*>(end));
                            ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1 error generated.
```

- Clang's __builtin___clear_cache takes a char* parameters, gcc's takes void*
- The fix is to use char* which automatically converts to void*

Empty structs in C vs C++

```
external/libunwind/include/libunwind-x86.h:158:9:
error: empty struct has size 0 in C, size 1 in C+
+ [-Werror,-Wextern-c-compat]

typedef struct unw_tdep_save_loc
        ^
```

- (Similar issues in libunwind-arm.h)
- -Werror in clang has different warnings to gcc
- Add a dummy member to the struct to ensure its size is consistent across C and C++

Implicit Exception Specification Mismatches

```
bionic/libstdc++/include/new:16:7: error:
function previously declared with an explicit
exception specification redeclared with an
implicit exception specification
[-Werror,-Wimplicit-exception-spec-mismatch]
void operator delete(void*);
     ^
```

- clang warns about exception specification mismatches (even when using -fno-exceptions)
- Add throw() to the prototype

Command Line Option Spell Checking Fail

- `error: unknown warning option '-Wno-maybe-uninitialized'; did you mean '-Wno-uninitialized'? [-Werror,-Wunknown-warning-option]`
- Clang tries to be helpful by suggesting command line option for options it doesn't recognize
- In this case clang guesses wrong for a gcc specific compiler flag used in libunwind and skia
- The fix is to add an “ifneq ($(LOCAL_CLANG),true)” wrapper

C++98 vs C++11 Issues

- rvalue references are used in chromium_org external
- This is a c++11 addition which isn't supported in c++98
- clang uses c++98 by default
- gcc does c++98 with backported features from c++11
- 2 solutions:
  - -std=c++11
  - -Wno-c++11-extensions

Current Status

- Image size is slightly larger than when built with gcc 4.9
- Build time is significantly faster (60min vs. 90min)
- However it doesn't fully boot
- Seems to be an Android userland issue in a component needed very early (possibly Bionic or init)
- Reasons for this boot failure still need to be found

Testing/Benchmarks: CTS

- Android Compatibility Test Suite
- Clang compiled kernel with gcc compiled Android userspace

Compiler	Passed	Failed	Not Executed
clang	14463	3470	46
gcc	14461	3472	46

Testing/Benchmarks: Antutu

Android 4.4.2	*clang*	*% diff*	*gcc*
AOSP on Grouper	**13848**	**99.6%**	**13904**
Multitask	2862	100.8%	2838
Dalvik	1030	98.9%	1041
CPU integer	1941	99.5%	1951
CPU float	1317	100.0%	1317
RAM Operation	1542	100.9%	1529
RAM Speed	473	100.0%	473
2D graphics	818	98.6%	830
3D graphics	2286	95.6%	2392
Storage I/O	954	105.1%	908
Database I/O	626	100.2%	625

Not entirely scientifically gathered data

Gather your own data and average over a large number to get more accurate numbers.

Data hurriedly generated right before this conference...

Is this enough qualification for you?

Integration with LAVA

- The LLVMLinux Project is working with Linaro to integrate a Clang compiled kernel with Linaro's extensive HW based LAVA test system
- Currently working on Vexpress HW
- This will eventually include testing on both Android and non-Android targets (though for the moment with a gcc compiled userspace)
- The LLVMLinux project already has tested kernels for a number of Android based devices

Embrace the Dragon. He's cuddly.

Thank you

http://llvm.linuxfoundation.org

Special thanks to

- Bernhard Rosenkränzer
- Renato Golin
- Vinicius Tinti
- Mark Charlebois
- Jan-Simon Möller

Contribute to the LLVMLinux Project

- Project wiki page
  - http://llvm.linuxfoundation.org
- Project Mailing List
  - http://lists.linuxfoundation.org/mailman/listinfo/llvmlinux
  - http://lists.linuxfoundation.org/pipermail/llvmlinux/
- IRC Channel
  - #llvmlinux on OFTC
  - http://buildbot.llvm.linuxfoundation.org/irclogs/OFTC/%23llvmlinux/
- LLVMLinux Community on Google Plus