Multiarch crossbuilding
How to use it, and what still needs work

Wookey

The Cross-building victim

MultiarchCross

- Historical Context
- Autobuilder
- Toolchains and $stuff
- Multiarch for cross-deps
- Examples of things that break
- Current Status & Outstanding issues
- Bootstrapping

Outline

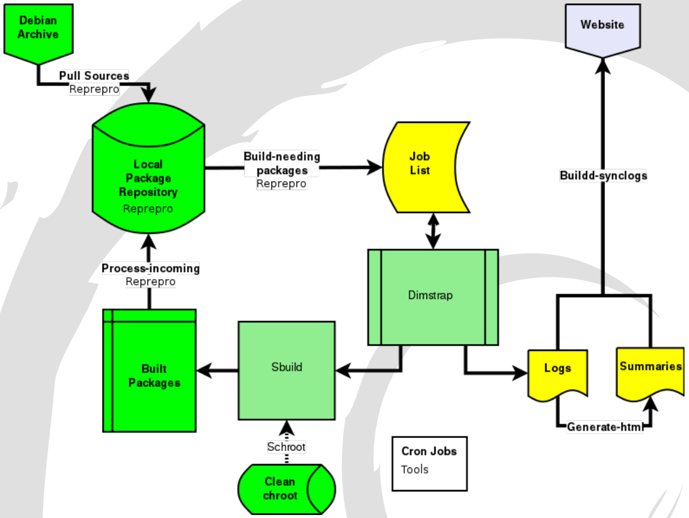
Nomenclature

`Build`	:	Machine/architecture you are building on
`Host`	:	Machine/architecture package is being built for
`Target`	:	Machine/architecture a compiler generates code for

Potted History

- 1997 - dpkg-cross (Roman hodek, Dave Schleef, Nikita Youschenko, Neil Williams)
- 2003 - emdebian cross-toolchains (Wookey, Hector Oron)
- 2004 - apt-cross
- 2007 - xapt, pdebuild-cross
- 2009 - chromiumos-build $\rightarrow$ xdeb
- 2010 - linaro cross-toolchains (Marcin)
- 2011 - cross-build daemon
- 2012 - sbuild cross-support
- 2012 - multiarch-built cross-toolchains (Thibault Girka)

Cross Build Daemon



xbuilder package in Linaro PPA

`https://gitorious.org/debian-bootstrap/gsoc2013`

Cross Build Daemon - Stats

- http://people.linaro.org/~wookey/buildd/
- http://people.canonical.com/~cjwatson/cross/armhf/raring/

Distro	Updated	Total	Builds	Fails	Deps
Sid	Feb 2013	99	27	6	65
Raring	July 2013	1362	464	332	566
Quantal	dead	93	37	24	32
Precise	dead	94	51	18	25

Parts needed

- Toolchain
- Cross-build-dependencies
- Build-system helpers (autotools, cmake)
- Avoid running wrong-arch binaries (or qemu)

Toolchain

There are 2 aspects to multiarching toolchains

- System search paths
  - path for libs and system headers (<> includes)
  - Previously previously /usr/include/ (native), /usr/<triplet>/include (cross)
  - Now always /usr/include/<triplet>:/usr/include/
  - Previously previously /usr/lib/ (native), /usr/<triplet>/lib (cross)
  - Now always /usr/lib/<triplet>:/usr/lib:/lib/<triplet>:/lib
- Build mechanism
  - Emdebian dpkg-cross libc6 for armel to make libc6-armel-cross arch all
  - Multiarch Depend on libc6:armel (libgomp:armel, libmudflap:armel, etc)

Autoconf caching

dpkg-cross provides `/etc/dpkg-cross/cross-config.cache` and `/etc/dpkg-cross/cross-config.<arch>`

- ac_cv_sizeof_float=4
- coreutils gl_cv_func_fstatat_zero_flag=yes
- dbus ac_cv_have_abstract_sockets=yes
- shadow ac_cv_func_setpgrp_void=yes
- bash bash_cv_job_control_missing=present
- sudo sudo_cv_func_unsetenv_void=no

Bits and Bobs

Other things are needed for a smooth experience

- build-essential-<arch> packages (waiting for toolchains)
- <triplet>-pkg-config
- toolchain defaults links (arm-linux-gnueabi-gcc → arm-linux-gnueabi-gcc-4.7)
- autoconf cache (in dpkg-cross)
- cmake TOOLCHAIN file (in dpkg-cross)
- <triplet>-tools
- sbuild support (upstream)
- scons? ant? MakeMaker? improvements ...

Actually building

Build locally:

- sbuild –host <arch>

or in a chroot:

- sbuild –host <arch> -d <distro> <package>_<version>

1. chroot into <distro> (clean) chroot
2. update/upgrade
3. dpkg –add-architecture <arch>
4. apt-get install crossbuild-essential-<arch>
5. apt-get -a <arch> build-dep <package>
6. DEB_BUILD_OPTIONS=nocheck CONFIG_SITE=/etc/dpkg-cross/cross-config.<arch> dpkg-buildpackage -a <arch>
7. clean up after build (throw away chroot)

Crossable Build-deps

crossbuild-essential is a bit hacky
Empty package depending on: gcc-<triplet>, g++-<triplet>, pkg-config-<triplet>, libc-dev:<arch>, build-essential:native

Some build-deps change name

binutils → binutils-<triplet>
gcc-4.6 → gcc-4.6-<triplet>
pkg-config → pkg-config-<triplet>
g-ir-scanner → g-ir-scanner-<triplet>

Propose:
Cross-name: Yes
in control file

Outline

Cross Toolchains in the archive!

- Emdebian toolchains broken in wheezy
- Emdebian toolchains don't use multiarch paths
- Ubuntu has had them for ages
- Linaro cross-toolchains don't use multiarch paths
- Some people want bare-metal cross-toolchains (and libcs)

Toolchain Bootstrap

'3-stage' bootstrap

1	Linux	linux-libc-dev headers
2	GCC stage1	Bare C-compiler
3	eglibc stage1	Minimal libc
4	GCC stage2	C-compiler against eglibc
5	eglibc stage2	Full libc build (without libselinux)
6	GCC stage3	All compilers

Currently automated by arm64-cross-toolchain-base
armhf-cross-toolchain-base

Multiarch Toolchain build

- Source: linux ⇒ linux-libc-dev:any (already true)
- Source: binutils-cross ⇒ binutils-<triplet> for all arches
```
TARGET=armel dpkg-buildpackage -d -T control-stamp
TARGET=armel dpkg-buildpackage -b
```
- Source: gcc-4.8 ⇒ gcc-4.8:any, libgcc1:any, libstdc++:any (already true)
- Source: gcc-4.8-<triplet> ⇒ builds gcc-4.8-<triplet>:any
```
DEB_TARGET_ARCH=armel with_deps_on_target_arch_pkgs=yes
dpkg-buildpackage -d -T control
DEB_TARGET_ARCH=armel with_deps_on_target_arch_pkgs=yes
dpkg-buildpackage -b
```
  Build-depends: gcc-4.8-source, libc6-dev:<arch>, libgcc1:<arch>, <libstdc++-dev>:<arch>

Considerations

- Two libc, libstdc++, libgcc let configure find the wrong one
- Two libc, libstdc++, libgcc lets the linker use the wrong one
- 3-stage bootstrap slow and builds a pile of stuff we already have (kernel headers, libc, libgcc, libstdc++)
- Standalone compilers *are* useful for new arch, not yet in archive
- Needs multiarch-ready buildds

Source Build Depends

Binary-source packages are a horrid workaround
Build-depends: binutils:src would be big improvement
gcc-4.8-source is not the same as gcc-4.8:source (it's patched) What

would it take to fix?
What directory to install to?
Why not just allow apt-get source foo during build for now?

Consitent Target Arch Specifier

We have this right for HOST but not TARGET

Consistent target arch env var

binutils: TARGET=arm64
gcc: DEB_GCC_TARGET=arm64 or GCC_TARGET (now fixed)

Should be DEB_TARGET_ARCH=arm64 everywhere

Consistent dpkg-buildpackage usage

dpkg-buildpackage -a<DEB_HOST_ARCH>
dpkg-buildpackage –target-arch<DEB_TARGET_ARCH>

–target is already used
-t is already used
therefore: –target-arch

Co-installable Toolchains

`https://wiki.debian.org/CoinstallableToolchains`

Currently not possible to install gcc:i386 and gcc:amd64 together

Currently

gcc-<ver> contains the native compiler

gcc-<ver>-<triplet> contains a cross-compiler

Proposed

gcc-<ver>-x86_64-linux-gnu

gcc-<ver>-i386-linux-gnu

gcc-<ver>-arm-linux-gnueabihf

Status

- Multiarch-built toolchains done in GSOC 2012
- Upstreamed to gcc Dec 2012 (gcc4.7)
- Was building fine in gcc4.7
- Broke when I tried it on Friday
- Bootstrapping: stage1 toolchain in Debian has missing multiarch include path
- Ubuntu/Linaro packaging fails due to different kernel packaging and patches
- Better is enemy of ‘good enough’?
- I worry about transitions - should I?
- Bare-metal cross-toolchains in NEW (multilib, not multiarch)

Outline

Multiarch terminology

Multi-arch-ready packages are given an extra field Multi-Arch

- same: *(libraries)*
  can be co-installed and can only satisfy deps within the arch
- foreign: *(tools)*
  can not be co-installed can satisfy deps for any arch
- allowed: *(both)*
  can be either. Depending packages specify which is wanted

dpkg has support for reference-counting of (doc-)files from co-installable packages that overlap

Dependency satisfaction

```
dpkg --add-architecture armhf
apt-get build-dep -a armhf <package>
```

- Described at `https://wiki.ubuntu.com/MultiarchCross`

	Build-Depends: foo	Build-Depends: foo:any	Build-Depends: foo:native
no Multi-Arch field	DEB_HOST_ARCH	disallowed	DEB_BUILD_ARCH
Multi-Arch: same	DEB_HOST_ARCH	disallowed	DEB_BUILD_ARCH
Multi-Arch: foreign	any, pref DEB_BUILD_ARCH	disallowed	disallowed
Multi-Arch: allowed	DEB_HOST_ARCH	any, pref DEB_BUILD_ARCH	DEB_BUILD_ARCH

- So tools all need to be marked Multi-Arch: foreign (over 1000)
- Or implement #666772 *apt cross-build-dep handling should be liberal with Arch: all packages*

Transitive Build-deps

A package Build-depends: libdb-dev

```
Package: libdb-dev
Depends: libdb5.1-dev
```

libdb-dev used to be arch all. Now needs to be arch any to get libdb5.1-dev:DEB_HOST_ARCH

Outline

Crossbuilding Issues - Wrong arch tools

- libnih: /≪PKGBUILDDIR≫/nih-dbus-tool/.libs/lt-nih-dbus-tool: No such file or directory
- help2man Runs command –help to get manpage

Crossbuilding Issues - config scripts

Arch-dependent config scripts

- tcl8.5 /usr/lib/tcl8.5/tclConfig.sh
- curl /usr/bin/curl-config
- freetype /usr/bin/freetype-config
- guile /usr/bin/guile-config
- icu /usr/bin/icu-config
- krb5 /usr/bin/krb5-config
- pcre /usr/bin/pcre-config –libs
  → `-L/usr/lib/x86_64-linux-gnu -lpcre`
- apr /usr/bin/apr-config –cc
  → `x86_64-linux-gnu-gcc`

Crossbuilding Issues - cross-install failures

M-A: same packages which run foreign-arch binaries during install

- libgvc5: libgvc5-config-update
- libglib2.0-0: glib-compile-schemas, gio-querymodules (fixed)
- libgdk-pixbuf2.0-0: gdk-pixbuf-query-loaders
- libgtk2.0-0: gtk-query-immodules-2.0
- libgtk-3-0: gtk-query-immodules-3.0

Crossbuilding Issues - Arch-dependent tools

- chrpath: Modifies rpath in binary (now fixed)
- gobject-introspection (atk, gstreamer, pango, udev, libsoup, gdk-pixbuf, gnome-everything)
  - `g-ir-scanner` dlopens binaries to scan for gobject interfaces and writes (arch-specific) xml descriptions

Making your packages cross-friendly

- `include /usr/share/dpkg/architecture.mk`
- Use pkg-config and autotools or cmake
- Don't run just-built binaries when crossing
- read `http://wiki.debian.org/CrossBuildPackagingGuidelines` and `https://wiki.linaro.org/Platform/DevPlatform/CrossCompile/CrossPatching`

The End

Thanks to various people

- Linaro for funding this work
- Team GSOC
  - Johanes Schauer (bootstrapping analysis & botch)
  - Thibaut Girka (multiarch cross-toochains)
  - Patrick McDermott (utils and dep-cycle breaking)
  - Gustavo Alkmim (bootstrap tool)
- Various useful people: Steve Langasek, Colin Watson, Marcin Juśkiewicz, Mattias Klose, Hector Oron, Neil Williams, Pietro Abate, Jonathan Austin, Harry Liebel, Loic Minier

Further reading: `https://wiki.linaro.org/Platform/DevPlatform/CrossCompile/CrossBuilding`