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

Workshop Coverage

Compiler Specifications
- Machine descriptions
- Front end hooks

Compiler Generator
- Configuration and building
- Retargetability mechanism

Generated Compiler
- Gray box probing
- Pass structure and IR
- Data Flow Analysis
- Parallelization, Vectorization
- Control flow
- Static and dynamic plugin mechanisms

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

The GNU Tool Chain for C

Source Program
- gcc
  - cc1
  - cpp
  - as
  - ld
  - glibc/newlib
  - Target Program

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
The Architecture of GCC

Configuring GCC

Bootstrapping: The Conventional View

A Native Build on i386
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

Stage 1 Build

- mips assembly
- 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

Requirement: B_S = H_S = i386, T_S = mips
Building a MIPS Cross Compiler on i386: A Closer Look

 GCC Source

 C mips

 i386

 cc

Stage 1 Build

 mips assembly

Stage 2 Build

 Requirement: BS = HS = i386

• 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

 requires

 target libraries

 uses

libgcc

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/ltol
• Objective C
  $BUILD/gcc/cc1obj
• Objective C++
  $BUILD/gcc/cc1objplus

Basic Transformations in GCC

Transformation from a language to a different language

Target Independent

Parse → Simplify → Tree SSA Optimize → Generate RTL

Optimize RTL → Generate ASM

GIMPLE → RTL

RTL → ASM

Target Dependent

GIMPLE Passes

RTL Passes
Instruction Specification and Translation: A Recap

- Target Independent
  - Parse → Gimplify → Tree SSA → Generate RTL → Optimize RTL → Generate
- Target Dependent
  - GIMPLE: target independent
  - RTL: target dependent
  - Need: associate the semantics

GCC Solution: Standard Pattern Names

D.1283 = 10;
(set (reg:SI 58 [D.1283]) (const int 10: [0xa]))
li $t0, 10

Retargetability Mechanism of GCC

Input Language
Compiler Generation Framework
Target Name
Language Specific Code
Language and Machine Independent Generic Code
Machine Dependent Generator Code
Machine Descriptions
Selected
Copied
Generated
Selected
Copied
Generated
GIMPLE → PN
PN → IR-RTL
IR-RTL → ASM
GIMPLE → IR-RTL
IR-RTL → ASM

Plugin Structure in cc1

toplevel
main
front end
pass manager
code for
pass 1
code for
pass 2

MD n
MD 2
MD 1
The Mechanism of Dynamic Plugin

Runtime initialization of the appropriate linked list of passes
Made possible by dynamic linking

Execution Order in Intraprocedural Passes

Execution Order in Interprocedural Passes
**Execution Order in Interprocedural Passes**

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5

**LTO Support in GCC**

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Whole Program Analysis</th>
<th>Call graph without function bodies</th>
<th>Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Call graph with function bodies</td>
<td>Supported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not supported</td>
</tr>
</tbody>
</table>

**cc1 and Single Process lto1**

toplevel_main
... compile_file
... cgraph_analyze_function
cgraph_optimize
... ipa_passes
... cgraph_expand_all_functions
... tree_rest_of_compilation
c1

lto1
... lto_main
... read_cgraph_and_symbols
... materialize_cgraph
cgraph_optimize
... ipa_passes
... cgraph_expand_all_functions
... tree_rest_of_compilation
The GNU Tool Chain for Single Process LTO Support

The GNU Tool Chain for Single Process LTO Support

Essential Abstractions: Summary

Multi Process LTO (aka WHOPR LTO)

Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

Option `-flto -c`

External View

Internal View

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

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

ecessary Abstractions in GCC
GCC Resource Center, IIT Bombay

ecessary Abstractions in GCC
GCC Resource Center, IIT Bombay

ecessary Abstractions in GCC
GCC Resource Center, IIT Bombay

ecessary Abstractions in GCC
GCC Resource Center, IIT Bombay
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

```
cc1  lto1  common
f1.c --> f1.o
f2.c --> f2.o
f3.c --> f3.o
```

`LGEN`

WPA

```
Option `-flto -o out`
```

large call graph without procedure bodies
(Interproc. analysis: ✓ Transformation: ✗)

```
/ccdKEyVB.ltrans0.o
(possibly multiple files)
```

```
/out
```

`LTRANS`

Option `-flto -o out
-flto-partition=none`

```
cc1  lto1  common
f1.c --> f1.o
f2.c --> f2.o
f3.c --> f3.o
```

large call graph with procedure bodies
(Interproc. analysis: ✓ Transformation: ✓)

```
/out
```

Single Process LTO

```
Option `-flto -c`
```

```
cc1  lto1  common
f1.c --> f1.o
f2.c --> f2.o
f3.c --> f3.o
```

```
/out
```

External View

```
Option `-flto -o out
-flto-partition=none`
```

large call graph with procedure bodies
(Interproc. analysis: ✓ Transformation: ✓)

```
/out
```

Internal View

```
cc1  lto1  common
```

This WPA can examine function bodies also

```
/ccdKEyVB.ltrans0.o
```

```
/out
```

IPA + LTRANS
Redundancy in MIPS Machine Descriptions: Example 3

\[
\text{(set (match_operand: } m \ 0 \ "register_operand" \ "c0") \ (plus: } m \ \text{(mult: } m \ \text{(match_operand: } m \ 1 \ "register_operand" \ "c1") \ \text{(match_operand: } m \ 2 \ "register_operand" \ "c2")))) \]

\[
\text{(match_operand: } m \ 3 \ "register_operand" \ "c3") \]

RTL Template

Structure

Details

<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><em>mul</em>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><em>mul</em>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><em>macc</em></td>
<td>SI</td>
<td>=l,d</td>
<td>d,d</td>
<td>d,d</td>
<td>0,1</td>
</tr>
<tr>
<td><em>madd4&lt;mode&gt;</em></td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td><em>madd3&lt;mode&gt;</em></td>
<td>ANYF</td>
<td>=f</td>
<td>f</td>
<td>f</td>
<td>0</td>
</tr>
</tbody>
</table>

Hooking up Back End Details

Optab Table

 insn_data

<table>
<thead>
<tr>
<th>optab_table</th>
</tr>
</thead>
</table>
| ...
| ...
| mov_optab |
| ...

Runtime initialization of data structure

OTI

insn_code

CODE_FOR_movsi

$SOURCE$gcc/optabs.h

$(SOURCE)/gcc/optabs.c

$BUILD$gcc/insn-output.c

insn_data

| ...
| ...
| ...
| "movsi"
| 1280
| gen_movsi

And the final realization...

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay