FreeBSD on ARMv6: Cross-Compile Performance Optimization for Poudriere

Important Announcements on FreeBSD-armv6 packages

While initially writing this article, I had the idea to establish a service where packages can be selected to build for armv6. As of February 2016 this service is now online.
If you just need current FreeBSD packages for armv6, this is the place to visit. Otherwise, keep on reading.

Whilst playing around with FreeBSD on Raspberry Pi, I started to dig into cross-compiling packages.

Well, if you follow the first tutorial you'll surely notice that there is no real speed-gain, because the use of full binary emulation on a x86 host through QEMU. So this is almost as slow as if packages were natively compiled on the Raspberry Pi itself even if done on a multi-cpu Xeon powerhouse.

So let's see how to get an actually performance gain.

First, create a two new jails. The one with the 'x' in the name will be using the host cross-compiler.

```
 # poudriere jails -c -j fbsd10armv6 -a armv6 -m tar=/build/fbsd10_2_release_armv6.tar -v 10.2-RELEASE
# poudriere jails -c -j xfbsd10armv6 -a armv6 -m tar=/build/fbsd10_2_release_armv6.tar -v 10.2-RELEASE
```

Then get the mount point:

```
# poudriere jail -l
JAILNAME VERSION ARCH METHOD TIMESTAMP PATH
fbsd10armv6 10.2-RELEASE arm.armv6 tar 2015-10-08 00:42:48 /build/poudriere/jails/fbsd10armv6
xfbsd10armv6 10.2-RELEASE armv6 tar 2015-10-11 12:47:23 /build/poudriere/jails/xfbsd10armv6
```

Go to host source directory. Set the export location to your jail's **/usr/obj** directory. Then build the toolchain as shown below.

```
cd /usr/src
mkdir -p /build/poudriere/jails/xfbsd10armv6/usr/obj
export MAKEOBJDIRPREFIX=/build/poudriere/jails/xfbsd10armv6/usr/obj
make XDEV=arm XDEV_ARCH=armv6 WITH_GCC=1 xdev
```

Once the host cross compiler is built, copy the directory tree **/usr/armv6-freebsd** into the jail.

```
cp -rpv /usr/armv6-freebsd /build/poudriere/jails/xfbsd10armv6/usr/armv6-freebsd
```

Ok, now important to know is how poudriere works internally.

Poudriere will clone the jail's base directory into each newly spawned jail off a ZFS snapshot called @clean.
In order to have our toolchain show up in the cloned jail, we'll need to create a new @clean snapshot.

However, the old snapshot should be kept as well, just in case.

Here's how to get list of snapshots:

```
# zfs list -t snapshot
NAME                                USED  AVAIL  REFER  MOUNTPOINT
build/poudriere/jails/fbsd10armv6@clean     35K      -   210M  -
build/poudriere/jails/xfbsd10armv6@clean  47.5K      -   210M  -
```

So, let's rename the snapshot.

```
zfs rename build/poudriere/jails/xfbsd10armv6@clean build/poudriere/jails/xfbsd10armv6@old_clean
```

And now create a new @clean snapshot.

```
zfs snapshot build/poudriere/jails/xfbsd10armv6@clean
```

Let's check the snapshots again.

```
# zfs list -t snapshot
NAME                                    USED  AVAIL  REFER  MOUNTPOINT
build/poudriere/jails/fbsd10armv6@clean         35K      -   210M  -
build/poudriere/jails/xfbsd10armv6@old_clean  47.5K      -   210M  -
build/poudriere/jails/xfbsd10armv6@clean          0      -   881M  -
```

Now create a custom make.conf for the second jail at **/usr/local/etc/poudriere.d/xfbsd10armv6-make.conf**:

```
CC=/usr/armv6-freebsd/usr/bin/cc
CPP=/usr/armv6-freebsd/usr/bin/cpp
CXX=/usr/armv6-freebsd/usr/bin/c++
AS=/usr/armv6-freebsd/usr/bin/as
NM=/usr/armv6-freebsd/usr/bin/nm
RANLIB=/usr/armv6-freebsd/usr/bin/ranlib
LD=/usr/armv6-freebsd/usr/bin/ld
OBJCOPY=/usr/armv6-freebsd/usr/bin/objcopy
SIZE=/usr/armv6-freebsd/usr/bin/size
STRIPBIN=/usr/armv6-freebsd/usr/bin/strip
```

That's it, we're read to see how a the build using the cross-compiler toolchain compares against QEMU.

Here's how I did the compile run:

```
# poudriere bulk -j fbsd10armv6 -p p10_2015 -f /build/armv6.pkglist -J 8
# poudriere bulk -j xfbsd10armv6 -p p10_2015 -f /build/armv6.pkglist -J 8
```

The package lists contains the ports as shown below. Counting in all build dependencies, there's some 42 ports to build in total.

```
shells/bash
editors/vim-lite
www/lighttpd
www/thttpd
ftp/wget
ftp/curl
lang/perl5.20
net/isc-dhcp43-server
net/isc-dhcp43-client
net-mgmt/net-snmp
net/rsync
sysutils/tree
security/sudo
dns/bind910
www/nginx
net/radvd
lang/python27
```

And here's the facts and figures.

```
SET PORTS    JAIL         BUILD                STATUS         QUEUE BUILT FAIL SKIP IGNORE REMAIN TIME      LOGS
-   p10_2015 fbsd10armv6  2015-10-08_01h17m51s parallel_build    42    42    0    0      0      0 13:22:48
/build/poudriere//data/logs/bulk/fbsd10armv6-p10_2015/2015-10-08_01h17m51s
-   p10_2015 xfbsd10armv6 2015-10-11_22h07m46s done             42    42    0    0      0      0 02:06:10
/build/poudriere//data/logs/bulk/xfbsd10armv6-p10_2015/2015-10-11_22h07m46s
```

While the first build was done using full binary emulation with QEMU, the second run used the host cross-compiler and was roughly 6 times faster.

Yet, there is still some overhead slowing things down. Most of the tools inside the jails, especially when it comes to shell scripts, libtool, make and some other tools, are still executed through QEMU. Thus some steps like configure will still suffer from a performance impact.

The tests described where carried out on a dual-quad Xeon-backed ESXi VM running FreeBSD 10.2 amd64. I set aside 4 v-cores, 8 GiB of RAM and a dedicated storage on SSD. Poudriere was configured to make use of "tmpfs" in order to get around possible I/O limits as well as ccache to reuse previously compiled files.