ÉCOLE SUPÉRIEURE DES TECHNOLOGIES
ÉLECTRONIQUE, INFORMATIQUE, INFOGRAPHIE

PROGRAMMATION LINUX

-

TP1 : Chaîne de Compilation Croisée

Thomas LEPOIX & Soumana AMADOU AMADOU
BACHELOR 3 SER 2016/2017

Table des matières

I – Généralités

A – Compilation

En informatique, la compilation est le processus par lequel un programme écrit dans un langage de programmation humainement compréhensible est traduit en un programme écrit dans un langage compréhensible par une machine, c'est-à-dire en binaire.

Un compilateur est en fait décomposé en quatre fonctions distinctes :

- Le préprocesseur : Ce programme analyse le code source et procède à différentes transformations de celui ci, comme l'inclusion d'autres fichiers, le remplacement de certains mots clés par d'autres, ou encore de la compilation conditionnelle.
- Le compilateur : Le compilateur à proprement parler, ce programme traduit le code source d'un programme écrit en un langage de haut niveau d'abstraction (plus compréhensible pour une personne) en un langage de bas niveau, l'assembleur, plus proche du fonctionnement réel de la machine.
- L'assembleur : Ce programme traduit le code source assembleur du programme précédemment obtenu en code binaire compréhensible par la machine. Cette étape est plus simple que la précédente du fait que les instructions assembleur correspondent chacune directement à un code binaire de plusieurs bits.
- L'éditeur de lien : Ce programme produit un fichier exécutable à partir de différents binaires issus de la compilation de différentes composantes d'un même programme (on parle de fichier objet), à savoir les fonctions, les bibliothèques...

Dans les environnement de type UNIX, l'installation d'un programme à partir de son code source se déroule en trois étapes.

- ./configure : Il s'agit de l'exécution d'un script fourni avec les sources. Selon les paramètres passés à ce script qui dépendent du logiciel lui même, il élaborera un makefile, fichier qui contient toutes les règles de compilation du programme. En général les paramètres que l'on peut passer au script ainsi que leur brève explication sont accessibles en passant le paramètre —help.
- make : Cette commande scan le dossier où elle est exécutée à la recherche d'un makefile. Elle va alors compiler le programme en tenant compte des instructions dictées dans le fichier.
- make install : Cette commande permet d'installer l'exécutable créé par la commande précédente, c'est à dire déployer les fichiers le composant à la place où ils doivent être dans l'ordinateur.

B – Compilation croisée

La compilation croisée a lieu lorsque le fichier exécutable produit par le compilateur est destiné à fonctionner sur une machine d'architecture différente de celle de la machine faisant fonctionner le compilateur. D'une façon générale le compilateur natif d'une machine produit des exécutables destinés à fonctionner sur la même architecture que celle de la machine elle même, c'est pourquoi il est de rigueur de créer soi même son compilateur croisé à partir de son code source. Celui ci sera capable de produire des exécutables pour une architecture quelconque donnée.

On distingue trois machines théoriques intervenant dans la chaîne de compilation croisée :

- BUILD : Machine sur laquelle est compilé le compilateur croisé.
- HOST : Machine sur laquelle est exécuté le compilateur croisé pour compiler un programme quelconque.
- TARGET : Machine sur laquelle est exécute le programme quelconque.

Certaines de ces trois machines théoriques peuvent être incarnées par une seule et même machine. Le cas où les trois machines sont incarnées par une unique machine est le cas le plus courant de la compilation classique. Le cas où les trois machines sont différentes, appelé canadienne, est relativement rare. Dans le cas étudié ici, la machine BUILD et la machine HOST sont la même machine, c'est le cas de figure le plus commun dans l'informatique embarquée.

C – Bootstrapping

La subtilité dans l'élaboration d'un compilateur croisé est que l'on ne peut directement créer un compilateur disposant de toutes les fonctionnalités dont nous avons besoin, il faut procéder par étapes successives et récursives. Il est en revanche possible de créer un compilateur C minimaliste n'étant capable de compiler que du code C simple, il sera alors utilisé pour compiler une bibliothèque C standard qui sera elle même utilisée pour créer un nouveau compilateur C capable de compiler la plupart des codes C. Ce nouveau compilateur peut alors être à son tour utilisé pour compiler d'autres bibliothèques dont le code s'appuie sur la bibliothèque C standard, par exemple les bibliothèques mathématiques, nécessaires pour la compilation d'un Gcc récent notamment. Ces nouvelles bibliothèques qui en plus de la bibliothèque C standard permettront de créer un troisième compilateur capable notamment de créer des programmes utilisant des bibliothèques dynamiques.

II – Chaîne mise en œuvre

A – Vue d’ensemble

Le cahier des charges de ce TP demande de construire un compilateur croisé fonctionnant sur machine x86_64 bits équipée d’un environnement GNU/Linux et produisant du code pour machine ARM. Une nomenclature des différents paquets sources à disposition est également fournie :

- Binutils-2.22 : Utilitaires binaires, comprend notamment l’assembleur et l’éditeur de liens.
- Gcc-4.7.3 : Compilateur multi langage et multi plateforme.
- Glibc-2.19 : Bibliothèque C standard.
- Gmp-5.0.5 : Bibliothèque mathématique.
- Mpfr-3.1.1 : Bibliothèque mathématique.
- Linux-3.5.3 : Système d’exploitation.

Comme expliqué précédemment la subtilité de la construction d’un compilateur croisé vient du fait qu’il faut procéder par étapes successives pour finalement obtenir un compilateur qui inclut des bibliothèques choisies. Cela permet d’obtenir un compilateur adapté à ce pour quoi il sera utilisé, très utile dans l’embarqué où les systèmes sont souvent minimalistes…

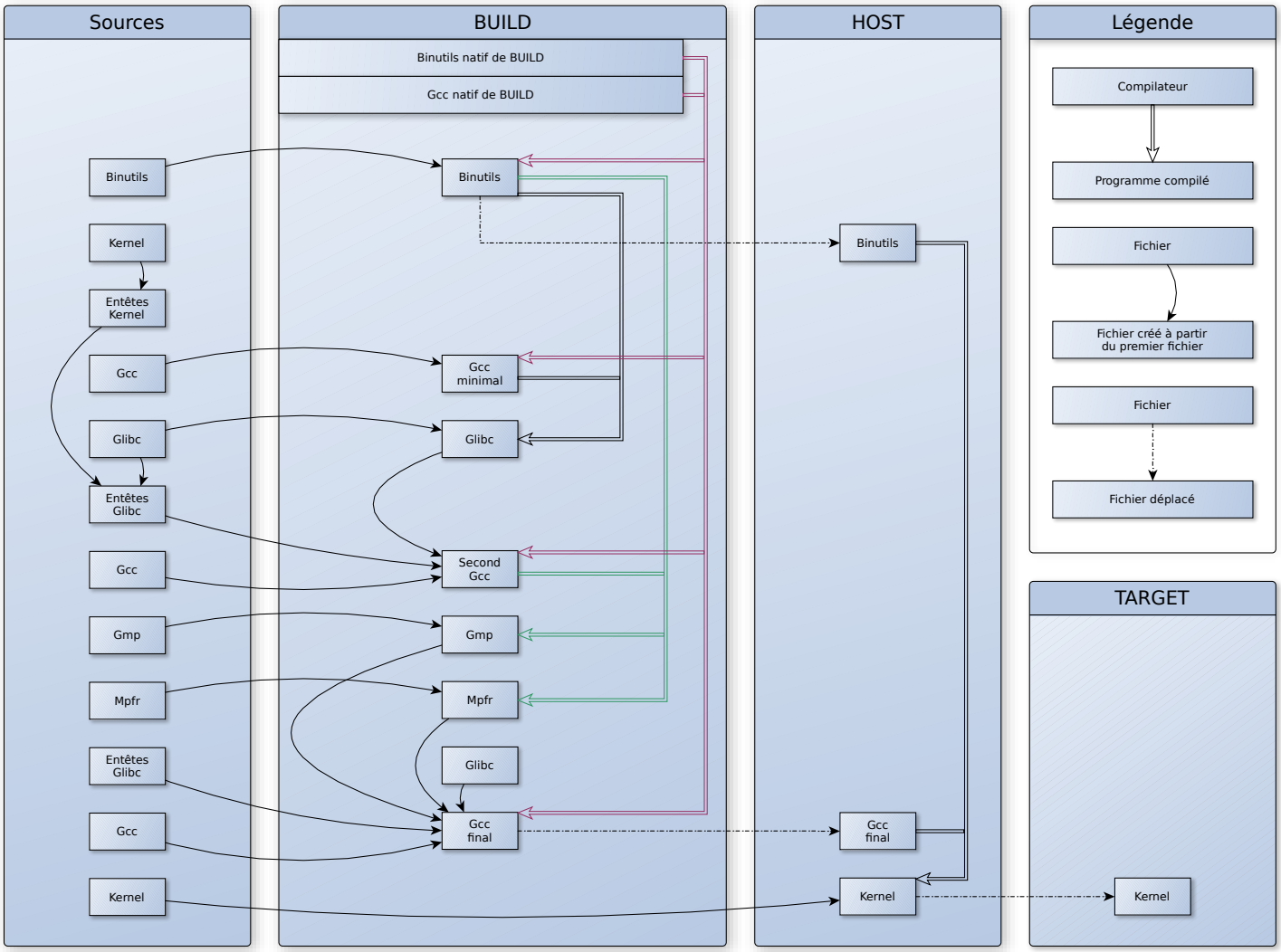
Ci suit un résumé des étapes de la chaîne de compilation croisée qui seront détaillées dans les chapitres suivants, seules les commandes clés y figureront, un script construisant la chaîne de compilation est fourni en annexe.

- Premièrement il faut préparer l’environnement de compilation, c’est à dire créer une arborescence dans laquelle sera installé le système, et créer des variables permettant un niveau d’abstraction par rapport à cette arborescence et à l’architecture des différentes machines dont il est question.
- Il faut ensuite compiler le paquet Binutils soit les utilitaires binaires. Il s’agit de la première compilation a effectuer car les autres compilations utiliseront ces utilitaires.
- Installer les entêtes du noyau Linux, cela permettra au compilateur de savoir que les logiciels qu’il compilera sont destinés à fonctionner par dessus le système d’exploitation Linux et permettra de les interfacer avec celui ci.
- Compiler un Gcc minimal « bootstrap », capable de compiler un autre Gcc minimal ou une bibliothèque C standard mais rien de plus.

- Créer des entêtes de la Glibc, le but est de créer une bibliothèque C précompilée, les fichiers d'entête serviront à faire le lien avec la Glibc binaire. La création des entêtes de la Glibc utilise les entêtes du noyau Linux.
- Utiliser le Gcc minimal et les utilitaires binaires de la seconde étape pour compiler la Glibc.
- Compiler un second Gcc incluant les entêtes de la Glibc et donc celles du noyau Linux. Ce Gcc est capable de compiler du code statiques.
- Utiliser ce nouveau Gcc et les utilitaires binaires pour compiler de nouvelles bibliothèques, elles mêmes utilisant les fonctions de la bibliothèque C standard. Ici deux bibliothèques servant au calcul mathématique seront compilées, la première étant Gmp.
- Répéter l'étape pour la seconde bibliothèque, Mpfr. Il convient de compiler Gmp avant Mpfr car la première est une dépendance de la seconde.
- Compiler un Gcc final incluant ces nouvelles bibliothèques en plus de la Glibc et étant capable de produire du code utilisant des bibliothèques dynamiques.

Toutes ces étapes se passent sur la machine BUILD, après la dernière étape le Gcc final peut être déplacé sur la machine HOST, en l'occurrence la même machine, puis exécuté pour produire des exécutables ARM à partir de leur code source, à commencer par le noyau Linux correspondant aux entêtes utilisées dans l'élaboration du compilateur croisé. Ces exécutables pourront enfin être déplacés sur la machine TARGET, où ils demeureront et seront exécutés.

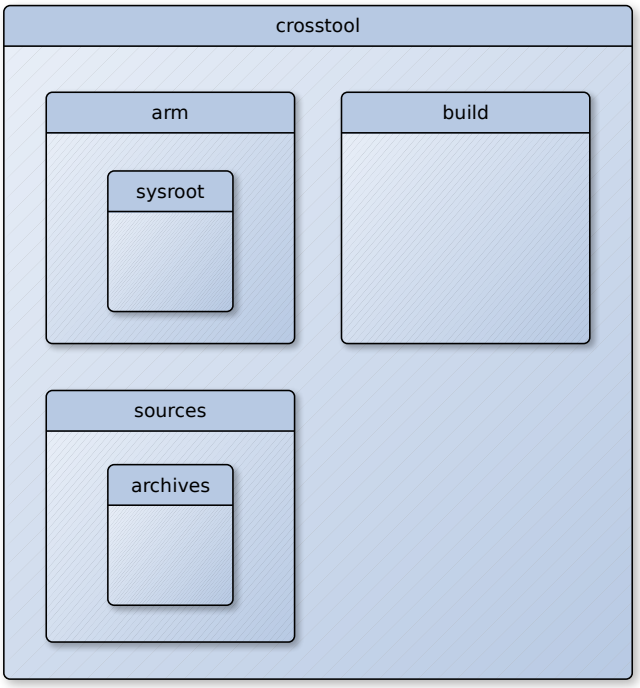
Ci joint un schéma récapitulant l'ensemble de la chaîne de compilation croisée.



B – Création de l’environnement de compilations

Voici une représentation de l’arborescence à créer dans un dossier quelconque, quelque part dans le dossier /home/user/ par exemple.

```
~ $ mkdir -p crosstool/{build,arm/sysroot,sources/archives}
```



Il faut ensuite définir des variables d’environnement qui seront réutilisées tout au long du processus.

```
~ $ export SRCDIR="$(pwd)/crosstool/sources"
~ $ export BUILDDIR="$(pwd)/crosstool/build"
~ $ export INSTALLDIR="$(pwd)/crosstool/arm"
~ $ export SYSROOTDIR="$(pwd)/crosstool/arm/sysroot"
~ $ export TARGET="arm-none-linux-gnueabi"
~ $ export BUILD="x86_64-pc-linux-gnu"
~ $ export THREADS="$(egrep -c 'processor' /proc/cpuinfo)"
```

Il y a là des références à l’arborescence précédente et à l’architecture des machines concernées x86_64 pour la machine BUILD et ARM pour la machine TARGET, THREADS correspond au nombre de cœurs du processeur de la machine BUILD.

Il faut également faire l’acquisition des paquets sources et les extraire de leurs archives, pour cela on peut utiliser la commande wget.

```
$SRCDIR/archives $ wget ftp://ftp.gnu.org/gnu/binutils/binutils-2.22.tar.bz2
$SRCDIR/archives $ wget ftp://ftp.gnu.org/gnu/gcc/gcc-4.7.3/gcc-4.7.3.tar.bz2
$SRCDIR/archives $ wget ftp://ftp.gnu.org/gnu/glibc/glibc-2.19.tar.xz
$SRCDIR/archives $ wget ftp://ftp.gnu.org/gnu/gmp/gmp-5.0.5.tar.xz
$SRCDIR/archives $ wget ftp://ftp.gnu.org/gnu/mpfr/mpfr-3.1.1.tar.xz
$SRCDIR/archives $ wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.5.3.tar.xz
$SRCDIR $ for i in archives/* ; do tar -xvf $i ; done
```

C – Compilation de Binutils

Voici les commandes à exécuter pour compiler Binutils suivis d'une description des arguments passés au configure :

```
$BUILDDIR $ mkdir binutils/ && cd binutils/
$BUILDDIR/binutils $ ../../sources/binutils-2.22/configure \
        --disable-werror \
        --build=$BUILD \
        --target=$TARGET \
        --with-sysroot=$SYSROOTDIR \
        --prefix=$INSTALLDIR
$BUILDDIR/binutils $ make
$BUILDDIR/binutils $ make install
```

- --disable-werror : Désactive les messages d'erreur et de warning.
- --build : Précise la machine sur laquelle sera construite la chaîne de compilation.
- --target : Précise la machine sur laquelle seront exécutés les programmes compilés par le compilateur croisé issu de la chaîne de compilation.
- --with-sysroot : Permet de définir une racine d'arborescence de notre choix, par défaut il s'agit de la racine de l'arborescence de la machine BUILD, ce qui convient dans le cas d'une installation classique d'un logiciel mais pas dans le cas présent où les machines BUILD et TARGET sont différentes. Le dossier SYSROOTDIR sera la future racine de l'arborescence de la machine TARGET, il accueillera notamment des bibliothèques.
- --prefix : Précise dans quel dossier installer les fichiers exécutables produits.

D – Création des entêtes du noyau Linux

Voici les commandes à exécuter pour créer les entêtes du noyau Linux. Ces entêtes sont ce qui permet aux programmes d'interagir avec le système d'exploitation.

```
$SRCDIR/linux-3.5.3 $ make mrproper
$SRCDIR/linux-3.5.3 $ make ARCH=arm integrator_defconfig
$SRCDIR/linux-3.5.3 $ make ARCH=arm headers_check
$SRCDIR/linux-3.5.3 $ make ARCH=arm INSTALL_HDR_PATH=$INSTALLDIR/sysroot/usr headers_install
```

- Ligne1 : Remet le dossier en l'état tel qu'à sa décompression de l'archive, efface toutes traces d'une configuration précédente.
- Ligne2 : Sélectionne une architecture ARM particulière, integrator est une option générique figurant parmi une longue liste de processeurs ARM pris en charge.
- Ligne3 : Vérifie et nettoie les entêtes du noyau. Ces trois premières étapes ne sont pas nécessaires dans la mesure où le dossier source linux-3.5.3/ vient d'être extrait de son archive et où ce TP étant à titre pédagogique, une architecture ARM spécifique n'est pas requise.
- Ligne4 : Installe les entêtes du noyau à l'emplacement spécifié.

E – Compilation d’un Gcc minimaliste

Ce Gcc minimal est ce qui servira à compiler une bibliothèque standard C qui servira elle même à élaborer un nouveau compilateur disposant de davantage de fonctionnalités que ce compilateur capable de compiler lui même et la bibliothèque C standard. Voici les commandes permettant de compiler ce Gcc :

```
$BUILDDIR $ mkdir gcc-bootstrap/ && cd gcc-bootstrap/
$BUILDDIR/gcc-bootstrap $ ../../sources/gcc-4.7.3/configure \
        --build=$BUILD \
        --host=$BUILD \
        --target=$TARGET \
        --prefix=$INSTALLDIR \
        --enable-bootstrap \
        --without-headers \
        --enable-languages=c \
        --disable-threads \
        --enable-__cxa_atexit \
        --disable-libmudflap \
        --with-gnu-ld \
        --with-gnu-as \
        --disable-libssp \
        --disable-libgomp \
        --disable-nls \
        --disable-shared
$BUILDDIR $ make all-gcc install-gcc
$BUILDDIR $ make all-target-libgcc install-target-libgcc
```

- --enable-bootstrap : Crée un compilateur minimaliste.
- --without-headers : Le compilateur n’inclura pas les entêtes du noyau.
- --enable-languages=c : Le compilateur sera capable de compiler du langage C (et rien d’autre).
- --disable -threads : Désactive le support du multi threads.
- --enable-__cxa_atexit : Implémente la fonction __cxa_atexit, liée aux destructeurs de fonctions.
- --disable-libmudflap : Désactive l’utilisation d’un bibliothèque liée au débug.
- --with-gnu-ld : Utilise l’éditeur de liens natif de la machine BUILD pour la compilation de Gcc.
- --with-gnu-as : Utilise l’assembleur natif de la machine BUILD pour la compilation de Gcc.
- --disable-libssp : Désactive l’utilisation d’une bibliothèque liée au dépassement mémoire.
- --disable-nls : Désactive le support d’autres langues que l’anglais.
- --disable-shared : Le compilateur ne sera pas capable de créer des programmes utilisant les bibliothèques dynamiques.

F – Création des entêtes de la Glibc

Il faut maintenant créer une bibliothèque C standard compilée, pour cela il faut créer les entêtes permettant d'interagir avec la bibliothèque C. Ces entêtes incluent les entêtes du noyau.

```
$INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.7.3 $ ln -s libgcc.a libgcc_sh.a
$BUILDDIR $ mkdir libc-headers/ && cd libc-headers/
$BUILDDIR/libc-headers $ export CROSS=arm-none-linux-gnueabi
$BUILDDIR/libc-headers $ export CC=${CROSS}-gcc
$BUILDDIR/libc-headers $ export LD=${CROSS}-ld
$BUILDDIR/libc-headers $ export AS=${CROSS}-as
$BUILDDIR/libc-headers $ export PATH=$INSTALLDIR/bin:$PATH
$BUILDDIR/libc-headers $ echo "libc_cv_forced_unwind=yes" > config.cache
$BUILDDIR/libc-headers $ echo "libc_cv_c_cleanup=yes" >> config.cach
$BUILDDIR/libc-headers $ ../../sources/glibc-2.19/configure \
        --build=$BUILD \
        --host=$TARGET \
        --prefix=$SYSROOTDIR \
        --with-headers=$SYSROOTDIR/usr/include \
        --config-cache \
        --enable-add-ons=ports,nptl \
        --enable-kernel=3.5.3
$BUILDDIR/libc-headers $ make -k install-headers cross_compiling=yes install_root=$SYSROOTDIR
$INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.7.3 $ ln -s libgcc.a libgcc_eh.a
$INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.7.3 $ ln -s libgcc.a libgcc_s.a
```

Les créations de liens symboliques ln, servent à forcer l'inclusion de ces bibliothèques en statique. Avant de procéder à la création de ces entêtes, il est nécessaire de modifier certaines variables d'environnement afin de ne plus utiliser le compilateur natif de la machine BUILD mais le Gcc minimaliste créé à l'étape précédente, se trouvant à l'emplacement $INSTALLDIR/bin/arm-none-linux-gnueabi-gcc et l'assembleur et l'éditeur de lien créés en début de chaîne de compilation et se trouvant respectivement aux emplacements $INSTALLDIR/bin/arm-none-linux-gnueabi-as et $INSTALLDIR/bin/arm-none-linux-gnueabi-ld.

- --host : On remarque que le HOST vis à vis des bibliothèques est la TARGET des compilateurs, ce qui est tout à fait logique puisqu'elles sont destinées à faire partie de programmes fonctionnant sur ARM.
- --with-headers : Indique où trouver les entêtes du noyau.
- --config-cache : Ordonne au configure de lire le fichier config.cache.
- --enable-add-ons : ajoute une bibliothèque supplémentaire.
- --enable-kernel : Conçois une bibliothèque destinée à fonctionner avec telle version du noyau.

G – Compilation de la Glibc

Il faut compiler la bibliothèque C standard toujours en utilisant le Gcc minimaliste.

```
$BUILDDIR $ mkdir glibc/ && cd glibc/
$BUILDDIR/glibc $ echo "libc_cv_forced_unwind=yes" > config.cache
$BUILDDIR/glibc $ echo "libc_cv_c_cleanup=yes" >> config.cache
```

```
$BUILDDIR/glibc $ ../../sources/glibc-2.19/configure \
        --build=$BUILD \
        --host=$TARGET \
        --prefix=/usr \
        --with-headers=$SYSROOTDIR/usr/include \
        --config-cache \
        --enable-add-ons=ports,nptl \
        --enable-kernel=3.5.3
$BUILDDIR/glibc $ make
$BUILDDIR/glibc $ make install_root=$SYSROOTDIR install
```

Les paramètres du configure sont très semblables à ceux du configure des entêtes de la Glibc.

H – Compilation d'un second Gcc

Maintenant que l'on dispose d'une bibliothèque standard C, il est possible de créer le second compilateur incluant cette bibliothèque.

```
$BUILDDIR $ mkdir gcc-intermediate/ && cd gcc-intermediate/
$BUILDDIR/gcc-intermediate $ unset CROSS
$BUILDDIR/gcc-intermediate $ unset CC
$BUILDDIR/gcc-intermediate $ unset AS
$BUILDDIR/gcc-intermediate $ unset LD
$BUILDDIR/gcc-intermediate $ echo "libc_cv_forced_unwind=yes" > config.cache
$BUILDDIR/gcc-intermediate $ echo "libc_cv_c_cleanup=yes" >> config.cache
$BUILDDIR/gcc-intermediate $ ../../sources/gcc-4.7.3/configure \
        --build=$BUILD \
        --target=$TARGET \
        --prefix=$INSTALLDIR \
        --with-sysroot=$SYSROOTDIR \
        --enable-languages=c \
        --with-gnu-as \
        --with-gnu-ld \
        --disable-multilib \
        --with-float=soft \
        --disable-sjlj-exceptions \
        --disable-nls \
        --enable-threads=posix \
        --enable-long-longx
$BUILDDIR/gcc-intermediate $ make all-gcc
$BUILDDIR/gcc-intermediate $ make install-gcc
```

Pour compiler ce nouveau Gcc, nous n'utilisons plus le Gcc minimaliste mais de nouveau le compilateur natif de la machine BUILD, d'où la désaffectation des variables d'environnement CC, AS et LD .

- --disable-multilib : Désactive la création de variantes des bibliothèques pour diverses architectures.
- --with-float : Active le support de nombres à virgules flottantes via le logiciel car les architectures ARM n'incluent pas de FPU matérielle.
- --disable-sjlj-exceptions : Désactive une option relative au C++ qui n'est pas compris par le compilateur.
- --enable-threads : Implémente le support des threads selon la norme POSIX

- --enable-long-longx : Active le support du type long long dans le compilateur C.

I – Compilation de la bibliothèque Gmp

Les deux bibliothèques seront compilées avec le Gcc nouvellement créé, d'où la remise en place des variables d'environnement pointant vers le dossier $INSTALLDIR/bin.

```
$BUILDDIR $ mkdir gmp/ && cd gmp/
$BUILDDIR/gmp $ export CROSS=arm-none-linux-gnueabi
$BUILDDIR/gmp $ export CC=${CROSS}-gcc
$BUILDDIR/gmp $ export LD=${CROSS}-ld
$BUILDDIR/gmp $ export AS=${CROSS}-as
$BUILDDIR/gmp $ export CFLAGS=-static
$BUILDDIR/gmp $ ../../sources/gmp-5.0.5/configure \
        --build=$BUILD \
        --host=$TARGET \
        --prefix=$INSTALLDIR \
        --disable-shared
$BUILDDIR/gmp $ make
$BUILDDIR/gmp $ make install
```

J – Compilation de la bibliothèque Mpfr

Il en va de même pour la bibliothèque Mpfr. Ces deux bibliothèques mathématiques servent au calcul de nombres à virgule flottante, l'intérêt de les intégrer au compilateur croisé vient du fait que les processeurs ARM n'incluent pas de FPU (Floating Point Unit), unité matérielle de calcul à virgule flottante. C'est à dire que ces bibliothèques remplaceront logiciellement cette unité.

```
$BUILDDIR $ mkdir mpfr/ && cd mpfr/
$BUILDDIR/mpfr $ ../../sources/gmp-5.0.5/configure \
        --build=$BUILD \
        --host=$TARGET \
        --prefix=$INSTALLDIR \
        --with-gmp=$INSTALLDIR
$BUILDDIR/mpfr $ make
$BUILDDIR/mpfr $ make install
```

- --with-gmp : Gmp est une dépendance de Mpfr d'où ce renseignement.

K – Compilation du Gcc final

Voilà le dernière étape de la chaîne de compilation, le Gcc final produit ici est le compilateur croisé qui s'exécutera sur machine x86_64 et produira du code pour machines d'architecture ARM. Comme pour le deuxième Gcc, il convient de désaffecter les variables d'environnement CC, AS et LD afin de faire appel au compilateur natif de la machine et non plus au Gcc précédent.

```
$BUILDDIR $ mkdir gcc-final/ && cd gcc-final/
$BUILDDIR/gcc-final $ unset CROSS
$BUILDDIR/gcc-final $ unset CC
$BUILDDIR/gcc-final $ unset AS
$BUILDDIR/gcc-final $ unset LD
$BUILDDIR/gcc-final $ unset CFLAGS
$BUILDDIR/gcc-final $ export CC=gcc
$BUILDDIR/gcc-final $ echo "libc_cv_forced_unwind=yes" > config.cache
```

```
$BUILDDIR/gcc-final $ echo "libc_cv_c_cleanup" >> config.cache
$BUILDDIR/gcc-final $ ../../sources/gcc-4.7.3/configure \
        --build=$BUILD \
        --target=$TARGET \
        --prefix=$INSTALLDIR \
        --with-sysroot=$SYSROOTDIR \
        --enable-languages=c \
        --with-gnu-ld \
        --with-gnu-as \
        --disable-multilib \
        --with-float=soft \
        --disable-sjls-exceptions \
        --disable-nls \
        --enable-threads=posix \
        --disable-libmudflap \
        --disable-libssp \
        --enable-long-longx \
        --with-shared \
        --with-gmp=$INSTALLDIR \
        --with-mpfr=$INSTALLDIR
$BUILDDIR/gcc-final $ make
$BUILDDIR/gcc-final $ make install
```

- --with-shared : Active le support des bibliothèques dynamiques.
- --with-gmp : Inclut Gmp et renseigne sur son emplacement.
- --with-mpfr : Inclut Mpfr et renseigne sur son emplacement.

La chaîne de compilation est maintenant terminée, il est possible de vérifier son fonctionnement. Pour cela on écrit un programme C simple, un « hello word » par exemple que l'on compile avec le Gcc final, puis on analyse l'exécutable obtenu avec la commande file. Le résultat est univoque, il doit s'agir d'un exécutable ARM.

```
~ $ $INSTALLDIR/bin/arm-none-linux-gnueabi-gcc helloword.c -o helloword
~ $ file helloword

helloword: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux.so.3, for GNU/Linux 3.5.3, not stripped
```

Conclusion

Nous avons mené à bien cette chaîne de compilation croisée et élaboré un script d'automatisation du processus. Les tests ont été effectués sur une machine virtuelle Debian 8, certains ont étés fait sur Ubuntu et se sont avérés infructueux, il serait maintenant intéressant de se pencher davantage sur les dépendances de cette chaîne de compilation afin de porter le script à d'autres distributions GNU/Linux, y compris des distributions utilisant d'autres gestionnaires de paquets que ceux de Debian (yum, pacman, etc.).