Workshop on Essential Abstractions in GCC

A Summary of Essential Abstractions

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

3 July 2013
Part 2

Methodology
Our Padagogy

Compiler Specifications

External View
- Machine descriptions
- Configuration and building

Internal View
- Front end hooks
- Retargetability mechanism
- Pass structure
- Control flow
- Static and dynamic plugin mechanisms

Compiled Compiler

Pass structure and IR
- Parallelization, Vectorization
- Gray box probing
- Configuration and building

Compiler Generator

Retargetability mechanism

Machine descriptions

Front end hooks

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Gray Box Probing

Black Box Probing

Observe

Observe
Gray Box Probing

White Box Probing
Gray Box Probing

Phase 1 → Phase 2 → ... → Phase n

Observe → Observe → Observe → Observe
Systematic Development 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
Part 3

The Framework
The GNU Tool Chain for C

Source Program

gcc

cc1

cpp

as

ld

glibc/newlib

Target Program
The Architecture of GCC

Input Language

Compiler Generation Framework

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Target Name

Selected

Copied

Copied

Copied

Generated

Generated

Generated

Parser

Gimplifier

Tree SSA Optimizer

Expander

Optimizer

Recognizer

Development Time

Build Time

Use Time

Source Program

Generated Compiler (cc1)

Assembly Program

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Part 4

The Generated Compiler
Compilation Models

**Aho Ullman Model**

Front End → AST → Optimizer → Target Indep. IR → Code Generator → Target Program

Aho Ullman: Instruction selection
- over optimized IR using
- cost based tree pattern matching

**Davidson Fraser Model**

Front End → AST → Expander → Register Transfers → Optimizer → Register Transfers → Recognizer → Target Program

Davidson Fraser: Instruction selection
- over AST using
- structural tree pattern matching
- naive code which is
  - target dependent, and is
  - optimized subsequently

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Basic Transformations in GCC

Transformation from a language to a "different" language

Target Independent --- Target Dependent

Parse → Gimplify → Tree SSA Optimize → Generate RTL → Optimize RTL → Generate ASM

GIMPLE → RTL

RTL → ASM

GIMPLE Passes

RTL Passes
For simplicity, we have included all passes in a single list. Actually passes are organized into five lists and are invoked as five different sequences.
The Mechanism of Dynamic Plugin

- pass manager
- code for pass
- code for pass
- code for dynamic plugin
- expander code
- optab_table
- recognizer code

Runtime initialization of the appropriate linked list of passes
Made possible by dynamic linking
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part 5

LTO
Partitioned and Non-Partitioned LTO

Analysis

Load complete call graph

Load function summaries but not bodies

Transformation

Load all function bodies

Load all function bodies one by one

Partitioned Mode

Load groups of function bodies

Sequential Analysis

All function bodies already loaded

No need to load the entire program in memory
IPA possible (multiple function bodies)
Parallel transformations possible
Analysis and transformations in independent processes
Partitioned and Non-Partitioned LTO

Analysis

Load complete call graph
Load function summaries but not bodies
Load all function bodies
Partitioned Mode

Sequential Analysis

Transformation

Load all function bodies
Load function bodies one by one
Load groups of function bodies
All function bodies already loaded

Balanced partitions: `-flto -flto-partitions=balanced`
One Partition per file: `-flto -flto-partitions=1to1`
Partitions by number: `-flto --params lto-partitions=n`
Partitions by size: `-flto --params lto-min-partition=s`
Partitioned and Non-Partitioned LTO

**Analysis**

- Load complete call graph
- Load function summaries but not bodies
- Load all function bodies

**Transformation**

- Load all function bodies
- Load function bodies one by one
- Load groups of function bodies
- All function bodies already loaded

**Non-Partitioned Mode**

Entire program needs to be loaded in memory
No partitions `-flto -flto-partitions=none`
Strictly sequential transformations
Analysis and transformations in the same processes
cc1 and Single Process

```c

toplev_main
...
  compile_file
...
  cgraph_analyze_function

cc1
...
  cgraph_optimize
...
  ipa_passes
...
  cgraph_expand_all_functions
...
  tree_rest_of_compilation
```
cc1 and Single Process lto1

toplev_main
... compile_file
... cgraph_analyze_function

lto_main
... read_cgraph_and_symbols
... materialize_cgraph

cgraph_optimize
... ipa_passes
... cgraph_expand_all_functions
... tree_rest_of_compilation

lto1
The GNU Tool Chain for Single Process LTO Support

**gcc**

- **cc1’**
- **lto1’**
- **common**
- "Fat" .s files
- "Fat" .o files
- Single .s file
- Single .o file + glibc/newlib
- a.out file

**collect2**

**as**

**ld**
Common Code (executed twice for each function in the input program for single process LTO. Once during LGEN and then during WPA + LTRANS)

```c

cgraph_optimize
  ipa_passes
    execute_ipa_pass_list(all_small_ipa_passes)/*!in lto*/
    execute_ipa_summary_passes(all_regular_ipa_passes)
    execute_ipa_summary_passes(all_lto_gen_passes)
  ipa_write_summaries
  execute_ipa_pass_list(all_late_ipa_passes)
  cgraph_expand_all_functions
    cgraph_expand_function
    /* Intraprocedural passes on GIMPLE, */
    /* expansion pass, and passes on RTL. */
```
Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`

- `f1.c` -> `cc1` -> `[lto1]` -> `f1.o`
- `f2.c` -> `cc1` -> `[lto1]` -> `f2.o`
- `f3.c` -> `cc1` -> `[lto1]` -> `f3.o`

Option `-flto -o out`

- `[lto1]` -> `common`
- `/tmp/ccdKEyVB.ltrans0.o` (possibly multiple files)

Large call graph without procedure bodies
(Interproc. analysis: √
Transformation: ×)

WPA

LGEN

LTRANS
Non-Partitioned LTO

- Option `-flto -c`
- `f1.c` -> `f1.o`
- `f2.c` -> `f2.o`
- `f3.c` -> `f3.o`

IPA + Transformations

- Option `-flto -o out`
- `-flto-partition=none`
- `large call graph with procedure bodies (Interproc. analysis: √ Transformation: √)`

This IPA can examine function bodies also

LGEN
Part 6

The Build Process
Configuring GCC

```
configure
config.guess
config.sub
configure.in
config/*
config.log
config.cache
config.h.in
config.status
Makefile.in
config.h
```
Bootstrapping: The Conventional View

Level $n$ C

Implementation language

$C_{n-1}$

Input language

$C_{n-2}$

Output language

$m/c$

Implementation language

$m/c$

Essential Abstractions: The Build Process

3 July 2013

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
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
Build for a Given Machine

This is what actually happens!

• Generation
  ▶ Generator sources
    \((\$(SOURCE\_D)/gcc/gen*.c)\) 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 \((\$(SOURCE\_D)\)
  and executables created in corresponding subdirectories of \((\$(BUILD)\)

• Installation
  Created executables and libraries are copied in
  \((\$(INSTALL)\)
More Details of an Actual Stage 1 Build for C

GCC sources

native cc, binutils, libraries

libraries

libliberty

fixincl

gen*

cc1

cpp

taxt target binutils, libraries

xgcc

libgcc

cc, binutils, libraries for stage 2
Building a MIPS Cross Compiler on i386: A Closer Look

**Stage 1 Build**
- Requirement: \( BS = HS = \text{i386}, TS = \text{mips} \)
  - *Stage 1 cannot build gcc but can build only cc1*
  - Stage 1 build cannot create executables
  - Library sources cannot be compiled for mips using stage 1 build
  - Stage 2 build is not possible

**Stage 2 build is infeasible for cross build**

-we have not built libraries for mips
Difficulty in Building a Cross Compiler

- gcc for target
- target libraries
- libgcc

The diagram illustrates the relationships:

- gcc for target requires libgcc
- libgcc uses target libraries
- target libraries require gcc for target
## Generated Compiler Executable for All Languages

- **Main driver**
  
  ```
  $BUILD/gcc/xgcc
  ```

- **C compiler**
  
  ```
  $BUILD/gcc/cc1
  ```

- **C++ compiler**
  
  ```
  $BUILD/gcc/cc1plus
  ```

- **Fortran compiler**
  
  ```
  $BUILD/gcc/f951
  ```

- **Ada compiler**
  
  ```
  $BUILD/gcc/gnat1
  ```

- **Java compiler**
  
  ```
  $BUILD/gcc/jcl
  ```

- **Java compiler for generating main class**
  
  ```
  $BUILD/gcc/jvgenmain
  ```

- **LTO driver**
  
  ```
  $BUILD/gcc/lto1
  ```

- **Objective C**
  
  ```
  $BUILD/gcc/cc1obj
  ```

- **Objective C++**
  
  ```
  $BUILD/gcc/cc1objplus
  ```
Part 7

Retargetability
Examples of Influences on the Machine Descriptions

- **Source Language**
  - INT_TYPE_SIZE
  - Activation Record
  - `<target>.h`

- **GCC Architecture**
  - Generation of `nop`
  - tree covers for instruction selection
  - `define_predicate`
  - `<target>.h`

- **Build System**
  - `hwint.h`

- **Host System**
  - `hwint.h`

- **Target System**
  - Instruction Set Architecture
  - Assembly and executable formats
  - `{<target>.md, <target>.h, other headers}`

---

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Redundancy in MIPS Machine Descriptions: Example 3

\[
\begin{align*}
\text{(set } (\text{match_operand:} m \ 0 \ "\text{register_operand}" \ "c0") \ (\text{plus:} m \\
(\text{mult:} m \ (\text{match_operand:} m \ 1 \ "\text{register_operand}" \ "c1") \\
(\text{match_operand:} m \ 2 \ "\text{register_operand}" \ "c2")) \\
(\text{match_operand:} m \ 3 \ "\text{register_operand}" \ "c3")))
\end{align*}
\]

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>m</th>
<th>c0</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>*mul_acc_si</td>
<td>SI</td>
<td>=l*???,d?</td>
<td>d,d</td>
<td>d,d</td>
<td>0,d</td>
</tr>
<tr>
<td>*mul_acc_si_r3900</td>
<td>SI</td>
<td>=l*???,d*??,d?</td>
<td>d,d,d</td>
<td>d,d,d</td>
<td>0,1,d</td>
</tr>
<tr>
<td>*macc</td>
<td>SI</td>
<td>=l,d</td>
<td>d,d</td>
<td>d,d</td>
<td>0,1</td>
</tr>
<tr>
<td>*madd4&lt;mode&gt;</td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>*madd3&lt;mode&gt;</td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>0</td>
</tr>
</tbody>
</table>
Instruction Specification and Translation: A Recap

- GIMPLE: target independent
- RTL: target dependent
- Need: associate the semantics
  ⇒ GCC Solution: Standard Pattern Names

(define_insn "movsi"
  [(set (match_operand 0 "register_operand" "r")
    (match_operand 1 "const_int_operand" "k"))]
  ";" /* C boolean expression, if required */
  "li %0, %1"
)
Translation Sequence in GCC

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

```c
D.1283 = 10;
(set
  (reg:SI 58 [D.1283])
  (const_int 10: [0xa])
)
```
Retargetability Mechanism of GCC

- **Input Language**
- **Target Name**
- **Compiler Generation Framework**

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

**Selected** → **Copied** → **Generated** → **Generated Compiler**

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

- **GIMPLE → PN**
  - Development Time
- **PN → IR-RTL**
- **IR-RTL → ASM**
  - Build Time

- **GIMPLE → IR-RTL**
  - Use Time
- **IR-RTL → ASM**

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Hooking up Back End Details

$(\text{SOURCE})/\text{gcc/optabs.h}$
$(\text{SOURCE})/\text{gcc/optabs.c}$

$\text{optab_table}$

Runtime initialization of data structure in $\text{cc1}$
through function $\text{init_all_optabs}$

OTI\_mov

SI
- $\text{insn_code}$
  - CODE\_FOR\_movsi

SF
- $\text{insn_code}$
  - CODE\_FOR\_nothing

insn\_data

... ...

"movsi"

1280
- $\text{gen_movsi}$
  ...

$\text{insn\_output.c}$

$\text{insn\_codes.h}$

$\text{insn\_opinit.c}$

$\text{optab_table}$

$(\text{BUILD})/\text{gcc/insn-output.c}$

$\text{insn\_data}$

... ...

"movsi"

1280
- $\text{gen_movsi}$
  ...

$\text{insn\_codes.h}$

$\text{insn\_opinit.c}$

...