Workshop on Essential Abstractions in GCC

The Retargetability Model of GCC

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

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

2 July 2011

Outline

- A Recap
- Generating the code generators
- Using the generator code generators

Retargetability Mechanism of GCC

Part 1

A Recap

Essential Abstractions in GCC

 GCC Resource Center, IIT Bombay
**Plugin Structure in cc1**

![Diagram showing plugin structure in cc1](image)

**What is “Generated”?**

- Info about instructions supported by chosen target, e.g.
  - Listing data structures (e.g. instruction pattern lists)
  - Indexing data structures, since diff. targets give diff. lists.
- C functions that generate RTL internal representation
- Any useful “attributes”, e.g.
  - Semantic groupings: arithmetic, logical, I/O etc.
  - Processor unit usage groups for pipeline utilisation

**Information supplied by the MD**

- The target instructions – as ASM strings
- A description of the semantics of each
- A description of the features of each like
  - Data size limits
  - One of the operands must be a register
  - Implicit operands
  - Register restrictions

<table>
<thead>
<tr>
<th>Information supplied</th>
<th>in define_insn as</th>
</tr>
</thead>
<tbody>
<tr>
<td>The target instruction</td>
<td>ASM string</td>
</tr>
<tr>
<td>A description of it's semantics</td>
<td>RTL Template</td>
</tr>
<tr>
<td>Operand data size limits</td>
<td>predicates</td>
</tr>
<tr>
<td>Register restrictions</td>
<td>constraints</td>
</tr>
</tbody>
</table>
Part 2

Generating the Code Generators

Retargetability ⇒ Multiple MD vs. One CGF!

How GCC uses target specific RTL as IR

GIMPLE ASSIGN "movsi" (set (<dest>) (<src>))

Separate CGF code and MD

Implement

GIMPLE ASSIGN "movsi" (set (<dest>) (<src>))

Unnecessary in CGF; hard code

Implement in MD

MD Information Data Structures

Two principal data structures

• struct optab – Interface to CGF
• struct insn_data – All information about a pattern
  ▶ Array of each pattern read
  ▶ Some patterns are SPNs
  ▶ Each pattern is accessed using the generated index

Supporting data structures

• enum insn_code: Index of patterns available in the given MD

Note

Data structures are named in the CGF, but populated at build time. Generating target specific code = populating these data structures.
Assume `movsi` is supported but `movsf` is not supported...

```
$(SOURCE_D)/gcc/optabs.h
$(SOURCE_D)/gcc/optabs.c
(BUILD)/gcc/insn-output.c

optab_table

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

$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 using function set_optab_handler**
- **Index of patterns**
- **Function calls**
- **Setting up of activation records**
- **Non-local jumps**
- **etc. (i.e. deeper study is required on this aspect)**
Handling C Code in `define_expand`

```c
(define_expand "movsi"
  [((set (op0) (op1)))]
  "/* C CODE OF DEFINE EXPAND */
"

rtx
gen_movsi (rtx operand0, rtx operand1)
{
  ...{
    /* C CODE OF DEFINE EXPAND */
    } emit_insn (gen_rtx_SET (VOIDmode, operand0, operand1)
    ...}
}
```

Part 3

Using the Code Generators

cc1 Control Flow: GIMPLE to RTL Expansion (pass_expand)

```c
gimple_expand_cfg
  expand_gimple_basic_block(bb)
  expand_gimple_cond(stmt)
  expand_gimple_stmt(stmt)
  expand_gimple_stmt_1(stmt)
  expand_expr_real_2
  expand_expr /* Operands */
  expand_expr_real
  optab_for_tree_code
  expand_binop /* Now we have rtx for operands */
  expand_binop_directly
  /* The plugin for a machine */
  code=optab_handler(binoptab,mode)
  GEN_FCN
  emit_insn
```

RTL Generation

```c
expand_binop_directly
  ... /* Various cases of expansion */
  /* One case: integer mode move */
  icode = mov_optab->handler[SImode].insn_code
  if (icode != CODE_FOR_nothing) {
    ... /* preparatory code */
    emit_insn (GEN_FCN(icode)(dest,src));
  }
```
RTL to ASM Conversion

- Simple pattern matching of IR RTLs and the patterns present in all named, un-named, standard, non-standard patterns defined using `define_insn`.
- A DFA (deterministic finite automaton) is constructed and the first match is used.

Conclusions

A Comparison with Davidson Fraser Model

- Retargetability in Davidson Fraser Model
  - Manually rewriting Expander and patter matcher
  - Expected to be simple for machines of 1984 Era
- Retargetability in GCC
  - Automatic construction possible by separating machine specific details in carefully designed data structures
  - List insns as they appear in the chosen MD
  - Index them
  - Supply index to the CGF