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
Part 1

A Recap
Retargetability Mechanism of GCC

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Retargetability Mechanism of GCC

Input Language

Parser
Gimplifier
Tree SSA Optimizer
Expander
Optimizer
Code Generator

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

Selected
Copied
Copied
Generated
Generated

GIMPLE → IR-RTL
IR-RTL → ASM

Development Time
Build Time
Use Time

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Retargetability Mechanism of GCC

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
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

Generated Compiler

Selected

Copied

Copied

Generated

Generated

Parser

Gimplifier

Tree SSA Optimizer

Expander

Optimizer

Code Generator

GIMPLE → PN

PN → IR-RTL

IR-RTL → ASM

GIMPLE → IR-RTL

IR-RTL → ASM

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Retargetability Mechanism of GCC

Compiler Generation Framework

Input Language
- Language Specific Code
- Language and Machine Independent Generic Code

Target Name
- Machine Descriptions
- Machine Dependent Generator Code

Parser
- Gimplifier
- Tree SSA Optimizer
- Expander
- Optimizer
- Code Generator

GIMPLE → PN

PN → IR-RTL

IR-RTL → ASM

GIMPLE → IR-RTL

IR-RTL → ASM

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Plugin Structure in cc1

toplev main

front end

pass manager

double arrow represents control flow whereas single arrow represents pointer or index

langhook ...

code for language 1

code for language 2

code for language n

pass 1

code for pass 1

pass 2

code for pass 2

expander
code
optab_table

pass n

code for pass n
Plugin Structure in cc1

- **toplevel**
  - `main`
- **frontend**
- **pass manager**
  - **pass 1**
  - **pass 2**
  - **pass expand**
  - **pass n**
- **langhook**
  - code for language 1
  - code for language 2
  - code for language n
- ** insn_data**
  - generated code for machine 1
- **code for**
  - pass 1
  - pass 2
  - pass expand
  - pass n
- **optab_table**

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Plugin Structure in \texttt{cc1}

- **toplev**
  - **main**
- **frontend**
- **pass manager**
  - **pass 1**
  - **code for pass 1**
  - **code for pass 2**
  - **pass 2**
  - **expander code**
  - **optab_table**
  - **insn_data**
  - **generated code for machine 2**
- **pass expand**
- **pass n**
  - **code for pass n**
- **langhook**
  - ...
Plugin Structure in cc1

- `toplevel`
  - `main`
  - `frontend`

- `pass_manager`
  - `pass 1`
    - `code for pass 1`
  - `pass 2`
    - `code for pass 2`
  - `pass expand`
    - `expander code`
      - `optab_table`

- `insn_data`
  - generated code for machine n

- `langhook`
  - `code for language 1`
  - `code for language 2`
  - `code for language n`
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
How GCC uses target specific RTL as IR

GIMPLE_ASSIGN

(set (<dest>)) (<src>))
How GCC uses target specific RTL as IR

GIMPLE_ASSIGN  "movsi"  (set (<dest>) (<src>))

Standard Pattern Name
How GCC uses target specific RTL as IR

- **GIMPLE**
- **ASSIGN**
- "movsi"
- `(set (<dest>) (<src>))`

**Standard Pattern Name**

**Separate CGF code and MD**

- **GIMPLE**
- **ASSIGN**
- "movsi"
- "movsi"
- `(set (<dest>) (<src>))`
How GCC uses target specific RTL as IR

- GIMPLE_ASSIGN
- "movsi"
- (set (<dest>) (<src>))

Standard Pattern Name

Separate CGF code and MD

- GIMPLE_ASSIGN
- "movsi"
- (set (<dest>) (<src>))

Implement

- GIMPLE_ASSIGN
- "movsi"
- (set (<dest>) (<src>))

Unnecessary in CGF; hard code

Implement in MD
Retargetability $\Rightarrow$ Multiple MD vs. One CGF!

CGF needs:
An interface **immune** to MD authoring variations
Retargetability $\Rightarrow$ Multiple MD vs. One CGF!

CGF needs:
An interface **immune** to MD authoring variations
Retargetability $\Rightarrow$ Multiple MD vs. One CGF!

CGF needs:
An interface **immune** to MD authoring variations

Basic Approach: Tabulate

GIMPLE – RTL

struct optab []

struct insn_data []
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
```

<table>
<thead>
<tr>
<th>optab_table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|}
\hline
\text{OTI\_mov} \\
\hline
\text{mov\_optab} \\
\hline
\end{array}
\]

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Assume `movsi` is supported but `movsf` is not supported...

$(SOURCE_D)/gcc/optabs.h
$(SOURCE_D)/gcc/optabs.c

```
OTI_mov
    mov_optab
    handler
```
Assume `movsi` is supported but `movsf` is not supported...

```
$(SOURCE_D)/gcc/optabs.h
$(SOURCE_D)/gcc/optabs.c
```
Assume `movsi` is supported but `movsf` is not supported...

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

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

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

```
Assume \texttt{movsi} is supported but \texttt{movsf} is not supported...

\begin{itemize}
  \item \texttt{\$(SOURCE_D)/gcc/optabs.h}
  \item \texttt{\$(SOURCE_D)/gcc/optabs.c}
\end{itemize}

\begin{itemize}
  \item \texttt{\$(BUILD)/gcc/insn-output.c}
\end{itemize}

\begin{itemize}
  \item \texttt{optab_table}
  \item \texttt{mov_optab}
  \item \texttt{handler}
  \item \texttt{insn_code}
\end{itemize}

\begin{itemize}
  \item \texttt{insn_data}
  \item \texttt{"movsi"}
  \item \texttt{1280}
  \item \texttt{gen_movsi}
\end{itemize}

\begin{itemize}
  \item \texttt{$BUILD/gcc/insn-codes.h}$
  \item \texttt{CODE\_FOR\_movsi=1280}
  \item \texttt{CODE\_FOR\_movsf=CODE\_FOR\_nothing}
\end{itemize}
Assume *movsi* is supported but *movsf* is not supported...

**optab_table**

```
<table>
<thead>
<tr>
<th>OTI_mov</th>
<th>mov_optab</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td></td>
</tr>
</tbody>
</table>
```

```
$\$(SOURCE_D)/gcc/optabs.h$
$\$(SOURCE_D)/gcc/optabs.c$
```

**insn_data**

```
<table>
<thead>
<tr>
<th>insn_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

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

| ... |
| "gen_movsi" |
```

```
$\$(BUILD)/gcc/insn-output.c$
```

```
$\$(BUILD)/gcc/insn-codes.h$
CODE_FOR_movsi=1280
CODE_FOR_movsf=CODE_FOR_nothing
```

```
$\$(BUILD)/gcc/insn-opinit.c$
...
Assume `movsi` is supported but `movsf` is not supported...

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

```
_insn_data

... ...

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

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

```
CODE_FOR_movsi=1280
CODE_FOR_movsf=CODE_FOR_nothing
```

```
... ...
```

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Assume **movsi** is supported but **movsf** is not supported...

```plaintext
$(SOURCE_D)/gcc/optabs.h
$(SOURCE_D)/gcc/optabs.c
```

```plaintext
$(BUILD)/gcc/insn-output.c
```

---

<table>
<thead>
<tr>
<th>optab_table</th>
<th>insn_data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;movsi&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1280</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>gen_movsi</td>
</tr>
</tbody>
</table>

---

```plaintext
$BUILD/gcc/insn-codes.h
CODE_FOR_movsi=1280
CODE_FOR_movsf=CODE_FOR_nothing
```

```plaintext
$BUILD/gcc/insn-opinit.c
...
```
Assume movsi is supported but movsf is not supported...

```
Assume movsi is supported but movsf is not supported...

$(SOURCE_D)/gcc/optabs.h
$(SOURCE_D)/gcc/optabs.c

optab_table
OTI_mov
  insn_code
    CODE_FOR_movsi
SF
  insn_code
    CODE_FOR_nothing

$(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

$(BUILD)/gcc/insn-output.c
insn_data
...

"movsi"
1280
...

gen_movsi
...
```

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
<table>
<thead>
<tr>
<th>Generator</th>
<th>Generated from MD</th>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genopinit</td>
<td>insn-opinit.c</td>
<td>void init_all_optabs (void);</td>
<td>Operations Table Initialiser</td>
</tr>
<tr>
<td>gencodes</td>
<td>insn-codes.h</td>
<td>enum insn_code = { ... CODE_FOR_movsi = 1280, ... }</td>
<td>Index of patterns</td>
</tr>
<tr>
<td>genooutput</td>
<td>insn-output.c</td>
<td>struct insn_data [CODE].genfun = /* fn ptr */</td>
<td>All insn data e.g. gen function</td>
</tr>
<tr>
<td>genemit</td>
<td>insn-emit.c</td>
<td>rtx gen_rtx_movsi /* args <em>/; /</em> body */</td>
<td>RTL emission functions</td>
</tr>
</tbody>
</table>
Explicit Calls to $\text{gen}<$SPN$>$ functions

- In some cases, an entry is not made in $\text{insn\_data}$ table for some SPNs.
- $\text{gen}$ functions for such SPNs are explicitly called.
- These are mostly related to
  - 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

(define_expand "movsi"
  
  
  ""
  
  "{" /* 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)

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
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 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));
}
```

Seek index
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,sr));
}
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 Generation

```
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));
}
```

Use `icode (= 1280)`

```
#define GEN_FCN(code) insn_data[code].genfun
```
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,
}

#define GEN_FCN(code) insn_data[code].genfun

insn-output.c
insn_data[1280].genfun
= gen_movsi
RTL Generation

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));
}

#define GEN_FCN(code) insn_data[code].genfun

Execute: gen_movsi (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.
Part 4

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