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 2011
Compilation Models

**Aho Ullman Model**

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

**Davidson Fraser Model**

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

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

**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
The GNU Tool Chain

Source Program

\[
gcc \\
\text{cpp} \quad \text{cc1} \quad \text{cpp}
\]

 GCC

\[
glibc/newlib \\
gcc \\
\text{as} \\
\text{ld}
\]

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

Selected

Copied

Copied

Generated

Generated

Development Time

Build Time

Use Time

Parser

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Generated Compiler (cc1)

Source Program

Assembly Program
Configuring GCC

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

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Bootstrapping: The Conventional View

\[ C_{n-1} \quad C_n \]

Input language \( C_{n-2} \) Implementation language

Output language
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 + native binutils → libraries

libiberty → fixincl → gen*

cc1 → target binutils → xgcc → libgcc

c + binutils for stage 2
Basic Transformations in GCC

Transformation from a language to a *different* language

Target Independent  Target Dependent

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

GIMPLE → RTL

RTL → ASM

GIMPLE Passes

RTL Passes
Instruction Specification and Translation: A Recap

- **GIMPLE**: target independent
- **RTL**: target dependent
- **Need**: associate the *semantics*

> GCC Solution: Standard Pattern Names

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

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Translation Sequence in GCC

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

D.1283 = 10;

```
(set
  (reg:SI 58 [D.1283])
  (const_int 10: [0xa])
)
```

li $t0, 10
Retargetability Mechanism of GCC

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Plugin Structure in cc1

- toplevel
- main
- frontend
- pass manager
- pass 1
- code for pass 1
- code for language 1
- code for language 2
- code for language n
- pass 2
- code for pass 2
- pass expand
- code for pass expand
- optab_table
- insn_data
- generated code for machine 1
- pass n
- code for pass n
- MD n
- MD 2
- MD 1
The GNU Tool Chain for LTO Support

- `gcc`
- `cc1`
- `lto1`
- `as`
- `collect2`
- `ld`

```
```

- "Fat" .s files
- "Fat" .o files
- Single .s file
- Single .o file + glibc/newlib
- a.out file

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The GNU Tool Chain for LTO Support

Common Code (executed twice for each function in the input program)

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
  cgraph_expand_all_functions
    cgraph_expand_function
      /* Intraprocedural passes on GIMPLE, */
      /* expansion pass, and passes on RTL. */

a.out file
Hooking up Back End Details

```
$(SOURCE)/gcc/optabs.h
$(SOURCE)/gcc/optabs.c
```

```
insn_data

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;movsi&quot;</td>
<td></td>
</tr>
<tr>
<td>1280</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
```

```
optab_table

<table>
<thead>
<tr>
<th>OTI_mov</th>
<th>SI</th>
<th>SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov_optab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insn_code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_FOR_movsi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```

```
$BUILD/gcc/insn-codes.h

CODE_FOR_movsi=1280
CODE_FOR_movsf=CODE_FOR_nothing
```

```
$BUILD/gcc/insn-opinit.c

... 
```

Runtime initialization of data structure