GCC Configuration and Building

Uday Khedker
(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

July 2008
Part 1

Configuration and Building
Configuring GCC

configure
Configuring GCC

config.guess

configure
Configuring GCC

- configure
- config.guess
- configure.in
Configuring GCC

```
configure
config.guess
config.in
config/*
configure
```
Configuring GCC

- configure
- config.guess
- config.sub
- config.in
- config/*
Configuring GCC

configure.in  config/*

config.guess  config.sub

config.log  config.cache  config.status
Configuring GCC

- configure.in
- configure
- config.guess
- config.sub
- config.h.in
- config.log
- config.cache
- config.status
- config.sub
- makefile.in
- config/*
Configuring GCC

- configure
- config.guess
- config.sub
- config.log
- config.cache
- config.status
- config.h.in
- makefile.in
- config.h
Alternatives in Configuration

GCC 3.3

$BUILD

configure

SubDir1

configure

SubDir2

configure
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure

SubDir2
configure
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
Alternatives in Configuration

GCC 3.3

$BUILD

configure

Makefile

SubDir1

configure

Makefile

SubDir2

configure

Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1

configure
Makefile

SubDir2

configure
Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure
make

SubDir1
configure
Makefile

SubDir2
configure
Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure
make

SubDir1
configure
make

SubDir2
configure
Makefile
## Alternatives in Configuration

### GCC 3.3

<table>
<thead>
<tr>
<th>$BUILD</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure</td>
</tr>
<tr>
<td>make</td>
</tr>
</tbody>
</table>

- **SubDir1**
  - configure
  - Makefile

- **SubDir2**
  - configure
  - make
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD

configure

SubDir1
configure

SubDir2
configure

SubDir2
configure
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD

configure
Makefile

SubDir1
configure

SubDir2
configure
Alternatives in Configuration

GCC 3.3

$BUILD
configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD
configure
make

SubDir1
configure

SubDir2
configure

SubDir2
configure
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD

configure
make

SubDir1
configure
Makefile

SubDir2
configure
Makefile
Alternatives in Configuration

<table>
<thead>
<tr>
<th>GCC 3.3</th>
<th>GCC 4.0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BUILD</td>
<td></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
<tr>
<td>Makefile</td>
<td></td>
</tr>
<tr>
<td>SubDir1</td>
<td></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
<tr>
<td>Makefile</td>
<td></td>
</tr>
<tr>
<td>SubDir2</td>
<td></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
<tr>
<td>Makefile</td>
<td></td>
</tr>
<tr>
<td>SubDir1</td>
<td></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
<tr>
<td>make</td>
<td></td>
</tr>
<tr>
<td>SubDir2</td>
<td></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
</tbody>
</table>
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD

configure
make

SubDir1
configure
Makefile

SubDir2
configure
Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD

configure

make

SubDir1
configure
Makefile

SubDir2
configure
Makefile
Alternatives in Configuration

GCC 3.3

$BUILD

configure

Makefile

SubDir1

configure

Makefile

SubDir2

configure

Makefile

GCC 4.0.2

$BUILD

configure

SubDir1

configure

Makefile

SubDir2

configure

make

make
Alternatives in Configuration

GCC 3.3

$BUILD

configure

Makefile

SubDir1

configure

Makefile

SubDir2

configure

Makefile

GCC 4.0.2

$BUILD

configure

make

SubDir1

configure

Makefile

SubDir2

configure

Makefile

UPK IIT Bombay
Alternatives in Configuration

GCC 3.3

$BUILD
configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile

GCC 4.0.2

$BUILD
configure
Makefile

SubDir1
configure
Makefile

SubDir2
configure
Makefile
## Steps in Configuration and Building

### Usual Steps

- Download and untar the source
# Steps in Configuration and Building

**Usual Steps**

- Download and untar the source
- `cd $SOURCE`
## Steps in Configuration and Building

### Usual Steps

- Download and untar the source
- `cd $SOURCE`
- `./configure`
Steps in Configuration and Building

Usual Steps

- Download and untar the source
- cd $SOURCE
- ./configure
- make
Steps in Configuration and Building

Usual Steps

- Download and untar the source
- cd $SOURCE
- ./configure
- make
- make install
### Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td></td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td></td>
</tr>
<tr>
<td>• ./configure</td>
<td></td>
</tr>
<tr>
<td>• make</td>
<td></td>
</tr>
<tr>
<td>• make install</td>
<td></td>
</tr>
</tbody>
</table>
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td></td>
</tr>
<tr>
<td>• ./configure</td>
<td></td>
</tr>
<tr>
<td>• make</td>
<td></td>
</tr>
<tr>
<td>• make install</td>
<td></td>
</tr>
</tbody>
</table>
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download and untar the source</td>
<td>Download and untar the source</td>
</tr>
<tr>
<td>cd $SOURCE</td>
<td>cd $BUILD</td>
</tr>
<tr>
<td>./configure</td>
<td></td>
</tr>
<tr>
<td>make</td>
<td></td>
</tr>
<tr>
<td>make install</td>
<td></td>
</tr>
</tbody>
</table>
# Steps in Configuration and Building

## Usual Steps
- Download and untar the source
- `cd $SOURCE`
- `./configure`
- `make`
- `make install`

## Steps in GCC
- Download and untar the source
- `cd $BUILD`
- `$SOURCE/configure`
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td>• cd $BUILD</td>
</tr>
<tr>
<td>• ./configure</td>
<td>• $SOURCE/configure</td>
</tr>
<tr>
<td>• make</td>
<td>• make</td>
</tr>
<tr>
<td>• make install</td>
<td></td>
</tr>
</tbody>
</table>
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td>• cd $BUILD</td>
</tr>
<tr>
<td>• ./configure</td>
<td>• $SOURCE/configure</td>
</tr>
<tr>
<td>• make</td>
<td>• make</td>
</tr>
<tr>
<td>• make install</td>
<td>• make install</td>
</tr>
</tbody>
</table>

UPK IIT Bombay
## Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td>• cd $BUILD</td>
</tr>
<tr>
<td>• ./configure</td>
<td>• $SOURCE/configure</td>
</tr>
<tr>
<td>• make</td>
<td>• make</td>
</tr>
<tr>
<td>• make install</td>
<td>• make install</td>
</tr>
</tbody>
</table>

$SOURCE$ and $BUILD$ must be distinct!
# Steps in Configuration and Building

<table>
<thead>
<tr>
<th>Usual Steps</th>
<th>Steps in GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Download and untar the source</td>
<td>• Download and untar the source</td>
</tr>
<tr>
<td>• cd $SOURCE</td>
<td>• cd $BUILD</td>
</tr>
<tr>
<td>• ./configure</td>
<td>• $SOURCE/configure</td>
</tr>
<tr>
<td>• make</td>
<td>• make</td>
</tr>
<tr>
<td>• make install</td>
<td>• make install</td>
</tr>
<tr>
<td>$SOURCE and $BUILD must be distinct!</td>
<td>$SOURCE and $BUILD must be distinct!</td>
</tr>
</tbody>
</table>

*GCC generates a large part of source code during configuration!*
Some Interesting Facts about GCC 4.0.2

Pristine compiler sources (downloaded tarball)

- Lines of C code: 1,098,306
- Lines of MD code: 217,888
- Lines of total code: 1,316,194
- Total Authors (approx): 63
- Backend directories: 34

For the targeted (= pristine + generated) C compiler

- Total lines of code: 810,827
- Total lines of pure code: 606,980
- Total pure code WITHOUT #includes: 602,351
- Total number of #include directives: 4,629
- Total #included files: 336
Some Interesting i386 MD Facts

General information

Number of .md files 8
Number of C files 72

Realistic code size information (excludes comments)

Total lines of code 47290
Total lines of .md code 23566
Total lines of header code 9986
Total lines of C code 16961
Other Facts about the Size of GCC

<table>
<thead>
<tr>
<th></th>
<th>GCC 4.0.2</th>
<th>GCC 4.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed tar file (.tar.bz2)</td>
<td>30.3 MB</td>
<td>6.6 MB</td>
</tr>
<tr>
<td>Uncompressed source</td>
<td>378 MB</td>
<td>511 MB</td>
</tr>
</tbody>
</table>

The size of $BUILD$ directory is usually larger than $SOURCE$ and depends on the target specification.
Some Terminology

- The sources of a compiler are compiled (i.e. built) on machine X
  X is called as the **Build system**

- The built compiler runs on machine Y
  Y is called as the **Host system**

- The compiler compiles code for target Z
  Z is called as the **Target system**

- **Note:** The built compiler itself **runs** on the Host machine and generates executables that run on Target machine!!!
Building a Compiler: General issues II

Some Definitions

Note: The built compiler itself runs on the Host machine and generates executables that run on Target machine!!!

A few interesting permutations of X, Y and Z are:

\[ X = Y = Z \] Native build
\[ X = Y \neq Z \] Cross compiler
\[ X \neq Y \neq Z \] Canadian Cross compiler

Example

Native i386: built on i386, hosted on i386, produces i386 code.
Sparc cross on i386: built on i386, hosted on i386, produces Sparc code.
Building a Compiler

Bootstrapping

A compiler is just another program

It is improved, bugs are fixed and newer versions are released

To build a new version given a built old version:

1. Stage 1: Build the new compiler using the old compiler
2. Stage 2: Build another new compiler using compiler from stage 1
3. Stage 3: Build another new compiler using compiler from stage 2
   Stage 2 and stage 3 builds must result in identical compilers

⇒ Building cross compilers stops after Stage 1!
A Native Build

GCC Source → Native Compiler → gcc1 Executable
A Native Build

GCC Source → Native Compiler → gcc1 Executable

Compiled using native compiler
A Native Build

Compiled using native compiler

GCC Source → Native Compiler → gcc1 Executable

GCC Source → gcc1 Executable → gcc2 Executable
A Native Build

GCC Source → Native Compiler → gcc1 Executable

Compiled using native compiler

GCC Source → gcc1 Executable → gcc2 Executable

Compiled using gcc
A Native Build

1. GCC Source → Native Compiler → gcc1 Executable
   - Compiled using native compiler

2. GCC Source → gcc1 Executable → gcc2 Executable
   - Compiled using gcc

3. GCC Source → gcc2 Executable → gcc3 Executable
A Native Build

gcc1 Executable
Compiled using native compiler

gcc2 Executable
Compiled using gcc

gcc3 Executable
A Cross Build

Let Host = Build = X and Target = Y

 GCC Source \[\rightarrow\] Native Compiler \((X,X)\) \[\rightarrow\] gcc1 Executable \((X,Y)\)
A Cross Build

Let Host = Build = X and Target = Y

GCC

Source

Native Compiler (X,X)

gcc1

Executable (X,Y)

Executes on X
Generates code for X

Executes on X
Generates code for Y
A Cross Build

Let Host = Build = X and Target = Y

 GCC Source → Native Compiler (X,X) → gcc1 Executable (X,Y) → Executes on X Generates code for X

 GCC Source → gcc1 Executable (X,Y) → gcc2 Executable (Y,Y) → Executes on X Generates code for Y
A Cross Build

Let Host = Build = X and Target = Y

GCC Source → Native Compiler (X,X) → gcc1 Executable (X,Y) → Executes on X
Generates code for X

GCC Source → gcc1 Executable (X,Y) → gcc2 Executable (Y,Y) → Executes on Y
Generates code for Y
A Cross Build

Let Host = Build = X and Target = Y

GCC Source → Native Compiler (X,X) → gcc1 Executable (X,Y) → Executes on X Generates code for X

GCC Source → gcc1 Executable (X,Y) → gcc2 Executable (Y,Y) → Executes on Y Generates code for Y

gcc2 Cannot execute on X
Logical parts are:

- Build configuration files
- Compiler sources
- Emulation libraries
  - libgcc → emulate operations not supported on the target
  - real.c → floating point
- Language Libraries (except C)
- Support software (e.g. garbage collector)
GCC Code Organization Overview

Logical parts are:

- Build configuration files
- Compiler sources
- Emulation libraries
  libgcc → emulate operations not supported on the target
  real.c → floating point
- Language Libraries (except C)
- Support software (e.g. garbage collector)

Our conventions

GCC source directory : $(GCCHOME)
GCC build directory : $(GCCBUILD)
GCC install directory : $(GCCINSTALL)

$(GCCHOME) ≠ $(GCCBUILD) ≠ $(GCCBUILD)
Organization of the Front End Code

- Source language dir: $(GCCHOME)/<lang dir>
- Source language dir contains:
  - Parsing code
    - either handwritten or parser generator input
  - Additional AST/Generic nodes, if any
  - Interface to Generic creation

**Except** for C – which is the “native” language of the compiler

C front end code in: $(GCCHOME)/gcc
GCC Backend Organization

- $(GCCHOME)/gcc/config/<target dir>/
  Directory containing backend code

- **Two** main files: `<target>.h` and `<target>.md`,
  e.g. for an i386 target, we have
  $(GCCHOME)/gcc/config/i386/i386.md and
  $(GCCHOME)/gcc/config/i386/i386.h

- Usually, also `<target>.c` for additional processing code
  (e.g. $(GCCHOME)/gcc/config/i386/i386.c)

- Some additional files
The GCC Build System I

Some Information

- Build-Host-Target systems inferred for native builds
- Specify Target system for cross builds
  Build $\equiv$ Host systems: inferred
- Build-Host-Target systems can be explicitly specified too
- For GCC: A “system” = three entities
  - “cpu”
  - “vendor”
  - “os”

  e.g. sparc-sun-sunos, i386-unknown-linux, i386-gcc-linux
The GCC Build System II

Basic GCC Building How To

- prompt$ cd $GCCBUILD

- prompt$ $GCCHOME configure <options>
  - Specify target: optional for native builds, necessary for others (option --target=<host-cpu-vendor string>)
  - Choose source languages (option --enable-languages=<CSV lang list (c,java))
  - Specify the installation directory (option --prefix=<absolute path of $(GCCBUILD)>)
    ⇒ configure output: customized Makefile

- prompt$ make 2> make.err > make.log

- prompt$ make install 2> install.err > install.log
Adding a New MD

To add a new backend to GCC

- **Define** a new system name, typically a triple.
  
e.g. spim-gnu-linux

- **Edit** $GCCHOME/config.sub to recognize the triple

- **Edit** $GCCHOME/gcc/config.gcc to define
  
  ▶ any backend specific variables
  ▶ any backend specific files
  ▶ $GCCHOME/gcc/config/<cpu> is used as the backend directory for recognized system names.

**Tip**

Read comments in $GCCHOME/config.sub & $GCCHOME/gcc/config/<cpu>.
The GCC Build Process I

GCC builds in two main phases:

- **Adapt** the compiler source for the specified build/host/target systems
  Consider a cross compiler:
  - Find the target MD in the source tree
  - “Include” MD info into the sources
    (details follow)
- **Compile** the adapted sources

**NOTE:**
- Incomplete MD specifications ⇒ Unsuccessful build
- Incorrect MD specification ⇒ Run time failures/crashes
  (either ICE or SIGSEGV)
The GCC Build Process

- make first compiles and runs a series of programs that process the target MD
- Typically, the program source file names are prefixed with gen
- The $GCCHOME/gcc/gen*.c programs
  - read the target MD files, and
  - extract info to create & populate the main GCC data structures

Example

Consider genconstants.c:

- `<target>.md` may define UNSPEC_* constants.
- `genconstants.c` – reads UNSPEC_* constants
- `genconstants.c` – generates corresponding #defines
- Collect then into the insn-constants.h
- `#include "insn-constants.h"` in the main GCC sources
The GCC Build Process

 GCC Sources

<table>
<thead>
<tr>
<th></th>
<th>Target Compiler</th>
<th>Target Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Source Generation</td>
<td>Data Structures</td>
</tr>
</tbody>
</table>

- genconditions
- genconstants
- genflags
- genconfig
- gencodes
- genattr
- genemit
- genextract
- genopinit
- genpeep

GCC Sources:
- libiberty.a
- gensupport.c
- rtl.c
- read-rtl.c
- print-rtl1.c
- errors.c
- bitmap.c
- ggc-none.c

Target Compiler Source Generation:
- insn-conditions.c
- insn-constants.c
- insn-config.h
- insn-codes.h
- insn-attr.h
- insn-emit.c
- insn-extract.c
- insn-opinit.c
- insn-peep.c

Target Compiler Data Structures:
- struct c_test insn_conditions[],
  size_t n_insn_conditions
- GCC_INSN_CONSTANTS_H
  HAVE_(md instructions)
  enum insn_code {
    CODE_FOR_(md inst)= ..
    ...
  }
  HAVE_ATTR_(md_inst_attribs)
- RTX emission functions
  for every insn in MD file
- Extract operands of RTL
  instructions in MD file
- Writes a function that initialises
  an array with the code for
  each insn/expand in MD file
- Extract peephole optimisation
  information in MD files
Building GCC – Summary

- Choose the source language: C
  \texttt{(\textit{--enable-languages=c})}

- Choose installation directory:
  \texttt{(\textit{--prefix=<absolute path>})}

- Choose the target for non native builds:
  \texttt{(\textit{--target=sparc-sunos-sun})}

- Run: \texttt{configure} with above choices

- Run: \texttt{make} to
  \begin{itemize}
    \item \texttt{generate} target specific part of the compiler
    \item \texttt{build} the entire compiler
  \end{itemize}

- Run: \texttt{make install} to install the compiler

**Tip**

Redirect \underline{all} the outputs:

\texttt{$ make > make.log 2> make.err$}