*Workshop on Essential Abstractions in GCC*

# GCC for Cross Compilation

GCC Resource Center

(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

July 2010

# Outline

- Overview
- Building Binutils
- Building First GCC
- Installing Header Files
- Building Second GCC
- Building Final C Libraries
- Final Build
- Using the Cross Compiler Tool Chain

*Part 1*

# *Overview*

# Overview of the Cross Compilation Procedure

1. Build a cross compiler with certain facilities disabled (First GCC build).
2. Configure the C library using the compiler built in Step 1. Build a few of the C run-time object files, but not rest of the library. Install the library's header files and run-time object file, and create dummy libc.so.
3. Build a second cross-compiler (Second GCC build), using the header files and object files installed in Step 2.
4. Configure, build and install fresh C library, using the compiler built in Step 3.
5. Build a third cross compiler (Third GCC build), based on the C library built in Step 4.

# Why Such a Complex Procedure?

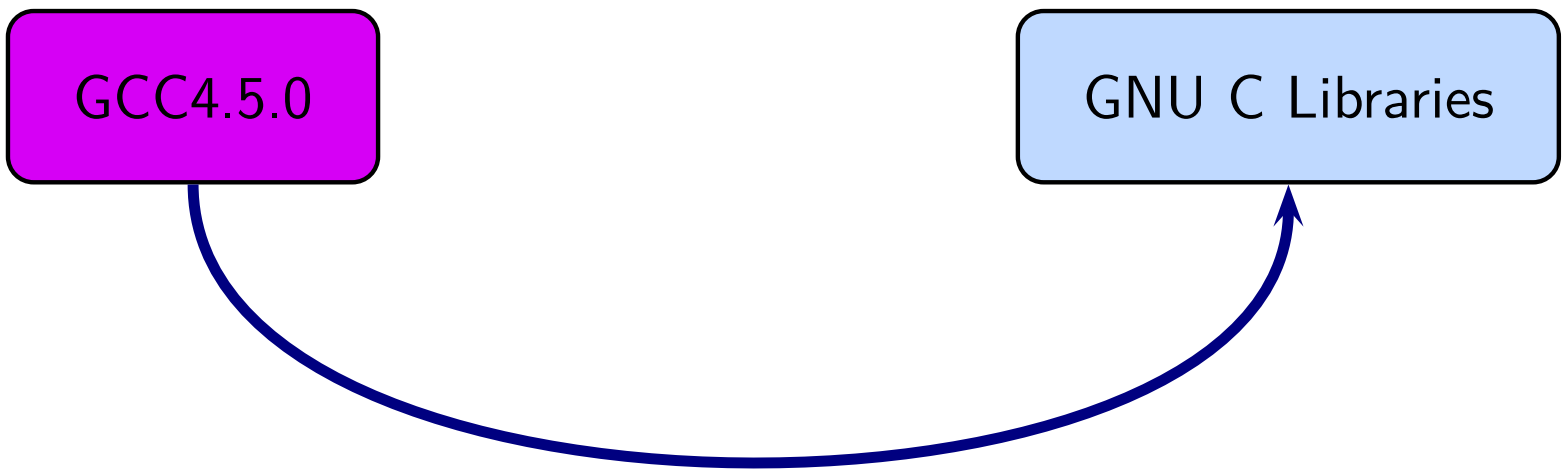
GCC4.5.0

# Why Such a Complex Procedure?

GCC4.5.0

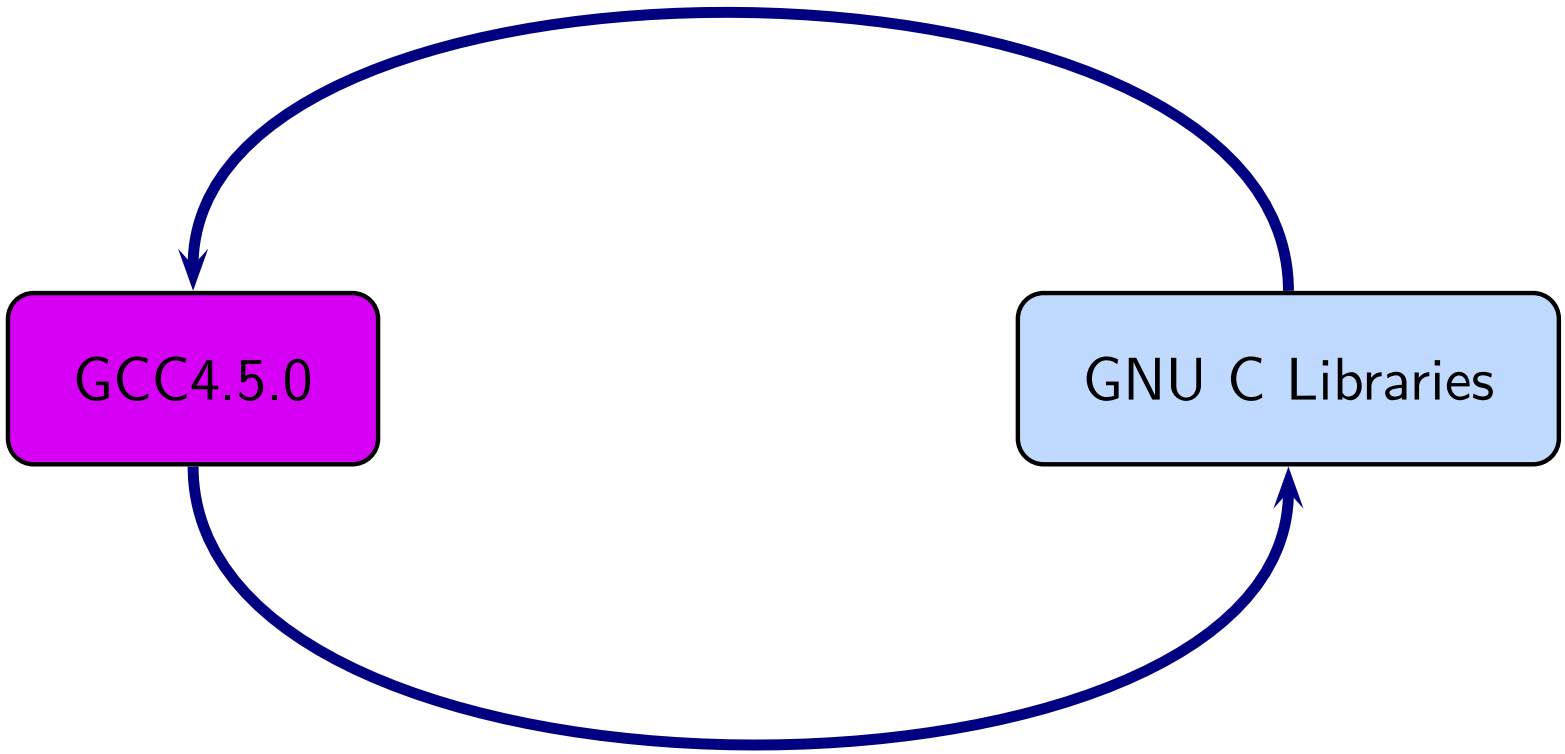
GNU C Libraries

# Why Such a Complex Procedure?




GCC requires the C library headers and some object file to compile its own libraries

# Why Such a Complex Procedure?




C library depends on GCC's libraries

# Downloading Source Tarballs

Download the latest version of source tarballs

| Tar File Name | Download URL |
|---|---|
| gcc-4.5.0.tar.gz | gcc.cybermirror.org/releases/gcc-4.5.0/ |
| binutils-2.20.tar.gz | ftp.gnu.org/gnu/binutils/ |
| Latest revision of EGLIBC | svn co svn://svn.eglibc.org/trunk eglibc |
| linux-2.6.33.3.tar.gz | www.kernel.org/pub/linux/kernel/v2.6/ |

# Setting Up the Environment for Cross Compilation

- Create a folder 'crossbuild' that will contain the crossbuilt compiler sources and binaries.

```
$.mkdir crossbuild
$.cd crossbuild
```

- Create independent folders that will contain the source code of gcc-4.5.0, binutil, and eglibc.

```
crossbuild$.mkdir gcc
crossbuild$.mkdir eglibc
crossbuild$.mkdir binutils
```

# Setting Up the Environment for Cross Compilation

- Create a folder that will contain the cross toolchain.

```
crossbuild$.mkdir install
```

- Create a folder that will have a complete EGLIBC installation, as well as all the header files, library files, and the startup C files for the target system.

```
crossbuild$.mkdir sysroot
```

# Setting Up the Environment for Cross Compilation

- Create a folder that will contain the cross toolchain.

```
crossbuild$.mkdir install
```

- Create a folder that will have a complete EGLIBC installation, as well as all the header files, library files, and the startup C files for the target system.

```
crossbuild$.mkdir sysroot
```

sysroot ≡ standard linux directory layout

# Setting the Environment Variables

Set the environment variables to generalize the later steps for cross build.

```
crossbuild$.export prefix=<path_to crossbuild/install>
crossbuild$.export sysroot=<path_to crossbuild/sysroot>
crossbuild$.export host=i686-pc-linux-gnu
crossbuild$.export build=i686-pc-linux-gnu
crossbuild$.export target=mips-linux OR
            export target=powerpc-linux
crossbuild$.export linuxarch=mips OR
            export linuxarch=powerpc
```

*Part 2*

# *Building Binutils*

# Building Binutils

- Change the working directory to binutils.

```
crossbuild$. cd binutils
```

- Untar the binutil source tarball here.

```
crossbuild/binutils$. tar -xvf binutils-2.20.tar.gz
```

- Make a build directory to configure and build the binutils, and go to that dicrectory.

```
crossbuild/binutils$. mkdir build
crossbuild/binutils$. cd build
```

# Building Binutils

- Configure the binutils:

```
crossbuild/binutils/build$. ../binutils-2.20/configure
--target=$target --prefix=$prefix --with-sysroot=$sysroot
```

- Install the binutils:

```
crossbuild/binutils/build$. make
crossbuild/binutils/build$. make install
```

- Change the working directory back to crossbuild.

```
crossbuild/binutils/build$. cd ~/crossbuild
```



*Part 3*

# *Building First GCC*

# Building First GCC

- Change the working directory to gcc.

```
crossbuild$. cd gcc
```

- Untar the gcc-4.5.0 source tarball here.

```
crossbuild/gcc$. tar -xvf gcc-4.5.0.tar.gz
```

- Make a build directory to configure and build gcc, and go to that directory.

```
crossbuild/gcc$. mkdir build
crossbuild/gcc$. cd build
```

libgcc and other libraries are built using libc headers. Shared libraries like 'libgcc_s.so' are to be compiled against EGLIBC headers (not installed yet), and linked against 'libc.so' (not built yet). We need configure time options to tell GCC not to build 'libgcc_s.so'.

# Building First GCC

- Configure gcc:

```
crossbuild/gcc/build$. ../gcc-4.5.0/configure
--target=$target --prefix=$prefix --without-headers
--with-newlib --disable-shared --disable-threads
--disable-libssp --disable-libgomp --disable-libmudflap
--enable-languages=c
```

'--without-headers' ⇒ build libgcc without any headers at all.
'--with-newlib' ⇒ use newlib header while building other libraries than libgcc.
Using both the options together results in libgcc being built without requiring the presence of any header, and other libraries being built with newlib headers.

# Building First GCC

- Install gcc in the install folder:

```
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make all-gcc
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make
install-gcc
```

- change the working directory back to crossbuild.

```
crossbuild/gcc/build$. cd ~/crossbuild
```

*Part 4*

# *Installing Header Files*

# Installing Linux Kernel Headers

Linux makefiles are target-specific

- Untar the linux kernel source tarball.

```
crossbuild$.tar -xvf linux-2.6.33.3.tar.gz
```

- Change the working directory to linux-2.6.33.3

```
crossbuild$.cd linux-2.6.33.3
```

- Install the kernel headers in the sysroot directory:

```
crossbuild/linux-2.6.33.3$.PATH=$prefix/bin:$PATH make
headers_install CROSS_COMPILE=$target-
INSTALL_HDR_PATH=$sysroot/usr ARCH=$linuxarch
```

- change the working directory back to crossbuild.

```
crossbuild/linux-2.6.33.3$.cd ~/crossbuild
```

# Installing EGLIBC Headers and Preliminary Objects

Using the cross compiler that we have just built, configure EGLIBC to install the headers and build the object files that the full cross compiler will need.

- Change the working directory to eglibc.

```
crossbuild$. cd eglibc
```

- Check the latest eglibc source revision here.

```
crossbuild/eglibc$. svn co svn://svn.eglibc.org/trunk
eglibc
```

- Some of the targets are not supported by glibc (e.g. mips). The support for such targets is provided in the 'ports' folder in eglibc. We need to copy this folder inside the libc folder to create libraries for the new target.

```
crossbuild/eglibc$. cp -r eglibc/ports eglibc/libc
```

# Installing EGLIBC Headers and Preliminary Objects

- Make a build directory to configure and build eglibc headers, and go to that directory.

```
crossbuild/eglibc$. mkdir build
crossbuild/eglibc$. cd build
```

- Configure eglibc:

```
crossbuild/eglibc/build$. BUILD_CC=gcc
CC=$prefix/bin/$target-gcc  AR=$prefix/bin/$target-ar
RANLIB=$prefix/bin/$target-ranlib ../eglibc/libc/configure
--prefix=/usr  --with-headers=$sysroot/usr/include
--build=$build  --host=$target  --disable-profile
--without-gd  --without-cvs  --enable-add-ons
```

EGLIBC must be configured with option '--prefix=/usr', because the EGLIBC build system checks whether the prefix is '/usr', and does special handling only if that is the case.

# Installing EGLIBC Headers and Preliminary Objects

- We can now use the 'install-headers' makefile target to install the headers:

```
crossbuild/eglibc/build$. make install-headers
install_root=$sysroot \install-bootstrap-headers=yes
```

'install-bootstrap-headers' variable requests special handling for certain tricky header files.

- There are a few object files that are needed to link shared libraries. We will build and install them by hand:

```
crossbuild/eglibc/build$. mkdir -p $sysroot/usr/lib
crossbuild/eglibc/build$. make csu/subdir_lib
crossbuild/eglibc/build$. cd csu
crossbuild/eglibc/build/csu$. cp crt1.o  crti.o  crtn.o
$sysroot/usr/lib
```

# Installing EGLIBC Headers and Preliminary Objects

- Finally, 'libgcc_s.so' requires a 'libc.so' to link against. However, since we will never actually execute its code, it doesn't matter what it contains. So, treating '/dev/null' as a C souce code, we produce a dummy 'libc.so' in one step:

```
crossbuild/eglibc/build/csu$. $prefix/bin/$target-gcc
-nostdlib -nostartfiles -shared -x c /dev/null -o
$sysroot/usr/lib/libc.so
```

- change the working directory back to crossbuild.

```
crossbuild/gcc/build$. cd ~/crossbuild
```

*Part 5*

# *Building Second GCC*

# Building the Second GCC

With the EGLIBC headers and the selected object files installed, build a GCC that is capable of compiling EGLIBC.

- Change the working directory to build directory inside gcc folder.

```
crossbuild$. cd gcc/build
```

- Clean the build folder.

```
crossbuild/gcc/build$. rm -rf *
```

- Configure the second gcc:

```
crossbuild/gcc/build$. ../gcc-4.5.0/configure
--target=$target  --prefix=$prefix  --with-sysroot=$sysroot
--disable-libssp  --disable-libgomp  --disable-libmudflap
--enable-languages=c
```

# Building the Second GCC

- install the second gcc in the install folder:

```
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make install
```

- change the working directory back to crossbuild.

```
crossbuild/gcc/build$. cd ~/crossbuild
```



*Part 6*

# *Building Final C Libraries*

# Building Complete EGLIBC

With the second compiler built and installed, build EGLIBC completely.

- Change the working directory to the build directory inside eglibc folder.

```
crossbuild$. cd eglibc/build
```

- Clean the build folder.

```
crossbuild/eglibc/build$. rm -rf *
```

- Configure eglibc:

```
crossbuild/eglibc/build$. BUILD_CC=gcc
CC=$prefix/bin/$target-gcc  AR=$prefix/bin/$target-ar
RANLIB=$prefix/bin/$target-ranlib ../eglibc/libc/configure
--prefix=/usr  --with-headers=$sysroot/usr/include
--build=$build  --host=$target  --disable-profile
--without-gd  --without-cvs  --enable-add-ons
```

# Building Complete EGLIBC

- install the required libraries in $sysroot:

```
crossbuild/eglibc/build$. PATH=$prefix/bin:$PATH make
crossbuild/eglibc/build$. PATH=$prefix/bin:$PATH make
install install_root=$sysroot
```

- change the working directory back to crossbuild.

```
crossbuild/gcc/build$. cd ~/crossbuild
```

At this point, we have a complete EGLIBC installation in '$sysroot', with header files, library files, and most of the C runtime startup files in place.

*Part 7*

# *The Final Build*

# Building fully Cross-compiled GCC

Recompile GCC against this full installation, enabling whatever languages and libraries you would like to use.

- Change the working directory to build directory inside gcc folder.

```
crossbuild$. cd gcc/build
```

- Clean the build folder.

```
crossbuild/gcc/build$. rm -rf *
```

- Configure the third gcc:

```
crossbuild/gcc/build$. ../gcc-4.5.0/configure
--target=$target  --prefix=$prefix  --with-sysroot=$sysroot
--disable-libssp  --disable-libgomp  --disable-libmudflap
--enable-languages=c
```

# Building fully Cross-compiled GCC

- Install the final gcc in the install folder:

```
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make
crossbuild/gcc/build$. PATH=$prefix/bin:$PATH make install
```

- change the working directory back to crossbuild.

```
crossbuild/gcc/build$. cd ~/crossbuild
```

# Maintaining $sysroot Folder

Since GCC's installation process is not designed to help construct sysroot trees, certain libraries must be manually copied into place in the sysroot.

- Copy the libgcc_s.so files to the lib folder in $sysroot.

```
crossbuild$.cp -d  $prefix/$target/lib/libgcc_s.so*
$sysroot/lib
```

- If c++ language was enabled, copy the libstdc++.so files to the usr/lib folder in $sysroot.

```
crossbuild$.cp -d  $prefix/$target/lib/libstdc++.so*
$sysroot/usr/lib
```

At this point, we have a ready cross compile toolchain in $prefix, and EGLIBC installation in $sysroot.

*Part 8*

# *Using the Cross Compiler Tool Chain*

# Testing the Cross Compiler

Sample input file test.c:

```
#include <stdio.h>
int main ()
{
        int a, b, c, *d;
        d = &a;
        a = b + c;
        printf ("%d", a);
        return 0;
}
```

```
$. $prefix/bin/$target-gcc -o test test.c
```

# Testing the Cross Compiler

For a powerpc architecture,

```
$. $prefix/bin/powerpc-unknown-linux-gnu-gcc -o test test.c
```

Use readelf to verify whether the executable is indeed for powerpc

```
$. $prefix/bin/powerpc-unknown-linux-gnu-readelf -lh test
```

```
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  ...
  Type:                              EXEC (Executable file)
  Machine:                           PowerPC
  ...
Program Headers:
  ...
      [Requesting program interpreter: /lib/ld.so.1]
  ...
```