Improving the Gnu Compiler Collection

Uday Khedker

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

Jan 2010
Outline

- GCC: The Great Compiler Challenge
- Understanding and Improving the Gnu Compiler Collection
  - Improving machine independent optimizations
  - Understanding machine descriptions
  - Improving machine descriptions and instruction selection
- Activities of GCC Resource Center
- Conclusions
Part 1

GCC ≡ The Great Compiler Challenge
The Gnu Tool Chain

Source Program

gcc

Target Program
The Gnu Tool Chain

Source Program

GCC

Target Program

cc1
The Gnu Tool Chain

Source Program

\[\text{gcc} \]

Target Program

\[\text{cpp}, \text{cc1} \]
The Gnu Tool Chain

Source Program

\[\text{gcc}\]

Target Program

\[\text{cpp} \quad \text{cc1} \quad \text{as}\]
The Gnu Tool Chain

Source Program

gcc

Target Program

cc1 ↔ cpp

as

ld
The Gnu Tool Chain

Source Program

 gcc

 Target Program

 cc1

 as

 ld

 cpp

 glibc/newlib
The Gnu Tool Chain

Source Program

gcc

cc1

cpp

as

GCC

glibc/newlib

ld

Target Program
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:
  - 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,
  - **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,
  - **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:**
  - **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:**
  - **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:**
    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:**

  ▶ **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:**
  - **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:**
  - **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:**
  - **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:
  - 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:**
  - **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:**
  - **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:**

  - **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:**
  
  - **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:**
  - **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:**

  ▶ **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

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

(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]

- **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 \textit{compiler} but a \textit{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

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions
The Architecture of GCC

Compiler Generation Framework

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

Parser | Gimplifier | Tree SSA Optimizer | RTL Generator | Optimizer | Code Generator

Source Program | Generated Compiler (cc1) | Assembly Program

Jan 2010
Uday Khedker, IIT Bombay
The Architecture of GCC

Compiler Generation Framework

- Input Language
- Target Name

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

- Selected
- Copied
- Generated

- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Source Program → Generated Compiler (cc1) → Assembly Program
The Architecture of GCC

Input Language

Compiler Generation Framework

Target Name

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Source Program

Generated Compiler (cc1)

Assembly Program

Jan 2010
An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```c
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

  ```c
  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

<table>
<thead>
<tr>
<th>File</th>
<th>Line</th>
<th>Function Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect2.c</td>
<td>766</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>fix-header.c</td>
<td>1074</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>fp-test.c</td>
<td>85</td>
<td>main (void )</td>
</tr>
<tr>
<td>gcc.c</td>
<td>6216</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gcov-dump.c</td>
<td>76</td>
<td>main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>gcov iov.c</td>
<td>29</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gcov.c</td>
<td>355</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>gen-protos.c</td>
<td>130</td>
<td>main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>genattr.c</td>
<td>89</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genattrtab.c</td>
<td>4438</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genautomata.c</td>
<td>9321</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genchecksum.c</td>
<td>65</td>
<td>main (int argc, char ** argv)</td>
</tr>
<tr>
<td>gencodes.c</td>
<td>51</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconditions.c</td>
<td>209</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconfig.c</td>
<td>261</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>genconstants.c</td>
<td>50</td>
<td>main (int argc, char **argv)</td>
</tr>
</tbody>
</table>
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 gengenrtl.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

<table>
<thead>
<tr>
<th>Files</th>
<th>i386</th>
<th>mips</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.md</td>
<td>35766</td>
<td>12930</td>
</tr>
<tr>
<td>*.c</td>
<td>28643</td>
<td>12572</td>
</tr>
<tr>
<td>*.h</td>
<td>15694</td>
<td>5105</td>
</tr>
</tbody>
</table>
## Meeting the GCC Challenge

<table>
<thead>
<tr>
<th>Goal of Understanding</th>
<th>Methodology</th>
<th>Needs</th>
<th>Examining</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Translation sequence of programs</strong></td>
<td>Gray box probing</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Build process</strong></td>
<td>Customizing the configuration and building</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Retargetability issues and machine descriptions</strong></td>
<td>Incremental construction of machine descriptions</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>IR data structures and access mechanisms</strong></td>
<td>Adding passes to massage IRs</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Retargetability mechanism</strong></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Jan 2010

Uday Khedker, IIT Bombay
Broad Research Goals of GCC Resource Center

• Using GCC as a means
  ▶ Adding new optimizations to GCC
  ▶ Adding flow and context sensitive analyses to GCC
    (In particular, pointer analysis)
  ▶ Translation validation of GCC
  ▶ Linear types in GCC

• Using GCC as an end in itself
  ▶ Changing the retargetability mechanism of GCC
  ▶ Cleaning up the machine descriptions of GCC
  ▶ Systematic construction of machine descriptions
  ▶ Facilitating optimizer generation in GCC
Part 2

Improving Machine Independent Optimizations in GCC
Improving Machine Independent Optimizations in GCC

• The Problems:

• Our Goals:

• Current Status:
Improving Machine Independent Optimizations in GCC

• The Problems:
  ▶ Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  ▶ Context and flow sensitive whole program analysis does not exist

• Our Goals:

• Current Status:
Improving Machine Independent Optimizations in GCC

• The Problems:
  ▶ Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  ▶ Context and flow sensitive whole program analysis does not exist

• Our Goals:
  ▶ Implement scalable context and flow sensitive pointer analysis
  ▶ Facilitate generation of optimizers from specifications
    – Clean specifications
    – Systematic local, global, and interprocedural analysis
    – Simple, efficient, generic, and precise algorithms
    – Incremental analyses for aggressive optimizations

• Current Status:

Jan 2010 Uday Khedker, IIT Bombay
The Problems:

- Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
- Context and flow sensitive whole program analysis does not exist

Our Goals:

- Implement scalable context and flow sensitive pointer analysis
- Facilitate generation of optimizers from specifications
  - Clean specifications
  - Systematic local, global, and interprocedural analysis
  - Simple, efficient, generic, and precise algorithms
  - Incremental analyses for aggressive optimizations

Current Status:

- \textit{gdfa}: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)
- Algorithms and formal theory required further is in place
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives:
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- **Objectives**: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures

- **Main Challenge**:

- **The State of Art**:

- **Our Breakthrough**:

- **The Consequences**:

- **Further Goal**:
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- **Objectives**: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures

- **Main Challenge**: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point

- **The State of Art:**

- **Our Breakthrough:**

- **The Consequences:**

- **Further Goal:**
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

• **Objectives:** Optimizations across procedure boundaries to incorporate
  ▶ the effects of procedure calls in the caller procedures, and
  ▶ the effects of calling contexts in the callee procedures

• **Main Challenge:** Precision requires distinguishing between an impractically large number (>> millions) of contexts at each program point

• **The State of Art:** Merge information across contexts for efficiency
  ⇒ Significant imprecision in recursive programs

• **Our Breakthrough:**

• **The Consequences:**

• **Further Goal:**

Defining Interprocedural Context for Static Analysis

\[
\begin{align*}
\text{Entry} & : a + b \\
\text{Call } p & : n_1 \quad d = a + b \\
\text{Endp} & \\
\text{Exit} & \\
\text{Startp} & \\
\text{Call } q & : a = 1 \\
\text{Endq} & : n_2 \\
\text{Startq} & \\
\text{Call } p & : n_4 \\
\text{Endq} & \\
\end{align*}
\]
Defining Interprocedural Context for Static Analysis

Entry

\[ a + b \]

Startp

\[ C_1 \]

Call p

\[ C_2 \]

Call q

Exit

\[ R_1 \]

\[ R_2 \]

Endp

\[ n_1 \]

\[ d = a + b \]

\[ C_3 \]

Call p

\[ n_2 \]

\[ a = 1 \]

\[ R_3 \]

\[ R_4 \]

\[ R_5 \]

\[ Endq \]

\[ n_3 \]

\[ n_4 \]

\[ n_5 \]
Defining Interprocedural Context for Static Analysis

Entry

\[ a + b \]

\[ C_1 \]

Call p

\[ R_1 \]

Exit

Startp

\[ C_2 \]

Call q

\[ R_2 \]

Startq

\[ n_1 \]

\[ d = a + b \]

\[ C_3 \]

Call p

\[ R_3 \]

Endp

\[ n_3 \]

Exit

\[ C_4 \]

Call p

\[ R_4 \]

Endq

\[ n_4 \]

\[ a = 1 \]

\[ n_2 \]
Defining Interprocedural Context for Static Analysis

```
Entry: a + b
C_1: Call p

R_1: Exit

Startp
C_2: Call q

R_2

Endp

n_1: d = a + b
C_3: Call p

R_3

n_3

C_4: Call p

R_4

Endq

Startq
a = 1
n_2

n_4
```

Jan 2010
Uday Khedker, IIT Bombay
Defining Interprocedural Context for Static Analysis

Entry

\[ a + b \]

Startp

\[ C_1 \]

Call p

\[ C_2 \]

Call q

\[ R_1 \]

Exit

\[ R_2 \]

Endp

\[ n_1 \]

\[ d = a + b \]

\[ n_3 \]

\[ n_4 \]

Endq

Startq

\[ n_2 \]

\[ a = 1 \]

\[ C_3 \]

Call p

\[ C_4 \]

Call p

\[ R_3 \]

\[ R_4 \]
Defining Interprocedural Context for Static Analysis

Entry

\[ a + b \]

Call p

\[ C_1 \]

Startp

Call q

\[ C_2 \]

Exit

\[ R_1 \]

Cap 1

Startp

\[ C_3 \]

Call p

Exit

\[ R_3 \]

Endp

Exit

\[ n_1 \]

\[ d = a + b \]

End q

\[ n_4 \]

\[ n_2 \]

\[ a = 1 \]

\[ n_3 \]

\[ R_4 \]
Defining Interprocedural Context for Static Analysis

Entry: $a + b$

Startp

Call p

C1

R1

Exit

Startq

$d = a + b$

n1

Call p

C3

R3

n3

Endp

C2

Call q

main

R2

n2

Endq

C4

Call p

R4

n4

stack

$p : C_1$
Defining Interprocedural Context for Static Analysis
Defining Interprocedural Context for Static Analysis
Defining Interprocedural Context for Static Analysis

Entry
\[ a + b \]

Startp
Call p

C1
Call p

R1
Exit

R2

Endp

C2
Call q

n1
\[ d = a + b \]

C3
Call p

n3

C4
Call p

n2

Startq

n4

Endq

stack

q : C2

p : C1

main

Jan 2010

Uday Khedker, IIT Bombay
Defining Interprocedural Context for Static Analysis

Diagram showing the flow of execution and calls between procedures.
Defining Interprocedural Context for Static Analysis

Context is defined by stack snapshot ⇒ Unbounded number of contexts

Jan 2010

Uday Khedker, IIT Bombay
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- **Objectives:** Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- **Main Challenge:** Precision requires distinguishing between an impractically large number (>> millions) of contexts at each program point
- **The State of Art:** Merge information across contexts for efficiency
  ⇒ Significant imprecision in recursive programs
- **Our Breakthrough:**

- **The Consequences:**

- **Further Goal:**
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- **Objectives:** Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- **Main Challenge:** Precision requires distinguishing between an impractically large number (≫ millions) of contexts at each program point
- **The State of Art:** Merge information across contexts for efficiency
  \[\Rightarrow\text{Significant imprecision in recursive programs}\]
- **Our Breakthrough:** Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- **The Consequences:**

- **Further Goal:**
Objectives: Optimizations across procedure boundaries to incorporate
- the effects of procedure calls in the caller procedures, and
- the effects of calling contexts in the callee procedures

Main Challenge: Precision requires distinguishing between an impractically large number (>> millions) of contexts at each program point

The State of Art: Merge information across contexts for efficiency
⇒ Significant imprecision in recursive programs

Our Breakthrough: Clean, formally provable characterizations to
- discard redundant contexts at the start of every procedure, and
- simulate regeneration contexts at the end of every procedure

The Consequences: Our implementation in GCC shows that our variant saves time and space by

Further Goal:
**Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]**

- **Objectives:** Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- **Main Challenge:** Precision requires distinguishing between an impractically large number (>> millions) of contexts at each program point
- **The State of Art:** Merge information across contexts for efficiency
  \[ \Rightarrow \] Significant imprecision in recursive programs
- **Our Breakthrough:** Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- **The Consequences:** Our implementation in GCC shows that our variant saves time and space by a factor of over a million!
- **Further Goal:**
Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- **Objectives:** Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- **Main Challenge:** Precision requires distinguishing between an impractically large number (>> millions) of contexts at each program point
- **The State of Art:** Merge information across contexts for efficiency
  ⇒ Significant imprecision in recursive programs
- **Our Breakthrough:** Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- **The Consequences:** Our implementation in GCC shows that our variant saves time and space by a factor of over a million!
- **Further Goal:** Design a scalable variant to analyze a million lines of code in a few minutes
Heap Reference Analysis [TOPLAS 2007]

• The Problem:

• Our Objectives:

• Main Challenge:

• Our Key Idea:

• Current status:

• Further Work:
Heap Reference Analysis [TOPLAS 2007]

• **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

• **Our Objectives:**

• **Main Challenge:**

• **Our Key Idea:**

• **Current status:**

• **Further Work:**
Heap Reference Analysis [TOPLAS 2007]

• **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

• **Our Objectives:** To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

• **Main Challenge:**

• **Our Key Idea:**

• **Current status:**

• **Further Work:**
Heap Reference Analysis [TOPLAS 2007]

• The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

• Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

• Main Challenge: Unlike stack and static data,
  ▶ heap data accessible to any procedure is unbounded. Hence,
  ▶ the mapping between object names and their addresses needs to change at runtime.

• Our Key Idea:

• Current status:

• Further Work:
Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is not executed even once.

```plaintext
1. w = x    // x points to ma
2. while (x.data < max)
3.     x = x.rptr
4. y = x.lptr
5. z = New class of z
6. y = y.lptr
7. z.sum = x.data + y.data
```
Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed once.

1. \( w = x \)  // \( x \) points to \( m_a \)
2. \( \text{while} \ (x.\text{data} < \text{max}) \)
3. \( x = x.\text{rptr} \)
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class}\_\text{of}_z \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed twice.

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while \( (x.data < \text{max}) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class of } z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
Heap Reference Analysis [TOPLAS 2007]

- **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

- **Our Objectives:** To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

- **Main Challenge:** Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.

- **Our Key Idea:**

- **Current status:**

- **Further Work:**
Heap Reference Analysis [TOPLAS 2007]

- **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

- **Our Objectives:** To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

- **Main Challenge:** Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.

- **Our Key Idea:** Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.

- **Current status:**

- **Further Work:**

Jan 2010

Uday Khedker, IIT Bombay
Our Solution

1. \( w = x \)
   \( w = null \)
2. \( \text{while} (x.data < \text{max}) \)
   \[ \begin{align*}
   &\{ \\
   &\quad x = x.rptr \\
   \} \\
   &x.rptr = x.lptr.rptr = null \\
   &x.lptr.lptr.lptr = null \\
   &x.lptr.lptr.rptr = null \\
   &x.rptr = x.lptr.rptr = null \\
   &x.lptr.lptr.lptr = null \\
   &x.lptr.lptr.rptr = null \\
\end{align*} \]
3. \( y = x.lptr \)
   \( x.lptr = y.rptr = null \)
   \( y.lptr.lptr = y.lptr.rptr = null \)
4. \( z = \text{New class of z} \)
   \( z.lptr = z.rptr = null \)
5. \( y = y.lptr \)
   \( y.lptr = y.rptr = null \)
6. \( z.sum = x.data + y.data \)
   \( x = y = z = null \)
Heap Reference Analysis: Our Solution

While loop is not executed even once

```
y = z = null
1 w = x
    w = null
2 while (x.data < max)
    { x.lptr = null
      x = x.rptr } x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
x.lptr.lptr.rptr = null
3 y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
4 z = New class of z
    z.lptr = z.rptr = null
5 z.sum = x.data + y.data
6 x = y = z = null
7
```

Stack

Heap
y = z = null
1  w = x
   w = null
2  while (x.data < max)
   { x.lptr = null
   x = x.rptr
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3  }
4  y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5  z = New class of z
6  z.lptr = z.rptr = null
7  y = y.lptr
   y.lptr = y.rptr = null
8  z.sum = x.data + y.data
9  x = y = z = null

While loop is not executed even once
Heap Reference Analysis: Our Solution

\[
y = z = \text{null}
\]

1. \( w = x \)
   \( w = \text{null} \)

2. \( \text{while } (x.\text{data} < \text{max}) \)
   \( \{ \)
   \( x.\text{lptr} = \text{null} \)
   \( x = x.\text{rptr} \)
   \( x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \)

3. \( x = x.\text{rptr} \)

4. \( y = x.\text{lptr} \)
   \( x.\text{lptr} = y.\text{rptr} = \text{null} \)
   \( y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \)

5. \( z = \text{New class of } z \)
   \( z.\text{lptr} = z.\text{rptr} = \text{null} \)

6. \( y = y.\text{lptr} \)
   \( y.\text{lptr} = y.\text{rptr} = \text{null} \)

7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \( x = y = z = \text{null} \)

While loop is not executed even once
Heap Reference Analysis: Our Solution

y = z = null
1  w = x
   w = null
2  while (x.data < max)
   {  x.lptr = null
     x = x.rptr   }
3     x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
4  y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5  z = New class of z
   z.lptr = z.rptr = null
6  y = y.lptr
   y.lptr = y.rptr = null
7  z.sum = x.data + y.data
   x = y = z = null

While loop is not executed even once
Heap Reference Analysis: Our Solution

y = z = null

1 w = x
   w = null

2 while (x.data < max)
   { x.lptr = null
     x = x.rptr   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4 z = New class of z
   z.lptr = z.rptr = null

5 y = y.lptr
   y.lptr = y.rptr = null

6 z.sum = x.data + y.data

7 x = y = z = null

While loop is not executed even once
Heap Reference Analysis: Our Solution

1. \( w = x \)
   \( w = \text{null} \)
2. \( \text{while (} x.\text{data} < \text{max}) \{
   x.\text{lptr} = \text{null}
   x = x.\text{rptr}
\} \)
3. \( x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \)
4. \( y = x.\text{lptr} \)
   \( x.\text{lptr} = y.\text{rptr} = \text{null} \)
   \( y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \)
5. \( z = \text{New class of z} \)
   \( z.\text{lptr} = z.\text{rptr} = \text{null} \)
6. \( y = y.\text{lptr} \)
   \( y.\text{lptr} = y.\text{rptr} = \text{null} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \( x = y = z = \text{null} \)

While loop is not executed even once
Heap Reference Analysis: Our Solution

While loop is not executed even once

1. \( w = x \)
   \( w = \text{null} \)
2. \( \text{while (x.data < max)} \)
   \{ \( x.lptr = \text{null} \)
   \( x = x.rptr \) \}
   \( x.rptr = x.lptr.rptr = \text{null} \)
   \( x.lptr.lptr.lptr = \text{null} \)
3. \( x.lptr.lptr.rptr = \text{null} \)
4. \( y = x.lptr \)
   \( x.lptr = y.rptr = \text{null} \)
   \( y.lptr.lptr = y.lptr.rptr = \text{null} \)
5. \( z = \text{New class of z} \)
   \( z.lptr = z.rptr = \text{null} \)
6. \( y = y.lptr \)
   \( y.lptr = y.rptr = \text{null} \)
7. \( z.sum = x.data + y.data \)
   \( x = y = z = \text{null} \)
Heap Reference Analysis: Our Solution

```
y = z = null
1  w = x
   w = null
2  while (x.data < max)
   { x.lptr = null
3     x = x.rptr  }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
4  y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5  z = New class of z
   z.lptr = z.rptr = null
6  y = y.lptr
   y.lptr = y.rptr = null
7  z.sum = x.data + y.data
   x = y = z = null
```

While loop is executed once
Heap Reference Analysis: Our Solution

1. \( w = x \)
   \( w = \text{null} \)

2. while \((x.\text{data} < \text{max})\)
   \{
   \( x.\text{lptr} = \text{null} \)
   \( x = x.\text{rptr} \)
   \( x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \)
   \( x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \)
   \}

3. \( x = x.\text{rptr} \)

4. \( y = x.\text{lptr} \)
   \( x.\text{lptr} = y.\text{rptr} = \text{null} \)
   \( y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \)

5. \( z = \text{New class of } z \)
   \( z.\text{lptr} = z.\text{rptr} = \text{null} \)

6. \( y = y.\text{lptr} \)
   \( y.\text{lptr} = y.\text{rptr} = \text{null} \)

7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \( x = y = z = \text{null} \)

While loop is executed twice
Some Observations

1. \( y = z = \text{null} \)
2. \( w = x \)
   \[ w = \text{null} \]
3. \( \text{while} (x.\text{data} < \text{max}) \)
   \[
   \begin{cases}
   x.\text{lptr} = \text{null} \\
   x = x.\text{rptr}
   \end{cases}
   \]
   \[ x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \]
   \[ x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \]
   \[ x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \]
4. \( y = x.\text{lptr} \)
   \[ x.\text{lptr} = y.\text{rptr} = \text{null} \]
   \[ y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \]
5. \( z = \text{New class of } z \)
   \[ z.\text{lptr} = z.\text{rptr} = \text{null} \]
6. \( y = y.\text{lptr} \)
   \[ y.\text{lptr} = y.\text{rptr} = \text{null} \]
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \[ x = y = z = \text{null} \]

Node \( i \) is live but link \( a \rightarrow i \) is nullified.

---

Jan 2010

Uday Khedker, IIT Bombay
Some Observations

New access expressions are created. Can they cause exceptions?

1. \( y = z = \text{null} \)
2. \( w = x \)
   \( w = \text{null} \)
3. \( \text{while} (x.\text{data} < \text{max}) \) \{
   \( x.\text{lptr} = \text{null} \)
   \( x = x.\text{rptr} \)
\}
4. \( x.\text{rptr} = x.\text{lptr}.\text{rptr} = \text{null} \)
5. \( x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \)
6. \( x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \)

4. \( y = x.\text{lptr} \)
   \( x.\text{lptr} = y.\text{rptr} = \text{null} \)
   \( y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \)
5. \( z = \text{New class of z} \)
   \( z.\text{lptr} = z.\text{rptr} = \text{null} \)
6. \( y = y.\text{lptr} \)
   \( y.\text{lptr} = y.\text{rptr} = \text{null} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
   \( x = y = z = \text{null} \)
Heap Reference Analysis [TOPLAS 2007]

• **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

• **Our Objectives:** To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

• **Main Challenge:** Unlike stack and static data,
  ▶ heap data accessible to any procedure is unbounded. Hence,
  ▶ the mapping between object names and their addresses needs to change at runtime.

• **Our Key Idea:** Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.

• **Current status:**

• **Further Work:**
Heap Reference Analysis [TOPLAS 2007]

- **The Problem**: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

- **Our Objectives**: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

- **Main Challenge**: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.

- **Our Key Idea**: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.

- **Current status**: Theory and prototype implementation (at the intraprocedural level) ready for Java.

- **Further Work**: 

Jan 2010

Uday Khedker, IIT Bombay
Heap Reference Analysis [TOPLAS 2007]

- **The Problem:** A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.

- **Our Objectives:** To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.

- **Main Challenge:** Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.

- **Our Key Idea:** Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.

- **Current status:** Theory and prototype implementation (at the intraprocedural level) ready for Java.

- **Further Work:**
  - Improve alias analysis
  - Interprocedural implementation and Performance tuning
BTW, What is Static Analysis of Heap?

Static

Dynamic
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Dynamic

Program Execution
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Dynamic

Program Execution

Heap Memory

Heap Memory

Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code
Summary Heap Data

Dynamic

Program Execution
Heap Memory
Heap Memory
Heap Memory

Jan 2010 Uday Khedker, IIT Bombay
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Summary Heap Data

Dynamic

Program Execution

Heap Memory

Profiling

Jan 2010

Uday Khedker, IIT Bombay
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Static Analysis

Summary Heap Data

Dynamic

Program Execution

Heap Memory

Heap Memory

Heap Memory
Part 3

Understanding GCC Machine Descriptions
Role of Machine Descriptions in Translation

Target Independent: Parse → Gimplify → Tree SSA Optimize → Generate RTL → Optimize RTL → Generate ASM

Target Dependent: Gimple → RTL

RTL → ASM
Role of Machine Descriptions in Translation

Target Independent: 
- Parse
- Gimplify
- Tree SSA
- Optimize

Target Dependent: 
- Generate RTL
- Optimize RTL
- Generate ASM

MD Info Required:
- Gimple → RTL
- RTL → ASM
A Target Instruction in Machine Descriptions

(define_insn
  "movsi"
  (set
   (match_operand 0 "register_operand" "r")
   (match_operand 1 "const_int_operand" "k")
  )
  "li %0, %1"
  /* C boolean expression, if required */
A Target Instruction in Machine Descriptions

Define instruction pattern

```
(define_insn
  "movsi"
  (set
   (match_operand 0 "register_operand" "r")
   (match_operand 1 "const_int_operand" "k"))
  ""
  /* C boolean expression, if required */
  "li %0, %1"
)
```

Standard Pattern Name

RTL Expression (RTX):
Semantics of target instruction

target asm inst. =
Concrete syntax for RTX
A Target Instruction in Machine Descriptions

```c
(define_insn
  "movsi"
  (set
   (match_operand 0 "register_operand" "r")
   (match_operand 1 "const_int_operand" "k"))
  "" /* C boolean expression, if required */
  "li %0, %1"
)
```
An Example of Translation

```lisp
(define_insn
  "movsi"
  (set
   (match_operand 0 "register_operand" "r")
   (match_operand 1 "const_int_operand" "k")
  )
  "li %0, %1"
  /* C boolean expression, if required */
)```
An Example of Translation

```
(define_insn
  "movsi"
  (set
    (match_operand 0 "register_operand" "r")
    (match_operand 1 "const_int_operand" "k")
  )
  "li %0, %1"
  /* C boolean expression, if required */
  "li %0, %1"
)
```

```
D.1283 = 10;
(set
  (reg:SI 58 [D.1283])
  (const_int 10: [0xa])
)
⇒ li $t0, 10
```
The Essence of Retargetability

When are the machine descriptions read?
The Essence of Retargetability

When are the machine descriptions read?

- During the build process
The Essence of Retargetability

When are the machine descriptions read?

- During the build process
- When a program is compiled by gcc the information gleaned from machine descriptions is consulted
Retargetability Mechanism of GCC

Compiler Generation Framework

- Input Language
- Target Name

Generated Compiler

- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Language Specific Code
Language and Machine Independent Generic Code
Machine Dependent Generator Code
Machine Descriptions

Selected
Copied
Copied
Generated
Generated

Development Time
Build Time
Use Time

Jan 2010
Uday Khedker, IIT Bombay
Retargetability Mechanism of GCC

Compiler Generation Framework

- Input Language
- Target Name

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

- Selected
- Copied
- Copied
- Generated

- Parser
- Gimplifier
- Tree SSA Optimizer
- RTL Generator
- Optimizer
- Code Generator

Development Time
Build Time
Use Time

Gimple → IR-RTL
IR-RTL → ASM
Retargetability Mechanism of GCC

Compiler Generation Framework

Input Language

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Gimple → PN

PN → IR-RTL

IR-RTL → ASM

Gimple → IR-RTL

IR-RTL → ASM

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Selected

Copied

Copied

Generated

Generated

Development Time

Build Time

Use Time

Generated Compiler

Jan 2010

Uday Khedker, IIT Bombay
Retargetability Mechanism of GCC

Input Language → Language Specific Code
Language Specific Code → Language and Machine Independent Generic Code
Language and Machine Independent Generic Code → Machine Dependent Generator Code
Machine Dependent Generator Code → Machine Descriptions
Machine Descriptions → Gimple → PN
PN → IR-RTL
IR-RTL → ASM
Gimple → IR-RTL
IR-RTL → ASM

Selected → Copied → Generated
Parser → Gimplifier → Tree SSA Optimizer
RTL Generator → Optimizer → Code Generator
Gimple → PN
PN → IR-RTL
Build Time
Development Time
Use Time
IITKgp Improving GCC: Understanding GCC Machine Descriptions

Retargetability Mechanism of GCC

### Compiler Generation Framework

- **Input Language**
- **Language Specific Code**
- **Language and Machine Independent Generic Code**
- **Machine Dependent Generator Code**
- **Machine Descriptions**

### Development Time

- Gimple → PN
  - PN → IR-RTL

### Build Time

- IR-RTL → ASM

### Use Time

- Gimple → IR-RTL
  - IR-RTL → ASM

Jan 2010 Uday Khedker, IIT Bombay
Some Other RTL Constructs

- **define_expan**: Generate possibly multiple RTL statements by using associated C code
- **define_attr**: Define attributes and specify their values
- **define_split**: Split complex insn into simpler ones e.g. for better use of delay slots
- **define_insn_and_split**: A combination of define_insn and define_split
  Used when the split pattern matches and insn exactly.
- **define_peephole2**: Peephole optimization over insns that substitutes insns. Run after register allocation, and before scheduling.
- **define_constants**: Use literal constants in rest of the MD.
### The Size of Machine Descriptions

Size in terms of line counts

<table>
<thead>
<tr>
<th>Files</th>
<th>i386</th>
<th>mips</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.md</td>
<td>35766</td>
<td>12930</td>
</tr>
<tr>
<td>*.c</td>
<td>28643</td>
<td>12572</td>
</tr>
<tr>
<td>*.h</td>
<td>15694</td>
<td>5105</td>
</tr>
</tbody>
</table>
In Search of Modularity in Retargetable Compilation
In Search of Modularity in Retargetable Compilation
In Search of Modularity in Retargetable Compilation
In Search of Modularity in Retargetable Compilation
In Search of Modularity in Retargetable Compilation

Phases of Compilation

Level 1

Source Features (Cumulative)

Level n

Minimal Target Features (Cumulative)
Incremental Construction of Machine Descriptions

[GREPS 2007]

- Conditional control transfers
- Function Calls
- Arithmetic Expressions
  - Sequence of Simple Assignments involving integers
    - MD Level 1
    - MD Level 2
    - MD Level 3
    - MD Level 4
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

- **Consequence**
  The usual ramp up period into GCC machine descriptions has been brought down from over six months to a couple of weeks

- **Current Status**
  - Has been extended to GCC 4.3.1
  - Has been extended to floating point operations

- **Further Work**
  - Extending it to optimizations specified in machine descriptions
  - Adapting it to possible simplifications in machine descriptions
Part 4

Improving Instruction Selection and Machine Descriptions
Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions
Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree machining instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions
Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree machining instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
- Adhoc design decisions
The Problems:

- Instruction selection algorithms are quite ad hoc
  - Full tree matching instead of tree tiling
- The specification mechanism for Machine descriptions is quite ad hoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
- Adhoc design decisions
  - Honouring operand constraints delayed to global register allocation
  - When required during gimple to RTL translation, a lot of C code is required
  - Choice of insertion of NOPs
Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple `define_insn` and `define_expand` patterns
  - Only Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ
  - One RTL expression may appear as a sub-expression of some other RTL expression
- Repetition of C code along with RTL expressions in these patterns.
Consequence of Design Flaws in Machine Descriptions

- The machine descriptions are too verbose, detailed, repetitive and require a lot of C code
- A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code
- The machine descriptions are difficult to construct, understand, maintain, and enhance
- GCC has become a hacker's paradise instead of a clean, production quality compiler generation framework
Step 1: Avoiding Verbosity in Machine Description

- New constructs to facilitate more concise machine descriptions
  - `define_rtltemplate`
    Introduces non-terminals for common RTL expressions instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_code`
    Introduces non-terminals for C/Assembly code instead of rewriting them in each `define_insn` or `define_expand` pattern
  - `define_pattern`
    Allows specification of multiple `define_insn` and `define_expand` sharing RTL template, assembly template, or C code
Step 1: Avoiding Verbosity in Machine Description

- New constructs to facilitate more concise machine descriptions
  - `define rtltemplate`
    Introduces non-terminals for common RTL expressions instead of rewriting them in each `define insn` or `define expand` pattern
  - `define code`
    Introduces non-terminals for C/Assembly code instead of rewriting them in each `define insn` or `define expand` pattern
  - `define pattern`
    Allows specification of multiple `define insn` and `define expand` sharing RTL template, assembly template, or C code

- Generate existing machine descriptions from new descriptions
  - No change in GCC source
  - Incremental changes with gradual transition to new descriptions
Step 2: Improving Instruction Selection

- Since rules become composable, tree tiling based instruction selection algorithms can be used.
  Currently rules are non-composable and GCC uses full tree matching algorithm.
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>= reg + reg</td>
<td></td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= reg * reg</td>
<td></td>
</tr>
</tbody>
</table>
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>reg</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>= a + b</td>
<td></td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * c</td>
<td></td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * c</td>
<td></td>
</tr>
</tbody>
</table>
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>= a + b c</td>
<td>= t * c</td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * b c</td>
<td>= a + a a</td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * b c</td>
<td>= a + a a</td>
</tr>
</tbody>
</table>
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>= a + b c</td>
<td>= t * b c</td>
</tr>
<tr>
<td>reg * reg</td>
<td>a * b c</td>
<td>a + a</td>
</tr>
<tr>
<td>reg * reg</td>
<td>a * b c</td>
<td>a + a</td>
</tr>
</tbody>
</table>
Full Tree Matching

Instructions are viewed as independent non-composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
<th>Modified Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg + reg</td>
<td>= a + b</td>
<td>= t * c</td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * c</td>
<td>= a + a</td>
</tr>
<tr>
<td>= reg * reg</td>
<td>= a * b</td>
<td>= t * c</td>
</tr>
</tbody>
</table>
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>reg Reg</td>
</tr>
<tr>
<td>+ Reg Reg</td>
<td></td>
</tr>
<tr>
<td>* Reg Reg</td>
<td></td>
</tr>
</tbody>
</table>
### Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Reg**: Register

- **Reg**: Register

- **Reg**: Register

- **Reg**: Register

- **Reg**: Register

- **Reg**: Register

- **Reg**: Register
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg Reg</td>
<td>= a + b c</td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← + Reg</td>
<td>a + b c</td>
</tr>
<tr>
<td>Reg ← * Reg</td>
<td>a b c</td>
</tr>
</tbody>
</table>
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg → Reg</td>
<td>=</td>
</tr>
<tr>
<td>Reg → reg</td>
<td></td>
</tr>
<tr>
<td>Reg → +</td>
<td></td>
</tr>
<tr>
<td>Reg → *</td>
<td></td>
</tr>
</tbody>
</table>

Jan 2010

Uday Khedker, IIT Bombay
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg = Reg</td>
<td>a + Reg * Reg</td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← + Reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← * Reg</td>
<td></td>
</tr>
</tbody>
</table>
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg = Reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← +</td>
<td></td>
</tr>
<tr>
<td>Reg ← *</td>
<td></td>
</tr>
</tbody>
</table>

Jan 2010 Uday Khedker, IIT Bombay
## Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg Reg =</td>
<td>= a + Reg</td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← + Reg Reg</td>
<td>Reg Reg</td>
</tr>
<tr>
<td>Reg ← * Reg Reg</td>
<td></td>
</tr>
</tbody>
</table>
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>= reg Reg</td>
<td>= a + (Reg Reg)</td>
</tr>
<tr>
<td>Reg ← reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← + Reg Reg</td>
<td></td>
</tr>
<tr>
<td>Reg ← * Reg Reg</td>
<td></td>
</tr>
</tbody>
</table>
### Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Reg} \rightarrow \text{reg} )</td>
<td>( \text{Reg} \rightarrow a )</td>
</tr>
<tr>
<td>( \text{Reg} \rightarrow \text{Reg} + \text{Reg} )</td>
<td>( \text{Reg} \rightarrow \text{Reg} \times \text{Reg} )</td>
</tr>
</tbody>
</table>
Tree Tiling

Instructions are viewed as composable rules

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Subject Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg → Reg</td>
<td>= a Reg</td>
</tr>
<tr>
<td>Reg → reg</td>
<td></td>
</tr>
<tr>
<td>Reg → + Reg</td>
<td></td>
</tr>
<tr>
<td>Reg → * Reg</td>
<td></td>
</tr>
</tbody>
</table>
Current Status:

- Preliminary investigations seem very promising
  - Fewer rules
  - Simple rules
Current Status:

- Preliminary investigations seem very promising
  - Fewer rules
  - Simple rules
- Prototype of new code generator generator (cgg) is being tested in a toy compiler set up
National Resource Center for F/OSS, Phase II

- Sponsored by Department of Information Technology (DIT), Ministry of Information and Communication Technology
- CDAC Chennai is the coordinating agency
- Participating agencies

<table>
<thead>
<tr>
<th>Organization</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDAC Chennai</td>
<td>SaaS Model, Mobile Internet Devices on BOSS, BOSS applications</td>
</tr>
<tr>
<td>CDAC Mumbai</td>
<td>FOSS Knowledge Base, FOSS Desktops</td>
</tr>
<tr>
<td>CDAC Hyderabad</td>
<td>E-Learning</td>
</tr>
<tr>
<td>IIT Bombay</td>
<td>Gnu Compiler Collection</td>
</tr>
<tr>
<td>IIT Madras</td>
<td>OO Linux kernel</td>
</tr>
<tr>
<td>Anna University</td>
<td>FOSS HRD</td>
</tr>
</tbody>
</table>
Objectives of GCC Resource Center

1. To support the open source movement
   Providing training and technical know-how of the GCC framework to academia and industry.

2. To include better technologies in GCC
   Whole program optimization, Optimizer generation, Tree tiling based instruction selection.

3. To facilitate easier and better quality deployments/enhancements of GCC
   Restructuring GCC and devising methodologies for systematic construction of machine descriptions in GCC.

4. To bridge the gap between academic research and practical implementation
   Designing suitable abstractions of GCC architecture
## GRC Training Programs

<table>
<thead>
<tr>
<th>Title</th>
<th>Target</th>
<th>Objectives</th>
<th>Mode</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop on Essential Abstractions in GCC</td>
<td>People interested in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Three days</td>
</tr>
<tr>
<td>Tutorial on Essential Abstractions in GCC</td>
<td>People interested in knowing about issues in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures and demonstrations</td>
<td>One day</td>
</tr>
<tr>
<td>Workshop on Compiler Construction with Introduction to GCC</td>
<td>College teachers</td>
<td>Explaining the theory and practice of compiler construction and illustrating them with the help of GCC</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Seven days</td>
</tr>
<tr>
<td>Tutorial on Demystifying GCC Compilation</td>
<td>Students</td>
<td>Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)</td>
<td>Lectures and demonstrations</td>
<td>Half day</td>
</tr>
</tbody>
</table>
## GRC Training Programs

<table>
<thead>
<tr>
<th>Title</th>
<th>Target</th>
<th>Objectives</th>
<th>Mode</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop on Essential Abstractions in GCC</td>
<td>People interested in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Three days</td>
</tr>
<tr>
<td>Tutorial on Essential Abstractions in GCC</td>
<td>People interested in knowing about issues in deploying or enhancing GCC</td>
<td>Explaining the essential abstractions in GCC to ensure a quick ramp up into GCC Internals</td>
<td>Lectures and demonstrations</td>
<td>One day</td>
</tr>
<tr>
<td>Workshop on Compiler Construction with Introduction to GCC</td>
<td>College teachers</td>
<td>Explaining the theory and practice of compiler construction and illustrating them with the help of GCC</td>
<td>Lectures, demonstrations, and practicals (experiments and assignments)</td>
<td>Seven days</td>
</tr>
<tr>
<td>Tutorial on Demystifying GCC Compilation</td>
<td>Students</td>
<td>Explaining the translation sequence of GCC through gray box probing (i.e. by examining the dumps produced by GCC)</td>
<td>Lectures and demonstrations</td>
<td>Half day</td>
</tr>
</tbody>
</table>

- **3, 4, and 5 July, 2009**
  - IIT Bombay, Mumbai

- **(modified version) 9 Jan 2010**
  - ACM PPoPP, Bangalore

- **7-13 Dec 2009,**
  - IIT Bombay, Mumbai

- **20 Jan 2010,**
  - Cummins College, Pune

Jan 2010

Uday Khedker, IIT Bombay
CS 715: The Design and Implementation of GNU Compiler Generation Framework

- 6 credits semester long course for M.Tech. (CSE) students at IIT Bombay
- Significant component of experimentation with GCC
- Introduced in 2008-2009
Part 6

Conclusions
Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems
Conclusions

GCC is a strange paradox

• Practically very successful
  ▶ Readily available without any restrictions
  ▶ Easy to use
  ▶ Easy to examine compilation without knowing internals
  ▶ Available on a wide variety of processors and operating systems
  ▶ Can be retargeted to new processors and operating systems

• Quite adhoc
Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems

- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation
Conclusions

GCC is a strange paradox

- Practically very successful
  - Readily available without any restrictions
  - Easy to use
  - Easy to examine compilation without knowing internals
  - Available on a wide variety of processors and operating systems
  - Can be retargeted to new processors and operating systems

- Quite adhoc
  - Needs significant improvements in terms of design
    Machine description specification, IRs, optimizer generation
  - Needs significant improvements in terms of better algorithms
    Retargetability mechanism, interprocedural optimizations, parallelization, vectorization,
Conclusions

GCC Resource Center at IIT Bombay

- Synergy from group activities
- Long term commitment to challenging research problems
- A desire to explore real issues in real compilers
  A dream to improve GCC
Last but not the least . . .

Thank You!