GCC for Parallelization

Uday Khedker, Supratim Biswas, Prashant Rawat

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

January 2010
Outline

- GCC: The Great Compiler Challenge
- Configuration and Building
- Introduction to parallelization and vectorization
- Observing parallelization and vectorization performed by GCC
- Activities of GCC Resource Center
About this Tutorial

- Expected background
  Some compiler background, no knowledge of GCC or parallelization
- Takeaways: After this tutorial you will be able to
  - Appreciate the GCC architecture
  - Configure and build GCC
  - Observe what GCC does to your programs
  - Study parallelization transformations done by GCC
  - Get a feel of the strengths and limitations of GCC
The Scope of this Tutorial

- What this tutorial does not address
  - Code or data structures of gcc
  - Algorithms used for parallelization and vectorization
  - Machine level issues related to parallelization and vectorization
  - Advanced theory such as polyhydral approach
  - Research issues

- What this tutorial addresses

  Basics of Discovering Parallelism using GCC
Part 1

GCC ≡ The Great Compiler Challenge
The Gnu Tool Chain

Source Program

\[ \text{gcc} \]

Target Program
The Gnu Tool Chain

Source Program → gcc → Target Program

cc1
The Gnu Tool Chain

Source Program

gcc

Target Program

cc1

cpp
The Gnu Tool Chain

Source Program

\[\text{gcc}\]

\[\text{cc1}\] \rightarrow \text{cpp}

\[\text{as}\]

Target Program
The Gnu Tool Chain

Source Program

gcc

Target Program

cc1

cpp

as

ld
The Gnu Tool Chain

Source Program

 kitt

Target Program

gcc

cc1

cpp

as

glibc/newlib

ld
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,
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, LatticeMicro32, 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]
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

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

UPK, SB, PR GRC, IIT Bombay
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

UPK, SB, PR
GRC, IIT Bombay
The Architecture of GCC

Input Language

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

Generated Compiler (cc1)

Source Program

Assembly Program

Selected

Copied

Generated

UPK, SB, PR

GRC, IIT Bombay
The Architecture of GCC

Compiler Generation Framework

Input Language

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Generated Compiler (cc1)

Source Program

Assembly Program

UPK, SB, PR

GRC, IIT Bombay

Development Time

Build Time

Use Time

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator
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!
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
  ```
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</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 collect2.c</td>
<td>766</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>1 fix-header.c</td>
<td>1074</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>2 fp-test.c</td>
<td>85</td>
<td>main (void )</td>
</tr>
<tr>
<td>3 gcc.c</td>
<td>6216</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>4 gcov-dump.c</td>
<td>76</td>
<td>main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>5 gcov iov.c</td>
<td>29</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>6 gcov.c</td>
<td>355</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>7 gen protos.c</td>
<td>130</td>
<td>main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>8 genattr.c</td>
<td>89</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>9 genattrtab.c</td>
<td>4438</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>a gen automata.c</td>
<td>9321</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>b gen checksum.c</td>
<td>65</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>c gen codes.c</td>
<td>51</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>d gen conditions.c</td>
<td>209</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>e gen config.c</td>
<td>261</td>
<td>main (int argc, char **argv)</td>
</tr>
<tr>
<td>f gen constants.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>
Part 2

Meeting the GCC Challenge
# Meeting the GCC Challenge

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

<table>
<thead>
<tr>
<th>Needs</th>
<th>Examining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefiles</td>
<td>Source</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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

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
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

Compiler Generation Framework

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

Input Language -> Compiler Generation Framework

- Selected
- Copied
- Generated

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

Target Name

Development Time
Build Time
Use Time

Generated Compiler

UPK, SB, PR
GRC, IIT Bombay
Understanding the Retargetability Mechanism

Compiler Generation Framework:

- **Input Language**
- **Target Name**

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

**Development Time**

**Build Time**

**Use Time**

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

**Generated Compiler**

- **Gimple → IR-RTL**
- **IR-RTL → ASM**
Understanding the Retargetability Mechanism

**Compiler Generation Framework**

- **Input Language**
- **Target Name**

**Selected**
- **Copied**
- **Generated**

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

- **Gimple → PN**
- **PN → IR-RTL**
- **IR-RTL → ASM**

- **Gimple → IR-RTL**
- **IR-RTL → ASM**

**Generated Compiler**

**UPK, SB, PR**

**GRC, IIT Bombay**
Understanding the Retargetability Mechanism

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

Built Time

Development Time

Use Time

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Generated Compiler
Understanding the Retargetability Mechanism

Compiler Generation Framework

- **Input Language**
- **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**

- **Development Time**
- **Build Time**
- **Use Time**

- **Gimple → PN**
- **PN → IR-RTL**
- **IR-RTL → ASM**
- **Gimple → IR-RTL**
- **IR-RTL → ASM**
Understanding the Retargetability Mechanism

Many more details need to be explained 😞
## Our Current Focus

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

<table>
<thead>
<tr>
<th>Makefiles</th>
<th>Source</th>
<th>MD</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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
  (e.g. 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)/<lang dir>

• 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) ≠ $(BUILD) ≠ $(INSTALL)
- None of the above directories should be contained in any of the above directories
Configuring GCC

configure
Configuring GCC

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

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

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

configure

config.guess

config.sub

makefile

config.h.in

Makefile.in

config.log

config.cache

config.status

Makefile

config.h
Steps in Configuration and Building

Usual Steps

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

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

UPK, SB, PR

GRC, IIT Bombay
### Steps in Configuration and Building

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

*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

<table>
<thead>
<tr>
<th>BS = HS = TS</th>
<th>Native Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS = HS ≠ TS</td>
<td>Cross Build</td>
</tr>
<tr>
<td>BS ≠ HS ≠ TS</td>
<td>Canadian Cross</td>
</tr>
</tbody>
</table>

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.
Bootstrapping

A compiler is just another program
It is improved, bugs are fixed and newer versions are released
To build a new version given a built old version:

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

⇒ Building cross compilers stops after Stage 1!
T Notation for a Compiler
T Notation for a Compiler

input language

C \rightarrow \text{i386}

\text{i386} \rightarrow \text{cc}
T Notation for a Compiler

input language

output language

C -> i386

i386 -> CC

UPK,SB,PR GRC, IIT Bombay
T Notation for a Compiler

input language

output language

C

i386

cc

execution language
T Notation for a Compiler

input language

C

i386

CC

output language

evaluation language

name of the translator
A Native Build on i386

GCC
Source

Requirement: \textbf{BS} = \textbf{HS} = \textbf{TS} = \text{i386}
A Native Build on i386

Requirement: $BS = HS = TS = i386$
A Native Build on i386

Requirement: \( BS = HS = TS = \text{i386} \)
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

Requirement: $BS = HS = TS = i386$
- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
A Native Build on i386

Requirement: \( BS = HS = TS = i386 \)
- Stage 1 build compiled using \( cc \)
- Stage 2 build compiled using \( gcc \)
- Stage 3 build compiled using \( gcc \)
A Native Build on i386

Requirement: \( BS = HS = TS = i386 \)

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
- Stage 3 build compiled using gcc
- Stage 2 and Stage 3 Builds must be identical for a successful native build
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: \( BS = HS = \text{i386}, \ TS = \text{mips} \)
- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
  Its \( HS \) = mips and not i386!
A More Detailed Look at Building

Source Program

\[ \text{gcc} \]

Target Program

\[ \text{glibc/newlib} \]

\[ \text{as} \]

\[ \text{cc1} \]

\[ \text{cpp} \]

\[ \text{ld} \]
A More Detailed Look at Building

Source Program

Partially generated and downloaded source is compiled into executables

Target Program

gcc

cc1

cpp

as

ld

glibc/newlib

UPK, SB, PR

GRC, IIT Bombay
A More Detailed Look at Building

Source Program

\[ \rightarrow \]

Target Program

**Partially generated and downloaded source is compiled into executables**

**Existing executables are directly used**

- **cc1**
- **cpp**
- **as**
- **ld**
- **glibc/newlib**

UPK, SB, PR

GRC, IIT Bombay
A More Detailed Look at Building

Source Program

Partially generated and downloaded source is compiled into executables

Target Program

Existing executables are directly used

cc1
cpp
as
ld
glibc/newlib
A More Detailed Look at Cross Build

Requirement: $BS = HS = i386, TS = mips$
A More Detailed Look at Cross Build

Requirement: \( BS = HS = i386, \ TS = \text{mips} \)

- Stage 1 build consists of only \text{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
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
A More Detailed Look at Cross Build

Stage 2 build is infeasible for cross build

Requirement: $BS = HS = \text{mips}$

- 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 GCC
  Some options differ in the two builds
Commands for Configuring and Building GCC

This is what we specify

- cd $(BUILD)
This is what we specify

- `cd $(BUILD)`
- `$(SOURCE)/configure <options>`
  - `configure output: customized Makefile`
This is what we specify

- `cd $(BUILD)`
- `$(SOURCE)/configure <options>`
  configure output: customized Makefile
- `make 2> make.err > make.log`
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
    - \((\$(\text{SOURCE})/\text{gcc/gen*.c})\) are read and generator executables are created in \(\$(\text{BUILD})/\text{gcc/build}\)
  - MD files are read by the generator executables and back end source code is generated in \(\$(\text{BUILD})/\text{gcc}\)

- **Compilation**
  - Other source files are read from \(\$(\text{SOURCE})\) and executables created in corresponding subdirectories of \(\$(\text{BUILD})\)

- **Installation**
  - Created executables and libraries are copied in \(\$(\text{INSTALL})\)
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 \(\$(INSTALL)\)
Build failures due to Machine Descriptions

Incomplete MD specifications  ⇒  Unsuccessful build
Incorrect MD specification  ⇒  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
```
Part 4

First Level Gray Box Probing of GCC
Gray Box Probing of GCC: Outline

- Overview of translation sequence in GCC
- Overview of intermediate representations
- Intermediate representations of programs across important phases
Important Phases of GCC

C Source Code

- Parser
- AST
- Gimplifier
- GIMPLE
- Linearizer
- Lower
- CFG Generator
- CFG
- RTL Generator
- RTL expand

Lowering of abstraction!

local reg allocator
- lregs
- global reg allocator
- Gregs
- pro_epilogue generation
- prologue-epilogue
- Pattern Matcher
- ASM Program
Phases of GCC

To see the output after each pass use the option

- fdump-<ir>-<pass>

where <ir> is

- -tree-<pass>
  <pass> could be: gimple, cfg etc.
  Use -all to see all dumps
- -rtl-<pass>
  <pass> could be: expand, greg, vreg etc.
  Use -all to see all dumps
  We can also use -da option

Example:

gcc -fdump-tree-all -fdump-rtl-all test.c
Example Program

```c
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
    if (a<=12)
    {
        a = a+b+c;
    }
}```
GCC 4.3.3 Dumps for Our Example Program

test.c.001t.tu

test.c.003t.original

test.c.004t.gimple

test.c.006t.vcg

test.c.007t.useless

test.c.010t.lower

test.c.011t.ehopt

test.c.012t.eh

test.c.013t.cfg

test.c.014t.cplxlower0

test.c.015t.veclower

test.c.021t.cleanup

test.c.051t.apply_inline

test.c.131r.expand

test.c.132r.sibling

test.c.134r.initvals

test.c.135r.unshare

test.c.136r.vregs

test.c.137r.into_cfglayout

test.c.138r.jump

test.c.157r.regclass

test.c.160r.outof_cfglayout

test.c.166r.split1

test.c.168r.dfini

test.c.169r.mode-sw

test.c.171r.asmcons

test.c.174r.subregs_of_mode_init

test.c.175r.lreg

test.c.176r.greg

test.c.177r.subregs_of_mode_finish

test.c.180r.split2

test.c.182r.pro_and_epilogue

test.c.196r.stack

test.c.197r.alignments

test.c.200r.mach

test.c.201r.barriers

test.c.204r.eh-ranges

test.c.205r.shorten

test.c.206r.dfinish

test.s
Selected Dumps for Our Example Program

test.c.001t.tu  test.c.157r.regclass  
test.c.003t.original  test.c.160r.outof_cfglayout  
test.c.004t.gimple  test.c.166r.split1  
test.c.006t.vcg  test.c.168r.dfinish  
test.c.007t.useless  test.c.169r.mode-sw  
test.c.010t.lower  test.c.171r.asmcons  
test.c.011t.ehopt  test.c.174r.subregs_of_mode_init  
test.c.012t.eh  test.c.175r.lreg  
test.c.013t.cfg  test.c.176r.greg  
test.c.014t.cplxlower0  test.c.177r.subregs_of_mode_finish  
test.c.015t.veclower  test.c.180r.split2  
test.c.021t.cleanup_cfg1  test.c.182r.pro_and_epilogue  
test.c.051t.apply_inline  test.c.196r.stack  
test.c.131r.expand  test.c.197r.alignments  
test.c.132r.sibling  test.c.200r.mach  
test.c.134r.initvals  test.c.201r.barriers  
test.c.135r.unshare  test.c.204r.eh-ranges  
test.c.136r.vregs  test.c.205r.shorten  
test.c.137r.into_cfglayout  test.c.206r.dfinish  
test.c.138r.jump  test.s  

UPK,SB,PR  
GRC, IIT Bombay
Important Phases of GCC

C Source Code

- Parser
- AST
- Gimplifier
- GIMPLE
- Linearizer
- Lower
- CFG Generator
- CFG
- RTL Generator
- RTL expand

local reg allocator
- lregs
- global reg allocator
- Gregs
- pro_epilogue generation
- prologue-epilogue
- Pattern Matcher
- ASM Program
Gimplifier

- Three-address representation derived from GENERIC
  - Computation represented as a sequence of basic operations
  - Temporaries introduced to hold intermediate values
- Control construct are explicated into conditional jumps
Gimple: Translation of Composite Expressions

File: test.c.004t.gimple

if (a <= 12)
{
    a = a+b+c;
}
else
{
    D.1199 = a + b;
    a = D.1199 + c;
}
Gimple: Translation of Higher Level Control Constructs

File: test.c.004t.gimple

```c
while (a <= 7)
{
    a = a + 1;
}
```

goto <D.1197>;
<D.1196>:

```c
a = a + 1;
<D.1197>:
```

if (a <= 7)
{
    goto <D.1196>;
}
else
{
    goto <D.1198>;
}

```
<D.1198>:
```

Gimple: Translation of Higher Level Control Constructs

File: test.c.004t.gimple

```c
while (a <= 7)
{
    a = a + 1;
}

goto <D.1197>;
<D.1196>:
    a = a + 1;
<D.1197>:
    if (a <= 7)
    {
        goto <D.1196>;
    }
    else
    {
        goto <D.1198>;
    }
<D.1198>:
```

UPK, SB, PR GRC, IIT Bombay
Gimple: Translation of Higher Level Control Constructs

File: test.c.004t.gimple

while (a <= 7)
{
    a = a+1;
}

go to <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7)
{
    goto <D.1196>;
}
else
{
    goto <D.1198>;
}
<D.1198>:
Gimple: Translation of Higher Level Control Constructs

File: test.c.004t.gimple

```c
while (a <= 7)
{
    a = a + 1;
}
goto <D.1197>;
<D.1196>:
    a = a + 1;
<D.1197>:
    if (a <= 7)
    {
        goto <D.1196>;
    }
    else
    {
        goto <D.1198>;
    }
<D.1198>:
```
Important Phases of GCC

C Source Code

Parser

AST

Gimplifier

GIMPLE

Linearizer

Lower

CFG Generator

CFG

RTL Generator

RTL expand

local reg allocator

lregs

global reg allocator

Gregs

pro_epilogue generation

prologue-epilogue

Pattern Matcher

ASM Program

UPK,SB,PR

GRC, IIT Bombay
File: test.c.010t.lower

if (a <= 12) {
    D.1199 = a + b;
    a = D.1199 + c;
} else goto <D.1201>;

if (a <= 12) goto <D.1200>;
else goto <D.1201>;

<D.1200>:
    D.1199 = a + b;
    a = D.1199 + c;

<D.1201>:
    return;
**File:** test.c.010t.lower

```c
if (a <= 12) {
    D.1199 = a + b;
    a = D.1199 + c;
}
```

```c
if (a <= 12) goto <D.1200>;
else goto <D.1201>;
<D.1200>:;
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:;
return;
```

**if-then translated in terms of conditional and unconditional gotos**
Lowering Gimple

File: test.c.010t.lower

```c
if (a <= 12)
{
    D.1199 = a + b;
    a = D.1199 + c;
}
else goto <D.1201>;
return;
```

if-then translated in terms of conditional and unconditional gotos

```c
if (a <= 12) goto <D.1200>;
else goto <D.1201>;
<D.1200>:;
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:;
return;
```
Important Phases of GCC

C Source Code

- Parser
  - AST
    - Gimplifier
      - GIMPLE
        - Linearizer
          - Lower
            - CFG Generator
              - CFG
                - RTL Generator
                  - RTL expand

local reg allocator

- Iregs
  - global reg allocator
    - Gregs
      - pro_epilogue generation
        - prologue-epilogue
          - Pattern Matcher
            - ASM Program

UPK,SB,PR
Constructing the Control Flow Graph

File: test.c.013t.cfg

if (a <= 12) goto <D.1200>;
else    goto <D.1201>;
<D.1200>;;
D.1199 = a + b;
a = D.1199 + c;
<D.1201>;;
return;

# BLOCK 5
if (a <= 7)
    goto <bb 6>;
else
    goto <bb 7>;
# SUCC: 6 (true) 7 (false)
# BLOCK 6
D.1199 = a + b;
a = D.1199 + c;
# SUCC: 7 (fallthru)
# BLOCK 7
return;
# SUCC: EXIT
**File:** test.c.013t.cfg

**Control Flow Graph**

```
while(a <= 7)
  a = a + 1;
```

```
if(a <= 12)
  a = a + b + c;
```

```
if(a <= 7)
```

```
D.1199 = a + b;
a = D.1199 + c;
```

```
return;
```

**Block 4:**

```
if(a <= 7)
```

**Block 5:**

```
if(a <= 12)
```

**Block 3:**

```
a = a + 1;
```

**Block 6:**

```
D.1199 = a + b;
a = D.1199 + c;
```

**Block 7:**

```
return;
```
Decisions that have been taken

- Three-address representation is generated
- All high level control flow structures are made explicit.
- Source code divided into interconnected blocks of sequential statements.
- This is a convenient structure for later analysis.
Important Phases of GCC

C Source Code

Parser

AST

Gimliplier

GIMPLE

Linearizer

Lower

CFG Generator

CFG

RTL Generator

RTL expand

local reg allocator

Iregs

global reg allocator

Gregs

pro_epilogue generation

prologue-epilogue

Pattern Matcher

ASM Program

UPK, SB, PR

GRC, IIT Bombay
Expansion into RTL for i386 Port

Translation of $a = a + 1$

**File:** test.c.031r.expand

$$\text{stack}($fp - 4) = \text{stack}($fp - 4) + 1$$

|| flags=？


*Plus operation computes $fp - 4$ as the address of variable $a*
Expansion into RTL for i386 Port

Translation of \( a = a + 1 \)

File: test.c.031r.expand

\[
\begin{align*}
\text{(insn 12 11 0 (parallel[} & \text{ set (mem/c/i:SI (plus:SI} \\
& \text{ (reg/f:SI 54 virtual-stack-vars)} \\
& \text{ (const int -4 [...]}) [...])} \\
& \text{(plus:SI} \\
& \text{ (mem/c/i:SI (plus:SI} \\
& \text{ (reg/f:SI 54 virtual-stack-vars)} \\
& \text{ (const int -4 [...]}) [...])} \\
& \text{ (const int 1 [...]})}) \\
& \text{ (clobber (reg:CC 17 flags))} \\
\text{]}) -1 (nil))}
\end{align*}
\]

\[
\text{stack($fp - 4) = stack($fp - 4) + 1} \\
|| \text{ flags=?}
\]

*Set denotes assignment*
Translation of \( a = a + 1 \)

File: \texttt{test.c.031r.expand}

\[
\begin{align*}
\text{stack($fp - 4$)} &= \text{stack($fp - 4$) + 1} \\
| | & \text{flags=}?
\end{align*}
\]

\[
\begin{align*}
(\text{insn 12 11 0 (parallel [}} & \\
&(\text{set (mem/c/i:SI (plus:SI}} & \\
&(\text{reg/f:SI 54 virtual-stack-vars}) & \\
&(\text{const int -4 [...]]})) [\ldots]) &
\end{align*}
\]

\[
\begin{align*}
(\text{plus:SI} & \\
&(\text{mem/c/i:SI (plus:SI}} & \\
&(\text{reg/f:SI 54 virtual-stack-vars}) & \\
&(\text{const int -4 [...]]})) [\ldots]) &
\end{align*}
\]

\[
\begin{align*}
&(\text{const int 1 [...]])]) &
\end{align*}
\]

\[
\begin{align*}
&(\text{clobber (reg:CC 17 flags))} &
\end{align*}
\]

\] -1 (nil))

1 is added to variable \( a \)
Expansion into RTL for i386 Port

Translation of $a = a + 1$

**File:** test.c.031r.expand

```
stack($fp - 4) = stack($fp - 4) + 1 || flags=?
```

(insn 12 11 0 (parallel [ 
  (set (mem/c/i:SI (plus:SI 
    (reg/f:SI 54 virtual-stack-vars) 
    (const int -4 [...]))) [...])
  (plus:SI 
    (mem/c/i:SI (plus:SI 
      (reg/f:SI 54 virtual-stack-vars) 
      (const int -4 [...]))) [...])
    (const int 1 [...])))
  (clobber (reg:CC 17 flags))
]) -1 (nil))

Condition Code register is clobbered to record possible side effect of plus
Flags in RTL Expressions

Meanings of some of the common flags

/c memory reference that does not trap
/i scalar that is not part of an aggregate
/f register that holds a pointer
Expansion into RTL in spim Port

Translation of $a = a + 1$

File: test.c.031r.expand

(r39=stack($fp - 4))
(r40=r39+1)
(stack($fp - 4)=r40)

In spim, a variable is loaded into register to perform any instruction, hence three instructions are generated.
Expansion into RTL in spim Port

Translation of \( a = a + 1 \)

**File:** test.c.031r.expand

\[
\begin{align*}
\text{(insn 7 6 8 test.c:6 (set (reg:SI 39))}
&\quad \text{(mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars))}
&\quad \text{(const_int -4 [...])}) [...]) -1 (nil))\\
\text{(insn 8 7 9 test.c:6 (set (reg:SI 40))}
&\quad \text{(plus:SI (reg:SI 39))}
&\quad \text{(const_int 1 [...])}) -1 (nil))\\
\text{(insn 9 8 0 test.c:6 (set)}
&\quad \text{(mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars))}
&\quad \text{(const_int -4 [...])}) [...])
&\quad \text{(reg:SI 40)) -1 (nil))}
\end{align*}
\]

In spim, a variable is loaded into register to perform any instruction,

hence three instructions are generated

- \( r39 = \text{stack($fp - 4$)} \)
- \( r40 = r39 + 1 \)
- \( \text{stack($fp - 4$)} = r40 \)
Expansion into RTL in spim Port

Translation of $a = a + 1$

File: test.c.031r.expand

```
(insn 7 6 8 test.c:6 (set (reg:SI 39)
   (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
      (const_int -4 [...]))) [...])) -1 (nil))
(insn 8 7 9 test.c:6 (set (reg:SI 40)
   (plus:SI (reg:SI 39)
      (const_int 1 [...]))) -1 (nil))
(insn 9 8 0 test.c:6 (set
   (mem/c/i:SI (plus:SI (reg/f:SI 33 virtual-stack-vars)
      (const_int -4 [...]))) [...])
   (reg:SI 40)) -1 (nil))
```

In spim, a variable is loaded into register to perform any instruction, hence three instructions are generated

r39 = stack($fp - 4$)
r40 = r39 + 1
stack($fp - 4$) = r40
Important Phases of GCC

C Source Code

Parser

AST

Gimplifier

GIMPLE

Linearizer

Lower

CFG Generator

CFG

RTL Generator

RTL expand

local reg allocator

lregs

global reg allocator

Gregs

pro_epilogue generation

prologue-epilogue

Pattern Matcher

ASM Program
Local Register Allocation: i386 Port

File: test.c.175r.lreg

(insn 12 11 13 4 (parallel [#]
    (set (mem/c/i:SI (plus:SI
        (reg/f:SI 20 frame)
        (const_int -4 [...]))) [...]>)
    (plus:SI (mem/c/i:SI
        (plus:SI
            (reg/f:SI 20 frame)
            (const_int -4 [...])]) [...]})
    (const_int 1 [...]])
    (clobber (reg:CC 17 flags))
)) 249 *addsi_1

(expr_list:REG_UNUSED (reg:CC 17 flags) (nil))

Identifies candidates for local register allocation ≡
Definitions which have all uses within the same block
Local Register Allocation: i386 Port

File: test.c.175r.lreg

Basic block 3, prev 2, next 4, loop_depth 0, count 0, freq 0.
Predecessors: 4

;; bb 3 artificial_uses: u-1(6) u-1(7) u-1(16) u-1(20)
;; lr in 6 [bp] 7 [sp] 16 [argp] 20 [frame]
;; lr use 6 [bp] 7 [sp] 16 [argp] 20 [frame]
;; lr def 17 [flags]

*Identifies candidates for local register allocation ≡
Definitions which have all uses within the same block*
Local Register Allocation: spim Port

File: test.c.175r.lreg

(insn 7 6 8 2 (set (reg:SI 39)
    (mem/c/i:SI (plus:SI (reg/f:SI 1 $at)
        (const_int -4 [...]))) [...]))
    4 *IITB_move_from_mem (nil))

(insn 8 7 9 2 (set (reg:SI 40)
    (plus:SI (reg:SI 39)
        (const_int 1 [...]))) 12 addsi3
    (expr_list:REG DEAD (reg:SI 39) (nil)))

(insn 9 8 12 2 (set
    (mem/c/i:SI (plus:SI (reg/f:SI 1 $at)
        (const_int -4 [...]))) [...])
    (reg:SI 40)) 5 *IITB_move_to_mem
    (expr_list:REG DEAD(reg:SI 40) (nil)))

*Discovers the last uses of reg:SI 39 and reg:SI 40*
Important Phases of GCC

C Source Code

Parser

AST

Gimiplifier

GIMPLE

Linearizer

Lower

CFG Generator

CFG

RTL Generator

RTL expand

local reg allocator

Iregs

global reg allocator

Gregs

pro_epilogue generation

prologue-epilogue

Pattern Matcher

ASM Program

UPK,SB,PR

GRC, IIT Bombay
Global Register Allocation: i386 Port

File: test.c.176r.greg

Global Register Allocation: spim Port

File: test.c.176r.greg

(insn 7 6 8 3 test.c:4 (set (reg:SI 2 $v0 [39]))
(mem/c/i:SI (plus:SI (reg/f:SI 1 $fp )
(const_int -4 [...]))) [...])
4 *IITB_move_from_mem (nil))

(insn 8 7 9 3 test.c:4 (set (reg:SI $v0 [40]))
(plus:SI (reg:SI $v0 [39])
(const_int 1 [...]))) 12 addsi3 (nil))

(insn 9 8 18 3 test.c:4 (set
(mem/c/i:SI (plus:SI (reg/f:SI 1 $fp )
(const_int -4 [...]))) [...])
(reg:SI $v0 40)) 5 *IITB_move_to_mem (nil))

$v0$ is used for both reg:SI 39 and reg:SI 40
Important Phases of GCC

C Source Code

- Parser
- AST
- Gimplifier
- GIMPLE
- Linearizer
- Lower
- CFG Generator
- CFG
- RTL Generator
- RTL expand

Local reg allocator
- lregs
- global reg allocator
- Gregs
- pro_epilogue generation
- prologue-epilogue
- Pattern Matcher
- ASM Program

UPK, SB, PR   GRC, IIT Bombay
Activation Record Structure in Spim

Caller’s Activation Record
Activation Record Structure in Spim

Caller’s Responsibility

Caller’s Activation Record

Parameter $n$
Activation Record Structure in Spim

- Caller’s Activation Record
  - Parameter $n$
  - Parameter $n-1$
Activation Record Structure in Spim

Caller’s Activation Record

Parameter $n$

Parameter $n - 1$

...
Activation Record Structure in Spim

- Caller's Activation Record
  - Parameter \( n \)
  - Parameter \( n - 1 \)
  - \( \ldots \)
  - Parameter 1

Caller's Responsibility

Argument Pointer
Activation Record Structure in Spim

Caller’s Activation Record

Parameter $n$

Parameter $n - 1$

...  

Parameter 1

Return Address

Caller’s Responsibility

Callee’s Responsibility

Argument Pointer
Activation Record Structure in Spim

Caller’s Activation Record

- Parameter $n$
- Parameter $n - 1$
- ...
- Parameter 1

Return Address

Caller’s FPR (Control Link)

Argument Pointer

Caller’s Responsibility

Callee’s Responsibility

UPK, SB, PR
Activation Record Structure in Spim

- **Caller’s Activation Record**
  - Parameter $n$
  - Parameter $n-1$
  - ...
  - Parameter 1
- **Return Address**
- **Caller’s FPR (Control Link)**
- **Caller’s SPR**

**Caller’s Responsibility**

**Callee’s Responsibility**

**Argument Pointer**
Activation Record Structure in Spim

<table>
<thead>
<tr>
<th>Caller’s Activation Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter $n$</td>
</tr>
<tr>
<td>Parameter $n-1$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Parameter 1</td>
</tr>
<tr>
<td>Return Address</td>
</tr>
<tr>
<td>Caller’s FPR (Control Link)</td>
</tr>
<tr>
<td>Caller’s SPR</td>
</tr>
<tr>
<td>Callee Saved Registers</td>
</tr>
</tbody>
</table>

**Caller’s Responsibility**

- Parameter $n$
- Parameter $n-1$
- ... (ellipsis)
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers

**Callee’s Responsibility**

- Argument Pointer

Size is known only after register allocation.
Activation Record Structure in Spim

Caller’s Activation Record

Parameter \( n \)

Parameter \( n - 1 \)

... 

Parameter 1

Return Address

Caller’s FPR (Control Link)

Caller’s SPR

Callee Saved Registers

Local Variable 1

Caller’s Responsibility

Callee’s Responsibility

Argument Pointer

Size is known only after register allocation

Initial Frame Pointer

UPK,SB,PR

GRC, IIT Bombay
Activation Record Structure in Spim

**Caller’s Activation Record**
- Parameter $n$
- Parameter $n - 1$
- ... (ellipsis)
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers
- Local Variable 1
- Local Variable 2

**Argument Pointer**
Size is known only after register allocation

**Initial Frame Pointer**

**Caller’s Responsibility**

**Callee’s Responsibility**
Activation Record Structure in Spim

Caller’s Activation Record

- Parameter $n$
- Parameter $n - 1$
- ... 
- Parameter 1
- Return Address
- Caller’s FPR (Control Link)
- Caller’s SPR
- Callee Saved Registers
- Local Variable 1
- Local Variable 2
- ... 

Caller’s Responsibility

Callee’s Responsibility

Size is known only after register allocation

Argument Pointer

Initial Frame Pointer
Activation Record Structure in Spim

Caller’s Activation Record

- Parameter $n$
- Parameter $n-1$
- ... 
- Parameter 1

Return Address

Caller’s FPR (Control Link)

Caller’s SPR

Callee Saved Registers

- Local Variable 1
- Local Variable 2
- ...
- Local Variable $n$

Caller’s Responsibility

Callee’s Responsibility

Argument Pointer

Size is known only after register allocation

Initial Frame Pointer

Stack Pointer
### RTL for Function Calls in spim

<table>
<thead>
<tr>
<th>Calling function</th>
<th>Called function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate memory for actual parameters on stack</td>
<td>Allocate memory for return value (push)</td>
</tr>
<tr>
<td>Copy actual parameters</td>
<td>Store mandatory callee save registers (push)</td>
</tr>
<tr>
<td><strong>Call function</strong></td>
<td>Set frame pointer</td>
</tr>
<tr>
<td>Get result from stack (pop)</td>
<td>Allocate local variables (push)</td>
</tr>
<tr>
<td>Deallocate memory for activation record (pop)</td>
<td><strong>Execute code</strong></td>
</tr>
<tr>
<td></td>
<td>Put result in return value space</td>
</tr>
<tr>
<td></td>
<td>Deallocate local variables (pop)</td>
</tr>
<tr>
<td></td>
<td>Load callee save registers (pop)</td>
</tr>
<tr>
<td></td>
<td>Return</td>
</tr>
</tbody>
</table>

UPK,SB,PR

GRC, IIT Bombay
Prologue and Epilogue: spim

File: test.c.182r.pro_and_epilogue

(insn 17 3 18 2 test.c:2
  (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8]))
  (reg:SI 31 $ra)) -1 (nil))
(insn 18 17 19 2 test.c:2
  (set (mem:SI (plus:SI (reg/f:SI 29 $sp)
                  (const_int -4 [...])) [...]))
  (reg/f:SI 29 $sp)) -1 (nil))
(insn 19 18 20 2 test.c:2 (set
  (mem:SI (plus:SI (reg/f:SI 29 $sp)
                  (const_int -8 [...])) [...]))
  (reg/f:SI 30 $fp)) -1 (nil))
(insn 20 19 21 2 test.c:2 (set (reg/f:SI 30 $fp)
  (reg/f:SI 29 $sp)) -1 (nil))
(insn 21 20 22 2 test.c:2 (set (reg/f:SI 29 $sp)
  (plus:SI (reg/f:SI 30 $fp)
            (const_int -32 [...])))) -1 (nil))
Prologue and Epilogue: spim

File: test.c.182r.pro_and_epilogue

(insn 17 3 18 2 test.c:2
  (set (mem:SI (reg/f:SI 29 $sp) [0 S4 A8]))
  (reg:SI 31 $ra)) -1 (nil))

(insn 18 17 19 2 test.c:2
  (set (mem:SI (plus:SI (reg/f:SI 29 $sp)
    (const_int -4 [...] [[...]]
    (reg/f:SI 29 $sp)) -1 (nil))
  (insn 19 18 20 2 test.c:2 (set
    (mem:SI (plus:SI (reg/f:SI 29 $sp)
      (const_int -8 [...] [[...]]
      (reg/f:SI 30 $fp)) -1 (nil))
  (insn 20 19 21 2 test.c:2 (set (reg/f:SI 30 $fp)
    (reg/f:SI 29 $sp)) -1 (nil))
  (insn 21 20 22 2 test.c:2 (set (reg/f:SI 29 $sp)
    (plus:SI (reg/f:SI 30 $fp)
      (const_int -32 [...] [[...]])) -1 (nil))

sw $ra, 0($sp)
sw $sp, 4($sp)
sw $fp, 8($sp)
move $fp,$sp
addi $sp,$fp,32
Important Phases of GCC

C Source Code

Parser

AST

Gimplifier

GIMPLE

Linearizer

Lower

CFG Generator

CFG

RTL Generator

RTL expand

local reg allocator

Iregs

global reg allocator

Gregs

pro_epilogue generation

prologue-epilogue

Pattern Matcher

ASM Program

UPK, SB, PR

GRC, IIT Bombay
Assembly

Assembly Code for $a = a + 1$

**File:** test.s

<table>
<thead>
<tr>
<th>For spim</th>
<th>For i386</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw  $v0, -8($fp)</td>
<td>addl $1, -8(%ebp)</td>
</tr>
<tr>
<td>addi $v0, $v0, 1</td>
<td></td>
</tr>
<tr>
<td>sw  $v0, -8($fp)</td>
<td></td>
</tr>
</tbody>
</table>

3 instruction required in spim and one in i386
Gray Box Probing of GCC: Conclusions

- Source code is transformed into assembly by lowering the abstraction level step by step to bring it close to machine architecture.
- This transformation can be understood to a large extent by observing inputs and output of the different steps in the transformation.
- In gcc, the output of almost all the passes can be examined.
- The complete list of dumps can be figured out by the command `man gcc`.
Part 5

Introduction to Parallelization and Vectorization
# A Taxonomy of Parallel Computation

<table>
<thead>
<tr>
<th></th>
<th>Single Program</th>
<th>Multiple Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data</td>
<td>SPSD</td>
<td>MPSD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SPMD</td>
<td>MPMD</td>
</tr>
</tbody>
</table>
# A Taxonomy of Parallel Computation

<table>
<thead>
<tr>
<th></th>
<th>Single Program</th>
<th>Multiple Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Instructions</td>
<td>MISD</td>
<td>MPSD</td>
</tr>
<tr>
<td>Single Data</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPMD</td>
</tr>
</tbody>
</table>
## A Taxonomy of Parallel Computation

<table>
<thead>
<tr>
<th></th>
<th>Single Program</th>
<th>Multiple Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single Instruction</td>
<td>Multiple Instructions</td>
</tr>
<tr>
<td>Single Data</td>
<td>SISD</td>
<td>?</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPMD</td>
</tr>
</tbody>
</table>

Redundant computation for validation of intermediate steps
### A Taxonomy of Parallel Computation

<table>
<thead>
<tr>
<th>Single Program</th>
<th>Multiple Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction</td>
<td>Multiple Instructions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Single Data</th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD</td>
<td>MISD</td>
<td>MPSD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiple Data</th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD</td>
<td>MIMD</td>
<td>MPMD</td>
</tr>
</tbody>
</table>

Transformations performed by a compiler
Vectorization: SISD $\Rightarrow$ SIMD

- Parallelism in executing operation on shorter operands (8-bit, 16-bit, 32-bit operands)
- Existing 32 or 64-bit arithmetic units used to perform multiple operations in parallel
  A 64 bit word $\equiv$ a vector of $2 \times (32 \text{ bits})$, $4 \times (16 \text{ bits})$, or $8 \times (8 \text{ bits})$
Example 1

Vectorization \( (\text{SISD} \Rightarrow \text{SIMD}) \) : Yes
Parallelization \( (\text{SISD} \Rightarrow \text{MIMD}) \) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

```
A[0..N]          B[0..N]
```

...
Example 1

**Vectorization** (SISD ⇒ SIMD) : Yes
**Parallelization** (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

- `A[0..N]`
- `B[0..N]`

Iteration # 1
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

```
A[0..N]  B[0..N]
```

Iteration #  1  2
### Example 1

**Vectorization** (SISD $\Rightarrow$ SIMD) : Yes

**Parallelization** (SISD $\Rightarrow$ MIMD) : Yes

**Original Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

- **A[0..N]**
- **B[0..N]**

Iteration # | 1 | 2 | 3
**Example 1**

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

**Original Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location.

A[0..N]  B[0..N]

Iteration #  1  2  3  4
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location...

- A[0..N]
- B[0..N]
- Iteration #: 1 2 3 4 5
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location...

A[0..N]...
B[0..N]...

Iteration # 1 2 3 4 5 6
**Example 1**

**Vectorization** (SISD $\Rightarrow$ SIMD) : Yes  
**Parallelization** (SISD $\Rightarrow$ MIMD) : Yes

**Original Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

A[0..N]  B[0..N]

Iteration # 1 2 3 4 5 6 7
**Example 1**

**Vectorization** (SISD ⇒ SIMD) : Yes

**Parallelization** (SISD ⇒ MIMD) : Yes

**Original Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location...

A[0..N]  B[0..N]

Iteration #  1  2  3  4  5  6  7  8  9
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

```
A[0..N]  B[0..N]
```

Iteration #

1  2  3  4  5  6  7  8  9  10  11
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example 1**

**Vectorization**  (SISD ⇒ SIMD) : Yes
**Parallelization**  (SISD ⇒ MIMD) : Yes

**Original Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

**Vectorized Code**

```c
int A[N], B[N], i;
for (i=1; i<N; i=i+4)
```

---

**Graphical Representation**

- **A[0..N]**
- **B[0..N]**
- Iteration #

---

**Vectorization Factor**

UPK, SB, PR  
GRC, IIT Bombay
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Vectorized Code

```c
int A[N], B[N], i;
for (i=1; i<N; i=i+4)
```

Vectorization Factor

![Diagram of vectorization and parallelization with arrows indicating operations on arrays A and B over iterations.](image)
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Vectorized Code

```c
int A[N], B[N], i;
for (i=1; i<N; i=i+4)
```

Vectorization Factor

Iteration #

A[0..N]  B[0..N]

1  2
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Vectorized Code

```c
int A[N], B[N], i;
for (i=1; i<N; i=i+4)
```

---

**Vectorization Factor**

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>A[0..N]</th>
<th>B[0..N]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1" alt="A[0..N] vectorized" /></td>
<td><img src="image2" alt="B[0..N] vectorized" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image1" alt="A[0..N] vectorized" /></td>
<td><img src="image2" alt="B[0..N] vectorized" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image1" alt="A[0..N] vectorized" /></td>
<td><img src="image2" alt="B[0..N] vectorized" /></td>
</tr>
<tr>
<td>...</td>
<td><img src="image1" alt="A[0..N] vectorized" /></td>
<td><img src="image2" alt="B[0..N] vectorized" /></td>
</tr>
</tbody>
</table>

---

UPK, SB, PR

GRC, IIT Bombay
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

A[0..N]  
B[0..N]  

. . .
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location

Iteration #
Example 1

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

Original Code

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Parallelized Code

```c
int A[N], B[N], i;
foreach (i=1; i<N; )
  A[i] = A[i] + B[i-1];
```

---

UPK, SB, PR
GRC, IIT Bombay
Example 1: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location
Example 1: The Moral of the Story

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iteration $i$</strong></td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

int A[N], B[N], i;

for (i=1; i<N; i++)

A[i] = A[i] + B[i-1];

When the same location is accessed across different iterations, the order of reads and writes must be preserved.
**Example 1: The Moral of the Story**

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration $i$</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

A[0..N]  . . .  B[0..N]  . . .
Example 1: The Moral of the Story

Vectorization \(\text{(SISD } \Rightarrow \text{ SIMD)}\) : Yes
Parallelization \(\text{(SISD } \Rightarrow \text{ MIMD)}\) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration (i)</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

int A[N], B[N], i;
for (i=1; i<N; i++)
A[i] = A[i] + B[i-1];
**Example 1: The Moral of the Story**

Vectorization (SISD $\Rightarrow$ SIMD) : Yes  
Parallelization (SISD $\Rightarrow$ MIMD) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
<th>Iteration $i$</th>
<th>Iteration $i + k$</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];
```

UPK,SB,PR  
GRC, IIT Bombay
Example 1: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iteration</strong> $i$</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```
Example 2

Vectorization \((\text{SISD} \Rightarrow \text{SIMD})\) : Yes
Parallelization \((\text{SISD} \Rightarrow \text{MIMD})\) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]    ...    B[0..N]    ...
Example 2

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

Iteration #
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location...
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]  B[0..N]

Iteration #  1  2
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0..N]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[0..N]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location...
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]  B[0..N]

Iteration #  1  2  3  4  5
Example 2

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

![Diagram of vectorized code execution]

Iteration #  1  2  3  4  5  6
Example 2

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]
B[0..N]

Iteration # 1 2 3 4 5 6 7
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]

B[0..N]

Iteration #  1  2  3  4  5  6  7  8
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

A[0..N]
B[0..N]
Iteration # 1 2 3 4 5 6 7 8 9 10
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location

```
A[0..N]  B[0..N]
```

Iteration #  1  2  3  4  5  6  7  8  9  10  11
Example 2

Vectorization \((\text{SISD} \Rightarrow \text{SIMD})\) : Yes
Parallelization \((\text{SISD} \Rightarrow \text{MIMD})\) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location
Example 2

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

- Vector instruction is synchronized: All reads before writes in a given instruction
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

- Vector instruction is synchronized: All reads before writes in a given instruction

![Diagram of vectorization and parallelization](image-url)
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

- Vector instruction is synchronized: All reads before writes in a given instruction
Example 2

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

- Vector instruction is synchronized: All reads before writes in a given instruction

![Diagram showing iteration and vectorization process]
Example 2

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```
int A[N], B[N], i;
for (i=0; i<N; i++)
```

- Vector instruction is synchronized: All reads before writes in a given instruction
- Read-writes across multiple instructions executing in parallel may not be synchronized
Example 2: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

Observe reads and writes into a given location
Example 2: The Moral of the Story

Vectorization  
(SISD ⇒ SIMD) : Yes

Parallelization  
(SISD ⇒ MIMD) : No

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration $i$</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

When the same location is accessed across different iterations, the order of reads and writes must be preserved.
Example 2: The Moral of the Story

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
<th>Iteration $i$</th>
<th>Iteration $i + k$</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

int A[N], B[N], i;
for (i = 0; i < N; i++)

A[0..N] 
B[0..N]
Example 2: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : No

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration $i$</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

A[0..N]... B[0..N]...
Example 2: The Moral of the Story

Vectorization (SISD ⇒ SIMD): Yes
Parallelization (SISD ⇒ MIMD): No

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration $i$</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

A[0..N]  . . .  B[0..N]  . . .
**Example 2: The Moral of the Story**

Vectorization (SISD $\Rightarrow$ SIMD) : Yes
Parallelization (SISD $\Rightarrow$ MIMD) : No

When the same location is accessed across different iterations, the order of reads and writes must be preserved.

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
<th>Iteration $i$</th>
<th>Iteration $i + k$</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
<td>Does not matter</td>
</tr>
</tbody>
</table>

Original Code

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
```

..

When the same location is accessed across different iterations, the order of reads and writes must be preserved.
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location:

```
A[0..N] .......... 
B[0..N] .......... 
```
**Example 3**

Vectorization (SISD $\Rightarrow$ SIMD) : No  
Parallelization (SISD $\Rightarrow$ MIMD) : No  

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

![Diagram showing memory access patterns](image-url)
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

<table>
<thead>
<tr>
<th>A[0..N]</th>
<th>B[0..N]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration #</td>
<td>1</td>
</tr>
</tbody>
</table>

UPK,SB,PR

GRC, IIT Bombay
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

```
A[0..N]  B[0..N]
```

Iteration # 1 2
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location.

Iteration #
1 2 3
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];

Observe reads and writes into a given location

\[ A[0..N] \]
\[ B[0..N] \]

Iteration #: 1 2 3 4
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

```
A[0..N]
B[0..N]
```

Iteration # 1 2 3 4 5
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];

Observe reads and writes into a given location

A[0..N]  B[0..N]

Iteration #  1  2  3  4  5  6  7

...  ...

UPK,SB,PR  GRC, IIT Bombay
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

```
A[0..N]  B[0..N]
```

Iteration #  1  2  3  4  5  6  7  8  ...
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[0..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location.

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
</table>
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];

Observe reads and writes into a given location

A[0..N]  B[0..N]
Iteration #  1  2  3  4  5  6  7  8  9  10  11  ...
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location

A[0..N] · · ·

B[0..N] · · ·

Iteration # 1 2 3 4 5 6 7 8 9 10 11 12 · · ·
**Example 3**

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
A[i+1] = A[i] + B[i+1];
```

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
<th>Iteration $i$</th>
<th>Iteration $i + k$</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Iteration #**

<table>
<thead>
<tr>
<th>A[0..N]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>B[0..N]</th>
</tr>
</thead>
</table>

| Iteration # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... |
```
**Example 3**

Vectorization (SISD ⇒ SIMD) : **No**

Parallelization (SISD ⇒ MIMD) : **No**

```
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

Nature of accesses in our example:

<table>
<thead>
<tr>
<th>Iteration ( i )</th>
<th>Iteration ( i + k )</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td>No</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td>Yes</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td></td>
</tr>
</tbody>
</table>

A[0..N] ... B[0..N] ...

Iteration #  1  2  3  4  5  6  7  8  9  10  11  12  ...
Example 3

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : No

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration $i$</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Write</td>
</tr>
<tr>
<td>Read</td>
</tr>
</tbody>
</table>

```
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```
Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

<table>
<thead>
<tr>
<th>Nature of accesses in our example</th>
<th>Iteration $i$</th>
<th>Iteration $i + k$</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Write</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>Read</td>
<td>Does not matter</td>
<td></td>
</tr>
</tbody>
</table>

```
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];
```

A[0..N] and B[0..N] are accessed in a sequential manner, indicating that vectorization is not possible. The nature of accesses in the example does not allow for parallelization either.
Example 4

Vectorization (SISD $\Rightarrow$ SIMD) : No
Parallelization (SISD $\Rightarrow$ MIMD) : Yes
Example 4

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : Yes

• This case is not possible
Example 4

Vectorization  (SISD $\Rightarrow$ SIMD)  : No
Parallelization (SISD $\Rightarrow$ MIMD)  : Yes

• This case is not possible
• Vectorization is a limited granularity parallelization
Example 4

Vectorization  (SISD $\Rightarrow$ SIMD)  : No
Parallelization  (SISD $\Rightarrow$ MIMD)  : Yes

- This case is not possible
- Vectorization is a limited granularity parallelization
- If parallelization is possible then vectorization is trivially possible
### Data Dependence

Let statements $S_i$ and $S_j$ access memory location $m$ at time instants $t$ and $t + k$

<table>
<thead>
<tr>
<th>Access in $S_i$</th>
<th>Access in $S_j$</th>
<th>Dependence</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read $m$</td>
<td>Write $m$</td>
<td>Anti (or Pseudo)</td>
<td>$S_i \bar{\delta} S_j$</td>
</tr>
<tr>
<td>Write $m$</td>
<td>Read $m$</td>
<td>Flow (or True)</td>
<td>$S_i \delta S_j$</td>
</tr>
<tr>
<td>Write $m$</td>
<td>Write $m$</td>
<td>Output (or Pseudo)</td>
<td>$S_i \delta^O S_j$</td>
</tr>
<tr>
<td>Read $m$</td>
<td>Read $m$</td>
<td>Does not matter</td>
<td></td>
</tr>
</tbody>
</table>

- Pseudo dependences may be eliminated by some transformations
- True dependence prohibits parallel execution of $S_i$ and $S_j$
Loop Carried and Loop Independent Dependences

Consider dependence between statements $S_i$ and $S_j$ in a loop

- **Loop independent dependence.** $t$ and $t + k$ occur in the same iteration of a loop
  - $S_i$ and $S_j$ must be executed sequentially
  - Different iterations of the loop can be parallelized

- **Loop carried dependence.** $t$ and $t + k$ occur in the different iterations of a loop
  - Within an iteration, $S_i$ and $S_j$ can be executed in parallel
  - Different iterations of the loop must be executed sequentially

- $S_i$ and $S_j$ may have both loop carried and loop independent dependences
Dependence in Example 1

• Program

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1]; /* S1 */
```

• Dependence graph

```
S₁

δ∞
```

• No loop carried dependence
Both vectorization and parallelization are possible
Dependence in Example 1

- Program

```c
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1]; /* S1 */
```

- Dependence graph

- No loop carried dependence
  Both vectorization and parallelization are possible
Dependence in Example 2

- **Program**

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i] = A[i+1] + B[i]; /* S1 */
```

- **Dependence graph**

```
S_1 \rightarrow \bar{\delta}_1
```

- **Loop carried anti-dependence**
  Parallelization is not possible
  Vectorization is possible since all reads are done before all writes
Dependence in Example 2

- Program

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i] = A[i+1] + B[i]; /* S1 */
```

- Dependence graph

- Loop carried anti-dependence
  Parallelization is not possible
  Vectorization is possible since all reads are done before all writes
Dependence in Example 3

• Program

```c
int A[N], B[N], i;
for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1]; /* S1 */
```

• Dependence graph

```
S_1 \rightarrow \delta_1
```

• Loop carried flow-dependence
  Neither parallelization nor vectorization is possible
Example 4: Dependence

Program to swap arrays

```
for (i=0; i<N; i++)
{
    T = A[i];  /* S1 */
    A[i] = B[i];  /* S2 */
    B[i] = T;  /* S3 */
}
```

Dependence Graph
Example 4: Dependence

Program to swap arrays

for (i=0; i<N; i++)
{
    T = A[i]; /* S1 */
    A[i] = B[i]; /* S2 */
    B[i] = T; /* S3 */
}

Loop independent anti dependence due to A[i]
Example 4: Dependence

Program to swap arrays

```
for (i=0; i<N; i++)
{   T = A[i];    /* S1 */
    A[i] = B[i]; /* S2 */
    B[i] = T;    /* S3 */
}
```

Dependence Graph

Loop independent anti dependence due to B[i]
Example 4: Dependence

Program to swap arrays

```
for (i=0; i<N; i++)
{
    T = A[i];     /* S1 */
    A[i] = B[i];  /* S2 */
    B[i] = T;     /* S3 */
}
```

Dependence Graph

Loop independent flow dependence due to T
Example 4: Dependence

Program to swap arrays

for (i=0; i<N; i++)
{
    T = A[i];      /* S1 */
    A[i] = B[i];   /* S2 */
    B[i] = T;      /* S3 */
}

Dependence Graph

Loop carried anti dependence due to T

UPK, SB, PR
GRC, IIT Bombay
Example 4: Dependence

Program to swap arrays

for (i=0; i<N; i++)
{
    T = A[i]; /* S1 */
    A[i] = B[i]; /* S2 */
    B[i] = T; /* S3 */
}

Dependence Graph

Loop carried output dependence due to T
Example 4: Dependence

**Program to swap arrays**

```c
for (i=0; i<N; i++)
{
    T = A[i];    /* S1 */
    A[i] = B[i]; /* S2 */
    B[i] = T;    /* S3 */
}
```

**Dependence Graph**

[Diagram of the dependence graph showing nodes S1, S2, S3 with edges δ₁, δ₂, δ₃, δ₁₀, δ₂₁, δ₃₂.]
Tutorial Problem for Discovering Dependence

Draw the dependence graph for the following program (Earlier program modified to swap 2-dimensional arrays)

```c
for (i=0; i<N; i++)
{
    for (j=0; j<N; j++)
    {
        T = A[i][j];      /* S1 */
        A[i][j] = B[i][j]; /* S2 */
        B[i][j] = T;      /* S3 */
    }
}
```
Data Dependence in Loops

- Analysis in loop is tricky, as
  - Loops may be nested
  - Different loop iterations may access same memory location
  - Arrays occur frequently
  - Far too many array locations to be treated as independent scalars
Data Dependence in Loops

• Analysis in loop is tricky, as
  ▶ Loops may be nested
  ▶ Different loop iterations may access same memory location
  ▶ Arrays occur frequently
  ▶ Far too many array locations to be treated as independent scalars

• Consider array location $A[4][9]$ in the following program

```c
for(i = 0; i <= 5; i++)
  for(j = 0; j <= 4; j++)
  {
    A[i+1][3*j] = ...; /* S1 */
    ... = A[i+3][2*j+1]; /* S2 */
  }
```
Data Dependence in Loops

• Analysis in loop is tricky, as
  ▶ Loops may be nested
  ▶ Different loop iterations may access same memory location
  ▶ Arrays occur frequently
  ▶ Far too many array locations to be treated as independent scalars

• Consider array location $A[4][9]$ in the following program

```c
for(i = 0; i <= 5; i ++)
    for(j = 0; j <= 4; j ++)
    {
        A[i+1][3*j] = ... ; /* S1 */
        ... = A[i+3][2*j+1]; /* S2 */
    }
```

S2 accesses in iteration (1,4), S1 accesses in iteration (3,3)
Iteration Vectors and Index Vectors: Example 1

```c
for (i=0, i<4; i++)
    for (j=0; j<4; j++)
    {
        a[i+1][j] = a[i][j] + 2;
    }
```

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHS</td>
<td>RHS</td>
</tr>
<tr>
<td>0,0</td>
<td>1,0</td>
</tr>
<tr>
<td>0,1</td>
<td>1,1</td>
</tr>
<tr>
<td>0,2</td>
<td>1,2</td>
</tr>
<tr>
<td>0,3</td>
<td>1,3</td>
</tr>
<tr>
<td>1,0</td>
<td>2,0</td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
</tr>
<tr>
<td>2,0</td>
<td>3,0</td>
</tr>
<tr>
<td>2,1</td>
<td>3,1</td>
</tr>
<tr>
<td>2,2</td>
<td>3,2</td>
</tr>
<tr>
<td>2,3</td>
<td>3,3</td>
</tr>
<tr>
<td>3,0</td>
<td>4,0</td>
</tr>
<tr>
<td>3,1</td>
<td>4,1</td>
</tr>
<tr>
<td>3,2</td>
<td>4,2</td>
</tr>
<tr>
<td>3,3</td>
<td>4,3</td>
</tr>
</tbody>
</table>
Iteration Vectors and Index Vectors: Example 1

for (i=0, i<4; i++)
    for (j=0; j<4; j++)
    {
        a[i+1][j] = a[i][j] + 2;
    }

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LHS</td>
</tr>
<tr>
<td>0,0</td>
<td>1,0</td>
</tr>
<tr>
<td>0,1</td>
<td>1,1</td>
</tr>
<tr>
<td>0,2</td>
<td>1,2</td>
</tr>
<tr>
<td>0,3</td>
<td>1,3</td>
</tr>
<tr>
<td>1,0</td>
<td>2,0</td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
</tr>
<tr>
<td>2,0</td>
<td>3,0</td>
</tr>
<tr>
<td>2,1</td>
<td>3,1</td>
</tr>
<tr>
<td>2,2</td>
<td>3,2</td>
</tr>
<tr>
<td>2,3</td>
<td>3,3</td>
</tr>
<tr>
<td>3,0</td>
<td>4,0</td>
</tr>
<tr>
<td>3,1</td>
<td>4,1</td>
</tr>
<tr>
<td>3,2</td>
<td>4,2</td>
</tr>
<tr>
<td>3,3</td>
<td>4,3</td>
</tr>
</tbody>
</table>

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical
Iteration Vectors and Index Vectors: Example 1

```c
for (i=0, i<4; i++)
    for (j=0; j<4; j++)
    {
        a[i+1][j] = a[i][j] + 2;
    }
```

Loop carried dependence exists if
- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

**Conclusion: Dependence exists**

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>1,0</td>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>0,1</td>
<td>1,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>0,2</td>
<td>1,2</td>
<td>0,2</td>
<td>0,2</td>
</tr>
<tr>
<td>0,3</td>
<td>1,3</td>
<td>0,3</td>
<td>0,3</td>
</tr>
<tr>
<td>1,0</td>
<td>2,0</td>
<td>1,0</td>
<td>1,0</td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>1,1</td>
<td>1,1</td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>1,2</td>
<td>1,2</td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>1,3</td>
<td>1,3</td>
</tr>
<tr>
<td>2,0</td>
<td>3,0</td>
<td>2,0</td>
<td>2,0</td>
</tr>
<tr>
<td>2,1</td>
<td>3,1</td>
<td>2,1</td>
<td>2,1</td>
</tr>
<tr>
<td>2,2</td>
<td>3,2</td>
<td>2,2</td>
<td>2,2</td>
</tr>
<tr>
<td>2,3</td>
<td>3,3</td>
<td>2,3</td>
<td>2,3</td>
</tr>
<tr>
<td>3,0</td>
<td>4,0</td>
<td>3,0</td>
<td>3,0</td>
</tr>
<tr>
<td>3,1</td>
<td>4,1</td>
<td>3,1</td>
<td>3,1</td>
</tr>
<tr>
<td>3,2</td>
<td>4,2</td>
<td>3,2</td>
<td>3,2</td>
</tr>
<tr>
<td>3,3</td>
<td>4,3</td>
<td>3,3</td>
<td>3,3</td>
</tr>
</tbody>
</table>

UPK, SB, PR GRC, IIT Bombay
Iteration Vectors and Index Vectors: Example 1

```
for (i=0, i<4; i++)
    for (j=0; j<4; j++)
    {
        a[i+1][j] = a[i][j] + 2;
    }
```

Iteration carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

**Conclusion: Dependence exists**

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LHS</td>
</tr>
<tr>
<td>0, 0</td>
<td>1, 0</td>
</tr>
<tr>
<td>0, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td>0, 2</td>
<td>1, 2</td>
</tr>
<tr>
<td>0, 3</td>
<td>1, 3</td>
</tr>
<tr>
<td>1, 0</td>
<td>2, 0</td>
</tr>
<tr>
<td>1, 1</td>
<td>2, 1</td>
</tr>
<tr>
<td>1, 2</td>
<td>2, 2</td>
</tr>
<tr>
<td>1, 3</td>
<td>2, 3</td>
</tr>
<tr>
<td>2, 0</td>
<td>3, 0</td>
</tr>
<tr>
<td>2, 1</td>
<td>3, 1</td>
</tr>
<tr>
<td>2, 2</td>
<td>3, 2</td>
</tr>
<tr>
<td>2, 3</td>
<td>3, 3</td>
</tr>
<tr>
<td>3, 0</td>
<td>4, 0</td>
</tr>
<tr>
<td>3, 1</td>
<td>4, 1</td>
</tr>
<tr>
<td>3, 2</td>
<td>4, 2</td>
</tr>
<tr>
<td>3, 3</td>
<td>4, 3</td>
</tr>
</tbody>
</table>
Iteration Vectors and Index Vectors: Example 1

for (i=0, i<4; i++)
  for (j=0; j<4; j++)
  {
    a[i+1][j] = a[i][j] + 2;
  }

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

Conclusion: Dependence exists
Iteration Vectors and Index Vectors: Example 2

for (i=0, i<4; i++)
    for (j=0; j<4; j++)
    {
        a[i][j] = a[i][j] + 2;
    }

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHS</td>
<td>RHS</td>
</tr>
<tr>
<td>0,0</td>
<td>0,0</td>
</tr>
<tr>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>0,2</td>
<td>0,2</td>
</tr>
<tr>
<td>0,3</td>
<td>0,3</td>
</tr>
<tr>
<td>1,0</td>
<td>1,0</td>
</tr>
<tr>
<td>1,1</td>
<td>1,1</td>
</tr>
<tr>
<td>1,2</td>
<td>1,2</td>
</tr>
<tr>
<td>1,3</td>
<td>1,3</td>
</tr>
<tr>
<td>2,0</td>
<td>2,0</td>
</tr>
<tr>
<td>2,1</td>
<td>2,1</td>
</tr>
<tr>
<td>2,2</td>
<td>2,2</td>
</tr>
<tr>
<td>2,3</td>
<td>2,3</td>
</tr>
<tr>
<td>3,0</td>
<td>3,0</td>
</tr>
<tr>
<td>3,1</td>
<td>3,1</td>
</tr>
<tr>
<td>3,2</td>
<td>3,2</td>
</tr>
<tr>
<td>3,3</td>
<td>3,3</td>
</tr>
</tbody>
</table>
Iteration Vectors and Index Vectors: Example 2

for (i=0, i<4; i++)
    for (j=0; j<4; j++)
        {
            a[i][j] = a[i][j] + 2;
        }

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical
Iteration Vectors and Index Vectors: Example 2

for (i=0, i<4; i++)
  for (j=0; j<4; j++)
  {
    a[i][j] = a[i][j] + 2;
  }

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

Conclusion: No dependence

<table>
<thead>
<tr>
<th>Iteration Vector</th>
<th>Index Vector</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,0</td>
<td>0,0</td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
<td></td>
</tr>
<tr>
<td>0,2</td>
<td>0,2</td>
<td>0,2</td>
<td></td>
</tr>
<tr>
<td>0,3</td>
<td>0,3</td>
<td>0,3</td>
<td></td>
</tr>
<tr>
<td>1,0</td>
<td>1,0</td>
<td>1,0</td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>1,1</td>
<td>1,1</td>
<td></td>
</tr>
<tr>
<td>1,2</td>
<td>1,2</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>1,3</td>
<td>1,3</td>
<td></td>
</tr>
<tr>
<td>2,0</td>
<td>2,0</td>
<td>2,0</td>
<td></td>
</tr>
<tr>
<td>2,1</td>
<td>2,1</td>
<td>2,1</td>
<td></td>
</tr>
<tr>
<td>2,2</td>
<td>2,2</td>
<td>2,2</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td>2,3</td>
<td>2,3</td>
<td></td>
</tr>
<tr>
<td>3,0</td>
<td>3,0</td>
<td>3,0</td>
<td></td>
</tr>
<tr>
<td>3,1</td>
<td>3,1</td>
<td>3,1</td>
<td></td>
</tr>
<tr>
<td>3,2</td>
<td>3,2</td>
<td>3,2</td>
<td></td>
</tr>
<tr>
<td>3,3</td>
<td>3,3</td>
<td>3,3</td>
<td></td>
</tr>
</tbody>
</table>
Data Dependence Theorem [KA02]

There exists a dependence from statement $S_1$ to statement $S_2$ in common nest of loops if and only if there exist two iteration vectors $i$ and $j$ for the nest, such that

1. $i < j$ or $i = j$ and there exists a path from $S_1$ to $S_2$ in the body of the loop,

2. statement $S_1$ accesses memory location $M$ on iteration $i$ and statement $S_2$ accesses location $M$ on iteration $j$, and

3. one of these accesses is a write access.
Implementation Issues

- Getting loop information (Loop discovery)
- Finding value spaces of induction variables, index expressions, and pointer accesses
- Analyzing data dependence
- Performing transformations
Loop Information

Loop0
{   Loop1
    {   Loop2
        {
        }
        Loop3
        {   Loop4
            {
            }
        }
    }
}{ Loop5
    {
    }
}
Representing Value Spaces of Variables and Expressions

Chain of Recurrences: 3-tuple \(\langle\text{Starting Value}, \text{modification}, \text{stride}\rangle\)
[BWZ94, KMZ98]

```
for (i=3; i<=15; i=i+3)
{
    for (j=11; j>=1; j=j-2)
    {
        A[i+1][2*j-1] = ...;
    }
}
```

<table>
<thead>
<tr>
<th>Entity</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Induction variable (i)</td>
<td>({3, +, 3})</td>
</tr>
<tr>
<td>Induction variable (j)</td>
<td>({11, +, -2})</td>
</tr>
<tr>
<td>Index expression (i+1)</td>
<td>({4, +, 3})</td>
</tr>
<tr>
<td>Index expression (2*j-1)</td>
<td>({21, +, -4})</td>
</tr>
</tbody>
</table>
Advantages of Chain of Recurrences

CR can represent any affine expression
⇒ Accesses through pointers can also be tracked

```c
int A[32], B[32];
int i, *p;
p = &B
for(i = 2; i<N; i++)
{
    *(p++) = A[i] + *p;
    A[i] = *p;
}
```
Advantages of Chain of Recurrences

CR can represent any affine expression
⇒ Accesses through pointers can also be tracked

```c
int A[32], B[32];
int i, *p;
p = &B
for(i = 2; i<N; i++)
{
    *(p++) = A[i] + *p;
    A[i] = *p;
}
```
Advantages of Chain of Recurrences

CR can represent any affine expression

⇒ Accesses through pointers can also be tracked

```
int A[32], B[32];
int i, *p;
p = &B
for(i = 2; i<N; i++)
{
    *(p++) = A[i] + *p;
    A[i] = *p;
}
```
Transformation Passes in GCC

- A total of 196 unique pass names initialized in
  `${SOURCE}/gcc/passes.c`
  - Some passes are called multiple times in different contexts
    Conditional constant propagation and dead code elimination are called thrice
  - Some passes are only demo passes (eg. data dependence analysis)
  - Some passes have many variations (eg. special cases for loops)
    Common subexpression elimination, dead code elimination

- The pass sequence can be divided broadly in two parts
  - Passes on Gimple
  - Passes on RTL

- Some passes are organizational passes to group related passes
## Passes On Gimple

<table>
<thead>
<tr>
<th>Pass Group</th>
<th>Examples</th>
<th>Number of passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowering</td>
<td>Gimple IR, CFG Construction</td>
<td>12</td>
</tr>
<tr>
<td>Interprocedural Optimizations</td>
<td>Conditional Constant Propagation, Inlining, SSA Construction</td>
<td>36</td>
</tr>
<tr>
<td>Intraprocedural Optimizations</td>
<td>Constant Propagation, Dead Code Elimination, PRE</td>
<td>40</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Vectorization, Parallelization</td>
<td>24</td>
</tr>
<tr>
<td>Remaining Intraprocedural Optimizations</td>
<td>Value Range Propagation, Rename SSA</td>
<td>23</td>
</tr>
<tr>
<td>Generating RTL</td>
<td></td>
<td>01</td>
</tr>
<tr>
<td>Total number of passes on Gimple</td>
<td></td>
<td>136</td>
</tr>
</tbody>
</table>
### Passes On Gimple

<table>
<thead>
<tr>
<th>Pass Group</th>
<th>Examples</th>
<th>Number of passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowering</td>
<td>Gimple IR, CFG Construction</td>
<td>12</td>
</tr>
<tr>
<td>Interprocedural Optimizations</td>
<td>Conditional Constant Propagation, Inlining, SSA Construction</td>
<td>36</td>
</tr>
<tr>
<td>Intraprocedural Optimizations</td>
<td>Constant Propagation, Dead Code Elimination, PRE</td>
<td>40</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Vectorization, Parallelization</td>
<td>24</td>
</tr>
<tr>
<td>Remaining Intraprocedural Optimizations</td>
<td>Value Range Propagation, Rename SSA</td>
<td>23</td>
</tr>
<tr>
<td>Generating RTL</td>
<td></td>
<td>01</td>
</tr>
<tr>
<td>Total number of passes on Gimple</td>
<td></td>
<td>136</td>
</tr>
</tbody>
</table>

Our Focus is Vectorization and Parallelization
## Passes On RTL

<table>
<thead>
<tr>
<th>Pass Group</th>
<th>Examples</th>
<th>Number of passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intraprocedural Optimizations</td>
<td>CSE, Jump Optimization</td>
<td>15</td>
</tr>
<tr>
<td>Loop Optimizations</td>
<td>Loop Invariant Movement, Peeling, Unswitching</td>
<td>7</td>
</tr>
<tr>
<td>Machine Dependent Optimizations</td>
<td>Register Allocation, Instruction Scheduling, Peephole Optimizations</td>
<td>59</td>
</tr>
<tr>
<td>Assembly Emission and Finishing</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td>Total number of passes on RTL</td>
<td></td>
<td>84</td>
</tr>
</tbody>
</table>
Loop Transformation Passes in GCC

NEXT_PASS (pass_tree_loop);
{
    struct opt_pass **p = &pass_tree_loop.pass.sub;
    NEXT_PASS (pass_tree_loop_init);
    NEXT_PASS (pass_copy_prop);
    NEXT_PASS (pass_dce_loop);
    NEXT_PASS (pass_lim);
    NEXT_PASS (pass_predcom);
    NEXT_PASS (pass_tree_unswitch);
    NEXT_PASS (pass_scev_cprop);
    NEXT_PASS (pass_empty_loop);
    NEXT_PASS (pass_record_bounds);
    NEXT_PASS (pass_check_data_deps);
    NEXT_PASS (pass_loop_distribution);
    NEXT_PASS (pass_linear_transform);
    NEXT_PASS (pass_graphite_transforms);
    NEXT_PASS (pass_if_canon);
    NEXT_PASS (pass_if_conversion);
    NEXT_PASS (pass_vectorize);
    {
        struct opt_pass **p = &pass_vectorize.pass.sub;
        NEXT_PASS (pass_lower_vector_ssa);
        NEXT_PASS (pass_dce_loop);
    }
    NEXT_PASS (pass_complete_unroll);
    NEXT_PASS (pass_parallelize_loops);
    NEXT_PASS (pass_loop_prefetch);
    NEXT_PASS (pass_iv_optimize);
    NEXT_PASS (pass_tree_loop_done);
}

- Passes on tree-SSA form
  A variant of Gimple IR
- Discover parallelism and transform IR
- Parameterized by some machine dependent features (Vectorization factor, alignment etc.)
- Mapping the transformed IR to machine instructions is achieved through machine descriptions
Loop Transformation Passes in GCC

NEXT_PASS (pass_tree_loop);
{
  struct opt_pass **p = &pass_tree_loop.pass.sub;
  NEXT_PASS (pass_tree_loop.init);
  NEXT_PASS (pass_copy_prop);
  NEXT_PASS (pass_dce_loop);
  NEXT_PASS (pass_lim);
  NEXT_PASS (pass_predcom);
  NEXT_PASS (pass_tree_unswitch);
  NEXT_PASS (pass_scev_cprop);
  NEXT_PASS (pass_empty_loop);
  NEXT_PASS (pass_record_bounds);
  NEXT_PASS (pass_check_data_deps);
  NEXT_PASS (pass_loop_distribution);
  NEXT_PASS (pass_linear_transform);
  NEXT_PASS (pass_graphite_transforms);
  NEXT_PASS (pass_iv_canon);
  NEXT_PASS (pass_if_conversion);
  NEXT_PASS (pass_vectorize);
    {
      struct opt_pass **p = &pass_vectorize.pass.sub;
      NEXT_PASS (pass_lower_vector_ssa);
      NEXT_PASS (pass_dce_loop);
    }
  NEXT_PASS (pass_complete_unroll);
  NEXT_PASS (pass_parallelize_loops);
  NEXT_PASS (pass_loop_prefetch);
  NEXT_PASS (pass_iv_optimize);
  NEXT_PASS (pass_tree_loop_done);
}

- Passes on tree-SSA form
- A variant of Gimple IR
- Discover parallelism and transform IR
- Parameterized by some machine dependent features (Vectorization factor, alignment etc.)
- Mapping the transformed IR to machine instructions is achieved through machine descriptions
Loop Transformation Passes in GCC

NEXT_PASS (pass_tree_loop);
{
    struct opt_pass **p = &pass_tree_loop.pass.sub;
    NEXT_PASS (pass_tree.loop.init);
    NEXT_PASS (pass_copy_prop);
    NEXT_PASS (pass_dce_loop);
    NEXT_PASS (pass_lim);
    NEXT_PASS (pass_predcom);
    NEXT_PASS (pass_tree.unswitch);
    NEXT_PASS (pass_scev_cprop);
    NEXT_PASS (pass_empty_loop);
    NEXT_PASS (pass_record_bounds);
    NEXT_PASS (pass_check_data_deps);
    NEXT_PASS (pass_loop_distribution);
    NEXT_PASS (pass_linear_transform);
    NEXT_PASS (pass_graphite_transforms);
    NEXT_PASS (pass_iv_canon);
    NEXT_PASS (pass_if_conversion);
    NEXT_PASS (pass_vectorize);
    {
        struct opt_pass **p = &pass_vectorize.pass.sub;
        NEXT_PASS (pass_lower_vector_ssa);
        NEXT_PASS (pass_dce_loop);
    }
    NEXT_PASS (pass_complete_unroll);
    NEXT_PASS (pass_parallelize_loops);
    NEXT_PASS (pass_loop_prefetch);
    NEXT_PASS (pass_iv_optimize);
    NEXT_PASS (pass_tree_loop_done);
}

• Passes on tree-SSA form
  A variant of Gimple IR
• Discover parallelism and transform IR
• Parameterized by some machine dependent features (Vectorization factor, alignment etc.)
• Mapping the transformed IR to machine instructions is achieved through machine descriptions
# Loop Transformation Passes in GCC: Our Focus

<table>
<thead>
<tr>
<th>Loop Transformation Passes</th>
<th>Pass variable name</th>
<th>Enabling switch</th>
<th>Dump switch</th>
<th>Dump file extension</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Dependence</strong></td>
<td>pass_check_data_deps</td>
<td>-fcheck-data-deps</td>
<td>-fdump-tree-ckdd</td>
<td>.ckdd</td>
</tr>
<tr>
<td><strong>Loop Distribution</strong></td>
<td>pass_loop_distribution</td>
<td>-ftree-loop-distribution</td>
<td>-fdump-tree-ldist</td>
<td>.ldist</td>
</tr>
<tr>
<td><strong>Vectorization</strong></td>
<td>pass_vectorize</td>
<td>-ftree-vectorize</td>
<td>-fdump-tree-vect</td>
<td>.vect</td>
</tr>
<tr>
<td><strong>Parallelization</strong></td>
<td>pass_parallelize_loops</td>
<td>-ftree-parallelize-loops=n</td>
<td>-fdump-tree-parloops</td>
<td>.parloops</td>
</tr>
</tbody>
</table>
Compiling for Emitting Dumps

- Other necessary command line switches
  - `-O3 -fdump-tree-all`
    - `-O3` enables `-ftree-vectorize`. Other flags must be enabled explicitly

- Processor related switches to enable transformations apart from analysis
  - `-mtune=pentium -msse4`

- Other useful options
  - Suffixing `-all` to all dump switches
  - `-S` to stop the compilation with assembly generation
  - `--verbose-asm` to see more detailed assembly dump
  - `-fno-predictive-commoning` to disable predictive commoning optimization
Example 1: Observing Data Dependence

Step 0: Compiling

```c
#include <stdio.h>
int a[200];
int main()
{
    int i, n;
    for (i=0; i<150; i++)
    {
        a[i] = a[i+1] + 2;
    }
    return 0;
}

gcc -fcheck-data-deps -fdump-tree-ckdd-all -O3 -S datadep.c
```
Example 1: Observing Data Dependence

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>#include &lt;stdio.h&gt;</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>int a[200];</td>
<td># i_13 = PHI &lt;i_4(4), 0(2)&gt;</td>
</tr>
<tr>
<td>int main()</td>
<td>i_4 = i_13 + 1;</td>
</tr>
<tr>
<td>{</td>
<td>D.1240_5 = a[i_4];</td>
</tr>
<tr>
<td>int i, n;</td>
<td>D.1241_6 = D.1240_5 + 2;</td>
</tr>
<tr>
<td>for (i=0; i&lt;150; i++)</td>
<td>a[i_13] = D.1241_6;</td>
</tr>
<tr>
<td>{</td>
<td>if (i_4 &lt;= 149)</td>
</tr>
<tr>
<td>a[i] = a[i+1] + 2;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>}</td>
<td>else</td>
</tr>
<tr>
<td>return 0;</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
</tbody>
</table>

<bb 4>: |
| goto <bb 3>; |
Example 1: Observing Data Dependence

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#include &lt;stdio.h&gt;</code></td>
<td><code>&lt;bb 3&gt;</code>:</td>
</tr>
<tr>
<td><code>int a[200];</code></td>
<td># i_13 = PHI \ i_4(4), 0(2) &gt;</td>
</tr>
<tr>
<td><code>int main()</code></td>
<td>i_4 = i_13 + 1;</td>
</tr>
<tr>
<td><code>{</code></td>
<td>D.1240_5 = a[i_4];</td>
</tr>
<tr>
<td><code>  int i, n;</code></td>
<td>D.1241_6 = D.1240_5 + 2;</td>
</tr>
<tr>
<td><code>  for (i=0; i&lt;150; i++)</code></td>
<td>a[i_13] = D.1241_6;</td>
</tr>
<tr>
<td><code>    {</code></td>
<td>if (i_4 &lt;= 149)</td>
</tr>
<tr>
<td><code>      a[i] = a[i+1] + 2;</code></td>
<td>\textbf{goto} &lt;bb 4&gt;;</td>
</tr>
<tr>
<td><code>    }</code></td>
<td>else</td>
</tr>
<tr>
<td><code>  return 0;</code></td>
<td>\textbf{goto} &lt;bb 5&gt;;</td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>&lt;bb 4&gt;</code>:</td>
</tr>
<tr>
<td></td>
<td><code>\textbf{goto} &lt;bb 3&gt;;</code></td>
</tr>
</tbody>
</table>
Example 1: Observing Data Dependence

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>#include &lt;stdio.h&gt;</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>int a[200];</td>
<td># i_13 = PHI &lt;i_4(4), 0(2)&gt;</td>
</tr>
<tr>
<td>int main()</td>
<td>i_4 = i_13 + 1;</td>
</tr>
<tr>
<td>{</td>
<td>D.1240_5 = a[i_4];</td>
</tr>
<tr>
<td>int i, n;</td>
<td>D.1241_6 = D.1240_5 + 2;</td>
</tr>
<tr>
<td>for (i=0; i&lt;150; i++)</td>
<td>a[i_13] = D.1241_6;</td>
</tr>
<tr>
<td>{</td>
<td>if (i_4 &lt;= 149)</td>
</tr>
<tr>
<td>a[i] = a[i+1] + 2;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>}</td>
<td>else</td>
</tr>
<tr>
<td>return 0;</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>}</td>
<td>&lt;bb 4&gt;:</td>
</tr>
</tbody>
</table>
|                                                                        | goto <bb 3>;
Example 1: Observing Data Dependence

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
</table>
| #include <stdio.h> int a[200]; int main() { int i, n; for (i=0; i<150; i++) { a[i] = a[i+1] + 2; } return 0; } | <bb 3>: 
  # i_13 = PHI <i_4(4), 0(2)> 
  i_4 = i_13 + 1; 
  D.1240_5 = a[i_4]; 
  D.1241_6 = D.1240_5 + 2; 
  a[i_13] = D.1241_6; 
  if (i_4 <= 149) goto <bb 4>; 
  else goto <bb 5>; 
  <bb 4>: goto <bb 3>;

Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:

# i_13 = PHI <i_4(4), 0(2)>

i_4 = i_13 + 1;
D.1240_5 = a[i_4];
D.1241_6 = D.1240_5 + 2;
a[i_13] = D.1241_6;
if (i_4 <= 149)
  goto <bb 4>;
else
  goto <bb 5>;

<bb 4>:
  goto <bb 3>;
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:

# i_13 = PHI <i_4(4), 0(2)>
i_4 = i_13 + 1;
D.1240_5 = a[i_4];
D.1241_6 = D.1240_5 + 2;
a[i_13] = D.1241_6;
if (i_4 <= 149)
    goto <bb 4>;
else
    goto <bb 5>;
<bb 4>:
    goto <bb 3>;

(evolution_function = 0, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:

# i_13 = PHI <i_4(4), 0(2)>

i_4 = i_13 + 1;
D.1240_5 = a[i_4];
D.1241_6 = D.1240_5 + 2;
a[i_13] = D.1241_6;
if (i_4 <= 149)
    goto <bb 4>;
else
    goto <bb 5>;
<bb 4>:

goto <bb 3>;

(scalar_evolution = 1, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:
    # i_13 = PHI <i_4(4), 0(2)>
    i_4 = i_13 + 1;
    D.1240_5 = a[i_4];
    D.1241_6 = D.1240_5 + 2;
    a[i_13] = D.1241_6;
    if (i_4 <= 149)
        goto <bb 4>;
    else
        goto <bb 5>;
<bb 4>:
    goto <bb 3>;

base_address: &a
offset from base address: 0
constant offset from base address: 4
aligned to: 128
(chrec = 1, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<i>bb 3</i>:
\[
\begin{align*}
# & \quad i_{13} = \text{PHI} \quad i_4(4), \ 0(2) \\
i_4 &= i_{13} + 1; \\
D.1240_5 &= a[i_4]; \\
D.1241_6 &= D.1240_5 + 2; \\
a[i_{13}] &= D.1241_6; \\
\text{if} \quad (i_4 \leq 149) \\
\quad \quad &\text{goto} \quad <bb \ 4>; \\
\text{else} \\
\quad \quad &\text{goto} \quad <bb \ 5>; \\
<bb \ 4>:\n\quad &\text{goto} \quad <bb \ 3>;
\end{align*}
\]

base_address: &a
offset from base address: 0
constant offset from base address: 0
aligned to: 128
base_object: a[0]
(chrec = 0, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:
  # i_13 = PHI <i_4(4), 0(2)>
  i_4 = i_13 + 1;
  D.1240_5 = a[i_4];
  D.1241_6 = D.1240_5 + 2;
  a[i_13] = D.1241_6;
  if (i_4 <= 149)
    goto <bb 4>;
  else
    goto <bb 5>;
<bb 4>:
  goto <bb 3>;
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:  
# i_13 = PHI <i_4(4), 0(2)>  
i_4 = i_13 + 1;  
D.1240_5 = a[i_4];  
D.1241_6 = D.1240_5 + 2;  
a[i_13] = D.1241_6;  
if (i_4 <= 149)  
goto <bb 4>;  
else  
goto <bb 5>;  
<bb 4>:  
goto <bb 3>;  

(evolution_function = 0, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:
   # i_13 = PHI <i_4(4), 0(2)>
   i_4 = i_13 + 1;
   D.1240_5 = a[i_4];
   D.1241_6 = D.1240_5 + 2;
   a[i_13] = D.1241_6;
   if (i_4 <= 149)
      goto <bb 4>;
   else
      goto <bb 5>;
<bb 4>:
   goto <bb 3>;

(scalar_evolution = 1, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:
  # i_13 = PHI <i_4(4), 0(2)>
  i_4 = i_13 + 1;
  D.1240_5 = a[i_4];
  D.1241_6 = D.1240_5 + 2;
  a[i_13] = D.1241_6;
  if (i_4 <= 149)
      goto <bb 4>;
  else
      goto <bb 5>;
<bb 4>:
  goto <bb 3>;

base_address: &a
offset from base address: 0
constant offset from base address: 4
aligned to: 128
(chrec = 1, +, 1_1)
Example 1: Observing Data Dependence

Step 2: Understanding the chain of recurrences

<bb 3>:

```c
# i_13 = PHI <i_4(4), 0(2)>
i_4 = i_13 + 1;
D.1240_5 = a[i_4];
D.1241_6 = D.1240_5 + 2;
a[i_13] = D.1241_6;
if (i_4 <= 149)
  goto <bb 4>;
else
  goto <bb 5>;
<bb 4>:
goto <bb 3>;
```

```c
base_address: &a
offset from base address: 0
constant offset from base address: 0
aligned to: 128
base_object: a[0]
(chrec = 0, +, 1_1)
```
Example 1: Observing Data Dependence

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant assignment is</td>
<td></td>
</tr>
<tr>
<td>$a[i] = a[i+1] + 2$</td>
<td></td>
</tr>
</tbody>
</table>
### Example 1: Observing Data Dependence

**Step 3: Understanding Banerjee’s test [Ban96]**

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is</td>
<td></td>
</tr>
<tr>
<td>( a[i] = a[i+1] + 2 )</td>
<td></td>
</tr>
<tr>
<td>• Solve for ( 0 \leq x, y &lt; 150 )</td>
<td></td>
</tr>
<tr>
<td>( y = x + 1 )</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: Observing Data Dependence

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is</td>
<td></td>
</tr>
<tr>
<td>(a[i] = a[i + 1] + 2)</td>
<td></td>
</tr>
<tr>
<td>• Solve for (0 \leq x, y &lt; 150)</td>
<td></td>
</tr>
<tr>
<td>(y = x + 1)</td>
<td></td>
</tr>
<tr>
<td>(\Rightarrow x - y + 1 = 0)</td>
<td></td>
</tr>
</tbody>
</table>
**Example 1: Observing Data Dependence**

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>\cdot Relevant assignment is (a[i] = a[i + 1] + 2)</td>
<td></td>
</tr>
<tr>
<td>\cdot Solve for (0 \leq x, y &lt; 150) (y = x + 1) (\Rightarrow x - y + 1 = 0)</td>
<td></td>
</tr>
<tr>
<td>\cdot Find min and max of LHS</td>
<td></td>
</tr>
</tbody>
</table>
Example 1: Observing Data Dependence

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is</td>
<td></td>
</tr>
<tr>
<td>$a[i] = a[i + 1] + 2$</td>
<td></td>
</tr>
<tr>
<td>• Solve for $0 \leq x, y &lt; 150$</td>
<td></td>
</tr>
<tr>
<td>$y = x + 1$</td>
<td></td>
</tr>
<tr>
<td>$\Rightarrow x - y + 1 = 0$</td>
<td></td>
</tr>
<tr>
<td>• Find min and max of LHS</td>
<td></td>
</tr>
<tr>
<td>$x - y + 1$</td>
<td></td>
</tr>
<tr>
<td>Min: -148 Max: +150</td>
<td></td>
</tr>
</tbody>
</table>
**Example 1: Observing Data Dependence**

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is ( a[i] = a[i+1] + 2 )</td>
<td></td>
</tr>
<tr>
<td>• Solve for ( 0 \leq x, y &lt; 150 )</td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{align*}
  y &= x + 1 \\
  \Rightarrow x - y + 1 &= 0
\end{align*}
\] | |
| • Find min and max of LHS \( x - y + 1 \) | |
| Min: -148 Max: +150 | |
| RHS belongs to \([-148, +150]\) and dependence may exist | |
## Example 1: Observing Data Dependence

### Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant assignment is</td>
<td>i₄ = i₃ + 1; D.1240₅ = a[i₄]; D.1241₆ = D.1240₅ + 2; a[i₃] = D.1241₆;</td>
</tr>
<tr>
<td>( a[i] = a[i + 1] + 2 )</td>
<td></td>
</tr>
<tr>
<td>Solve for ( 0 \leq x, y &lt; 150 )</td>
<td></td>
</tr>
<tr>
<td>( y = x + 1 )</td>
<td></td>
</tr>
<tr>
<td>( \Rightarrow x - y + 1 = 0 )</td>
<td></td>
</tr>
<tr>
<td>Find min and max of LHS</td>
<td></td>
</tr>
<tr>
<td>( x - y + 1 )</td>
<td></td>
</tr>
<tr>
<td>Min: -148 Max: +150</td>
<td></td>
</tr>
<tr>
<td>RHS belongs to ([-148, +150]) and dependence may exist</td>
<td></td>
</tr>
</tbody>
</table>
### Example 1: Observing Data Dependence

**Step 3: Understanding Banerjee’s test [Ban96]**

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is</td>
<td>• $i_4 = i_{13} + 1$;</td>
</tr>
<tr>
<td>$a[i] = a[i + 1] + 2$</td>
<td>D.1240.5 = $a[i_4]$;</td>
</tr>
<tr>
<td>• Solve for $0 \leq x, y &lt; 150$</td>
<td>D.1241.6 = D.1240.5 + 2;</td>
</tr>
<tr>
<td>$y = x + 1$</td>
<td>$a[i_{13}] = D.1241.6$;</td>
</tr>
<tr>
<td>$\Rightarrow x - y + 1 = 0$</td>
<td>• Chain of recurrences are</td>
</tr>
<tr>
<td>• Find min and max of LHS</td>
<td>For $a[i_4]$: ${1, +, 1}_1$</td>
</tr>
<tr>
<td>$x - y + 1$</td>
<td>For $a[i_{13}]$: ${0, +, 1}_1$</td>
</tr>
<tr>
<td>Min: $-148$</td>
<td>• RHS belongs to $[-148, +150]$ and dependence may exist</td>
</tr>
<tr>
<td>Max: $+150$</td>
<td>RHS belongs to $[-148, +150]$ and dependence may exist</td>
</tr>
</tbody>
</table>
### Example 1: Observing Data Dependence

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
</table>
| • Relevant assignment is  
  \[ a[i] = a[i + 1] + 2 \]  
• Solve for \( 0 \leq x, y < 150 \)  
  \[ y = x + 1 \]  
  \[ x - y + 1 = 0 \]  
• Find min and max of LHS  
  \[ x - y + 1 \]  
  \[ \text{Min: } -148 \quad \text{Max: } +150 \]  
  RHS belongs to \([-148, +150]\) and dependence may exist | • \( i_4 = i_{13} + 1; \)  
  \( D.1240.5 = a[i_4]; \)  
  \( D.1241.6 = D.1240.5 + 2; \)  
  \( a[i_{13}] = D.1241.6; \)  
• Chain of recurrences are  
  For \( a[i_4]: \) \( \{1, +, 1\}_1 \)  
  For \( a[i_{13}]: \) \( \{0, +, 1\}_1 \)  
• Solve for \( 0 \leq x_{-1} < 150 \)  
  \[ 1 + 1 \times x_{-1} - 0 + 1 \times x_{-1} = 0 \] |
Example 1: Observing Data Dependence

Step 3: Understanding Banerjee’s test [Ban96]

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is $a[i] = a[i + 1] + 2$</td>
<td>• $i_4 = i_{13} + 1$; $D.1240_5 = a[i_4]$; $D.1241_6 = D.1240_5 + 2$; $a[i_{13}] = D.1241_6$;</td>
</tr>
<tr>
<td>• Solve for $0 \leq x, y &lt; 150$</td>
<td>• Chain of recurrences are</td>
</tr>
<tr>
<td>$y = x + 1$</td>
<td>For $a[i_4]$: ${1, +, 1}_1$</td>
</tr>
<tr>
<td>$\Rightarrow x - y + 1 = 0$</td>
<td>For $a[i_{13}]$: ${0, +, 1}_1$</td>
</tr>
<tr>
<td>• Find min and max of LHS</td>
<td>• Solve for $0 \leq x_{-1} &lt; 150$</td>
</tr>
<tr>
<td>$x - y + 1$</td>
<td>$1 + 1<em>x_{-1} - 0 + 1</em>x_{-1} = 0$</td>
</tr>
<tr>
<td>Min: -148</td>
<td>• Min of LHS is -148, Max is +150</td>
</tr>
<tr>
<td>Max: +150</td>
<td></td>
</tr>
</tbody>
</table>

RHS belongs to $[-148, +150]$ and dependence may exist
### Example 1: Observing Data Dependence

**Step 3: Understanding Banerjee’s test [Ban96]**

<table>
<thead>
<tr>
<th>Source View</th>
<th>CFG View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Relevant assignment is</td>
<td></td>
</tr>
<tr>
<td>( a[i] = a[i + 1] + 2 )</td>
<td>• ( i_4 = i_{13} + 1; )</td>
</tr>
<tr>
<td>• Solve for ( 0 \leq x, y &lt; 150 )</td>
<td></td>
</tr>
<tr>
<td>( y = x + 1 )</td>
<td>( D.1240_5 = a[i_4]; )</td>
</tr>
<tr>
<td>( y = x + 1 )</td>
<td>( D.1241_6 = D.1240_5 + 2; )</td>
</tr>
<tr>
<td>( x - y + 1 = 0 )</td>
<td>( a[i_{13}] = D.1241_6; )</td>
</tr>
<tr>
<td>• Find min and max of LHS</td>
<td>• Chain of recurrences are</td>
</tr>
<tr>
<td>( x - y + 1 )</td>
<td>For ( a[i_4]: {1, +, 1}_1 )</td>
</tr>
<tr>
<td>Min: (-148)</td>
<td>For ( a[i_{13}]: {0, +, 1}_1 )</td>
</tr>
<tr>
<td>Max: (+150)</td>
<td>• Solve for ( 0 \leq x_1 &lt; 150 )</td>
</tr>
<tr>
<td></td>
<td>( 1 + 1\times x_1 - 0 + 1\times x_1 = 0 )</td>
</tr>
<tr>
<td></td>
<td>• Min of LHS is (-148), Max is (+150)</td>
</tr>
<tr>
<td></td>
<td>• Dependence may exist</td>
</tr>
</tbody>
</table>

RHS belongs to \([-148, +150]\) and dependence may exist
Example 2: Observing Vectorization and Parallelization

Step 0: Compiling with `-fno-predictive-commoning`

```c
int a[256], b[256];
int main()
{
    int i;
    for (i=0; i<256; i++)
    {
        a[i] = b[i];
    }
    return 0;
}
```

- Additional options for parallelization
  - `ftree-parallelize-loops=4`  `-fdump-tree-parloops-all`
- Additional options for vectorization
  - `fdump-tree-vect-all`  `-msse4`
Example 2: Observing Vectorization and Parallelization

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
</table>
| int a[256], b[256]; int main() {
    int i;
    for (i=0; i<256; i++) {
        a[i] = b[i];
    }
    return 0; } | <bb 3>:
    # i_14 = PHI <i_6(4), 0(2)>
    D.1666_5 = b[i_14];
    a[i_14] = D.1666_5;
    i_6 = i_14 + 1;
    if (i_6 <= 255)
        goto <bb 4>;
    else
        goto <bb 5>;
|
Example 2: Observing Vectorization and Parallelization

Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a[256], b[256]; int main()</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>{</td>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
</tr>
<tr>
<td>int i;</td>
<td>D.1666.5 = b[i_14];</td>
</tr>
<tr>
<td>for (i=0; i&lt;256; i++)</td>
<td>a[i_14] = D.1666.5;</td>
</tr>
<tr>
<td>{</td>
<td>i_6 = i_14 + 1;</td>
</tr>
<tr>
<td>a[i] = b[i];</td>
<td>if (i_6 &lt;= 255)</td>
</tr>
<tr>
<td>}</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>return 0;</td>
<td>else</td>
</tr>
<tr>
<td>}</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td></td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
**Example 2: Observing Vectorization and Parallelization**

### Step 1: Examining the control flow graph

<table>
<thead>
<tr>
<th>Program</th>
<th>Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a[256], b[256];</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>int main()</td>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
</tr>
<tr>
<td>{</td>
<td>D.1666_5 = b[i_14];</td>
</tr>
<tr>
<td>int i;</td>
<td>a[i_14] = D.1666_5;</td>
</tr>
<tr>
<td>for (i=0; i&lt;256; i++)</td>
<td>i_6 = i_14 + 1;</td>
</tr>
<tr>
<td>{</td>
<td>if (i_6 &lt;= 255)</td>
</tr>
<tr>
<td>a[i] = b[i];</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>}</td>
<td>else</td>
</tr>
<tr>
<td>return 0;</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>}</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td></td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Step 2: Observing the final decision about vectorization

parvec.c:9: note: LOOP VECTORIZED.
parvec.c:6: note: vectorized 1 loops in function.
### Example 2: Observing Vectorization and Parallelization

**Step 3: Examining the vectorized control flow graph**

<table>
<thead>
<tr>
<th>Original control flow graph</th>
<th>Transformed control flow graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>...</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>vect_var_.31_18 = *vect_pb.25_16;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>*vect_pa.32_21 = vect_var_.31_18;</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>vect_pb.25_17 = vect_pb.25_16 + 16;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>vect_pa.32_22 = vect_pa.32_21 + 16;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>ivtmp.38_24 = ivtmp.38_23 + 1;</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>if (ivtmp.38_24 &lt; 64)</td>
</tr>
<tr>
<td>else</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>else</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>...</td>
</tr>
</tbody>
</table>
### Example 2: Observing Vectorization and Parallelization

**Step 3: Examining the vectorized control flow graph**

<table>
<thead>
<tr>
<th>Original control flow graph</th>
<th>Transformed control flow graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bb 3&gt;</code>:</td>
<td><code>...</code></td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td><code>vect_var_.31_18 = *vect_pb.25_16;</code></td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td><code>*vect_pa.32_21 = vect_var_.31_18;</code></td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td><code>vect_pb.25_17 = vect_pb.25_16 + 16;</code></td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td><code>vect_pa.32_22 = vect_pa.32_21 + 16;</code></td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td><code>ivtmp.38_24 = ivtmp.38_23 + 1;</code></td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td><code>if (ivtmp.38_24 &lt; 64)</code></td>
</tr>
<tr>
<td>else</td>
<td><code>  goto &lt;bb 4&gt;;</code></td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td><code>else</code></td>
</tr>
<tr>
<td><code>&lt;bb 4&gt;</code>:</td>
<td><code>  goto &lt;bb 5&gt;;</code></td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td><code>...</code></td>
</tr>
</tbody>
</table>
Example 2: Observing Vectorization and Parallelization

Step 3: Examining the vectorized control flow graph

Original control flow graph

```
<bb 3>:
    # i_14 = PHI <i_6(4), 0(2)>
    D.1666_5 = b[i_14];
    a[i_14] = D.1666_5;
    i_6 = i_14 + 1;
    if (i_6 <= 255)
        goto <bb 4>;
    else
        goto <bb 5>;
<bb 4>:
    goto <bb 3>;
```

Transformed control flow graph

```
... vect_var_.31_18 = *vect_pb.25_16;
    *vect_pa.32_21 = vect_var_.31_18;
    vect_pb.25_17 = vect_pb.25_16 + 16;
    vect_pa.32_22 = vect_pa.32_21 + 16;
    ivtmp.38_24 = ivtmp.38_23 + 1;
    if (ivtmp.38_24 < 64)
        goto <bb 4>;
    else
        goto <bb 5>;
...```
Example 2: Observing Vectorization and Parallelization

Step 3: Examining the vectorized control flow graph

<table>
<thead>
<tr>
<th>Original control flow graph</th>
<th>Transformed control flow graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &lt;bb\ 3&gt;:)</td>
<td>( \ldots )</td>
</tr>
<tr>
<td># i_{14} = PHI &lt;i_{6}(4), 0(2)&gt;</td>
<td>vect_{var}.31.18 = \ast vect_{pb}.25.16;</td>
</tr>
<tr>
<td>D.1666.5 = b[i_{14}];</td>
<td>\ast vect_{pa}.32.21 = vect_{var}.31.18;</td>
</tr>
<tr>
<td>a[i_{14}] = D.1666.5;</td>
<td>vect_{pb}.25.17 = vect_{pb}.25.16 + 16;</td>
</tr>
<tr>
<td>i_{6} = i_{14} + 1;</td>
<td>vect_{pa}.32.22 = vect_{pa}.32.21 + 16;</td>
</tr>
<tr>
<td>if ( (i_{6} \leq 255) )</td>
<td>ivtmp.38.24 = ivtmp.38.23 + 1;</td>
</tr>
<tr>
<td>\quad goto &lt;bb 4&gt;;</td>
<td>if ( (ivtmp.38.24 &lt; 64) )</td>
</tr>
<tr>
<td>else</td>
<td>\quad goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>\quad goto &lt;bb 5&gt;;</td>
<td>else</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>\quad goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>\quad goto &lt;bb 3&gt;;</td>
<td>\ldots</td>
</tr>
</tbody>
</table>
Example 2: Observing Vectorization and Parallelization

Step 3: Examining the vectorized control flow graph

<table>
<thead>
<tr>
<th>Original control flow graph</th>
<th>Transformed control flow graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>...</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>vect_var_.31_18 = *vect_pb.25_16;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>*vect_pa.32_21 = vect_var_.31_18;</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>vect_pb.25_17 = vect_pb.25_16 + 16;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>vect_pa.32_22 = vect_pa.32_21 + 16;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>ivtmp.38_24 = ivtmp.38_23 + 1;</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>if (ivtmp.38_24 &lt; 64)</td>
</tr>
<tr>
<td>else</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>else</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>...</td>
</tr>
</tbody>
</table>
Example 2: Observing Vectorization and Parallelization

Step 4: Understanding the strategy of parallel execution

- Create threads $t_i$ for $1 \leq i \leq \text{MAX\_THREADS}$
- Assigning start and end iteration for each thread
  $\Rightarrow$ Distribute iteration space across all threads
Example 2: Observing Vectorization and Parallelization

Step 4: Understanding the strategy of parallel execution

- Create threads $t_i$ for $1 \leq i \leq \text{MAX\_THREADS}$
- Assigning start and end iteration for each thread
  $\Rightarrow$ Distribute iteration space across all threads
- Create the following code body for each thread $t_i$

  ```c
  for (j=start_for_thread_i; j<=end_for_thread_i; j++)
  {
    /* execute the loop body to be parallelized */
  }
  ```
PPoPP’10

123/147

GCC-Par: Parallelization and Vectorization in GCC

Example 2: Observing Vectorization and Parallelization

Step 4: Understanding the strategy of parallel execution
• Create threads ti for 1 ≤ i ≤ MAX THREADS
• Assigning start and end iteration for each thread

⇒ Distribute iteration space across all threads
• Create the following code body for each thread ti

for (j=start_for_thread_i; j<=end_for_thread_i; j++)
{
/* execute the loop body to be parallelized */
}
• All threads are executed in parallel

UPK,SB,PR

GRC, IIT Bombay


Step 5: Examining the thread creation in parallelized control flow graph

```
D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
    goto <bb 3>;
```
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

```
D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
  goto <bb 3>;
```

Get the number of threads
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

```
D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
    goto <bb 3>;
```

Get thread identity
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

D.1299\_7 = \texttt{\_\_builtin\_omp\_get\_num\_threads}();
D.1300\_9 = \texttt{\_\_builtin\_omp\_get\_thread\_num}();
D.1302\_10 = 255 / D.1299\_7;
D.1303\_11 = D.1302\_10 * D.1299\_7;
D.1304\_12 = D.1303\_11 != 255;
D.1305\_13 = D.1304\_12 + D.1302\_10;
ivtmp.28\_14 = D.1305\_13 * D.1300\_9;
D.1307\_15 = ivtmp.28\_14 + D.1305\_13;
D.1308\_16 = \texttt{MIN\_EXPR} <D.1307\_15, 255>;
if (ivtmp.28\_14 >= D.1308\_16)
goto <bb 3>;

Perform load calculations
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

```c
D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
  goto <bb 3>;
```

Assign start iteration to the chosen thread
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
    goto <bb 3>;

Assign end iteration to the chosen thread
Example 2: Observing Vectorization and Parallelization

Step 5: Examining the thread creation in parallelized control flow graph

D.1299_7 = __builtin_omp_get_num_threads ();
D.1300_9 = __builtin_omp_get_thread_num ();
D.1302_10 = 255 / D.1299_7;
D.1303_11 = D.1302_10 * D.1299_7;
D.1304_12 = D.1303_11 != 255;
D.1305_13 = D.1304_12 + D.1302_10;
ivtmp.28_14 = D.1305_13 * D.1300_9;
D.1307_15 = ivtmp.28_14 + D.1305_13;
D.1308_16 = MIN_EXPR <D.1307_15, 255>;
if (ivtmp.28_14 >= D.1308_16)
    goto <bb 3>;

Start execution of iterations of the chosen thread
Example 2: Observing Vectorization and Parallelization

Step 6: Examining the loop body to be executed by a thread

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Parallel loop body</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>i.29_21 = (int) ivtmp.28_18;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>D.1312_23 = (*b.31_4)[i.29_21];</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>(*a.32_5)[i.29_21] = D.1312_23;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>ivtmp.28_19 = ivtmp.28_18 + 1;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>if (D.1308_16 &gt; ivtmp.28_19)</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Step 6: Examining the loop body to be executed by a thread

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Parallel loop body</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td># i_14 = PHI i_6(4), 0(2)</td>
<td>i.29_21 = (int) ivtmp.28_18;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>D.1312_23 = (*b.31_4)[i.29_21];</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>(*a.32_5)[i.29_21] = D.1312_23;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>ivtmp.28_19 = ivtmp.28_18 + 1;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>if (D.1308_16 &gt; ivtmp.28_19)</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Example 2: Observing Vectorization and Parallelization

Step 6: Examining the loop body to be executed by a thread

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Parallel loop body</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bb 3&gt;</code>:</td>
<td><code>&lt;bb 4&gt;</code>:</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>i.29_21 = (int) ivtmp.28_18;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>D.1312_23 = (*b.31_4)[i.29_21];</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>(*a.32_5)[i.29_21] = D.1312_23;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>ivtmp.28_19 = ivtmp.28_18 + 1;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>if (D.1308_16 &gt; ivtmp.28_19)</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td><code>&lt;bb 4&gt;</code>:</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
**Example 2: Observing Vectorization and Parallelization**

Step 6: Examining the loop body to be executed by a thread

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Parallel loop body</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>i.29_21 = (int) ivtmp.28_18;</td>
</tr>
<tr>
<td>D.1666.5 = b[i_14];</td>
<td>D.1312.23 = (*b.31_4)[i.29_21];</td>
</tr>
<tr>
<td>a[i_14] = D.1666.5;</td>
<td>(*a.32_5)[i.29_21] = D.1312.23;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>ivtmp.28_19 = ivtmp.28_18 + 1;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>if (D.1308_16 &gt; ivtmp.28_19)</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Example 2: Observing Vectorization and Parallelization

Step 6: Examining the loop body to be executed by a thread

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Parallel loop body</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>&lt;bb 4&gt;:</td>
</tr>
<tr>
<td># i_14 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>i.29_21 = (int) ivtmp.28_18;</td>
</tr>
<tr>
<td>D.1666_5 = b[i_14];</td>
<td>D.1312_23 = (*b.31_4)[i.29_21];</td>
</tr>
<tr>
<td>a[i_14] = D.1666_5;</td>
<td>(*a.32_5)[i.29_21] = D.1312_23;</td>
</tr>
<tr>
<td>i_6 = i_14 + 1;</td>
<td>ivtmp.28_19 = ivtmp.28_18 + 1;</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>if (D.1308_16 &gt; ivtmp.28_19)</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>else</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Example 3: Vectorization but No Parallelization

Step 0: Compiling with
-fno-predictive-commoning -fdump-tree-vect-all -msse4

```c
int a[256];
int main()
{
    int i;
    for (i=0; i<256; i++)
    {
        a[i] = a[i+4];
    }
    return 0;
}
```
Example 3: Vectorization but No Parallelization

Step 1: Observing the final decision about vectorization

vecnopar.c:8: note: LOOP VECTORIZED.
vecnopar.c:5: note: vectorized 1 loops in function.
Example 3: Vectorization but No Parallelization

Step 2: Examining vectorization

Control Flow Graph

```
<bb 3>:
  # i_13 = PHI <i_6(4), 0(2)>
  D.1665_4 = i_13 + 4;
  D.1666_5 = a[D.1665_4];
  a[i_13] = D.1666_5;
  i_6 = i_13 + 1;
  if (i_6 <= 255)
    goto <bb 4>;
  else
    goto <bb 5>;
```

Vectorized Control Flow Graph

```
a.31_11 = (vector int *) &a;
vect_pa.30_15 = a.31_11 + 16;
vect_pa.25_16 = vect_pa.30_15;
vect_pa.38_20 = (vector int *) &a;
vect_pa.33_21 = vect_pa.38_20;

<bb 3>:
  vect_var_.32_19 = *vect_pa.25_17;
  *vect_pa.33_22 = vect_var_.32_19;
  vect_pa.25_18 = vect_pa.25_17 + 16;
  vect_pa.33_23 = vect_pa.33_22 + 16;
  ivtmp.39_25 = ivtmp.39_24 + 1;
  if (ivtmp.39_25 < 64)
    goto <bb 4>;
```

UPK, SB, PR

GRC, IIT Bombay
### Example 3: Vectorization but No Parallelization

#### Step 2: Examining vectorization

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Vectorized Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;bb 3&gt;</code>:</td>
<td><code>a.31_11 = (vector int *) &amp;a;</code></td>
</tr>
<tr>
<td></td>
<td><code>vect_pa.30_15 = a.31_11 + 16;</code></td>
</tr>
<tr>
<td></td>
<td><code>vect_pa.25_16 = vect_pa.30_15;</code></td>
</tr>
<tr>
<td></td>
<td><code>vect_pa.38_20 = (vector int *) &amp;a;</code></td>
</tr>
<tr>
<td></td>
<td><code>vect_pa.33_21 = vect_pa.38_20;</code></td>
</tr>
<tr>
<td># i_13 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td><code>&lt;bb 3&gt;</code>:</td>
</tr>
<tr>
<td>D.1665_4 = i_13 + 4;</td>
<td><code>vect_var_.32_19 = *vect_pa.25_17;</code></td>
</tr>
<tr>
<td>D.1666_5 = a[D.1665_4];</td>
<td><code>*vect_pa.33_22 = vect_var_.32_19;</code></td>
</tr>
<tr>
<td>a[i_13] = D.1666_5;</td>
<td><code>vect_pa.25_18 = vect_pa.25_17 + 16;</code></td>
</tr>
<tr>
<td>i_6 = i_13 + 1;</td>
<td><code>vect_pa.33_23 = vect_pa.33_22 + 16;</code></td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td><code>ivtmp.39_25 = ivtmp.39_24 + 1;</code></td>
</tr>
<tr>
<td></td>
<td>if (ivtmp.39_25 &lt; 64)</td>
</tr>
<tr>
<td></td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
<tr>
<td>else</td>
<td>goto &lt;bb 5&gt;;</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>goto &lt;bb 3&gt;;</td>
</tr>
</tbody>
</table>
Example 3: Vectorization but No Parallelization

Step 2: Examining vectorization

Control Flow Graph

<bb 3>:
# i_13 = PHI <i_6(4), 0(2)>
D.1665_4 = i_13 + 4;
D.1666_5 = a[D.1665_4];
a[i_13] = D.1666_5;
i_6 = i_13 + 1;
if (i_6 <= 255)
  goto <bb 4>;
else
  goto <bb 5>;

<bb 4>:
  goto <bb 3>;

Vectorized Control Flow Graph

a.31_11 = (vector int *) &a;
vect_pa.30_15 = a.31_11 + 16;
vect_pa.25_16 = vect_pa.30_15;
vect_pa.38_20 = (vector int *) &a;
vect_pa.33_21 = vect_pa.38_20;

<bb 3>:
vect_var_.32_19 = *vect_pa.25_17;
*vect_pa.33_22 = vect_var_.32_19;
vect_pa.25_18 = vect_pa.25_17 + 16;
vect_pa.33_23 = vect_pa.33_22 + 16;
ivtmp.39_25 = ivtmp.39_24 + 1;
if (ivtmp.39_25 < 64)
  goto <bb 4>;}
Example 3: Vectorization but No Parallelization

Step 2: Examining vectorization

Control Flow Graph

```
<bb 3>:
  # i_13 = PHI <i_6(4), 0(2)>
  D.1665_4 = i_13 + 4;
  D.1666_5 = a[D.1665_4];
  a[i_13] = D.1666_5;
  i_6 = i_13 + 1;
  if (i_6 <= 255)
    goto <bb 4>;
  else
    goto <bb 5>;

<bb 4>:
  goto <bb 3>;
```

Vectorized Control Flow Graph

```
a.31_11 = (vector int *) &a;
vect_pa.30_15 = a.31_11 + 16;
vect_pa.25_16 = vect_pa.30_15;
vect_pa.38_20 = (vector int *) &a;
vect_pa.33_21 = vect_pa.38_20;

<bb 3>:
  vect_var_.32_19 = *vect_pa.25_17;
  *vect_pa.33_22 = vect_var_.32_19;
  vect_pa.25_18 = vect_pa.25_17 + 16;
  vect_pa.33_23 = vect_pa.33_22 + 16;
  ivtmp.39_25 = ivtmp.39_24 + 1;
  if (ivtmp.39_25 < 64)
    goto <bb 4>;
```
**Example 3: Vectorization but No Parallelization**

**Step 2: Examining vectorization**

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Vectorized Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>a.31_11 = (vector int *) &amp;a;</td>
</tr>
<tr>
<td># i_13 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>vect_pa.30_15 = a.31_11 + 16;</td>
</tr>
<tr>
<td>D.1665_4 = i_13 + 4;</td>
<td>vect_pa.25_16 = vect_pa.30_15;</td>
</tr>
<tr>
<td>D.1666_5 = a[D.1665_4];</td>
<td>vect_pa.38_20 = (vector int *) &amp;a;</td>
</tr>
<tr>
<td>a[i_13] = D.1666_5;</td>
<td>vect_pa.33_21 = vect_pa.38_20;</td>
</tr>
<tr>
<td>i_6 = i_13 + 1;</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>vect_var__.32_19 = *vect_pa.25_17;</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>*vect_pa.33_22 = vect_var__.32_19;</td>
</tr>
<tr>
<td>else</td>
<td>vect_pa.25_18 = vect_pa.25_17 + 16;</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>vect_pa.33_23 = vect_pa.33_22 + 16;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>ivtmp.39_25 = ivtmp.39_24 + 1;</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>if (ivtmp.39_25 &lt; 64)</td>
</tr>
<tr>
<td></td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
</tbody>
</table>
**Example 3: Vectorization but No Parallelization**

Step 2: Examining vectorization

<table>
<thead>
<tr>
<th>Control Flow Graph</th>
<th>Vectorized Control Flow Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bb 3&gt;:</td>
<td>a.31_11 = (vector int *) &amp;a;</td>
</tr>
<tr>
<td># i_13 = PHI &lt;i_6(4), 0(2)&gt;</td>
<td>vect_pa.30_15 = a.31_11 + 16;</td>
</tr>
<tr>
<td>D.1665_4 = i_13 + 4;</td>
<td>vect_pa.25_16 = vect_pa.30_15;</td>
</tr>
<tr>
<td>D.1666_5 = a[D.1665_4];</td>
<td>vect_pa.38_20 = (vector int *) &amp;a;</td>
</tr>
<tr>
<td>a[i_13] = D.1666_5;</td>
<td>vect_pa.33_21 = vect_pa.38_20;</td>
</tr>
<tr>
<td>i_6 = i_13 + 1;</td>
<td>&lt;bb 3&gt;:</td>
</tr>
<tr>
<td>if (i_6 &lt;= 255)</td>
<td>vect_var_.32_19 = *vect_pa.25_17;</td>
</tr>
<tr>
<td>goto &lt;bb 4&gt;;</td>
<td>*vect_pa.33_22 = vect_var_.32_19;</td>
</tr>
<tr>
<td>else</td>
<td>vect_pa.25_18 = vect_pa.25_17 + 16;</td>
</tr>
<tr>
<td>goto &lt;bb 5&gt;;</td>
<td>vect_pa.33_23 = vect_pa.33_22 + 16;</td>
</tr>
<tr>
<td>&lt;bb 4&gt;:</td>
<td>ivtmp.39_25 = ivtmp.39_24 + 1;</td>
</tr>
<tr>
<td>goto &lt;bb 3&gt;;</td>
<td>if (ivtmp.39_25 &lt; 64)</td>
</tr>
<tr>
<td></td>
<td>goto &lt;bb 4&gt;;</td>
</tr>
</tbody>
</table>
Example 3: Vectorization but No Parallelization

- Step 3: Observing the conclusion about dependence information

  inner loop index: 0  
  loop nest: (1 )  
  distance_vector:  4  
  direction_vector:  +

- Step 4: Observing the final decision about parallelization

  FAILED: data dependencies exist across iterations
Example 4: No Vectorization and No Parallelization

Step 0: Compiling with `-fno-predictive-commoning`

```c
int a[256], b[256];
int main ()
{
    int i;
    for (i=0; i<256; i++)
    {
        a[i+2] = b[i] + 5;
        b[i+3] = a[i] + 10;
    }
    return 0;
}
```

- Additional options for parallelization
  `-ftree-parallelize-loops=4 -fdump-tree-parloops-all`
- Additional options for vectorization
  `-fdump-tree-vect-all -msse4`
Example 4: No Vectorization and No Parallelization

- Step 1: Observing the final decision about vectorization
  
  `noparvec.c:5: note: vectorized 0 loops in function.`

- Step 2: Observing the final decision about parallelization
  
  `FAILED: data dependencies exist across iterations`
Example 4: No Vectorization and No Parallelization

Step 3: Understanding the dependencies that prohibit vectorization and parallelization

\[ a[i+2] = b[i] + 5 \]

\[ \delta_1 \]

\[ b[i+3] = a[i] + 10 \]
GCC Parallelization and Vectorization: Conclusions

- Chain of recurrences seems to be a useful generalization
- Data dependence information is not stored across passes
- Interaction between different transformations is not clear
  Predictive commoning and SSA seem to prohibit many opportunities
- Scalar dependences are not reported. Not clear if they are computed
- May report dependence where there is none

Other passes need to be studied to arrive at a better judgement
References

Utpal K. Banerjee.  
*Dependence Analysis.*  

Ken Kennedy and John R. Allen.  
*Optimizing Compilers for Modern Architectures: A Dependence-Based Approach.*  
References

Olaf Bachmann, Paul S. Wang, and Eugene V. Zima.
Chains of recurrences - a method to expedite the evaluation of closed-form functions.

V. Kislenkov, V. Mitrofanov, and E. Zima.
Multidimensional chains of recurrences.
Part 7

GCC Resource Center
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
Broad Research Goals of GCC Resource Center

• Using GCC as a means
  ▶ Adding new optimizations to GCC
  ▶ Adding flow and context sensitive whole program analyses to GCC
    (In particular, pointer analysis)

• Using GCC as an end in itself
  ▶ Changing the retargetability mechanism of GCC
  ▶ Cleaning up the machine descriptions of GCC
  ▶ Facilitating specification driven optimizations
  ▶ Improving vectorization/parallelization in GCC
# 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 (experiences 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 (experiences 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

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
Progress on Research Work

- Released GDFA (Generic Data Flow Analyser)
  Currently: Intraprocedural data flow analysis for any bit vector framework
- Identified exact changes required in the machine descriptions and instruction selection mechanism
  New constructs in machine descriptions
  May reduce the size of specifications by about 50%
This workshop is a 3-day instructional workshop (and not a forum for contributed presentations) and involves lectures and laboratory exercises aimed at providing details of the internals of GCC which is an acronym for GNU Compiler Collection. It is the de-facto standard compiler generation framework on GNU/Linux and many variants of Unix. In the last 20 years of its existence, it has seen a rapid growth and wide acceptability.

**Take-aways from the Workshop**

After attending this workshop

- A teacher of compiler construction will be able to take examples of real compilation processes to illustrate the difference phases of compilation
- A compiler developer wanting to retarget GCC to a new machine will know how to write machine descriptions systematically
- A researcher exploring retargetable compilation will be exposed to real issues in an industry strength compiler
- A researcher exploring machine independent optimizations will be able to add data flow analysis based optimization passes to GCC
- A software engineer will be exposed to the architecture of a very large and very successful software

**Who should attend this workshop?**

Anybody who has done at least a first level undergraduate course in compiler construction and has some experience of either working in compilers or teaching compilers. A sound understanding of the process of compilation is a must. Familiarity with Unix/Linux (particularly, the command line style of working) is absolutely necessary.

**About GCC**

GCC, an acronym for GNU Compiler Collection, is a compiler generation framework which generates production quality optimizing compilers from descriptions of target platforms. It follows an open development model whereby its source is available for all for inspection and modification. It supports a wide variety of source languages and target machines (including operating system specific variants) in a ready-to-deploy form. Besides, new machines can be added by describing instruction set architectures and some other information (eg. calling conventions).

Novices may want to see the Wikipedia introduction to GCC. For experts, the GCC page contains a wealth of information including installation instructions, reference manuals (which include users' guides as well as details of GCC internals), a set of frequently asked questions, a wiki page for
July 2009 Workshop

- Conduct of the workshop:
  - Number of lecture sessions: 12
  - Number of lab sessions: 7
  - Assignments were done in groups of 2
  - Number of lab TAs: 15

- Participants:
  - External Participants: 60
  - Participants from private industry: 34
    (KPIT, ACME, HP, Siemens, Google, Bombardier, Morgan Stanley, AMD, Mercelloworld, Selec, Synopsys)
  - Participants from academia: 16
  - Participants from govt. research org.: 10
    (NPCIL, VSSC)
Part 8

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

- The availability of source code and the availability of dumps makes GCC a very useful case study for compiler researchers.
Conclusions

 GCC Resource Center at IIT Bombay

- **Our Goals**
  - Demystifying GCC
  - A dream to improve GCC
  - Spreading GCC know-how

- **Our Strength**
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers

- **On the horizon**
  - Enhancements to data flow analyser
  - Overall re-design of instruction selection mechanism
Last but not the least . . .

Thank You!