Bootstrapping Debian for a new architecture

Pietro Abate

Universite Paris Diderot / Irill

24-11-2012

Acknowledgements

- Most of the work done by Johannes Schauer during its GSOC 2012
- Mentoring myself and Wookey

Problem

- Debian was ported to more than 20 architectures so the process is executed roughly once per year
- Debian packages are neither made to be cross compilable nor to be built without an existing full Debian installation
- For each new port a `set` of source packages has to be cross compiled and/or built manually
- Bootstrap a new architecture often involves foreign distributions and a lot of hacking

Wish List

- Porting Debian to a new architecture should be less time consuming and less problematic.
- No foreign distributions during porting (self hosted).
- Automatic cross compiling for architectures that cannot build themselves.
- Sub-arch builds optimized for a specific CPU should be easier.

The final Goal : Deducing a build order

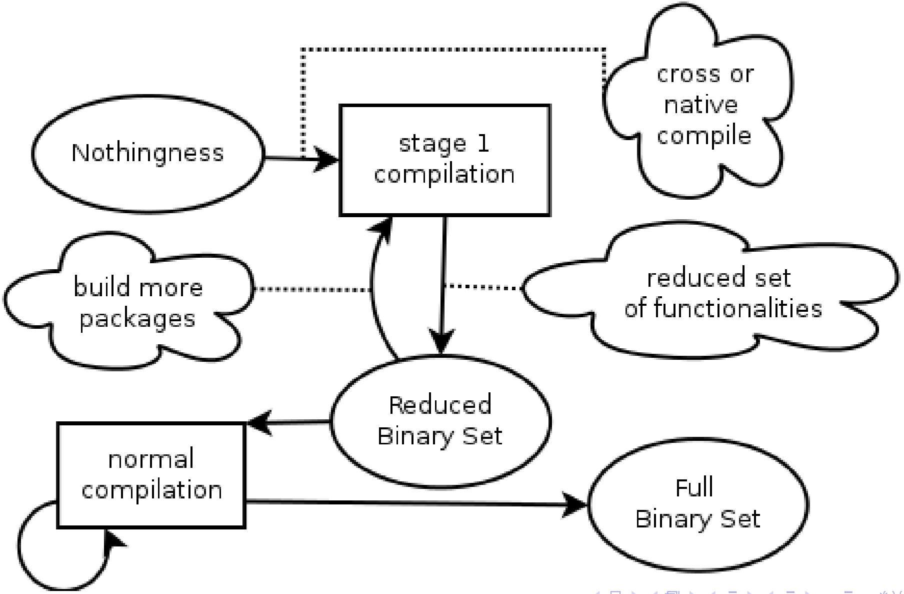
1. Step zero : Bare metal.
2. Cross compilation : create a minimal build system (*XC*).
3. Automatic (cross) compile *XC*.
4. Switch to native compilation.
5. Find the largest number of sources that can be natively built (*NC*).
6. Automatic compile *NC* (we need a build order).

We need correct Multi-Arch annotations and build profiles.

Stage Compilation

- Directly building fully fledged binary packages is impossible because of the presence of `build dependency cycles`
- We need to weak build dependencies in order to remove these dependency cycles.
- `Build Profiles` are the proposed solution solution.
  - A build profile is a global build dependency filter
  - It is the form : `Build-Depends:  foo [i386 arm] <!stage1>`
  - The format similar to architecture specifiers

Stage Compilation



Why we need Cross Compilation ?

- A new architecture cannot be bootstrapped from thin air
- At least a minimal system must be cross built
- This system should be large enough to compile the entire distribution
- Native compilation should be preferred over cross compilation

Cross compilation. Package selection.

The minimal set of packages that must be cross compiled (XC) are those with the following properties :

- `Essential: yes`
- `Build-Essential: yes`
- `Priority: required`

Plus debhelper as 79% of the archive depend on it

Minimal build system

- How many packages are in the minimal build system for Sid ?

	Debian Sid	Ubuntu Precise
`Priority: required`	37	70
`Essential: yes`	25	24
`Build-Essential: yes`	11	44
how many bianry packages	106	140
how many source packages	55	75

- Many packages in *XC* would cross-build just fine if their cross-build-dependencies could be resolved using Multi-Arch.
- `Challenge N. 1` : Automatically Cross compile the minimal build system.

Test cross-build-dependency resolution

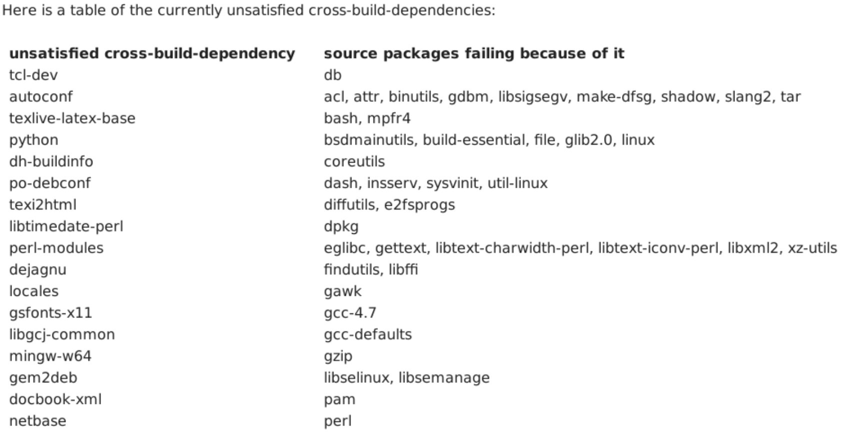
With `apt-get` (adding an armel as foreign architecture):

```
apt-get --simulate --host-architecture=armel build-dep <package>
```

Or with `dose-buildebcheck` (static check):

```
dose-builddebcheck --success --failures --explain \
 --deb-native-arch=amd64 \
 --deb-host-arch=armhf \
 ubuntu_dists_quantal_main_binary-amd64_Packages \
 ubuntu_dists_quantal_main_binary-armhf_Packages \
 ubuntu_dists_quantal_main_source_Sources
```

We can't cross compile the minimal build system (yet !)



Here is a table of the currently unsatisfied cross-build-dependencies:

unsatisfied cross-build-dependency	source packages failing because of it
tcl-dev	db
autoconf	acl, attr, binutils, gdbm, libsigsegv, make-dfsg, shadow, slang2, tar
texlive-latex-base	bash, mpfr4
python	bsdmainutils, build-essential, file, glib2.0, linux
dh-buildinfo	coreutils
po-debconf	dash, insserv, sysvinit, util-linux
texi2html	diffutils, e2fsprogs
libtimedate-perl	dpkg
perl-modules	eglibc, gettext, libtext-charwidth-perl, libtext-iconv-perl, libxml2, xz-utils
dejagnu	findutils, libffi
locales	gawk
gsfonts-x11	gcc-4.7
libgcj-common	gcc-defaults
mingw-w64	gzip
gem2deb	libselinux, libsemanage
docbook-xml	pam
netbase	perl

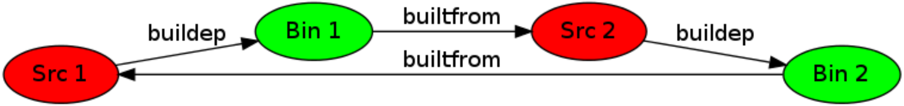
Which packages can be natively compiled from XC ?

Maximal set of source package that can be compiled natively.

- B_i : set of binary packages that are currently available.
- S set of packages that we want to compile.
- S_i set of source packages that can be successfully compiled.

```
procedure $\textsc{Build}(S_i, B_i, S)$
    $\mathtt{S_{i+1}} \leftarrow \mathit{find\_installable}(B_i, S)$
    if $\mathtt{S_{i+1}} = \emptyset$ then
        return $S_i$
    else
        $\mathtt{B_{i+1}} \leftarrow \mathit{Bin}(S_{i+1}) \cup B_i$
        return $\mathtt{BUILD}(S_i \cup S_{i+1}, B_{i+1}, S \setminus S_{i+1})$
$\mathtt{ALLNATIVE} \leftarrow \mathtt{BUILD}(\emptyset, \mathit{Bin}(XC), S)$
```

The dependency graph



- Two types of vertex
  - source packages.
  - build-dependency set (binaries needed to build a source package)
- Two types of edges
  - build-dep (source → binary)
  - built-from (binary → source)

- Built iteratively by adding connecting each source package to the set of its build dependencies and each build dependencies set to all source packages whose binaries are build from.
- Packages that are cross-built ($p \in XC$) or with `Architecture:all` are excluded from the dependency graph.

Simplify the Build Dependency Graph. Challenge N. 2

- The control fields `Build-Depends-Indep` and `Build-Conflicts-Indep` identify dependencies or conflicts for building architecture:all packages
- We are not interested to build architecture:all packages therefore we can drop `Build-Depends-Indep` and `Build-Conflicts-Indep` dependencies
- Find Weak dependencies :
  - Manually identify packages that are not strictly needed to compile a working, albeit not full, package
  - Use external information to identify weak packages (gentoo compile flags)
  - Add **build profiles** (ex. stage1, embedded, nodoc, etc) to source packages (more later about build profiles).

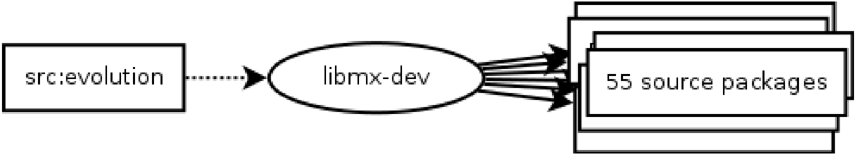
Some numbers on the build graph

- the dependency graph generated for Debian Sid has 39486 vertices.
- it has only one central SCC with 1027 vertices.
- eight other SCC with 2 to 7 vertices.
- contains not-nice packages like: nautilus, iceweasel, metacity, evolution, etc
- contains many build dependency cycles.
- `Challenge N. 3` (Automatically) Remove build dependencies

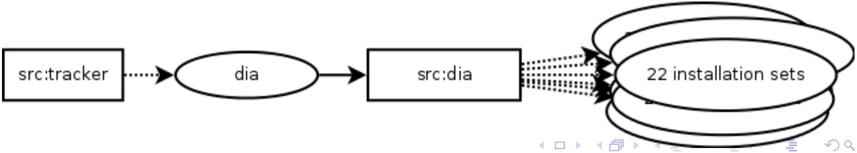
Dependency graph analysis

We can easily identify :

- binary/source nodes with most/least incoming/outgoing edges
- most/least connected nodes
- source packages only missing a few build dependencies
- binary packages with highest ratio of source packages it needs to be built and source packages that build depend on it



- source packages with highest ratio of build dependencies and source packages that build-depend on packages that depend on it



Current Unresolved Issue in Debian

- Provide a build order is still difficult because :
  - unsatisfied cross build dependencies because of missing multi-arch annotation
  - insufficient number of reduced build dependencies to solve dependency cycles
- what is blocking the above:
  - wanna-build doesn't support architecture qualifiers (`pkg:any`, `pkg:native`, `pkg:amd64`, ...)
  - no decision on format of reduced build dependencies
- after both issues are solved, changes have to be manually implemented into actual packages

Future work

- Identify a list of plausible weak dependencies (Work in progress to use Gentoo build-flags)
- Devise an algorithm to automatically break build cycles using weak dependencies (almost done)
- Create a topological sort of the build dependency graph (almost done)
- Provide a build order to be used to bootstrap debian of a foreign architecture.
- Generalize this solution to a larger class of problems.

Tools and Resources

All our tools and experiments are available :

- Debian Bootstrap :
  `https://gitorious.org/debian-bootstrap/bootstrap`
- Dose : `https://gforge.inria.fr/projects/dose/`
- dose-builddebcheck :
  `http://packages.debian.org/wheezy/dose-builddebcheck`
- Main page : `http://wiki.debian.org/DebianBootstrap`
- Lots of details :
  `http://wiki.debian.org/DebianBootstrap/TODO`
- Multi-Arch Cross spec
  `https://wiki.ubuntu.com/MultiarchCross`
- Multi-Arch spec : `https://wiki.ubuntu.com/MultiarchSpec`
- Linaro Cross Compile Howto `https://wiki.linaro.org/Platform/DevPlatform/CrossCompile/UsingMultiArch`