

The Design and Implementation of Gnu Compiler Generation Framework

Uday Khedker

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay



January 2010

Outline

- GCC: The Great Compiler Challenge
- Meeting the GCC Challenge: CS 715
- Configuration and Building



Part 1

*$GCC \equiv$ The **G**reat **C**ompiler **C**hallenge*

The Gnu Tool Chain

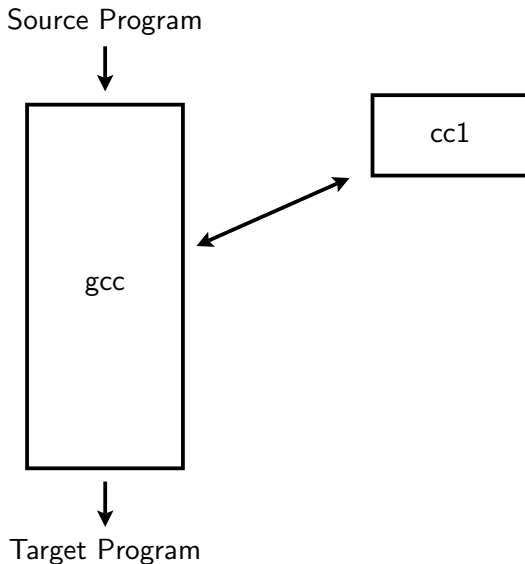
Source Program



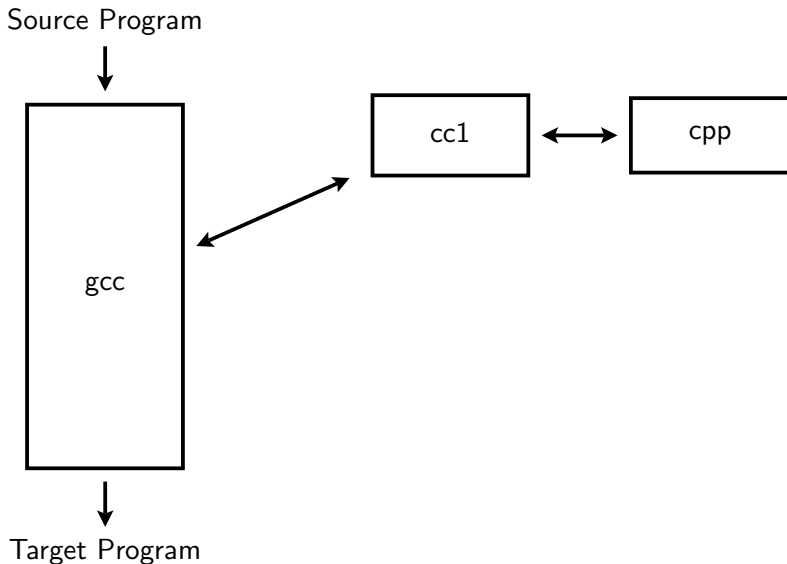
Target Program



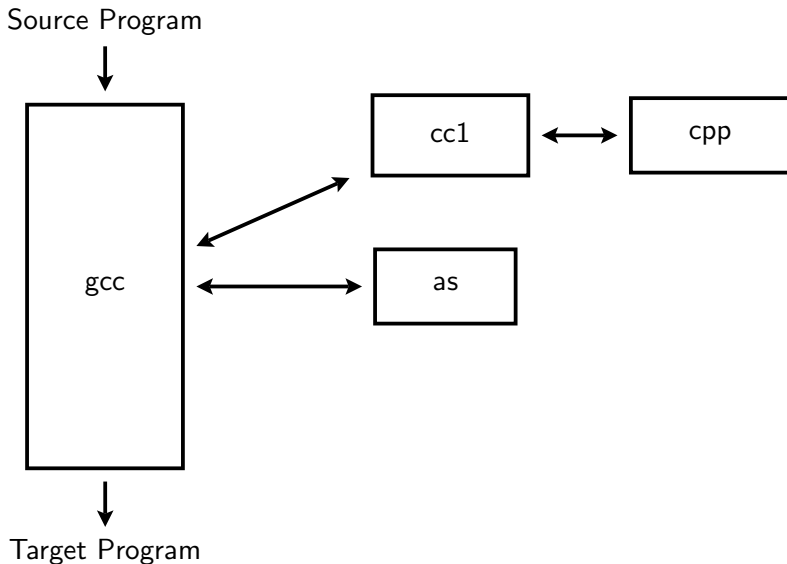
The Gnu Tool Chain



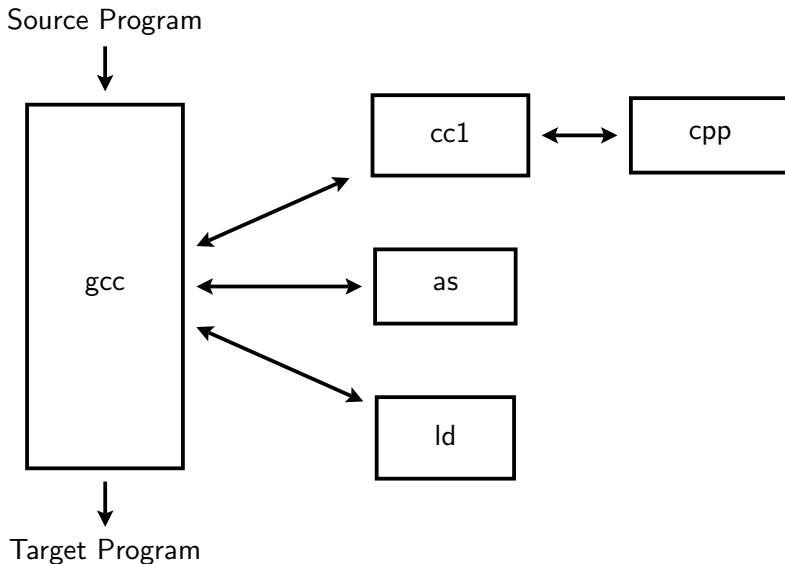
The Gnu Tool Chain



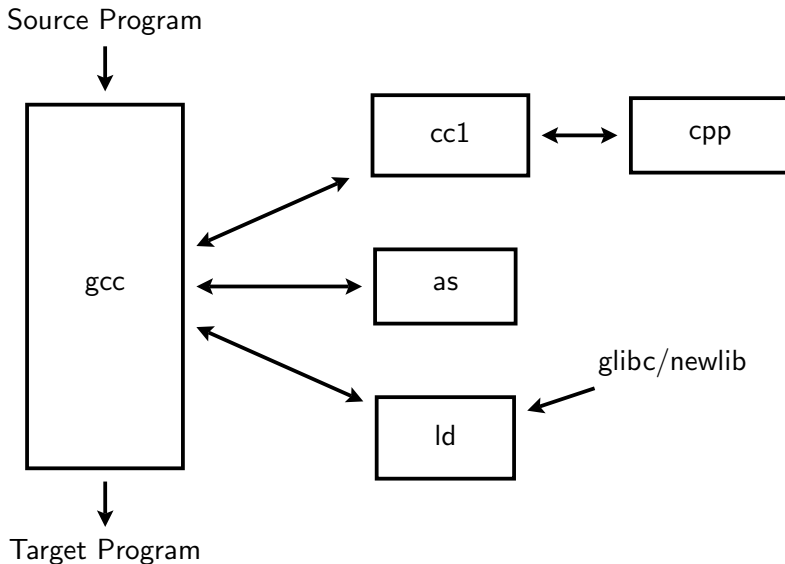
The Gnu Tool Chain



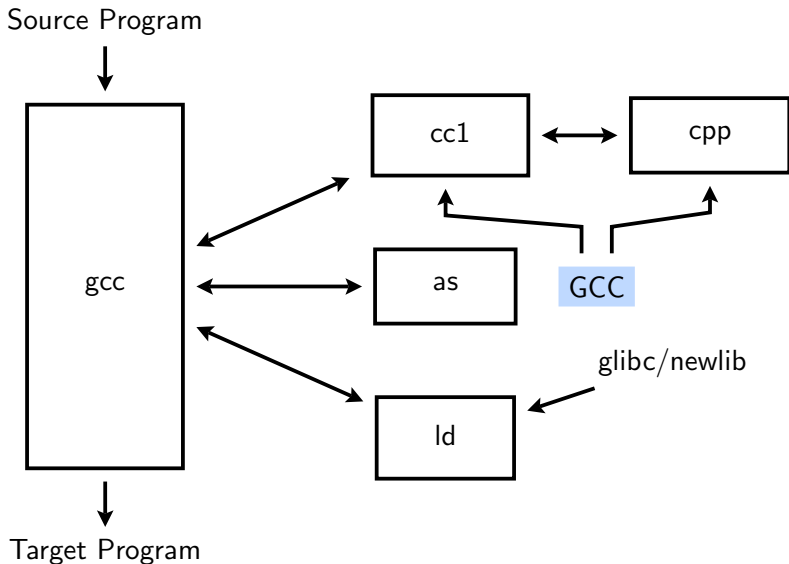
The Gnu Tool Chain



The Gnu Tool Chain



The Gnu Tool Chain



Why is Understanding GCC Difficult?

Some of the obvious reasons:

- **Comprehensiveness**

GCC is a production quality framework in terms of completeness and practical usefulness

- **Open development model**

Could lead to heterogeneity. Design flaws may be difficult to correct

- **Rapid versioning**

GCC maintenance is a race against time. Disruptive corrections are difficult



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM, Atmel AVR,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM, Atmel AVR, Blackfin,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM, Atmel AVR, Blackfin, HC12,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86),
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- Input languages supported:
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
 - ▶ Common processors:
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC,
 - ▶ Lesser-known target processors:
 - ▶ Additional processors independently supported:



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU,
 - ▶ **Lesser-known target processors:**
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries,
 - ▶ **Lesser-known target processors:**

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC,
 - ▶ **Lesser-known target processors:**

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K,

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC,

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS,

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V,

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx,

 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850,

- ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa,
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP, Motorola 6809,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32

- ▶ **Additional processors independently supported:**

D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32

- ▶ **Additional processors independently supported:**

D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32

- ▶ **Additional processors independently supported:**

D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**

C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- **Processors supported in standard releases:**

- ▶ **Common processors:**

Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX

- ▶ **Lesser-known target processors:**

A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32

- ▶ **Additional processors independently supported:**

D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant),



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC,



Comprehensiveness of GCC 4.3.1: Wide Applicability

- **Input languages supported:**
C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- **Processors supported in standard releases:**
 - ▶ **Common processors:**
Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
 - ▶ **Lesser-known target processors:**
A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
 - ▶ **Additional processors independently supported:**
D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC, NEC SX architecture



Comprehensiveness of GCC 4.3.1: Size

| | | |
|--------------|------------------------------------|-----------|
| Source Lines | Number of lines in the main source | 2,029,115 |
| | Number of lines in libraries | 1,546,826 |
| Directories | Number of subdirectories | 3527 |
| Files | Total number of files | 57825 |
| | C source files | 19834 |
| | Header files | 9643 |
| | C++ files | 3638 |
| | Java files | 6289 |
| | Makefiles and Makefile templates | 163 |
| | Configuration scripts | 52 |
| | Machine description files | 186 |

(Line counts estimated by the program `sloccount` by David A. Wheeler)



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control
Design, implement, test, release



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

Design, implement, test, release

- Bazaar: Total Decentralization

Release early, release often, make users partners in software development



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

Design, implement, test, release

- Bazaar: Total Decentralization

Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

Design, implement, test, release

- Bazaar: Total Decentralization

Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”

Code errors, logical errors, and architectural errors



Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

Design, implement, test, release

- Bazaar: Total Decentralization

Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”

Code errors, logical errors, and architectural errors

A combination of the two seems more sensible



The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

- GCC Steering Committee: Free Software Foundation has given charge
 - ▶ Major policy decisions
 - ▶ Handling Administrative and Political issues
- Release Managers:
 - ▶ Coordination of releases
- Maintainers:
 - ▶ Usually area/branch/module specific
 - ▶ Responsible for design and implementation
 - ▶ Take help of reviewers to evaluate submitted changes



Why is Understanding GCC Difficult?

Deeper reason: GCC is not a *compiler* but a *compiler generation framework*

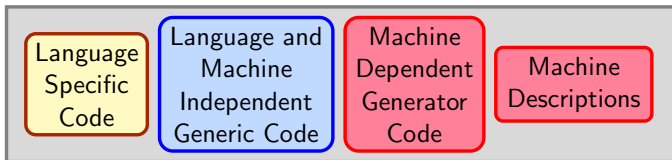
There are two distinct gaps that need to be bridged:

- Input-output of the generation framework: The target specification and the generated compiler
- Input-output of the generated compiler: A source program and the generated assembly program



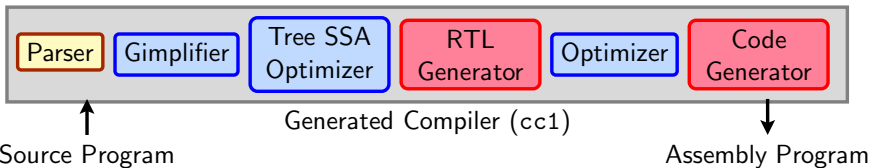
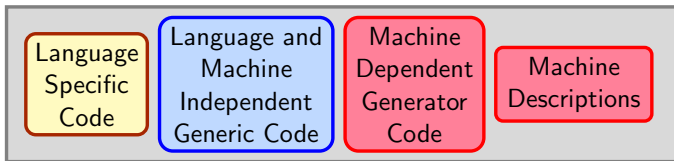
The Architecture of GCC

Compiler Generation Framework

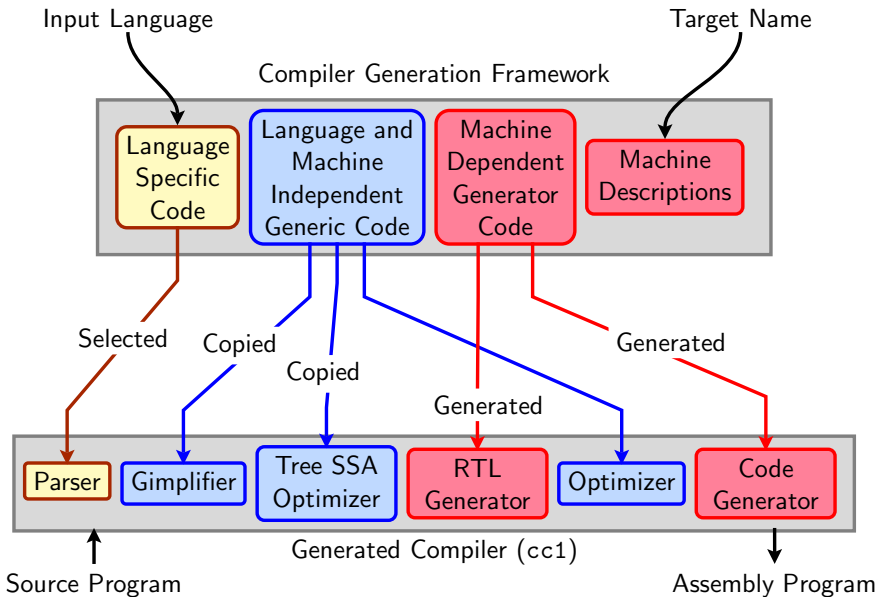


The Architecture of GCC

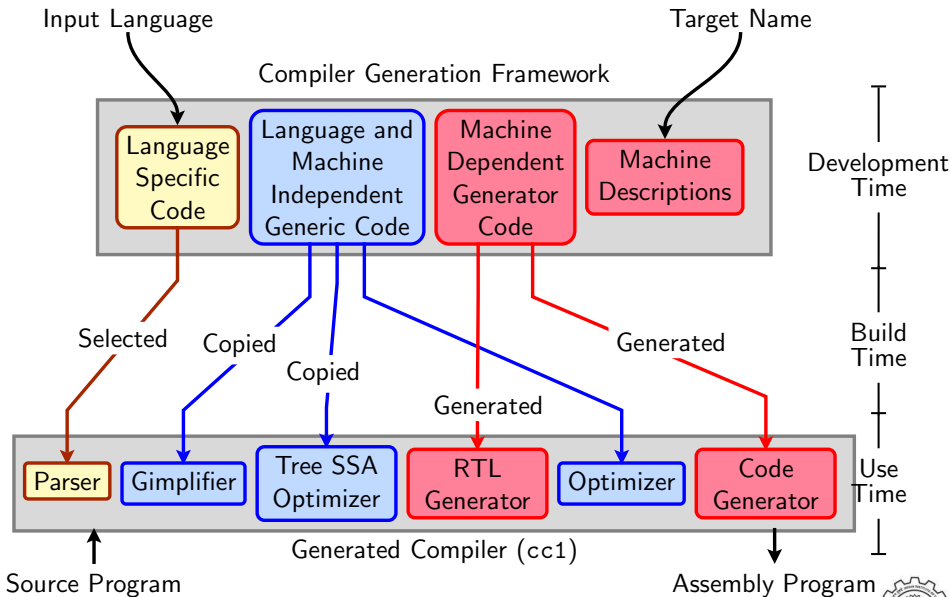
Compiler Generation Framework



The Architecture of GCC



The Architecture of GCC



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
    return flag_tree_loop_distribution != 0;
}
```



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
    return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
    return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
    return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```

- The required C statements are generated during the build



Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.4.2/gcc` using `cscope`



Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.4.2/gcc using cscope

| File | Line | |
|-------------------|------|-----------------------------------------------|
| 0 collect2.c | 766 | main (int argc, char **argv) |
| 1 fix-header.c | 1074 | main (int argc, char **argv) |
| 2 fp-test.c | 85 | main (void) |
| 3 gcc.c | 6216 | main (int argc, char **argv) |
| 4 gcov-dump.c | 76 | main (int argc ATTRIBUTE_UNUSED, char **argv) |
| 5 gcov-iov.c | 29 | main (int argc, char **argv) |
| 6 gcov.c | 355 | main (int argc, char **argv) |
| 7 gen-protos.c | 130 | main (int argc ATTRIBUTE_UNUSED, char **argv) |
| 8 genattr.c | 89 | main (int argc, char **argv) |
| 9 genattrtab.c | 4438 | main (int argc, char **argv) |
| a genautomata.c | 9321 | main (int argc, char **argv) |
| b genchecksum.c | 65 | main (int argc, char ** argv) |
| c gencodes.c | 51 | main (int argc, char **argv) |
| d genconditions.c | 209 | main (int argc, char **argv) |
| e genconfig.c | 261 | main (int argc, char **argv) |
| f genconstants.c | 50 | main (int argc, char **argv) |



Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.4.2/gcc using cscope

```
g genemit.c           820 main (int argc, char **argv)
h genextract.c        394 main (int argc, char **argv)
i genflags.c          231 main (int argc, char **argv)
j gengentrtl.c        350 main (int argc, char **argv)
k gengtype.c          3584 main (int argc, char **argv)
l genmddeps.c         45 main (int argc, char **argv)
m genmodes.c          1376 main (int argc, char **argv)
n genopinit.c         472 main (int argc, char **argv)
o genoutput.c         1005 main (int argc, char **argv)
p genpeep.c           353 main (int argc, char **argv)
q genpreds.c          1399 main (int argc, char **argv)
r genrecog.c          2718 main (int argc, char **argv)
s main.c              33 main (int argc, char **argv)
t mips-tdump.c        1393 main (int argc, char **argv)
u mips-tfile.c        655 main (void )
v mips-tfile.c        4690 main (int argc, char **argv)
w protoize.c          4373 main (int argc, char **const argv)
```



The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism
- Large size of specifications



The GCC Challenge: Poor Retargetability Mechanism

- Symptom of poor retargetability mechanism

Large size of specifications

- Size in terms of line counts

| Files | i386 | mips |
|-------|-------|-------|
| *.md | 35766 | 12930 |
| *.c | 28643 | 12572 |
| *.h | 15694 | 5105 |



Part 2

Meeting the GCC Challenge

Meeting the GCC Challenge

| Goal of Understanding | Methodology | Needs Examining | | |
|-------------------------------------------------|--------------------------------------------------|-----------------|--------|-----|
| | | Makefiles | Source | MD |
| Translation sequence of programs | Gray box probing | No | No | No |
| Build process | Customizing the configuration and building | Yes | No | No |
| Retargetability issues and machine descriptions | Incremental construction of machine descriptions | No | No | Yes |
| IR data structures and access mechanisms | Adding passes to massage IRs | No | Yes | Yes |
| Retargetability mechanism | | Yes | Yes | Yes |



What is Gray Box Probing of GCC?

- **Black Box probing:**
Examining only the input and output relationship of a system
- **White Box probing:**
Examining internals of a system for a given set of inputs
- **Gray Box probing:**
Examining input and output of various components/modules
 - ▶ Overview of translation sequence in GCC
 - ▶ Overview of intermediate representations
 - ▶ Intermediate representations of programs across important phases



Customizing the Configuration and Build Process

- Creating only cc1
- Creating bare metal cross build
Complete tool chain without OS support
- Creating cross build with OS support

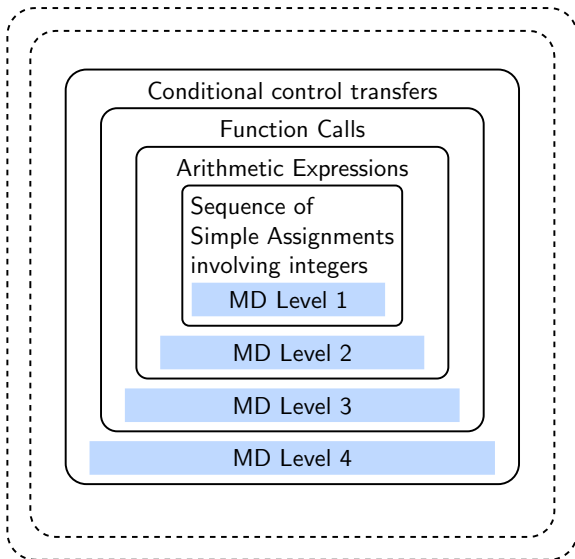


Incremental Construction of Machine Descriptions

- Define different levels of source language
- Identify the minimal set of features in the target required to support each level
- Identify the minimal information required in the machine description to support each level
 - ▶ Successful compilation of any program, and
 - ▶ correct execution of the generated assembly program
- Interesting observations
 - ▶ It is the increment in the source language which results in understandable increments in machine descriptions rather than the increment in the target architecture
 - ▶ If the levels are identified properly, the increments in machine descriptions are monotonic



Incremental Construction of Machine Descriptions

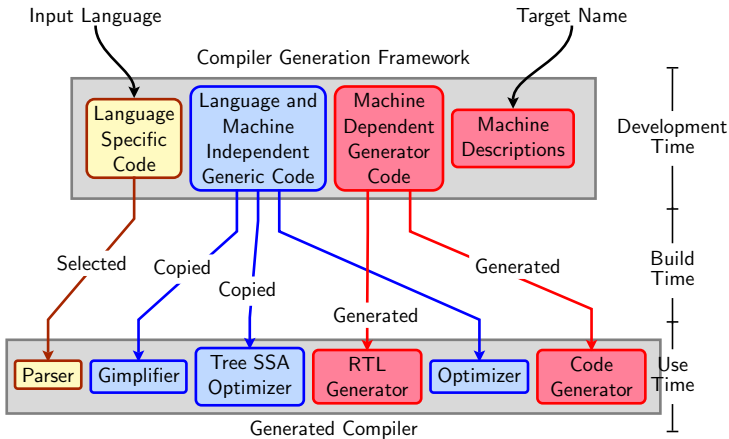


Adding Passes to Massage IRs

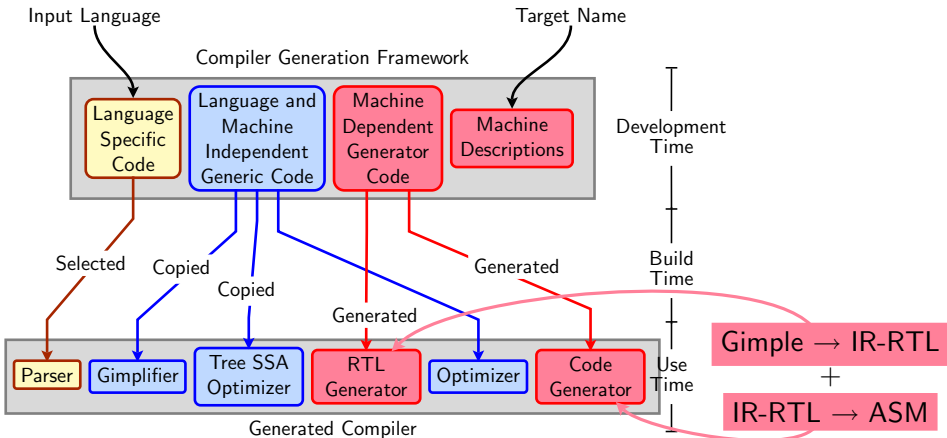
- Understanding the pass structure
- Understanding the mechanisms of traversing a call graph and a control flow graph
- Understanding how to access the data structures of IRs
- Simple exercises such as:
 - ▶ Count the number of copy statements in a program
 - ▶ Count the number of variables declared "const" in the program
 - ▶ Count the number of occurrences of arithmetic operators in the program
 - ▶ Count the number of references to global variables in the program



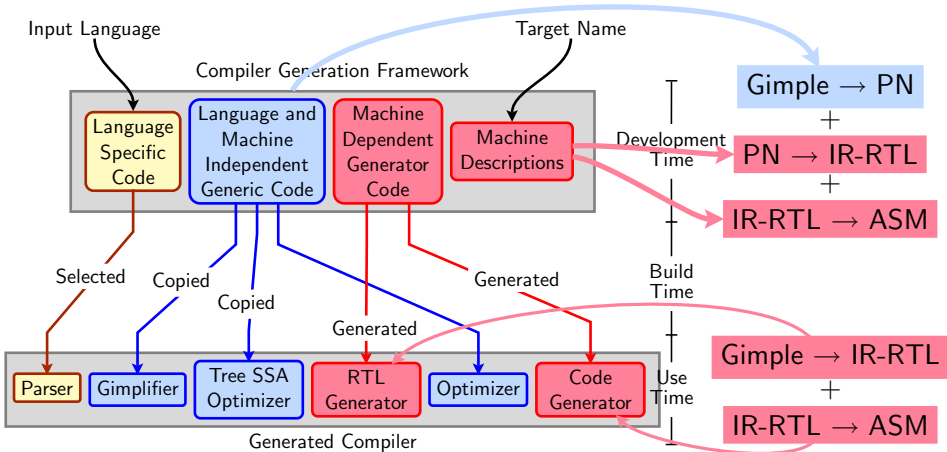
Understanding the Retargetability Mechanism



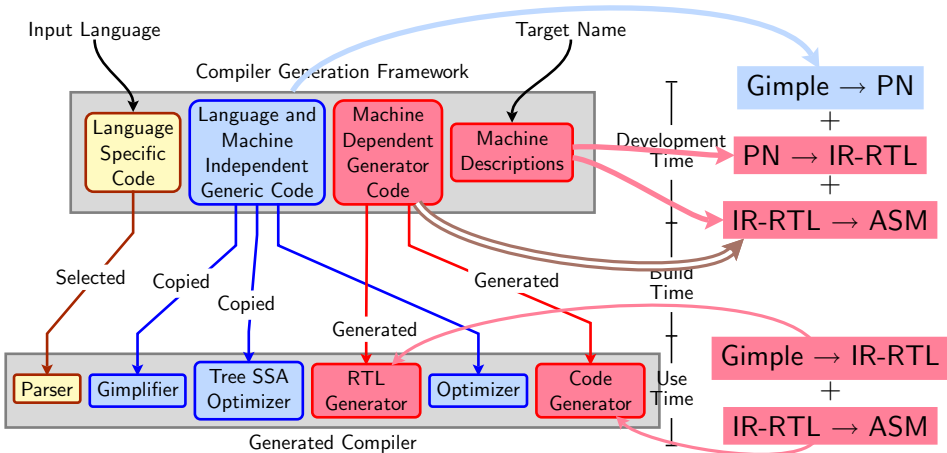
Understanding the Retargetability Mechanism



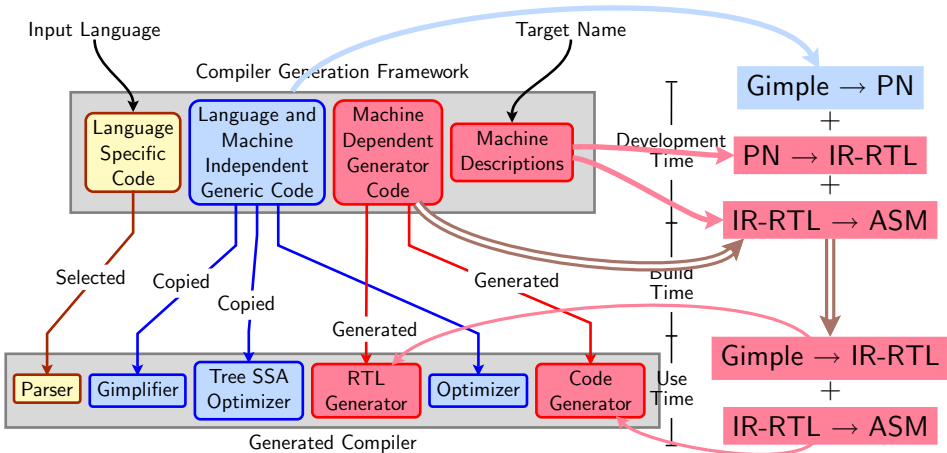
Understanding the Retargetability Mechanism



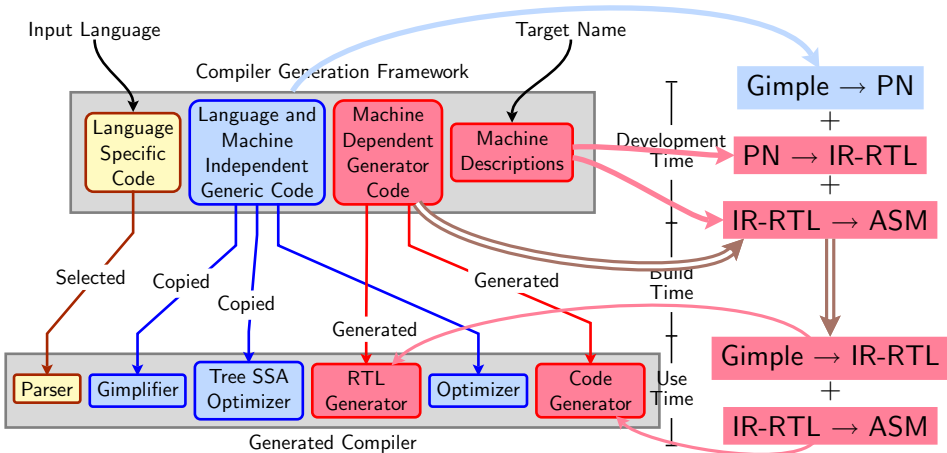
Understanding the Retargetability Mechanism



Understanding the Retargetability Mechanism



Understanding the Retargetability Mechanism



Many more details need to be explained



CS 715 Coverage

| Goal of Understanding | Methodology | Needs Examining | | |
|-------------------------------------------------|--------------------------------------------------|-----------------|--------|-----|
| | | Makefiles | Source | MD |
| Translation sequence of programs | Gray box probing | No | No | No |
| Build process | Customizing the configuration and building | Yes | No | No |
| Retargetability issues and machine descriptions | Incremental construction of machine descriptions | No | No | Yes |
| IR data structures and access mechanisms | Adding passes to massage IRs | No | Yes | Yes |
| Retargetability mechanism | | Yes | Yes | Yes |



CS 715 Pedagogy

- Introductory lecture for each topic followed by lab work
- Many the lecture hours will be used as lab hours
- Sequence of topics
 - ▶ Configuration and building
 - ▶ Examining and manipulating Gimple IR
 - ▶ Examining and manipulating RTL IR
 - ▶ Machine descriptions



CS 715 Lab Work

- Lab exercises:
 - ▶ Pre-defined experiments
 - ▶ Ungraded
- Assignments: Graded lab work
 - ▶ Implementation to modify gcc
 - ▶ Graded
 - ▶ To be submitted in about a couple of weeks or 10 days
- Projects
 - ▶ Specific Study + Implementation
 - ▶ Graded
 - ▶ To be submitted in about a couple of months
 - ▶ Possible topics
 - Extensions of GDFAs, MD rewriting with newer constructs, Detailing the retargetability mechanism, MD parsers



CS 715 Assessment Scheme

- No written examination
- Marks for lab work

| Head | Number | Weightage |
|-------------|--------|--------------------|
| Assignments | 4 | $4 \times 15 = 60$ |
| Project | 1 | 40 |
| Total | | 100 |

- Assignments need not be same for all students
However, they will be comparable and students would have the choice of opting for a particular assignment



Part 3

Configuration and Building

Configuration and Building: Outline

- Code Organization of GCC
- Configuration and Building
- Native build Vs. cross build
- Testing GCC



GCC Code Organization

Logical parts are:

- Build configuration files
- Front end + generic + generator sources
- Back end specifications
- Emulation libraries
(eg. `libgcc` to emulate operations not supported on the target)
- Language Libraries (except C)
- Support software (e.g. garbage collector)



GCC Code Organization

Front End Code

- Source language dir: $\$(SOURCE)/\langle\text{lang dir}\rangle$
- Source language dir contains
 - ▶ Parsing code (Hand written)
 - ▶ 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: $\$(SOURCE)/gcc$

Optimizer Code and Back End Generator Code

- Source language dir: $\$(SOURCE)/gcc$



Back End Specification

- `$(SOURCE)/gcc/config/<target dir>/`
Directory containing back end code
- Two main files: `<target>.h` and `<target>.md`,
e.g. for an i386 target, we have
`$(SOURCE)/gcc/config/i386/i386.md` and
`$(SOURCE)/gcc/config/i386/i386.h`
- Usually, also `<target>.c` for additional processing code
(e.g. `$(SOURCE)/gcc/config/i386/i386.c`)
- Some additional files



Configuration

Preparing the GCC source for local adaptation:

- The platform on which it will be compiled
- The platform on which the generated compiler will execute
- The platform for which the generated compiler will generate code
- The directory in which the source exists
- The directory in which the compiler will be generated
- The directory in which the generated compiler will be installed
- The input languages which will be supported
- The libraries that are required
- etc.



Pre-requisites for Configuring and Building GCC 4.4.2

- ISO C90 Compiler / GCC 2.95 or later
- GNU bash: for running configure etc
- Awk: creating some of the generated source file for GCC
- bzip/gzip/untar etc. For unzipping the downloaded source file
- GNU make version 3.8 (or later)
- GNU Multiple Precision Library (GMP) version 4.2 (or later)
- MPFR Library version 2.3.2 (or later)
(multiple precision floating point with correct rounding)
- MPC Library version 0.8.0 (or later)
- Parma Polyhedra Library (PPL) version 0.10
- CLooG-PPL (Chunky Loop Generator) version 0.15
- jar, or InfoZIP (zip and unzip)
- libelf version 0.8.12 (or later)

(for LTO)



Our Conventions for Directory Names

- GCC source directory : $\$(SOURCE)$
- GCC build directory : $\$(BUILD)$
- GCC install directory : $\$(INSTALL)$
- Important
 - ▶ $\$(SOURCE) \neq \$(BUILD) \neq \$(INSTALL)$
 - ▶ None of the above directories should be contained in any of the above directories

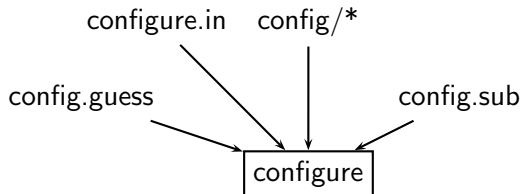


Configuring GCC

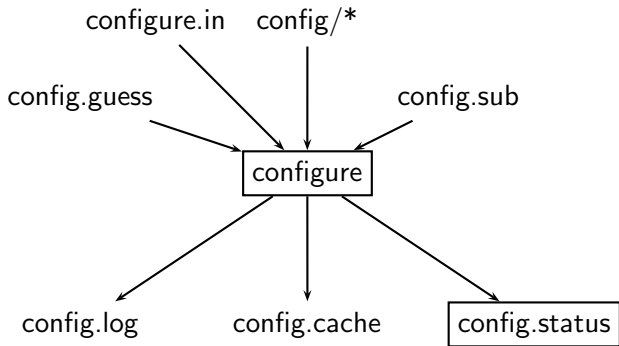
configure



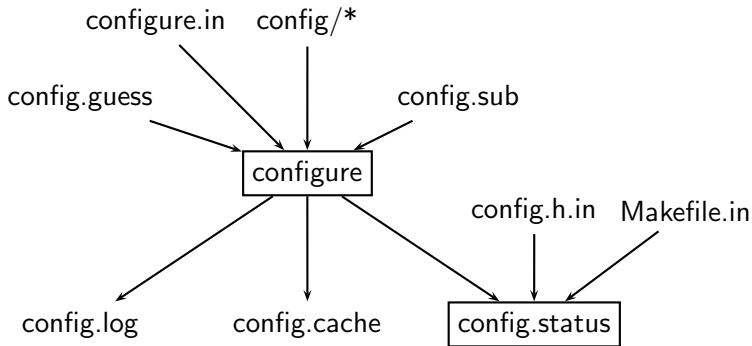
Configuring GCC



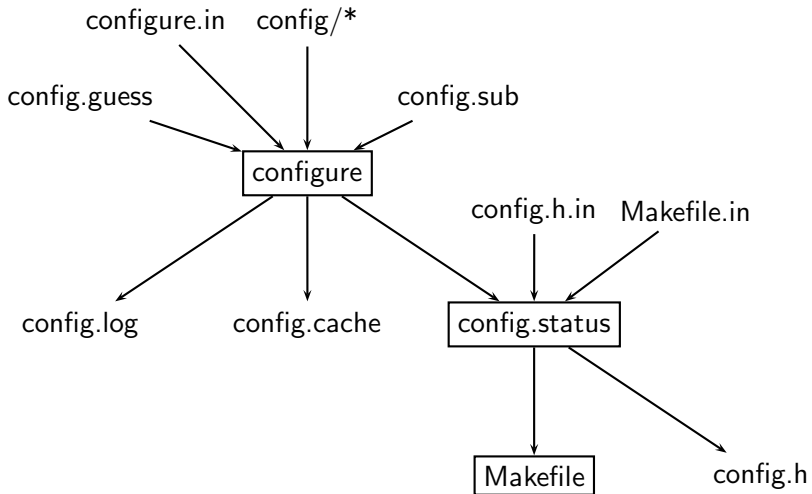
Configuring GCC



Configuring GCC



Configuring GCC



Steps in Configuration and Building

Usual Steps

- Download and untar the source
- `cd $(SOURCE)`
- `./configure`
- `make`
- `make install`



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`
- `make`
- `make install`



Steps in Configuration and Building

| Usual Steps | Steps in GCC |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">• Download and untar the source• <code>cd \$(SOURCE)</code>• <code>./configure</code>• <code>make</code>• <code>make install</code> | <ul style="list-style-type: none">• Download and untar the source• <code>cd \$(BUILD)</code>• <code>\$(SOURCE)/configure</code>• <code>make</code>• <code>make install</code> |

GCC generates a large part of source code during a build!



Building a Compiler: Terminology

- The sources of a compiler are compiled (i.e. built) on *Build system*, denoted **BS**.
- The built compiler runs on the *Host system*, denoted **HS**.
- The compiler compiles code for the *Target system*, denoted **TS**.

The built compiler itself **runs** on **HS** and generates executables that run on **TS**.



Variants of Compiler Builds

| | |
|----------------------|----------------|
| $BS = HS = TS$ | Native Build |
| $BS = HS \neq TS$ | Cross Build |
| $BS \neq HS \neq TS$ | Canadian Cross |

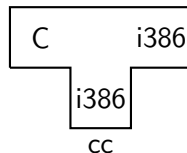
Example

Native i386: built on i386, hosted on i386, produces i386 code.

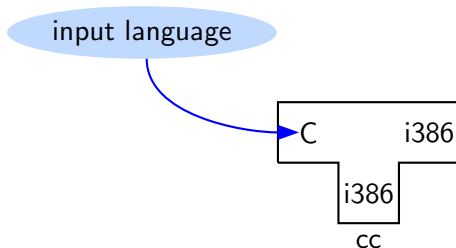
Sparc cross on i386: built on i386, hosted on i386, produces Sparc code.



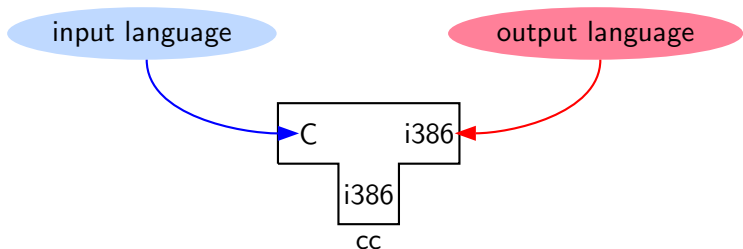
T Notation for a Compiler



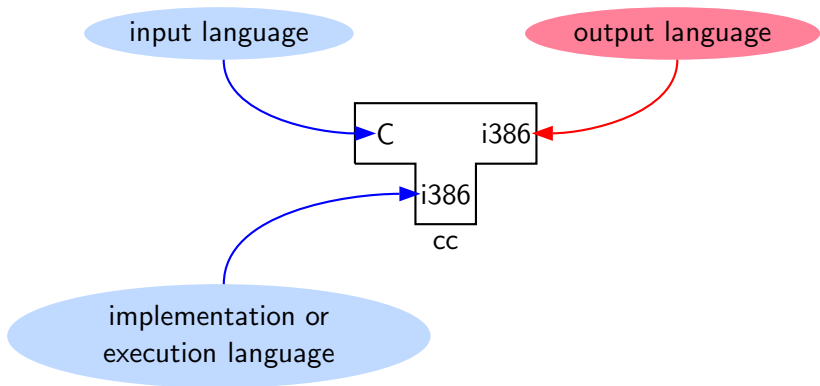
T Notation for a Compiler



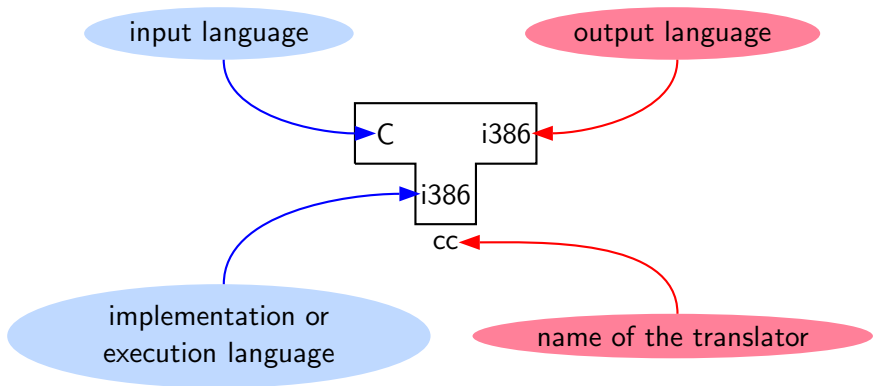
T Notation for a Compiler



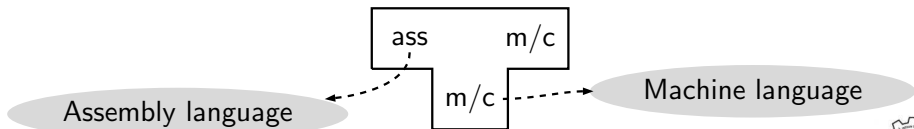
T Notation for a Compiler



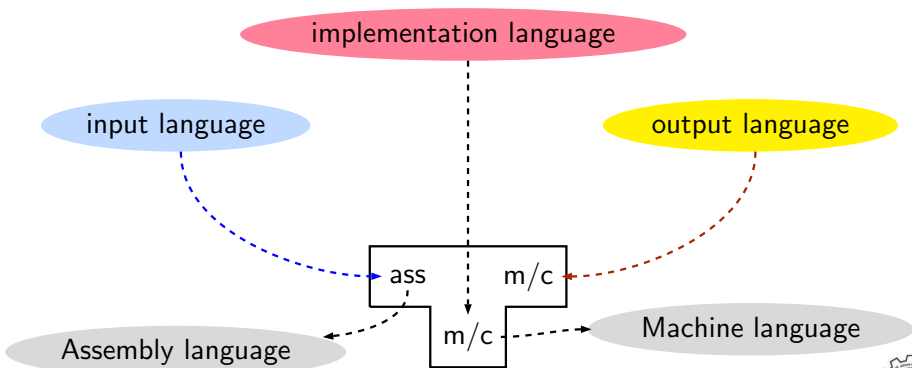
T Notation for a Compiler



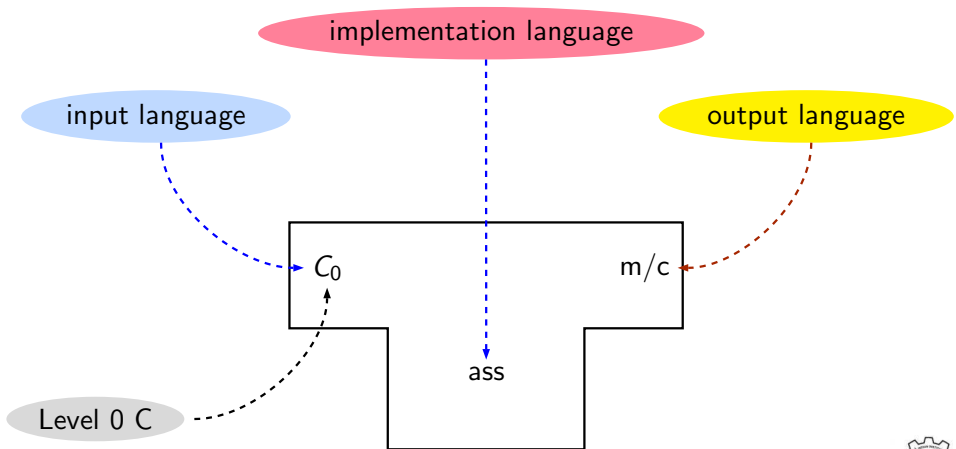
Bootstrapping: The Conventional View



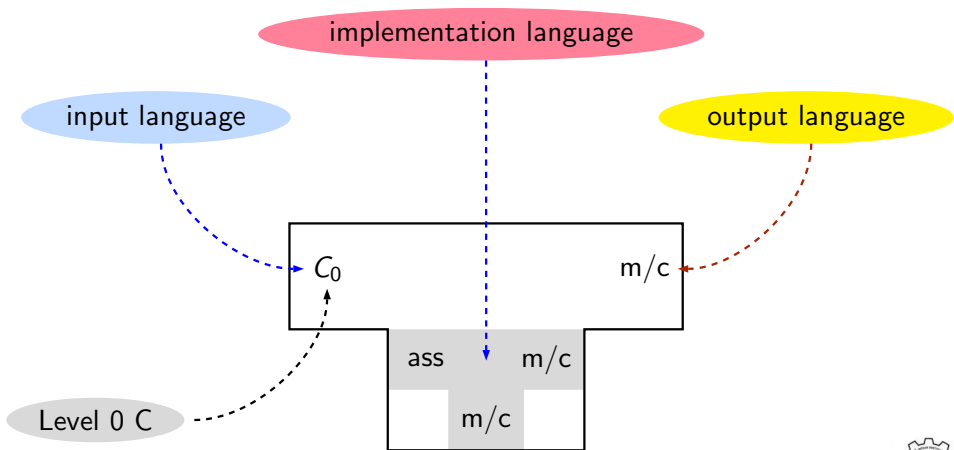
Bootstrapping: The Conventional View



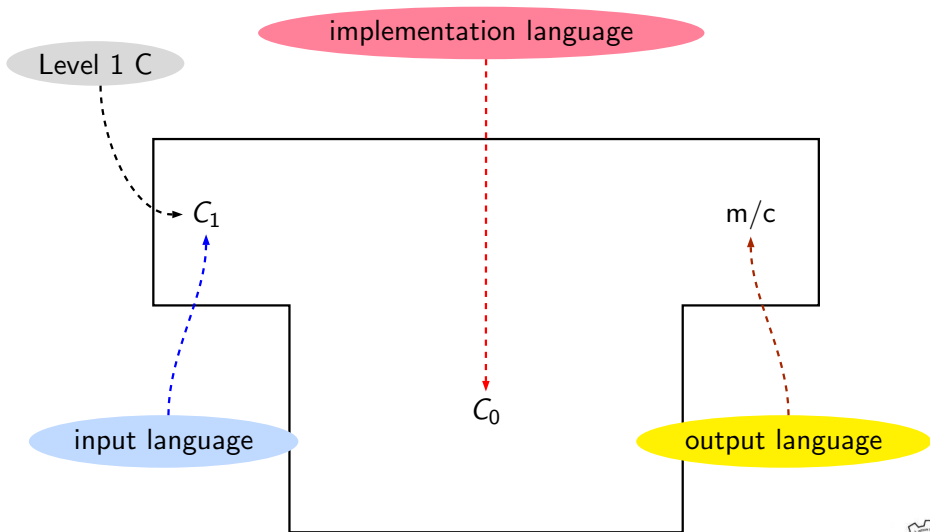
Bootstrapping: The Conventional View



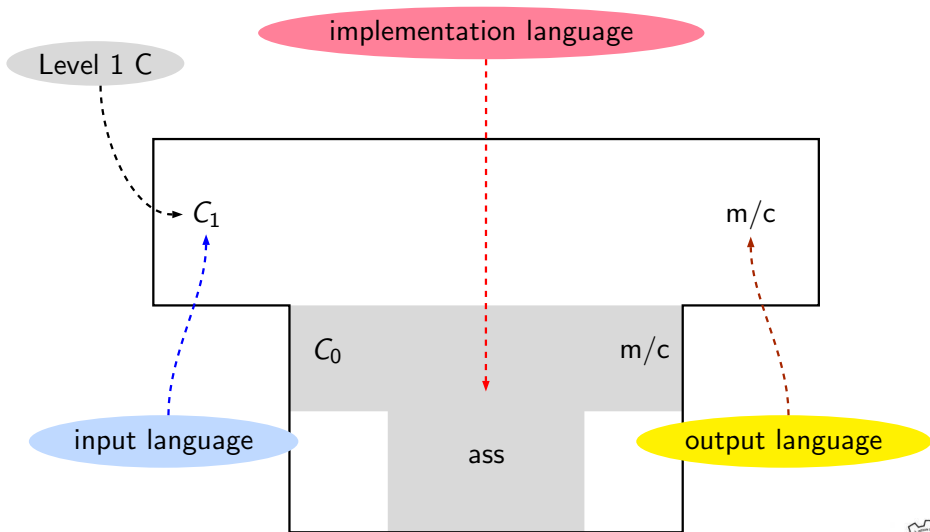
Bootstrapping: The Conventional View



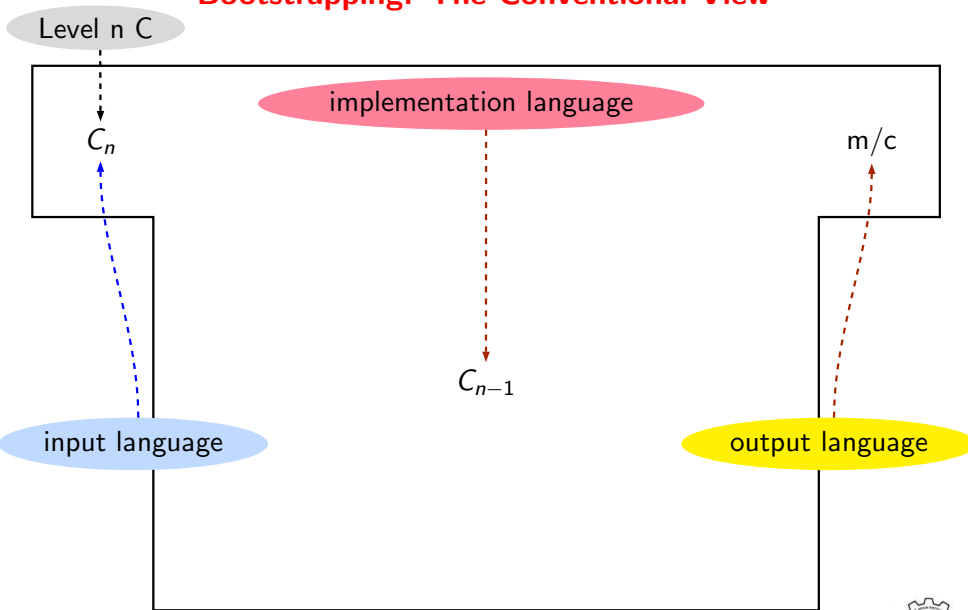
Bootstrapping: The Conventional View



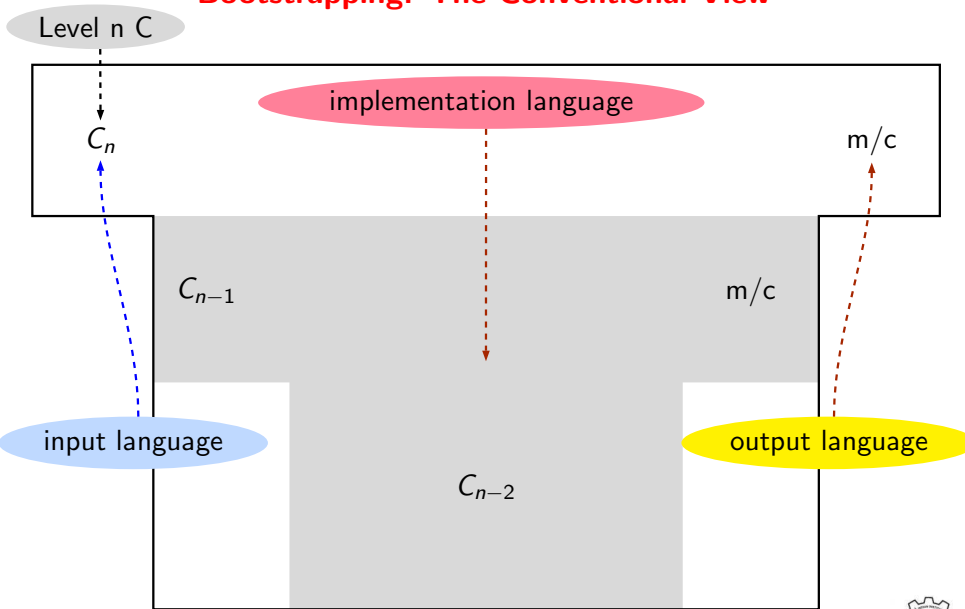
Bootstrapping: The Conventional View



Bootstrapping: The Conventional View



Bootstrapping: The Conventional View



Bootstrapping: GCC View

- Language need not change, but the compiler may change
Compiler is improved, bugs are fixed and newer versions are released
 - To build a new version of a compiler given a **built** old version:
 - ▶ Stage 1: Build the new compiler using the old compiler
 - ▶ Stage 2: Build another new compiler using compiler from stage 1
 - ▶ Stage 3: Build another new compiler using compiler from stage 2Stage 2 and stage 3 builds must result in identical compilers
- ⇒ Building cross compilers **stops** after Stage 1!



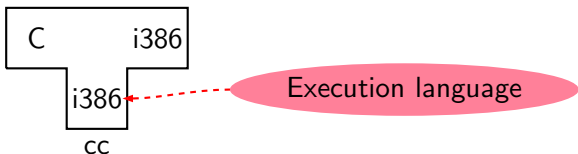
A Native Build on i386

GCC
Source

Requirement: $BS = HS = TS = i386$



A Native Build on i386

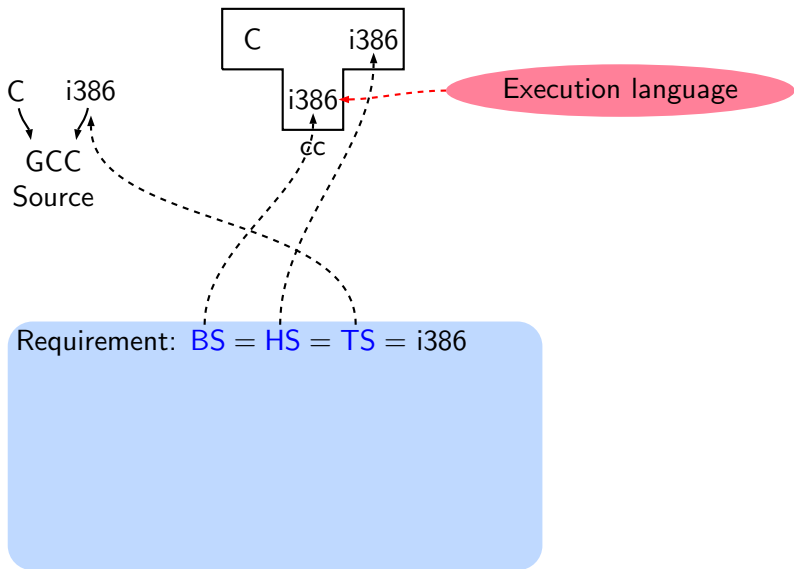


GCC
Source

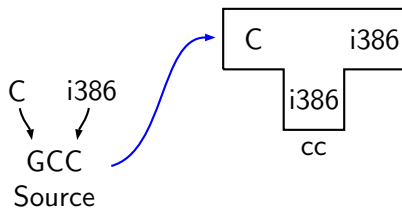
Requirement: $BS = HS = TS = i386$



A Native Build on i386



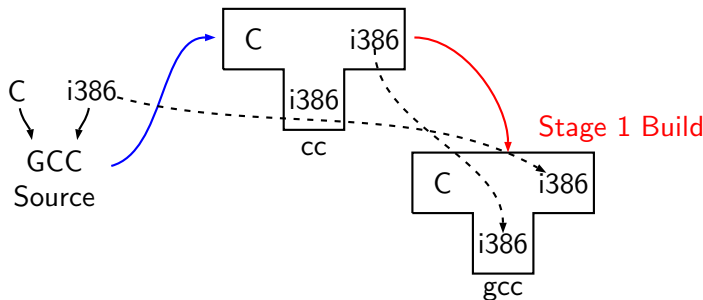
A Native Build on i386



Requirement: $BS = HS = TS = i386$



A Native Build on i386

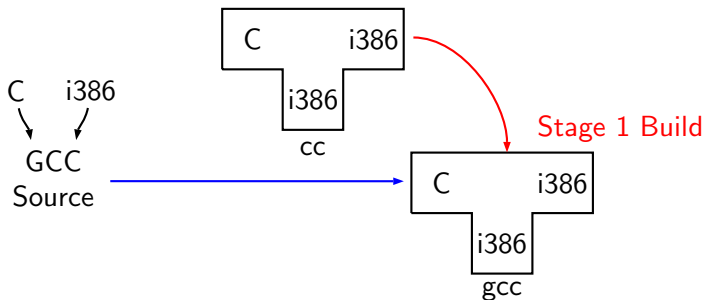


Requirement: $BS = HS = TS = i386$

- Stage 1 build compiled using `cc`



A Native Build on i386

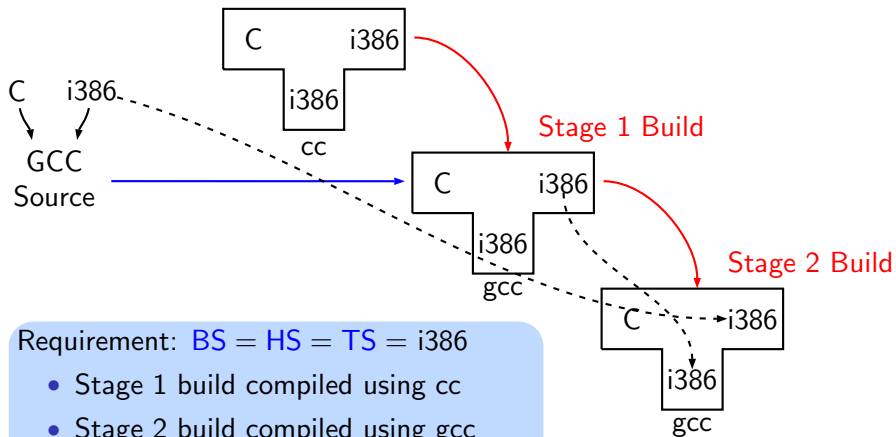


Requirement: $BS = HS = TS = i386$

- Stage 1 build compiled using cc



A Native Build on i386

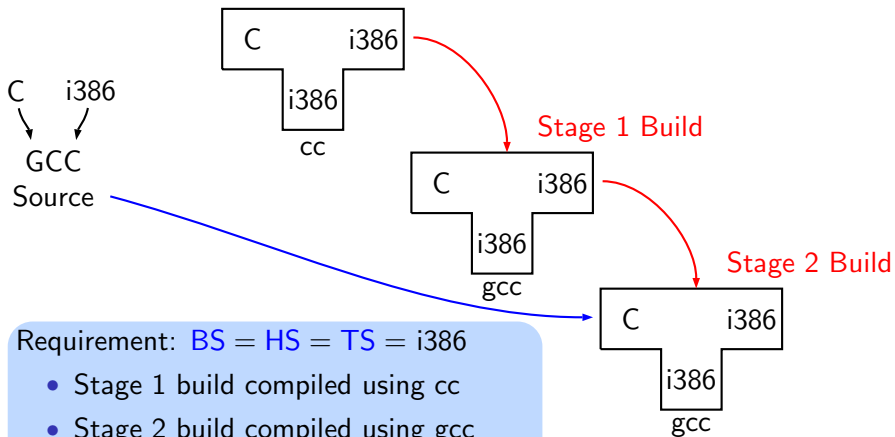


Requirement: $BS = HS = TS = i386$

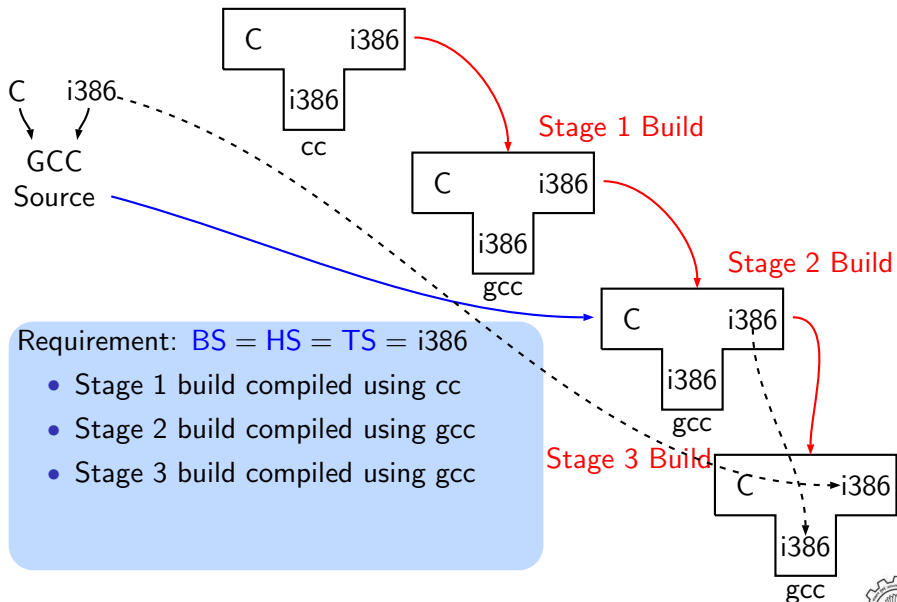
- Stage 1 build compiled using `cc`
- Stage 2 build compiled using `gcc`



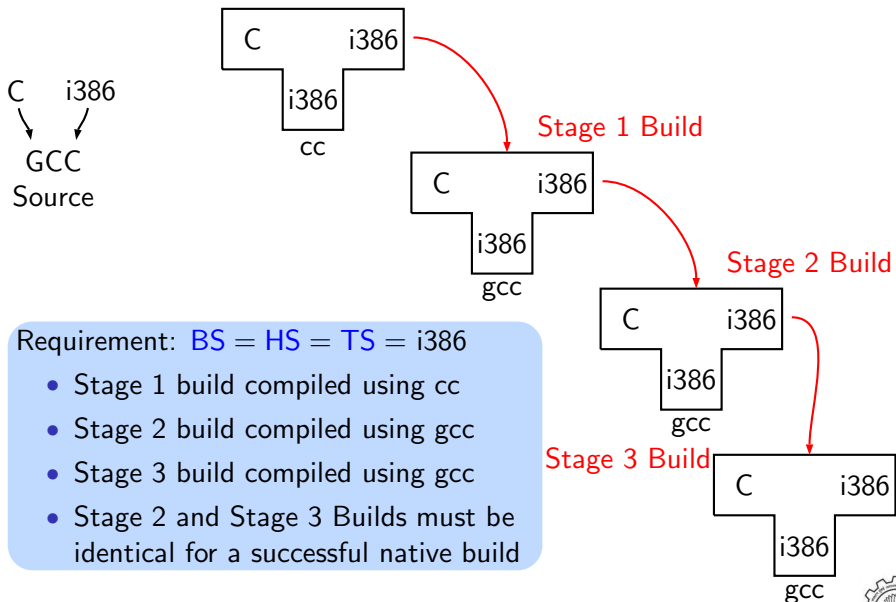
A Native Build on i386



A Native Build on i386



A Native Build on i386



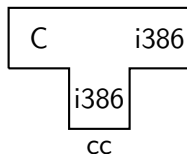
A Cross Build on i386

GCC
Source

Requirement: $BS = HS = i386$, $TS = mips$



A Cross Build on i386

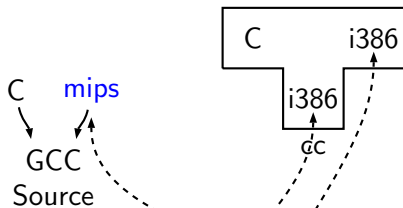


GCC
Source

Requirement: $BS = HS = i386$, $TS = mips$



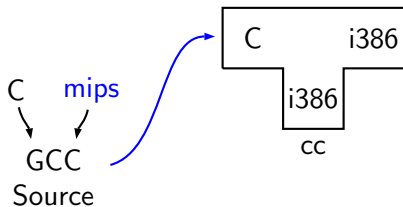
A Cross Build on i386



Requirement: $BS = HS = i386, TS = mips$



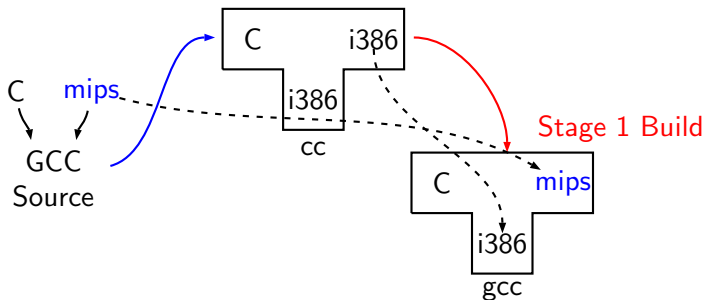
A Cross Build on i386



Requirement: $BS = HS = i386$, $TS = mips$



A Cross Build on i386

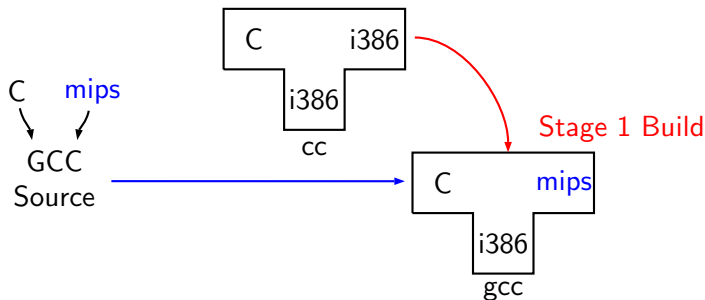


Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build compiled using cc



A Cross Build on i386

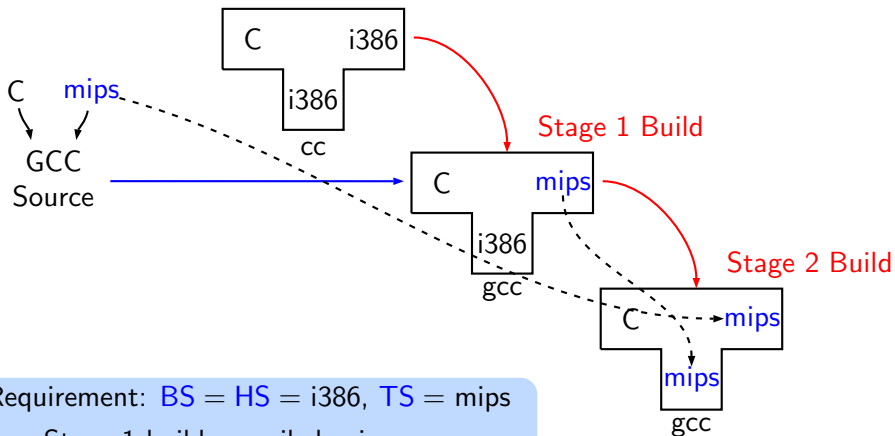


Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build compiled using cc



A Cross Build on i386

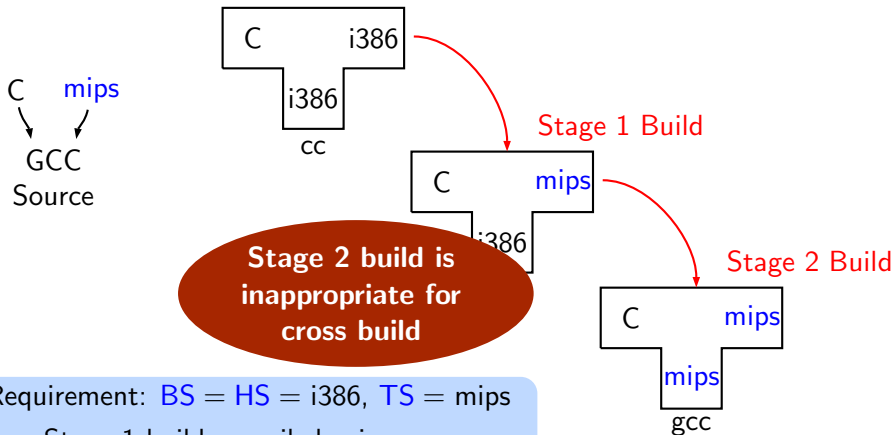


Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
Its $HS = mips$ and not $i386$!



A Cross Build on i386

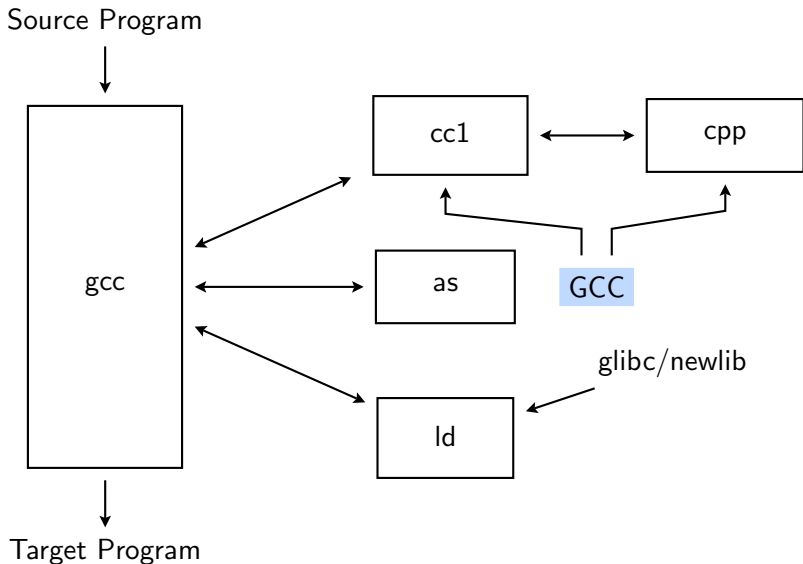


Requirement: $BS = HS = i386$, $TS = mips$

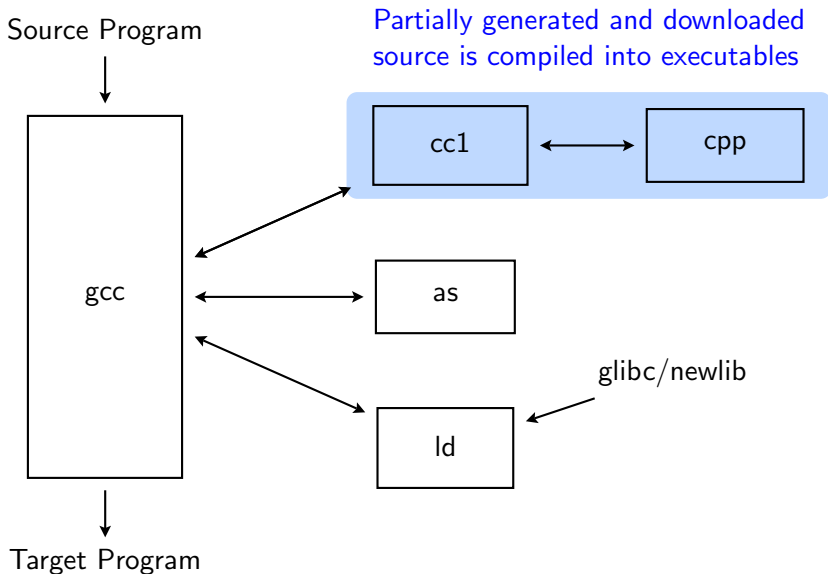
- Stage 1 build compiled using `cc`
- Stage 2 build compiled using `gcc`
Its $HS = mips$ and not `i386`!



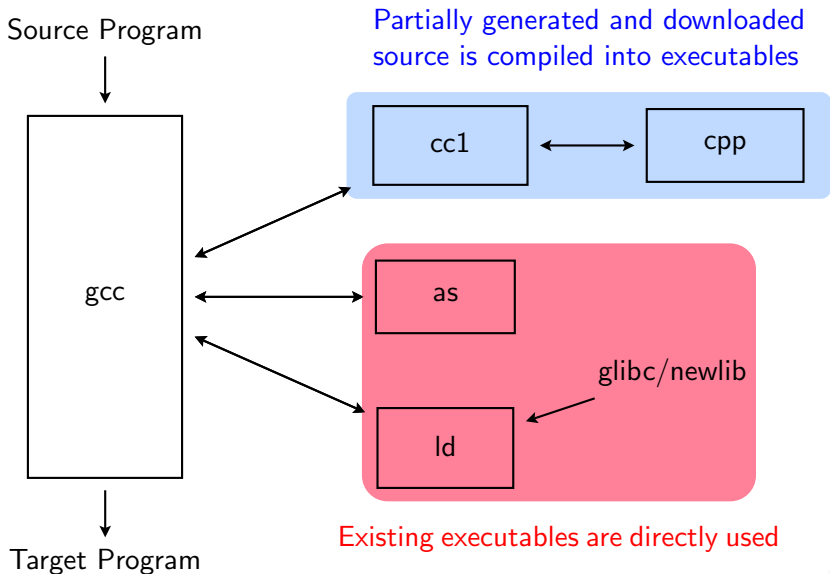
A More Detailed Look at Building



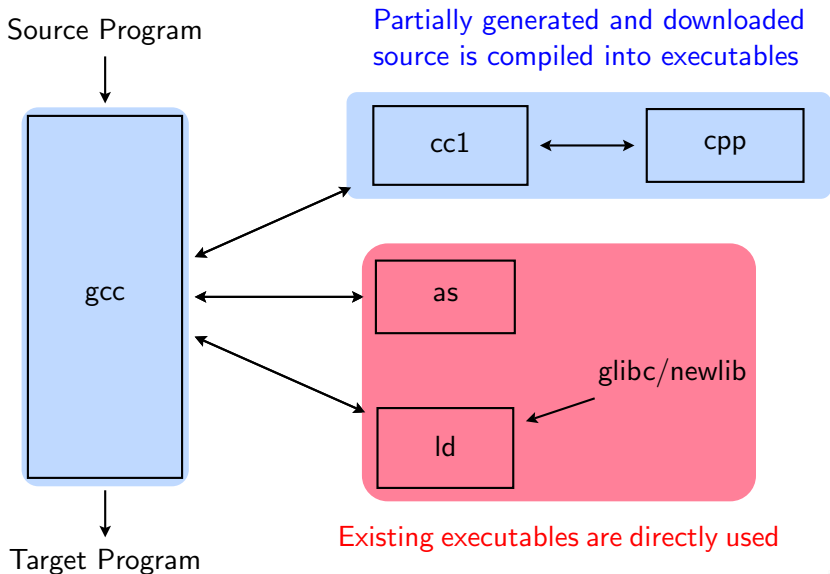
A More Detailed Look at Building



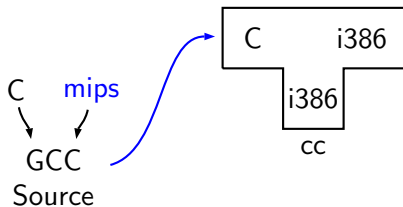
A More Detailed Look at Building



A More Detailed Look at Building



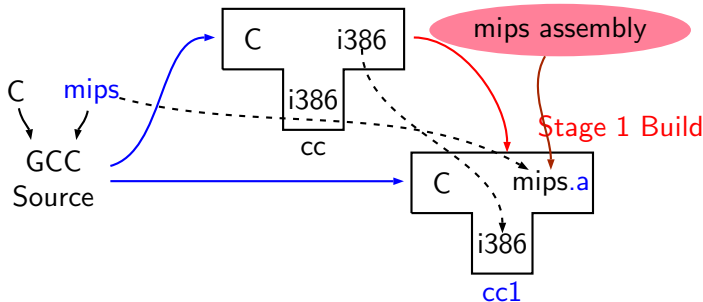
A More Detailed Look at Cross Build



Requirement: $BS = HS = i386$, $TS = mips$



A More Detailed Look at Cross Build

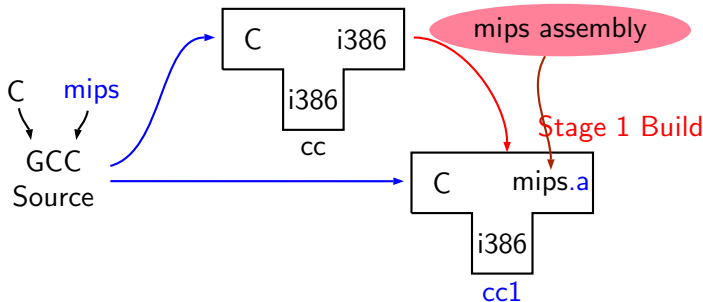


Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build consists of only cc1 and not gcc



A More Detailed Look at Cross Build



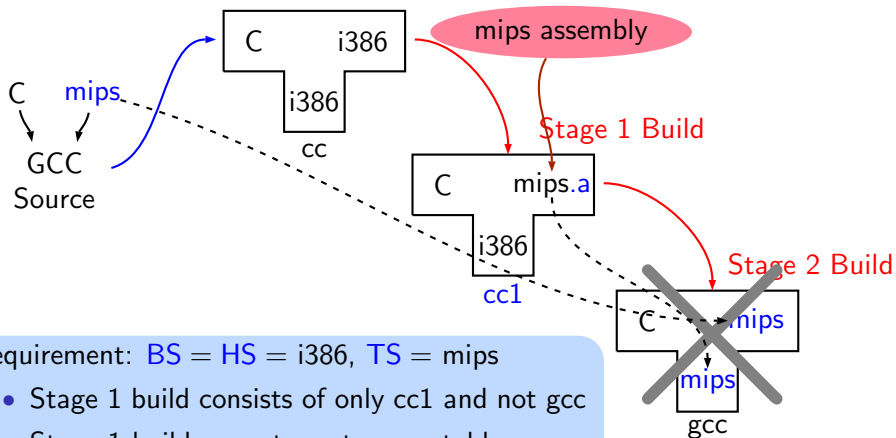
Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build consists of only cc1 and not gcc
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build

binutils
are not available
for mips



A More Detailed Look at Cross Build



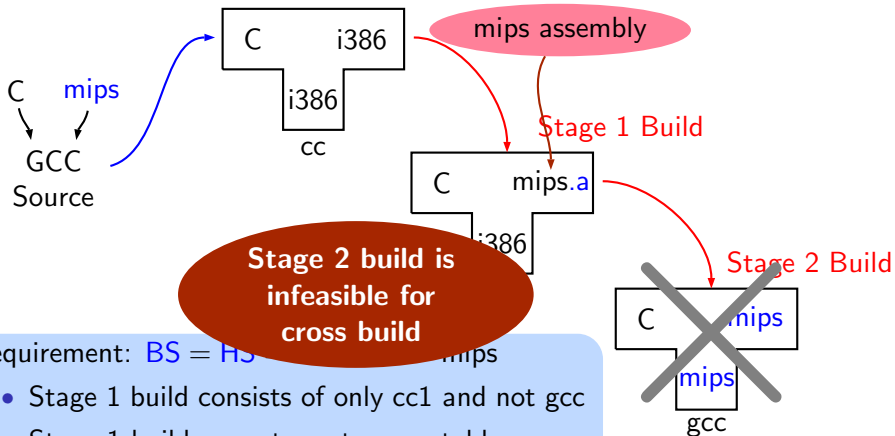
Requirement: $BS = HS = i386$, $TS = mips$

- Stage 1 build consists of only cc1 and not gcc
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build
- Stage 2 build is not possible

binutils
are not available
for mips



A More Detailed Look at Cross Build



Stage 2 build is infeasible for cross build

Requirement: $BS = HS$ and $BS = HS$ for mips

- Stage 1 build consists of only cc1 and not gcc
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build
- Stage 2 build is not possible



Cross Build Revisited

- Option 1: Build binutils in the same source tree as gcc
Copy binutils source in $\$(SOURCE)$, configure and build stage 1
- Option 2:
 - ▶ Compile cross-assembler (as), cross-linker (ld), cross-archiver (ar), and cross-program to build symbol table in archiver (ranlib),
 - ▶ Copy them in $\$(INSTALL)/bin$
 - ▶ Build stage GCC
 - ▶ Install newlib
 - ▶ Reconfigure and build GCCSome options differ in the two builds



Commands for Configuring and Building GCC

This is what we specify

- `cd $(BUILD)`



Commands for Configuring and Building GCC

This is what we specify

- `cd $(BUILD)`
- `$(SOURCE)/configure <options>`
configure output: customized Makefile



Commands for Configuring and Building GCC

This is what we specify

- `cd $(BUILD)`
- `$(SOURCE)/configure <options>`
configure output: customized Makefile
- `make 2> make.err > make.log`



Commands for Configuring and Building GCC

This is what we specify

- `cd $(BUILD)`
- `$(SOURCE)/configure <options>`
configure output: customized Makefile
- `make 2> make.err > make.log`
- `make install 2> install.err > install.log`



Build for a Given Machine

This is what actually happens!

- Generation
 - ▶ Generator sources ($\$(SOURCE)/gcc/gen*.c$) are read and generator executables are created in $\$(BUILD)/gcc/build$
 - ▶ MD files are read by the generator executables and back end source code is generated in $\$(BUILD)/gcc$
- Compilation

Other source files are read from $\$(SOURCE)$ and executables created in corresponding subdirectories of $\$(BUILD)$
- Installation

Created executables and libraries are copied in $\$(INSTALL)$



Build for a Given Machine

This is what actually happens!

- Generation
 - ▶ Generator sources ($\$(SOURCE)/gcc/gen*.c$) are read and generator executables are created in $\$(BUILD)/gcc/build$
 - ▶ MD files are read by the generator executables and back end source code is generated in $\$(BUILD)/gcc$
- Compilation

Other source files are read from $\$(SOURCE)$ and executables created in corresponding subdirectories of $\$(BUILD)$
- Installation

Created executables and libraries are copied in $\$(INSTALL)$

```
gcov-io\n genoutput\n genmddeps\n gencheck\n gengentr\n genchecksum\n genconditions\n genmodes\n genflags\n genattr\n genopinit\n genrecog\n gencondmd.c\n genattrtab\n genautomata\n genpreds\n genemit\n gencondmd\n gengtype\n gencodes\n genconfig\n genpeep\n genconstants\n genextract
```



Build failures due to Machine Descriptions

- Incomplete MD specifications \Rightarrow Unsuccessful build
- Incorrect MD specification \Rightarrow Successful build but run time failures/crashes
(either ICE or SIGSEGV)



Building cc1 Only

- Add a new target in the `Makefile.in`
`cc1:`
`make all-gcc TARGET-gcc=cc1$(exeext)`
- Configure and build with the command `make cc1`.



Common Configuration Options

`--target`

- Necessary for cross build
- Possible host-cpu-vendor strings: Listed in `$(SOURCE)/config.sub`

`--enable-languages`

- Comma separated list of language names
- Default names: `c, c++, fortran, java, objc`
- Additional names possible: `ada, obj-c++, treelang`

`--prefix=$(INSTALL)`

`--program-prefix`

- Prefix string for executable names

`--disable-bootstrap`

- Build stage 1 only



Registering New Machine Descriptions

- Define a new system name, typically a triple.
e.g. spim-gnu-linux
- Edit `$(SOURCE)/config.sub` to recognize the triple
- Edit `$(SOURCE)/gcc/config.gcc` to define
 - ▶ any back end specific variables
 - ▶ any back end specific files
 - ▶ `$(SOURCE)/gcc/config/<cpu>` is used as the back end directory for recognized system names.

Tip

Read comments in `$(SOURCE)/config.sub` &
`$(SOURCE)/gcc/config/<cpu>`.



Testing GCC

- Pre-requisites - Dejagnu, Expect tools
- Option 1: Build GCC and execute the command
make check
or
make check-gcc
- Option 2: Use the configure option `--enable-checking`
- Possible list of checks
 - ▶ Compile time consistency checks
assert, fold, gc, gcac, misc, rtl, rtlflag, runtime, tree, valgrind
 - ▶ Default combination names
 - ▶ yes: assert, gc, misc, rtlflag, runtime, tree
 - ▶ no
 - ▶ release: assert, runtime
 - ▶ all: all except valgrind



GCC Testing framework

- make will invoke runtest command
- Specifying runtest options using RUNTESTFLAGS to customize torture testing
make check RUNTESTFLAGS="compile.exp"
- Inspecting testsuite output: $\$(BUILD)/gcc/testsuite/gcc.log$



Interpreting Test Results

- PASS: the test passed as expected
- XPASS: the test unexpectedly passed
- FAIL: the test unexpectedly failed
- XFAIL: the test failed as expected
- UNSUPPORTED: the test is not supported on this platform
- ERROR: the testsuite detected an error
- WARNING: the testsuite detected a possible problem

[GCC Internals document](#) contains an exhaustive list of options for testing



Configuring and Building GCC – Summary

- Choose the source language: C (`--enable-languages=c`)
- Choose installation directory: (`--prefix=<absolute path>`)
- Choose the target for non native builds:
(`--target=sparc-sunos-sun`)
- Run: `configure` with above choices
- Run: `make` to
 - ▶ generate target specific part of the compiler
 - ▶ build the entire compiler
- Run: `make install` to install the compiler

Tip

Redirect all the outputs:

```
$ make > make.log 2> make.err
```



Exercise 1 and Assignment 1

- Exercises:
 - ▶ i386 native build for GCC-4.4.2
 - ▶ Build only cc1
- Assignment 1
 - ▶ Build a bare metal cross cross compiler for your choice of target
Complete tool chain without OS support
 - ▶ Build a cross compiler with OS support for your choice of target
 - ▶ Deadline: Monday 25 Jan 2009

