Improving the Gnu Compiler Collection

Uday Khedker

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

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

October 2009

Outline

- Introduction and motivation
- Understanding GCC
- Improving machine independent optimizations in GCC
- Improving machine descriptions and intruction selection in GCC

Part 1

Introduction and Motivation

Why GCC?

- An Informal Group

- A Desire

Why GCC?

- An Informal Group
  - CSE faculty members at IITB: Uday Khedker
    Amitabha Sanyal
    Supratim Biswas

- A Desire

Why GCC?

- An Informal Group
  - CSE faculty members at IITB: Uday Khedker
    Amitabha Sanyal
    Supratim Biswas
  - Reasonably long and deep experience of research in compilers
- A Desire

Why GCC?

- An Informal Group
  - CSE faculty members at IITB: Uday Khedker
    Amitabha Sanyal
    Supratim Biswas
  - Reasonably long and deep experience of research in compilers

- A Desire

  Performing research grounded in theory and corroborated by empirical evidence

Why GCC?

- An Informal Group
  - CSE faculty members at IITB: Uday Khedker
    Amitabha Sanyal
    Supratim Biswas
  - Reasonably long and deep experience of research in compilers

- A Desire

  Performing research grounded in theory and corroborated by empirical evidence
  - Exploring research issues in real compilers
  - Demonstrating the relevance and effectiveness of our research in real compilers

A Modest Start in 2003...

- Our Tool of Experiment

- Our Guinea Pigs

A Modest Start in 2003...

- Our Tool of Experiment The Gnu Compiler Collection
  - Compiler generation framework
    Stable compiler generated for several dozen targets
  - Millions of users

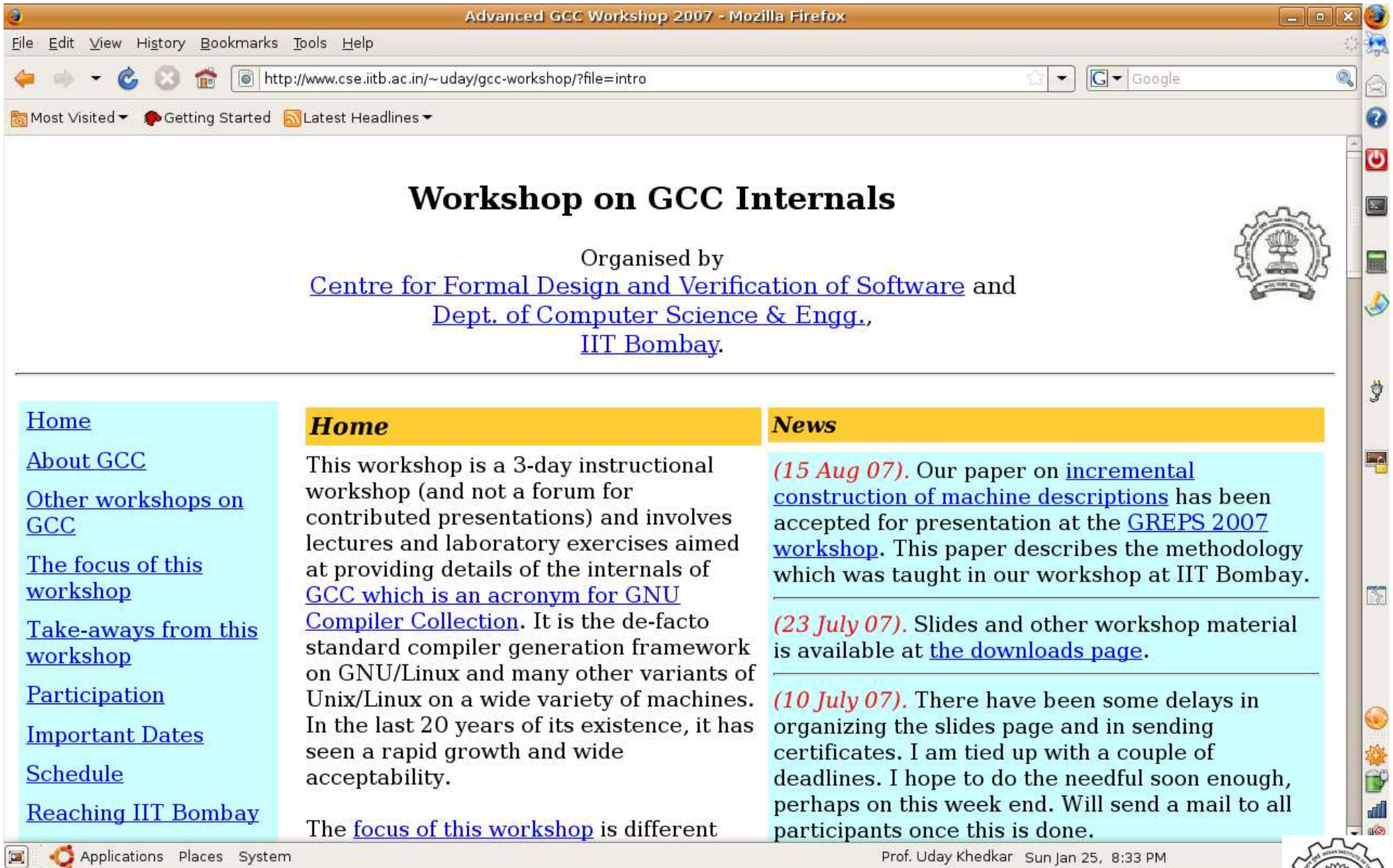
- Our Guinea Pigs

A Modest Start in 2003...

- Our Tool of Experiment The Gnu Compiler Collection
  - Compiler generation framework
    Stable compiler generated for several dozen targets
  - Millions of users

- Our Guinea Pigs
  - Several unsuspecting M.Tech. students, external B.E. students, and project engineers

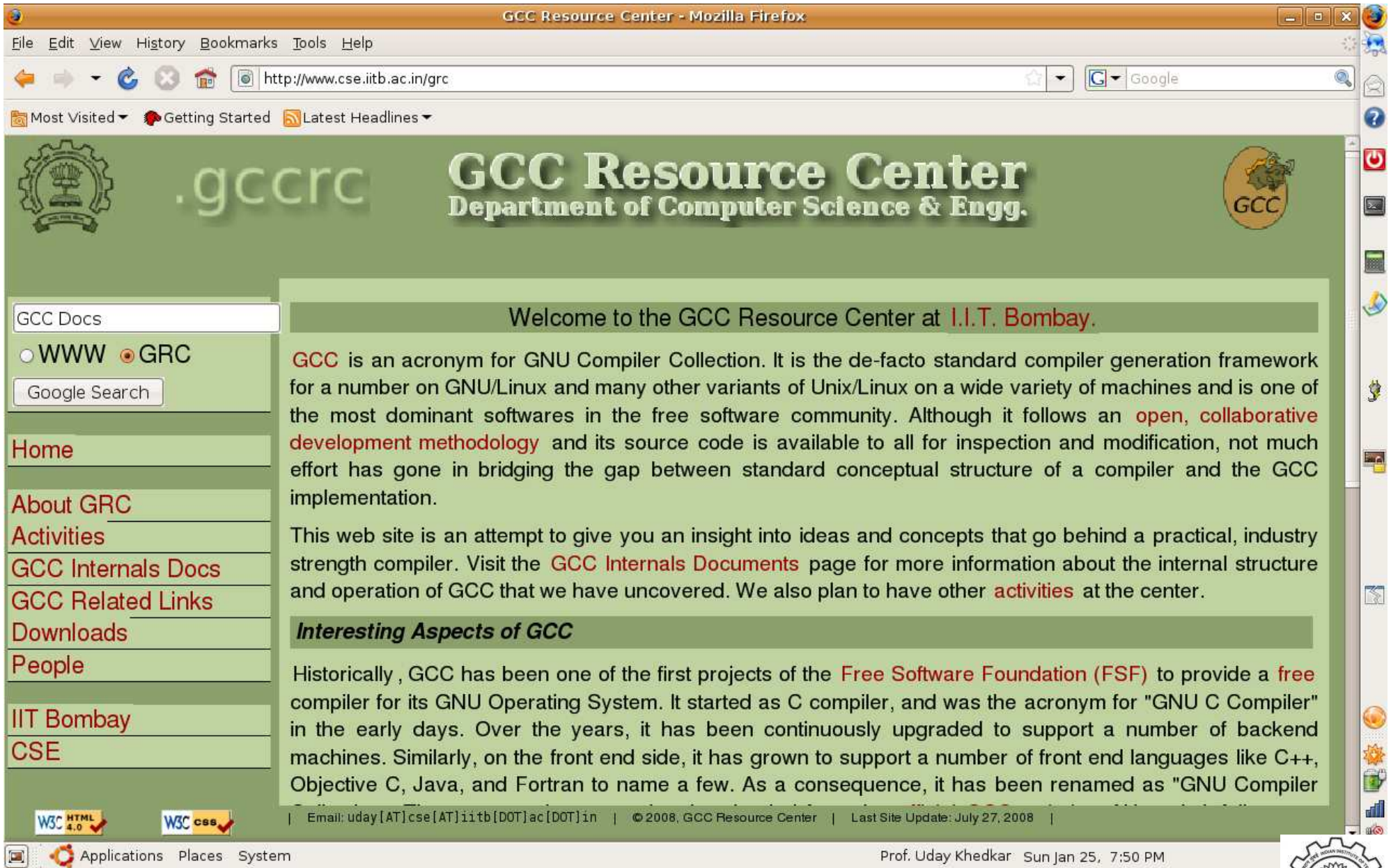
And Then in 2007...



And then in 2008...

Thanks to small seed grants from IITB and IBM Faculty Award...

And then in 2008...



Finally in 2009...

- A generous grant from the Department of Information Technology, Ministry of Communication and Information Technology, Government of India

July 2009...


Essential Abstractions in GCC '09, Workshop on GCC Internals - Konqueror
Login
Contact Us
Our Team
GCC
Essential Abstractions in GCC '09
A Workshop on GCC Internals by GCC Resource Center
Department of Computer Science & Engineering
Indian Institute of Technology, Bombay
Home
Updates
Coverage
Schedule
Registration
How to Reach
Downloads
FAQ
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 architctures 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
Applications Places System
Prof. Uday Khedkar Sat May 23, 9:20 AM

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

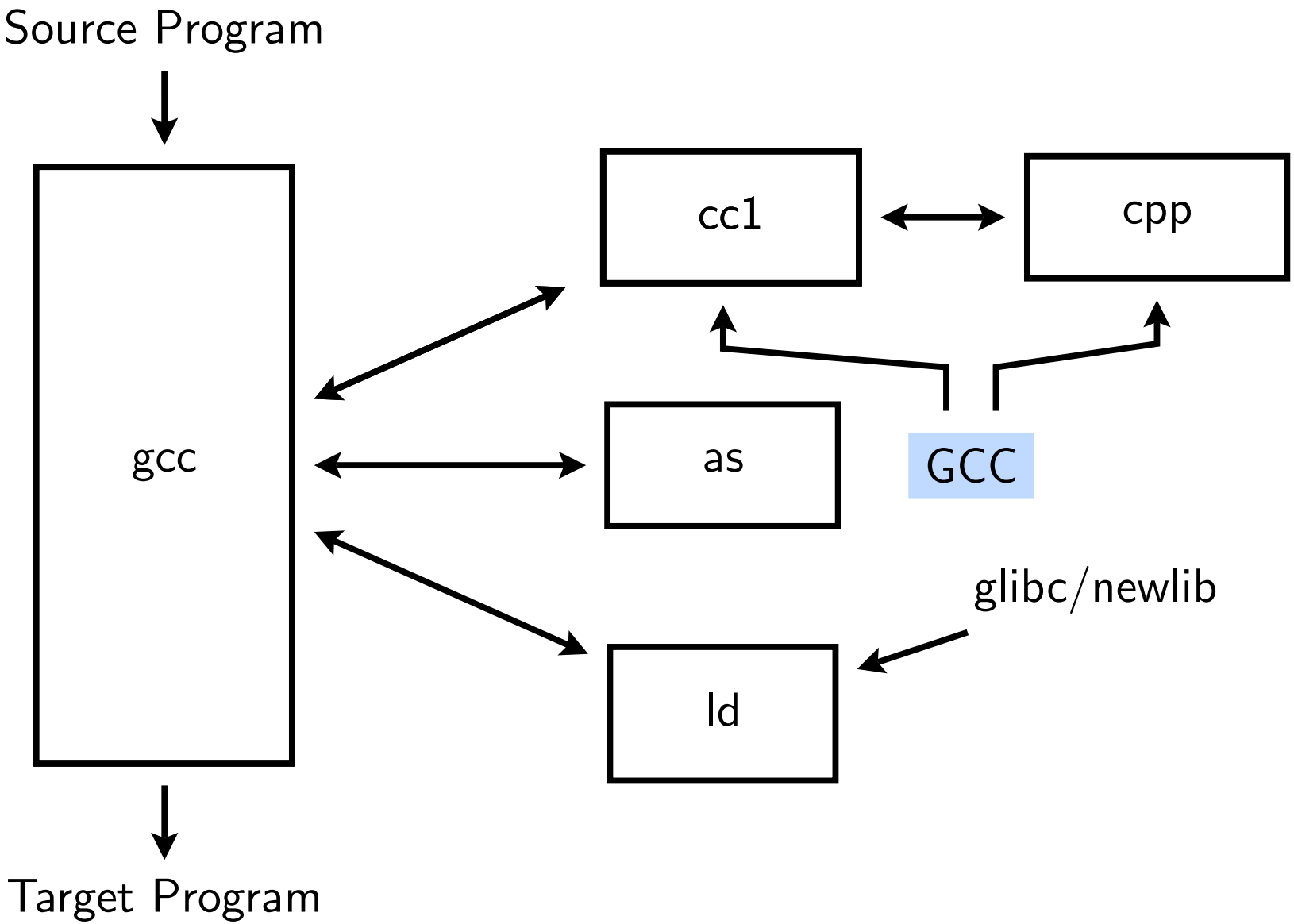
Broad Research Goals of GCC Resource Center

- Using GCC as a means
  - Adding new optimizations to GCC
  - Adding flow and context sensitive 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
  - Systematic construction of machine descriptions
  - Facilitating optimizer generation in GCC

Part 2

Understanding GCC

The Gnu Tool Chain



Why is Understanding GCC Difficult?

Some of the obvious reasons:

- Comprehensiveness

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

- Open development model

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

- Rapid versioning

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

Comprehensiveness of GCC 4.3.1: Wide Applicability

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

Comprehensiveness of GCC 4.3.1: Size

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

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

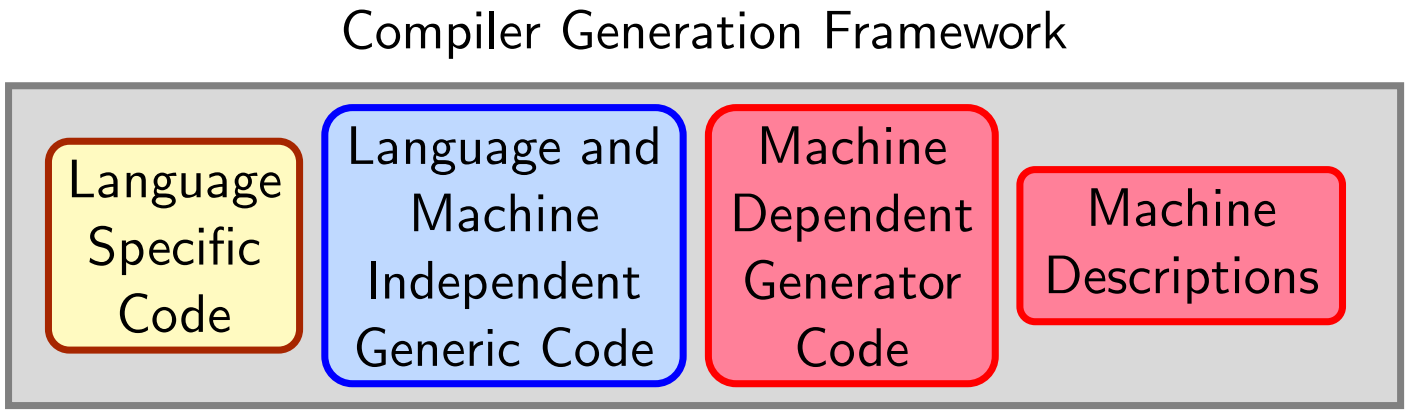
Why is Understanding GCC Difficult?

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

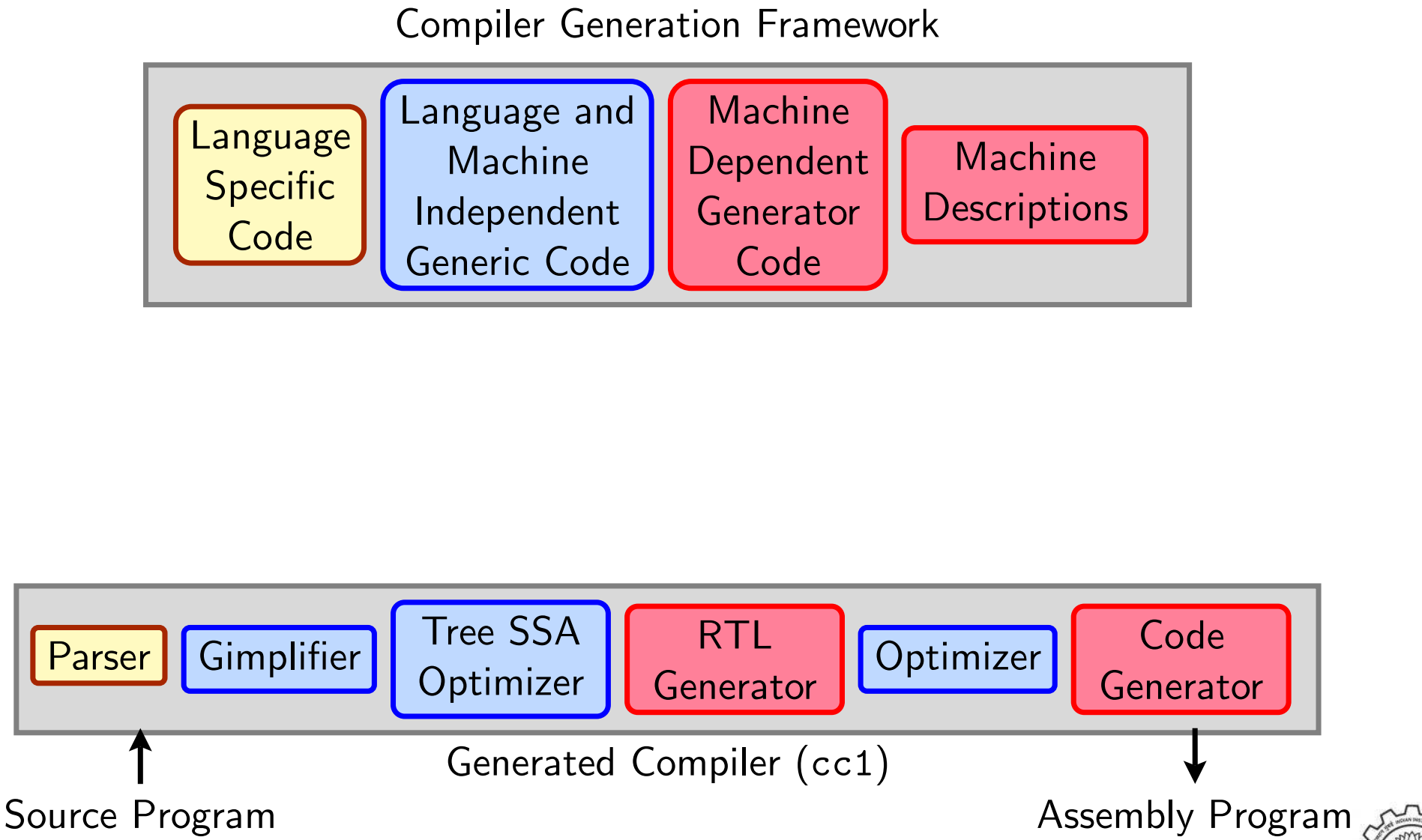
There are two distinct gaps that need to be bridged:

- 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

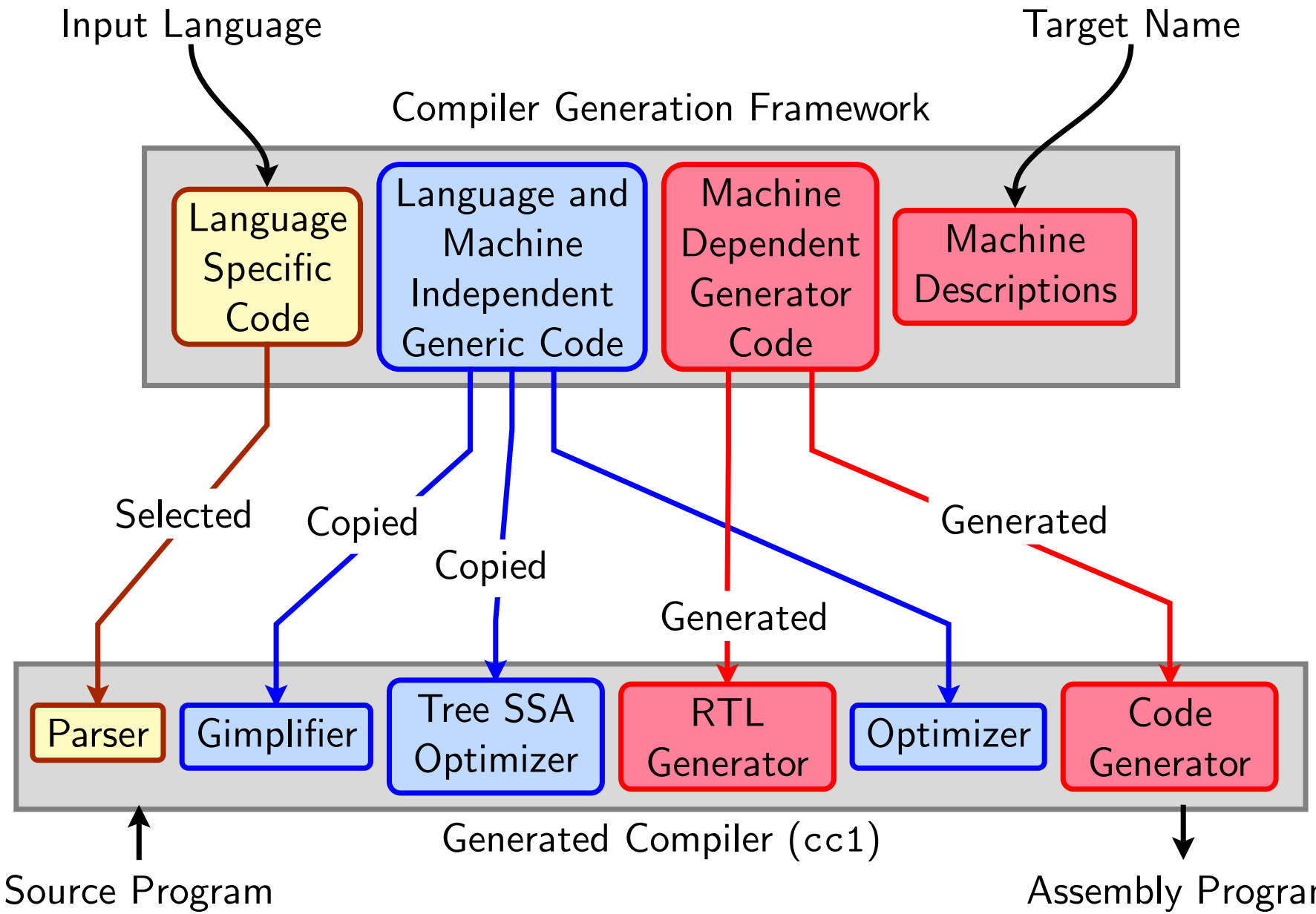
The Architecture of GCC



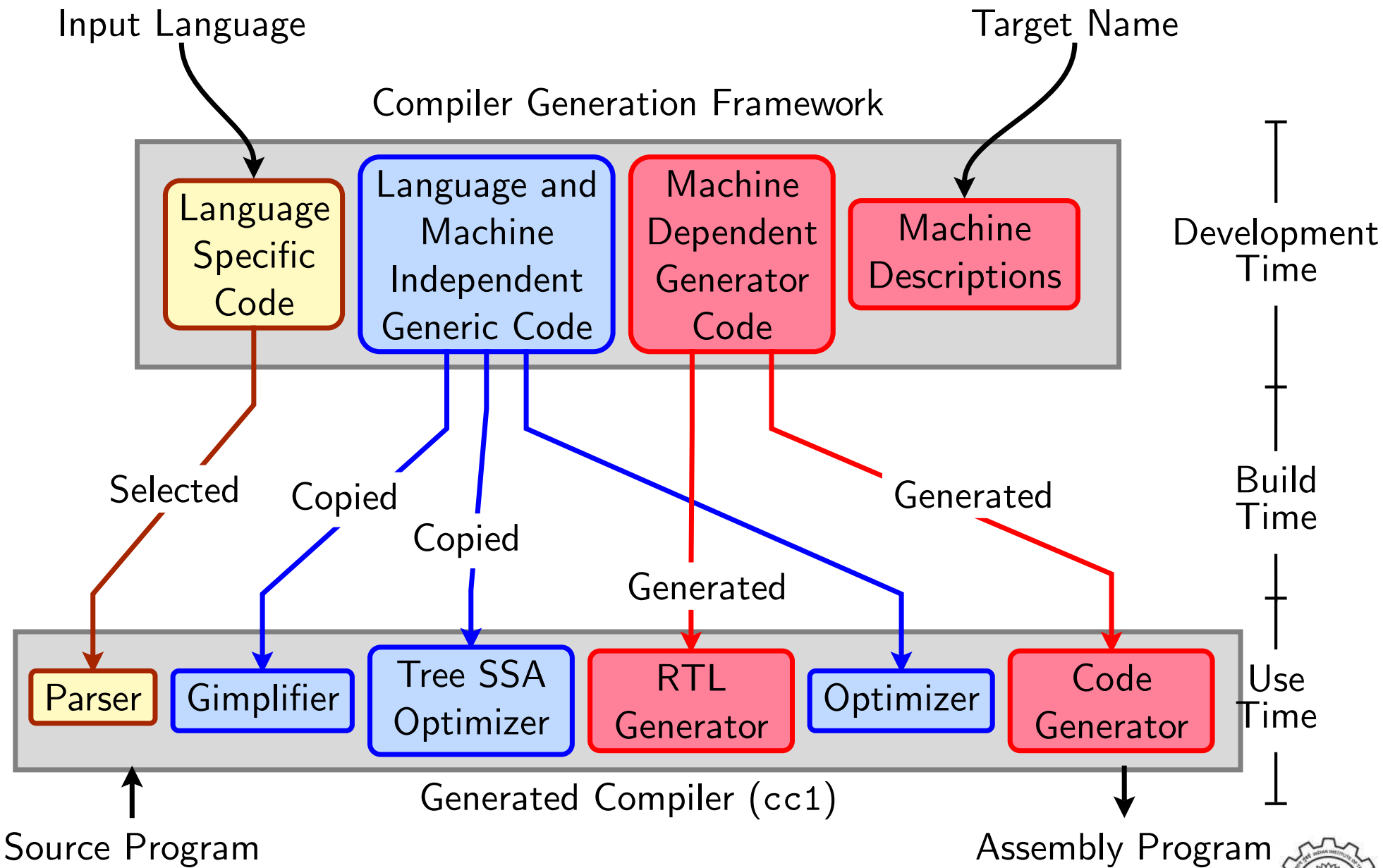
The Architecture of GCC



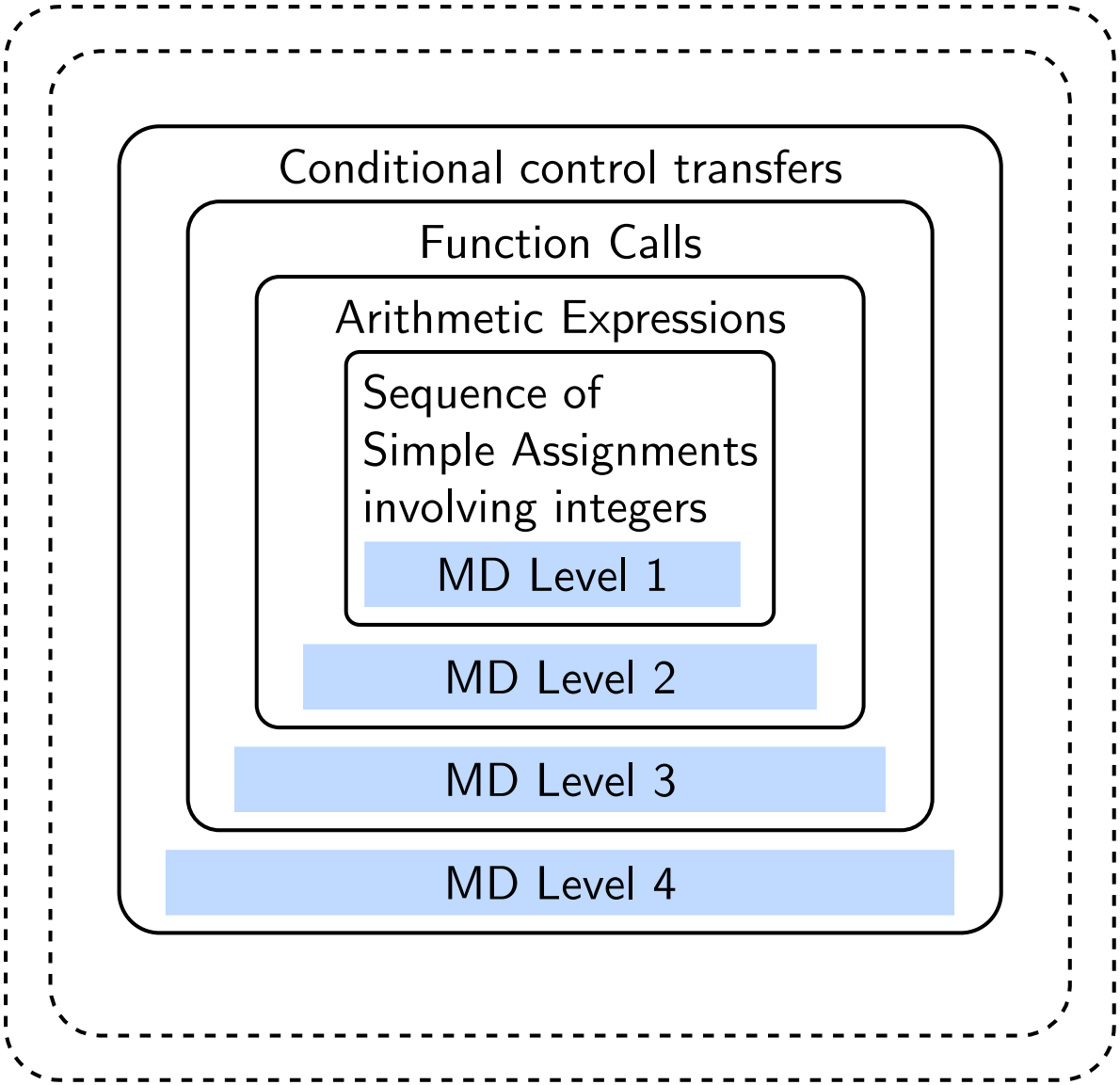
The Architecture of GCC



The Architecture of GCC



Systematic Development of Machine Descriptions [GREPS 2007]



Systematic Development of Machine Descriptions

- Define different levels of source language
- 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

Systematic Development of Machine Descriptions

- Consequence

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

- Current Status
  - Has been used for GCC 4.0.2 for mips target
  - Many people the world over have found it very useful
  - Has been extended to GCC 4.3.1
  - Has been extended to floating point operations

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

Part 3

Improving Optimizations in GCC

Improving Machine Independent Optimizations in GCC

- The Problems:
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

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

- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis

- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis
  - Facilitate generation of optimizers from specifications
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis
  - Facilitate generation of optimizers from specifications
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - *gdfa*: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR)
  - Context and flow sensitive whole program analysis does not exist
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis
  - Facilitate generation of optimizers from specifications
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - *gdfa*: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)
  - Algorithms and formal theory required further is in place

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives:
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art:
- Our Breakthrough:
- The Consequences:
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

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

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

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

- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point
- The State of Art: Merge information across contexts for efficiency
  $\Rightarrow$ Significant imprecision in recursive programs
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure
- The Consequences: Our implementation in GCC shows time and space savings by
- Further Goal:

Interprocedural Data Flow Analysis [TOPLAS2007, CC2008]

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

Heap Reference Analysis [TOPLAS 2007]

- The Problem:
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge:

- Our Key Idea:

- Current status:

- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea:
- Current status:
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status:

- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Further Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data,
  - heap data accessible to any procedure is unbounded. Hence,
  - the mapping between object names and their addresses needs to change at runtime.
- Our Key Idea: Build bounded abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Further Work:
  - Improve alias analysis
  - Interprocedural implementation and Performance tuning

Part 4

Improving Machine Descriptions and Instruction Selection

Improving Machine Descriptions and Instruction Selection

The Problems:

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

Improving Machine Descriptions and Instruction Selection

The Problems:

- Instruction selection algorithms are quite adhoc
  - Full tree maching instead of tree tiling
- The specification mechanism for Machine descriptions is quite adhoc
- Adhoc design decisions

Improving Machine Descriptions and Instruction Selection

The Problems:

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

Improving Machine Descriptions and Instruction Selection

The Problems:

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

Improving Machine Descriptions and Instruction Selection

The Problems:

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

Improving Machine Descriptions and Instruction Selection

The Consequences:

- The machine descriptions are too verbose, detailed, repetitive and require a lot of C code

Improving Machine Descriptions and Instruction Selection

The Consequences:

- The machine descriptions are too verbose, detailed, repetitive and require a lot of C code
- A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code

Improving Machine Descriptions and Instruction Selection

The Consequences:

- The machine descriptions are too verbose, detailed, repetitive and require a lot of C code
- A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code
- The machine descriptions are difficult to construct, understand, maintain, and enhance

Improving Machine Descriptions and Instruction Selection

The Consequences:

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

Improving Machine Descriptions and Instruction Selection

Our Goals:

- Define new contructs to facilitate more concise machine descriptions

Improving Machine Descriptions and Instruction Selection

Our Goals:

- Define new contructs to facilitate more concise machine descriptions
  - Generate exisiting machine descriptions from higher level descriptions

    $\Rightarrow$ No change in GCC source
    $\Rightarrow$ Incremental changes with gradual transition to new descriptions

Improving Machine Descriptions and Instruction Selection

Our Goals:

- Define new contructs to facilitate more concise machine descriptions
  - Generate exisiting machine descriptions from higher level descriptions

    ⇒ No change in GCC source
    ⇒ Incremental changes with gradual transition to new descriptions
  - Modify GCC to understand new descriptions

    ⇒ Disruptive change

Improving Machine Descriptions and Instruction Selection

Our Goals:

- Define new contructs to facilitate more concise machine descriptions
  - Generate exisiting machine descriptions from higher level descriptions

    ⇒ No change in GCC source
    ⇒ Incremental changes with gradual transition to new descriptions
  - Modify GCC to understand new descriptions

    ⇒ Disruptive change
- Use tree tiling based instruction selection algorithms

Improving Machine Descriptions and Instruction Selection

Our Goals:

- Define new contructs to facilitate more concise machine descriptions
  - Generate exisiting machine descriptions from higher level descriptions

    ⇒ No change in GCC source
    ⇒ Incremental changes with gradual transition to new descriptions
  - Modify GCC to understand new descriptions

    ⇒ Disruptive change
- Use tree tiling based instruction selection algorithms
- Honour constraints earlier during the gimple to RTL translation

Improving Machine Descriptions and Instruction Selection

Current Status:

- Preliminary investigations with spim target seem very promising
  - Fewer rules
  - Simple rules

Improving Machine Descriptions and Instruction Selection

Current Status:

- Preliminary investigations with spim target seem very promising
  - Fewer rules
  - Simple rules
- Prototype of new code generator generator (cgg) is being tested in a toy compiler set up

Part 5

Translation Validation of GCC

Translation Validation of GCC

- Problem:
- Our Objectives:
- Our approach:
- Current Status:
- Further work:

Translation Validation of GCC

- Problem:
  - Verifying a real compiler is very difficult
- Our Objectives:
- Our approach:
- Current Status:
- Further work:

Translation Validation of GCC

- Problem:
  - Verifying a real compiler is very difficult
- Our Objectives: To build a system to verify the correctness of the translation of given program
- Our approach:

- Current Status:

- Further work:

Translation Validation of GCC

- Problem:
  - Verifying a real compiler is very difficult
- Our Objectives: To build a system to verify the correctness of the translation of given program
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program
  - Derive the conditions under which the observables correspond at the start of the program
- Current Status:
- Further work:

Translation Validation of GCC

- Problem:
  - Verifying a real compiler is very difficult
- Our Objectives: To build a system to verify the correctness of the translation of given program
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program
  - Derive the conditions under which the observables correspond at the start of the program
- Current Status:
  - Formal theory and prototype implementation exists
- Further work:

Translation Validation of GCC

- Problem:
  - Verifying a real compiler is very difficult
- Our Objectives: To build a system to verify the correctness of the translation of given program
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program
  - Derive the conditions under which the observables correspond at the start of the program
- Current Status:
  - Formal theory and prototype implementation exists
  - Soundness has been shown
- Further work:
  - Cleaning up the theory to systematize the termination criteria
  - Ensuring completeness (more optimizations, more programs)

Linear Types in GCC

- The Problems:
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
  - Allow reasonable limited relaxations of linearity constraints
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
  - Allow reasonable limited relaxations of linearity constraints
  - Define appropriate type system and enforce it
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
  - Allow reasonable limited relaxations of linearity constraints
  - Define appropriate type system and enforce it
- Current Status:
  - Linearity aspects in C have been studied in details

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
  - Allow reasonable limited relaxations of linearity constraints
  - Define appropriate type system and enforce it
- Current Status:
  - Linearity aspects in C have been studied in details
  - Variants of linearity have been identified

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced
  - Parallelization and Vectorization becomes difficult
  - Synchronization and correctness problems in threads
- Our Goals:
  - Use linear types to prohibit aliasing
  - Allow reasonable limited relaxations of linearity constraints
  - Define appropriate type system and enforce it
- Current Status:
  - Linearity aspects in C have been studied in details
  - Variants of linearity have been identified
  - An initial draft of the type system is in place

Part 6

Conclusions

Conclusions

- GCC is a strange paradox
  - Practically very successful but quite adhoc.
  - Needs significant improvements
- GCC Resource Center at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC

Last but not the least . . .

Thank You!