GCC Toulibre 20091216

- Thanks to Toulibre
- http://toulibre.org/
- Laurent GUERBY
- http://guerby.org/

GCC

- GNU Compiler Collection
- Both Native and Cross compiler
- Multi languages
- Multi platform for host and target
- Free Software with FSF owning copyright
- http://gcc.gnu.org/

Multi Languages / in tree

- C (gcc)
- C++ (g++)
- Java (gcj)
- Ada (GNAT)
- Objective-C (gobjc)
- Objective-C++ (gobjc++)
- Fortran (gfortran)

Multi Languages / out of tree

- Modula-2
- Modula-3
- Pascal (gpc)
- PL/I
- D (gdc)
- Mercury
- VHDL (ghdl)

Multi Targets / in tree

- gcc-4.5/gcc/config subdirectories :
- alpha arc arm avr bfin cris crx fr30 frv h8300 i386 ia64 iq2000 lm32 m32c m32r m68hc11 m68k mcore mep mips mmix mn10300 moxie pa pdp11 picochip rs6000 rx s390 score sh soft-fp sparc spu stormy16 v850 vax vms xtensa

Multi Targets / out of tree

- Wikipedia GNU_Compiler_Collection
- Lists 14 targets like PDP-10 or Z8000

Easy to build

- At least for native
- $ tar xfj gcc-4.4.2.tar.bz2
- $ mkdir build && cd build
- $ ../gcc-4.4.2/configure –prefix=/opt/gcc
- $ make bootstrap && make install
- $ make -k check

Fast to build

- Depending on host
- About 15 minutes on 2x4 core x86_64
- About 7 hours on ARM Cortex A8 @ 800 Mhz
- Works with about 512 MB of RAM
- (depending on target)
- Warning: more internal checks enabled in devel version so slower to compile

Bootstrap

- Compiler is written in a programming language and compiles to machine code
- So a compiler written in a given language should be able to compile itself! => ”bootstrap”
- System compiler used to compile ”stage1”
- Which is then used to compile itself as ”stage2”
- Then again for ”stage3” and result should be identical to ”stage2”

Available precompiled

- On most distributions 4.3 or now 4.4
- $ apt-get build-dep gcc
- Also on windows Cygwin or Mingw

Building a cross compiler

- More complicated because GCC, GNU libc and Linux kernel are linked and depend on each other for headers and constants
- Plus you might need out of tree patches
- Better to rely on precompiled toolchain like gcc-avr
- Or tools like Crosstool-ng

Project History

- 1985 Richard Stallman starts GCC as GNU C Compiler
- 1991 GCC 1.x => GCC 2.x
- 1997 EGCS fork
- 1999 EGCS becomes official GCC 2.95 with GCC Steering Committee
- 2001 GCC 3.x
- 2005 GCC 4.x then from 6 month to one year release

Contributors

- YEAR == CHANGELOG == EMAIL
- 1998 == 1685 == 107
- 1999 == 2629 == 196
- 2000 == 5588 == 326
- 2001 == 6356 == 355
- 2002 == 7259 == 371
- 2003 == 10440 == 443
- 2004 == 12857 == 440

Contributors 2

- 2005 == 11657 == 407
- 2006 == 9221 == 356
- 2007 == 8611 == 366
- 2008 == 7789 == 357
- 2009 == 9028 == 341
- Based on quick ChangeLog parsing

Paid to work on GCC

- Red Hat, Novell, ...
- AdaCore, CodeSourcery, ...
- IBM, AMD, Intel, ST Microelectronics, ...
- Google
- … 15 on C++ support
- … 15 on GCC infrastructure
- Academia like INRIA Saclay

Fairly Big Project

- gcc-4.5-20091210/gcc excluding testsuite
- SLOCCOUNT
- Totals grouped by language (dominant language first):
- ansic: 1067922 (67.18%)
- ada: 485301 (30.53%)
- asm: 30379 (1.91%)

More Statistics

- 31006 == 108 == asm
- 170569 == 62 == texi
- 189382 == 999 == ads
- 230459 == 714 == h
- 300237 == 229 == md
- 707707 == 871 == adb
- 1365750 == 772 == c
- 3033097 total identified

GCC Options

- -O0 => no optimization, different from other compilers
- -O1 => optimize but minimize compile time
- -O2 => more costly optimizations
- -O3 => really costly optimizations like auto inlining and vectorizer
- -Os => optimize for code size

GCC Options 2

- $ gcc -Q --help=optimizers -O1
- The following options control optimizations:

-
  ```
    -falign-jumps                     [disabled]
  ```

-
  ```
    -falign-labels                    [disabled]
  ```

-
  ```
    -falign-loops                     [enabled]
  ```
- …
- ~140 for 4.3, ~160 for 4.4, ~180 for 4.5

GCC Options 3

- $ gcc -Q --target-help
- Target specific options:
-   -m128bit-long-double sizeof(long double) is 16
-   -m32 Generate 32bit i386 code
-   -m3dnow Support 3DNow! built-in functions
-   -m64   Generate 64bit x86-64 code
- ....

GCC Options 4

- -march=... => selects an instruction set
- Resulting executable might not work
- -mtune=... => within the instruction set, optimize for a specific target (atom,core2,opteron, …)
- =native => autodetect current processor

How to find interesting options

- http://www.spec.org
- Look at SPECint and SPECfp detailed results
- Option used for ”peak” mode is documented
- Or use automatic tool like Acovea
- http://www.coyotegulch.com/products/acovea/

Builtins

- GCC specific, but some other compilers provide them. Interesting because target independant.
- Example:

int __builtin_ffs (unsigned int x)

Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

- Target dependant inline assembly:
- asm ("fsinx %1,%0" : "=f" (result) : "f" (angle));

How does GCC work?

- $ ls -l /usr/bin/gcc-4.1
- -rwxr-xr-x 1 root root **205952** 2006-12-11 00:12 /usr/bin/gcc
- => a bit small
- The ”gcc” binary is just a ”driver”
- It will invoke various programs depending on command line

The real compilers

- 5.3M /usr/lib/gcc/x86_64-linux-gnu/4.1.2/cc1
- 5.8M/usr/lib/gcc/x86_64-linux-gnu/4.1.2/cc1plus
- 7.7M /usr/lib/gcc/x86_64-linux-gnu/4.1.2/gnat1
- 5.2M /usr/lib/gcc/x86_64-linux-gnu/4.1.2/jc1
- So gcc => xx1 => as or ld
- See: gcc -v myfile.c

Binutils

- GCC is just a text to text file converter
- Binutils does the work to get to machine code and executable
- $ gcc -S file.c
- Will generate file.s => assembly

GCC Organization

- Front-end : one per programming language
- Lexer, parser, semantic analysis for the programming language
- Translates to GENERIC intermediate representation
- Which is then lowered to GIMPLE

Middle and Back End

- Middle End : manipulate GIMPLE
- For target independant and then target dependant transformations and optimizations
- Back End : generate target assembly using target machine description
- RTL Register Transfer Language
- Hundreds of ”passes” when optimizing

Machine Description

- Pattern matching engine
- Describes what a machine instruction does
- Text file with fragment of C code, eg: gcc/config/i386/i386.md
- Quality of the description will impact quality of the generated code for the target
- Done once per target and you get all languages

Moxie

- http://moxielogic.org/blog/
- Virtual ISA design for simplicity
- Contributed to the whole GNU toolchain
- Binutils, GCC, qemu, Linux so you can run it!
- gcc/config/moxie/*.md is 560 lines
- gcc/config/moxie/* is less than 2000 lines

Intermediate Forms

```
int f(int x, int y) {
  int tmp;
  tmp=x;
  if (x>0) tmp=-x;
  return tmp+y;
}
$ gcc -S -O3 -fdump-tree-all  t.c
```

Intermediate Forms 2

- Generates more than 100 files!
- t.c.003t.original

```
f (int x, int y)
{
  int D.2705;
  int tmp;
  tmp = x;
  if (x > 0) goto <D.2703>; else goto <D.2704>;
  <D.2703>:
  tmp = -x;
  <D.2704>:
  D.2705 = tmp + y;
  return D.2705;
}
```

Intermediate Form 3

- t.c.024t.ssa

```
f (int x, int y) {
  int tmp;
  int D.2705;
<bb 2>:  tmp_3 = x_2(D);
  if (x_2(D) > 0)  goto <bb 3>;
  else    goto <bb 4>;
<bb 3>:
  tmp_4 = -x_2(D);
<bb 4>:
  # tmp_1 = PHI <tmp_3(2), tmp_4(3)>
  D.2705_6 = tmp_1 + y_5(D);
  return D.2705_6;
}
```

GCC and GDB

- -g enables generation of debugging information
- More stuff in the assembly, unused for execution. DWARF (and ELF :)
- GDB reads this stuff to map line, memory and registers at all points of execution
- GCC can generate debug information even when optimizing, but result might be difficult to follow

GCC and GDB bugs

- GCC can generate wrong debug information
- GDB can misread debug information
- User Interface can miscommunicate with GDB
- So in practice help from both projects is needed to fix bugs
- Hopefully very good cooperation

GCC bugs

- GCC is always released with hundreds of known bugs (as are other compilers)
- Worst bug kinds (bugzilla keyword)
- wrong-code
- accepts-invalid
- Easier to find when Internal Compiler Error
- ice-on-valid
- ice-on-invalid

GCC bugs 2

- wrong-debug
- missed-optimization
- rejects-valid
- memory-hog
- compile-time-hog
- assemble-failure
- build

How to report GCC bugs

- http://gcc.gnu.org/bugs
- Is a very good ressources including common ”non bugs”
- gcc -v -save-temps -O1 myfile.c
- Generates myfile.i
- Usually enough to attach it together with platform name and compiler output
- Version working and not working helpful

GCC 4.5 New Features

- http://gcc.gnu.org/gcc-4.5/changes.html
- A new link-time optimizer has been added (-flto). When this flag is used, GCC generates a bytecode representation of each input file and writes it to special ELF sections in each object file. When the object files are linked together, all the function bodies are read from these ELF sections and instantiated as if they had been part of the same translation unit. This enables interprocedural optimizations to work across different files (and even different languages), potentially improving the performance of the generated code. To use the link-timer optimizer, -flto needs to be specified at compile time and during the final link. If the program does not require any symbols to be exported, it is possible to combine -flto and -fwhopr with -fwhole-program to allow the interprocedural optimizers to use more aggressive assumptions.
- Automatic parallelization can be enabled as part of Graphite. In addition to -ftree-parallelize-loops=, specify -floop-parallelize-all to enable the Graphite-based optimization.
- It is now possible to extend the compiler without having to modify its source code. A new option -fplugin=file.so tells GCC to load the shared object file.so and execute it as part of the compiler. The internal documentation describes the details on how plugins can interact with the compiler.
- http://gcc.gnu.org/wiki/GCC_PluginAPI

Asking and Contributing

- http://gcc.gnu.org/ has manuals
- gcc-help@gcc.gnu.org
- IRC user help => irc.freenode.org #gcc
- http://gcc.gnu.org/ml => mailing list archives
- gcc-patches@gcc.gnu.org
- gcc@gcc.gnu.org
- gcc-testresults@gcc.gnu.org
- IRC dev channel => irc.oftc.net #gcc

Other Compilers

- Free: LLVM, Open64
- => Interesting with plugins!
- Non free: ICC (Intel), XLC (IBM), …
- Very hard to compare compilers
- Test on your own code in your own settings
- Beware of bugs and support
- If you do multi platform GCC is hard to beat

GCC Compile Farm

- http://gcc.gnu.org/wiki/CompileFarm
- Goal: provide easy access to various architectures and build/test machines for free software developpers. **Not limited to GCC**.
- FSF France sponsored
- Many institutions, individuals and companies providing machines, hosting and help
- AMD and Genesi donated machines

Conclusion

- GCC is a big and old project
- But it's still alive and kicking!
- Supported by many companies
- Learning it is hard
- Good bug reports are contributions!
- Questions?