Architectures vs the Ports tree: a losing battle ?

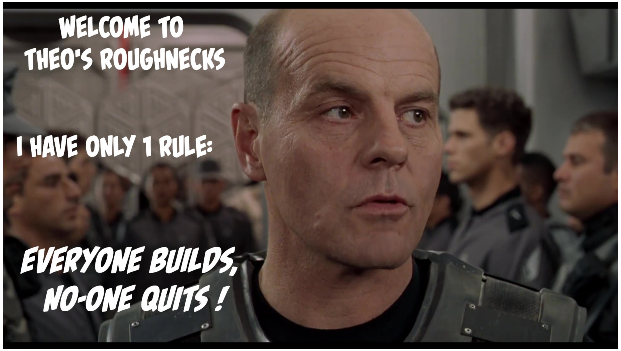
Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>

OpenBSD project & Laboratoire de Recherche de l'Epita

September 18, 2022

Elephant

OpenBSD rules



Only on OpenBSD

- we only use cross-compilation for bootstrap
- every arch builds its own packages
- best stress-test ever

Interesting variations

- strict alignment architectures
- big endian vs little endian
- character signedness (not really interesting)
- reverse stack
- ghostguard
- smallkva
- (compiler bugs)

Why this talk

- Cumulative work over the past 20 years or so
- Lots of (smallish) topics I haven't talked about ever
- No big plan, just lots of small improvements and know-how

What's an architecture

What's in a name

- `ARCH` describes the exact machine (e.g., macppc)
- `MACHINE_ARCH` is the "cpu make" (e.g., powerpc)
- details like "i386" vs "pentium" are generally not encoded
- $\rightarrow$ generally, packages target `MACHINE_ARCH`

Compiler subversion

- Compilers offer `-march=native` options
- This should **never** be used for building packages
- Instead the base OS targets a baseline cpu, and everything should work on this cpu and later versions
- (notable exception: the altivec extensions to ppc, a while ago)
- slowly, the bar gets raised, from i386 to i586 to...

not gentoo

talking to upstream

- explaining that we're software vendors, and we need reliable builds that will work on every machine
- so no tests during builds to optimize the compilation to the exact machine we have
- specifically for multimedia software: no hand-crafted assembly code selected at runtime

good practices

- provide at least a way to build that doesn't hardcode machine details
- replace compile-time tests with runtime tests to select hand-crafted code (for instance, relying on cpuid on intel boxess)

compilers

Compiler options

- in general, upstream is bad with compiler options
- those do break on some arches
- so we standardize on `-O2` and `-O2 -g`
- porters try to help heeding `CFLAGS` and `CXXFLAGS`
- we hate build systems without an easy way to specify options
- even compilers change options with hilarious effects

Variation on available stuff

- coding tests on `MACHINE_ARCH` is an extraordinarily bad idea
- prefer `ONLY_FOR_ARCHS` and `NOT_FOR_ARCHS`
- (or eventually `BROKEN`)
- that way everything is referenced properly

tools

dpb

- gets information through `make dump-vars`
- should be resilient to errors
- will flag as errors missing information for ports
- removes stuff if marked as not available for this arch
- can even be run on a different architecture for listing

sqlports

- we also run `make dump-vars` to create a db of everything
- that one errors out if something does not work, possibly a pkgpath

example I

```
===> archivers
===> archivers/arc
archivers/arc.IS_INTERACTIVE=No
archivers/arc.SUBPACKAGE=-
archivers/arc.BUILD_PACKAGES= -
archivers/arc.MULTI_PACKAGES=-
archivers/arc.DISTFILES=arc-5.21p.tar.gz
archivers/arc.MASTER_SITES=https://downloads.sourceforge.net/sourceforge/arc/
archivers/arc.CHECKSUM_FILE=/usr/ports/archivers/arc/distinfo
archivers/arc.FETCH_MANUALLY=No
archivers/arc.PERMIT_DISTFILES=Yes
archivers/arc.NO_TEST=Yes
archivers/arc.TEST_IS_INTERACTIVE=No
archivers/arc.DISTNAME=arc-5.21p
archivers/arc.HOMEPAGE=http://arc.sourceforge.net/
archivers/arc.MAINTAINER=The OpenBSD ports mailing-list <ports@openbsd.org>
archivers/arc.USE_GMAKE=No
```

example II

```
archivers/arc.USE_GROFF=No
archivers/arc.NO_BUILD=No
archivers/arc.USE_LIBTOOL=Yes
archivers/arc.SEPARATE_BUILD=No
archivers/arc.TARGETS= do-install
archivers/arc.MAKEFILE_LIST=/usr/share/mk/sys.mk Makefile /usr/share/mk/bsd.port.
archivers/arc.USE_LLD=Yes
archivers/arc.USE_WXNEEDED=No
archivers/arc.COMPILER=base-clang base-gcc gcc3
archivers/arc.COMPILER_LANGS=c c++
archivers/arc.COMPILER_LINKS= clang /usr/bin/clang  clang++ /usr/bin/clang++  cc /
archivers/arc.SUBST_VARS=ARCH BASE_PKGPATH FLAVOR_EXT FULLPKGNAME HOMEPAGE  LOCAL
archivers/arc.PKGPATHS=archivers/arc
archivers/arc.FULLPKGNAME=arc-5.21pp0
archivers/arc.PERMIT_PACKAGE=Yes
archivers/arc.COMMENT=create & extract files from DOS .ARC files
archivers/arc.PKGNAME=arc-5.21p
```

example III

```
archivers/arc.PKGSPEC=arc-*
archivers/arc.PKGSTEM=arc
archivers/arc.PREFIX=/usr/local
archivers/arc.WANTLIB=c
archivers/arc.CATEGORIES=archivers
archivers/arc.DESCR=/usr/ports/archivers/arc/pkg/DESCR
archivers/arc.REVISION=0
archivers/arc.STATIC_PLIST=Yes
archivers/arc.PKG_ARCH=amd64
===> archivers/blosc
archivers/blosc.BUILD_DEPENDS=devel/cmake devel/ninja
archivers/blosc.IS_INTERACTIVE=No
archivers/blosc.SUBPACKAGE=-
archivers/blosc.BUILD_PACKAGES= -
archivers/blosc.MULTI_PACKAGES=-
archivers/blosc.DISTFILES=c-blosc-1.21.1.tar.gz
```

naming game

- every location in the ports tree has a unique `fullpkgpath`
- for instance, `archivers/arc` or `lang/python/3.10,-tests`
- there are `FLAVORS` and `MULTI_PACKAGES`

Not building

- variations are often specific parts that do not build on an architecture
- we can setup a `MULTI_PACKAGES` port with that part in a separate `SUBPACKAGE`
- tests won't work because those subpackages won't be reachable
- so instead we remove stuff: `MULTI_PACKAGES` → `BUILD_PACKAGES`

Example I

```
ONLY_FOR_ARCHS-java =           aarch64 amd64 i386

CATEGORIES =                   graphics devel
COMMENT-main =                   library for computer vision real-time processing
COMMENT-java =                   Java bindings for OpenCV

V =                           4.6.0
GH_ACCOUNT =                   opencv
GH_PROJECT =                   opencv
GH_TAGNAME =                   ${V}

PKGNAME-main =                   opencv-${V}
PKGNAME-java =                   opencv-java-${V}

HOMEPAGE =                   https://www.opencv.org/

MAINTAINER =                   Rafael Sadowski <rsadowski@openbsd.org>
```

Example II

```

.for i in opencv_calib3d opencv_core opencv_features2d \
  opencv_flann opencv_highgui opencv_imgproc opencv_ml opencv_objdetect \
  opencv_photo opencv_stitching opencv_video opencv_imgcodecs \
  opencv_videoio opencv_dnn
SHARED_LIBS += $i 10.0
.endfor

WANTLIB-main += ${COMPILER_LIBCXX} avcodec avformat avutil OpenEXR-3_1
WANTLIB-main += c cairo gdk-3 gdk_pixbuf-2.0 glib-2.0 gobject-2.0 gstapp-1.0
WANTLIB-main += gstbase-1.0 gstaudio-1.0 gstpbutils-1.0 gstreamer-1.0
WANTLIB-main += gstriff-1.0 gstvideo-1.0 gtk-3 jpeg m openjp2 png swscale tiff
WANTLIB-main += webp z

WANTLIB-java += ${COMPILER_LIBCXX} opencv_calib3d opencv_core opencv_dnn
WANTLIB-java += opencv_features2d opencv_flann opencv_imgcodecs
WANTLIB-java += opencv_imgproc opencv_ml opencv_objdetect opencv_photo
```

Example III

```
WANTLIB-java += opencv_video opencv_videoio

COMPILER =                     base-clang ports-gcc

MULTI_PACKAGES =            -main -java
PSEUDO_FLAVORS =            no_java
FLAVOR ?=

# BSDL
PERMIT_PACKAGE =            Yes

MODULES =                    devel/cmake \
                            lang/python

BUILD_DEPENDS =                     math/eigen3 \
                            math/py-numpy${MODPY_FLAVOR}

```

Example IV

```
RUN_DEPENDS-main =          math/py-numpy${MODPY_FLAVOR}

RUN_DEPENDS-java =          ${MODJAVA_RUN_DEPENDS}

LIB_DEPENDS-main =          ${LIB_DEPENDS} \
                          graphics/ffmpeg \
                          graphics/jpeg \
                          graphics/libwebp \
                          graphics/openexr \
                          graphics/openjp2 \
                          graphics/png \
                          graphics/tiff \
                          multimedia/gstreamer1/core \
                          multimedia/gstreamer1/plugins-base \
                          x11/gtk+3

LIB_DEPENDS-java =          ${BUILD_PKGPATH},-main=${V}
```

Example V

```

# XXX PIE cannot be produced due to problems with inline assembly.
# Since OpenCV is mostly used as a LIBrary, switch to PIC.
.if ${MACHINE_ARCH:Mi386}
CFLAGS +=         -fPIC
CXXFLAGS +=         -fPIC
.endif

CONFIGURE_ARGS =            -DBUILD_DOCS=OFF \
                            -DBUILD_EXAMPLES=OFF \
                            -DBUILD_IPP_IW=OFF \
                            -DBUILD_ITT=OFF \
                            -DBUILD_PERF_TESTS=OFF \
                            -DBUILD_TESTS=OFF \
                            -DBUILD_opencv_python2=OFF \
                            -DINSTALL_PYTHON_EXAMPLES=OFF \
                            -DINSTALL_TESTS=OFF \
```

Example VI

```
                        -DOPENCV_SKIP_PYTHON_WARNING=ON \
                        -DPYTHON_DEFAULT_EXECUTABLE=${MODPY_BIN} \
                        -DWITH_1394=OFF \
                        -DWITH_ADE=OFF \
                        -DWITH_CUDA=OFF \
                        -DWITH_EIGEN=OFF \
                        -DWITH_IPP=OFF \
                        -DWITH_OPENCL=OFF \
                        -DWITH_V4L=ON \
                        -DWITH_VTK=OFF \
                        -DOPENCV_GENERATE_PKGCONFIG=ON

.include <bsd.port.arch.mk>

.if ${BUILD_PACKAGES:M-java}
MODULES +=                  java
MODJAVA_VER =                  1.8+
```

Example VII

```
BUILD_DEPENDS +=         devel/apache-ant
.else
# Safe: Java will be detected, if present, but won't be used
CONFIGURE_ARGS +=         -DBUILD_opencv_java=OFF
.endif

CONFIG_ADJ_CMD =         perl -pi
.for _l _v in ${SHARED_LIBS}
CONFIG_ADJ_CMD +=         -e 's,lib${_l}.so([^.]),lib${_l}.so.${_v}$$1,g;'
.endfor

NO_TEST =         Yes
# Enable to run the regression tests
#TEST_IS_INTERACTIVE =         X11
#
#CONFIGURE_ARGS +=         -DDBUILD_TESTS=ON \
#                          -DBUILD_PERF_TESTS=ON
```

Example VIII

```

post-patch:
	perl -pi -e 's@^.*(#\s*include)@$$1@' \
		${WRKSRC}/samples/cpp/tutorial_code/core/how_to_scan_images/how_to

post-install:
	${MODPY_BIN} ${MODPY_LIBDIR}/compileall.py ${WRKINST}${MODPY_SITEPKG}

do-test:
	cd ${WRKBUILD}; \
	${MODPY_BIN} ${WRKSRC}/modules/ts/misc/run.py

.include <bsd.port.mk>
```

bsd.port.arch.mk

- a part of `bsd.port.mk`
- if you don't include it yourself, it will be done automatically
- set up `BUILD_PACKAGES` according to `PSEUDO_FLAVORS` and arches
- then you test according to `BUILD_PACKAGES` for configure tests

also properties I

```
# architecture constants

ARCH ?!= uname -m

ALL_ARCHS = aarch64 alpha amd64 arm arm64 armv7 hppa i386 landisk loongson \
        luna88k m88k macppc mips64 mips64el octeon powerpc64 riscv64 sgi \
        sh sparc64
# normally only list MACHINE_ARCH (uname -p) names in these variables,
# but not all powerpc have apm(4), hence the use of macppc
APM_ARCHS = arm64 amd64 i386 loongson macppc sparc64
BE_ARCHS = hppa m88k mips64 powerpc powerpc64 sparc64
LE_ARCHS = aarch64 alpha amd64 arm i386 mips64el riscv64 sh
LP64_ARCHS = aarch64 alpha amd64 mips64 mips64el powerpc64 riscv64 sparc64
GCC4_ARCHS = alpha hppa sh sparc64
GCC3_ARCHS = m88k
# XXX easier for ports that depend on mono
MONO_ARCHS = aarch64 amd64 i386
```

also properties II

```
OCAML_NATIVE_ARCHS = aarch64 amd64 i386
OCAML_NATIVE_DYNLINK_ARCHS = aarch64 amd64 i386
GO_ARCHS = aarch64 amd64 arm armv7 i386 mips64
RUST_ARCHS = aarch64 amd64 i386 powerpc64 riscv64 sparc64

# arches where the base compiler is clang
CLANG_ARCHS = aarch64 amd64 arm i386 mips64 mips64el powerpc powerpc64 riscv64
# arches using LLVM's linker (ld.lld); others use binutils' ld.bfd
LLD_ARCHS = aarch64 amd64 arm i386 powerpc powerpc64 riscv64

# arches where ports devel/llvm builds - populates llvm ONLY_FOR_ARCHS
# as well as available for PROPERTIES checks.
LLVM_ARCHS = aarch64 amd64 arm i386 mips64 mips64el powerpc powerpc64 riscv64 spar
# arches where ports-gcc >4.9 exists.  To be used again for modules
GCC49_ARCHS = aarch64 alpha amd64 arm hppa i386 mips64 mips64el powerpc powerpc64

MODGCC4_VERSION?=8
```

also properties III

```
# arches where there is a C++11 compiler, either clang in base or ports-gcc
CXX11_ARCHS = ${CLANG_ARCHS} ${GCC49_ARCHS}
DEBUGINFO_ARCHS = aarch64 amd64
```

numbers

- 9700 Makefiles and fragments
- 200 uses of `bsd.port.arch.mk`
- 90 tests on `BUILD_PACKAGES`

lazy make to the rescue

Stuff like this actually works:

```
ONLY_FOR_ARCHS-sub = ${RUST_ARCHS}

.include <bsd.port.arch.mk>
.if ${BUILD_PACKAGES:M-sub}
...
.endif
```

Industrialisation

- we had binary packages in 2000
- dpb dates back from 2010
- dedicated build farms for most architectures
- takes between 24 hours and a few weeks
- regular build stats for everything (thanks landry@)

architecture issues

- intel 64 bits acts as "the bellwether" (most stuff always builds)
- other architectures get fixed depending on needs
- some big stuff is (sometimes) not even built because of practicality

compilers

- this was painful to create but works
- there's a variable `COMPILER` you can set to choose "the best" compiler
- some systems have `gcc3` in base, others have `gcc 4.2` and others have `clang`
- there's also a more modern gcc in ports and an llvm port
- `COMPILER` is a list of preferred compilers: `base-gcc, base-clang, gcc3, ports-gcc, ports-clang`
- either it's there, or it's not
- links under `WRKDIR/bin` will be created

odds and ends

- bootstrapping stuff like go and rust is painful
- we got a mechanism for `PSEUDO_FLAVORS` to help dpb and preserve bootstrap

location is everything

- lazy make: variable definitions first
- then tests and targets
- but `MODULES`
- but `COMPILER`
- but bsd.port.arch.mk
- very specific location (best of both worlds)

That losing battle

- language support is the #1 problem (modern C++, rust, go)
- 32 bit arches are losing
- we got dpb annotations to help (`lonesome`) but it's still a problem

???

Any questions ?