Workshop on Essential Abstractions in GCC

An Overview of Compilation and GCC

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

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

30 June 2012

Outline

• Introduction to Compilation
• An Overview of Compilation Phases
• An Overview of GCC

Part 1

Introduction to Compilation

30 June 2012

Essential Abstractions in GCC

 GCC Resource Center, IIT Bombay

30 June 2012

Overview: Introduction to Compilation

Binding

Nothing is known except the problem
Overall strategy, algorithm, data structures etc.
Functions, variables, their types etc.
Machine instructions, registers etc.
Addresses of functions, external data etc.
Actual addresses of code and data
Values of variables

No. of unbound objects

Conceptualisation Coding Compiling Linking Loading Execution

Time

Essential Abstractions in GCC

 GCC Resource Center, IIT Bombay
We will look at different binding times related to compiling.

Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution

Input C statement

\[ a = b < 10 ? b : c; \]

Spim Assembly Equivalent

\[
\begin{align*}
&\text{lw } \$t0, 4(\$fp) ; \ t0 \leftarrow b \\
&\text{slti } \$t0, \$t0, 10 ; \ t0 \leftarrow t0 < 10 \\
&\text{not } \$t0, \$t0 ; \ t0 \leftarrow \neg t0 \\
&\text{bgtz } \$t0, \text{L0} : ; \ \text{if } t0 > 0 \ \text{goto } \text{L0} \\
&\text{lw } \$t0, 4(\$fp) ; \ t0 \leftarrow b \\
&\text{L0: lw } \$t0, 8(\$fp) ; \text{L0: } t0 \leftarrow c \\
&\text{L1: sw } 0(\$fp), \$t0 ; \text{L1: } a \leftarrow t0 \\
\end{align*}
\]
### High and Low Level Abstractions

**Input C statement**

\[
a = \text{b<10?b:c;}
\]

**Spim Assembly Equivalent**

\begin{align*}
\text{lw } & \textit{t0}, 4(\textit{fp}) ; \textit{t0} \leftarrow \textit{b} \quad \# \text{Is b smaller} \\
\text{slti} & \textit{t0}, \textit{t0}, 10 ; \textit{t0} \leftarrow \textit{t0} < 10 \quad \# \text{than 10?} \\
\text{not} & \textit{t0}, \textit{t0} ; \textit{t0} \leftarrow \neg \textit{t0} \\
\text{bgtz} & \textit{t0}, \textit{L0} ; \text{if t0>0 goto L0} \\
\text{lw} & \textit{t0}, 4(\textit{fp}) ; \textit{t0} \leftarrow \textit{b} \quad \# \text{YES} \\
& \textit{b L1} ; \text{goto L1} \\
\textit{L0}: & \text{lw } \textit{t0}, 8(\textit{fp}) ; \textit{L0}: \textit{t0} \leftarrow \textit{c} \quad \# \text{NO} \\
\textit{L1}: & \text{sw } 0(\textit{fp}), \textit{t0} ; \textit{L1}: \textit{a} \leftarrow \textit{t0}
\end{align*}

### Implementation Mechanisms

- **Translation** = Analysis + Synthesis
- **Interpretation** = Analysis + Execution

- **Translation** Instructions $\longrightarrow$ Equivalent Instructions
- **Interpretation** Instructions $\longrightarrow$ Actions Implied by Instructions

### Language Implementation Models

- **Analysis**
- **Synthesis**
- **Compilation**
- **Execution**
- **Interpretation**
30 June 2012 Overview: Introduction to Compilation

Language Processor Models

C, C++, Java, C#

Part 2

An Overview of Compilation Phases

The Structure of a Simple Compiler

Translation Sequence in Our Compiler: Parsing

- Grammar rules, terminals, non-terminals
- Order of application of grammar rules eg. is it \( a = b<10? \) followed by \( b:c \)?
- Values of terminal symbols eg. string “10” vs. integer number 10.
Translation Sequence in Our Compiler: Semantic Analysis

\[
\text{Input} \quad \text{AsgnStmnt} \quad \text{Lhs} = \begin{cases} E \quad &\text{a=b<10?b:c;} \\ \text{name} \quad &\text{(a,int)} \\ \text{name} \quad &\text{(b,int) < E} \\ \text{name} \quad &\text{name (c,int)} \\ \text{name} \quad &\text{num} \end{cases}
\]

Issues:
- Symbol tables: Have variables been declared? What are their types? What is their scope?
- Type consistency of operators and operands: The result of computing b<10? is bool and not int

Translation Sequence in Our Compiler: IR Generation

\[
\text{Input} \quad \text{AsgnStmnt} \quad \text{Lhs} = \begin{cases} E \quad &\text{a=b<10?b:c;} \\ \text{name} \quad &\text{(a,int)} \\ \text{name} \quad &\text{(b,int) < E} \\ \text{name} \quad &\text{name (c,int)} \\ \text{name} \quad &\text{num} \end{cases}
\]

Issues:
- Convert to maximal trees which can be implemented without altering control flow: Simplifies instruction selection and scheduling, register allocation etc.
- Linearise control flow by flattening nested control constructs

Translation Sequence in Our Compiler: Instruction Selection

\[
\text{Input} \quad \text{AsgnStmnt} \quad \text{Lhs} = \begin{cases} E \quad &\text{a=b<10?b:c;} \\ \text{name} \quad &\text{(a,int)} \\ \text{name} \quad &\text{(b,int) < E} \\ \text{name} \quad &\text{name (c,int)} \\ \text{name} \quad &\text{num} \end{cases}
\]

Issues:
- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

Translation Sequence in Our Compiler: Emitting Instructions

\[
\text{Input} \quad \text{AsgnStmnt} \quad \text{Lhs} = \begin{cases} E \quad &\text{a=b<10?b:c;} \\ \text{name} \quad &\text{(a,int)} \\ \text{name} \quad &\text{(b,int) < E} \\ \text{name} \quad &\text{name (c,int)} \\ \text{name} \quad &\text{num} \end{cases}
\]

Issues:
- Offsets of variables in the stack frame
- Actual register numbers and assembly mnemonics
- Code to construct and discard activation records

Assembly Code:

\[
\begin{align*}
\text{lw} & \quad \$t0, 4(\text{fp}) \\
\text{slti} & \quad \$t0, \$t0, 10 \\
\text{not} & \quad \$t0, \$t0 \\
\text{bgtz} & \quad \$t0, \text{L0:} \\
\text{lw} & \quad \$t0, 4(\text{fp}) \\
\text{sw} & \quad 0(\text{fp}), \$t0 \\
\end{align*}
\]
Part 3

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
- Register Transfers
- Recognizer
  - Target Program

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

Davidson Fraser: Instruction selection
- over AST using
- simple full tree matching based algorithms that generate
  - naive code which is
target dependent, and is
  optimized subsequently
Typical Back Ends in Aho Ullman Model

- Compile time evaluations
- Eliminating redundant computations

<table>
<thead>
<tr>
<th>Aho Ullman Model</th>
<th>Davidson Fraser Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Selection</td>
<td>Machine independent IR is expressed in the form of trees</td>
</tr>
<tr>
<td>Optimization</td>
<td>Machine dependent</td>
</tr>
<tr>
<td>Cost based tree pattern matching</td>
<td>Structural tree pattern matching</td>
</tr>
<tr>
<td>Key Insight: Register transfers are target specific but their form is target independent</td>
<td></td>
</tr>
</tbody>
</table>

What is GCC?

- For the GCC developer community: The GNU Compiler Collection
- For other compiler writers: The Great Compiler Challenge 😊
The GNU Tool Chain for C

Source Program → cc1 ← cpp

cc1 → gcc ← as → ld

glibc/newlib

Target Program

Why is Understanding GCC Difficult?

Some of the obvious reasons:

- **Comprehensiveness**
  GCC is a production quality framework in terms of completeness and practical usefulness

- **Open development model**
  Could lead to heterogeneity. Design flaws may be difficult to correct

- **Rapid versioning**
  GCC maintenance is a race against time. Disruptive corrections are difficult

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- **Cathedral**: Total Centralized Control
  Design, implement, test, release

- **Bazaar**: Total Decentralization
  Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”
Code errors, logical errors, and architectural errors

A combination of the two seems more sensible

The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

- **GCC Steering Committee**: Free Software Foundation has given charge
  - Major policy decisions
  - Handling Administrative and Political issues

- **Release Managers**:
  - Coordination of releases

- **Maintainers**:
  - Usually area/branch/module specific
  - Responsible for design and implementation
  - Take help of reviewers to evaluate submitted changes
Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  - C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    - Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SPARC, VAX
  - Lesser-known target processors:
- Additional processors independently supported:
  - D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios. PDP-10, TIGCC (m68k variant), Z8000, PIC24/dSPIC, NEC SX architecture

Comprehensiveness of GCC: Size

- Overall size
  - Subdirectories Files
    - gcc-4.4.2 3794 62301
    - gcc-4.5.0 4056 65639
    - gcc-4.6.0 4333 71096
  
- Core size (src/gcc)
  - Subdirectories Files
    - gcc-4.4.2 257 30163
    - gcc-4.5.0 283 32723
    - gcc-4.6.0 336 36503

- Machine Descriptions (src/gcc/config)
  - Subdirectories c files h files md files
    - gcc-4.4.2 36 241 426 206
    - gcc-4.5.0 42 275 478 206
    - gcc-4.6.0 42 275 466 259
### Line Count of gcc-4.6.0

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Comment %</th>
<th>Blank</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>18463</td>
<td>210037</td>
<td>444313</td>
<td>11.5%</td>
<td>418290</td>
<td>2626262</td>
</tr>
<tr>
<td>g++</td>
<td>22002</td>
<td>98108</td>
<td>259451</td>
<td>16.6%</td>
<td>216781</td>
<td>1429396</td>
</tr>
<tr>
<td>java</td>
<td>9287</td>
<td>58590</td>
<td>138505</td>
<td>18.0%</td>
<td>169645</td>
<td>2484699</td>
</tr>
<tr>
<td>ada</td>
<td>4605</td>
<td>60043</td>
<td>135956</td>
<td>31.7%</td>
<td>23464</td>
<td>1220466</td>
</tr>
<tr>
<td>html</td>
<td>457</td>
<td>18330</td>
<td>5669</td>
<td>3.3%</td>
<td>38145</td>
<td>213710</td>
</tr>
<tr>
<td>rites</td>
<td>88</td>
<td>17216</td>
<td>3869</td>
<td>2.1%</td>
<td>19578</td>
<td>948827</td>
</tr>
<tr>
<td>thinhair</td>
<td>2957</td>
<td>138031</td>
<td>50692</td>
<td>14.7%</td>
<td>88987</td>
<td>1986781</td>
</tr>
<tr>
<td>shell</td>
<td>147</td>
<td>44032</td>
<td>10451</td>
<td>17.9%</td>
<td>6358</td>
<td>56069</td>
</tr>
<tr>
<td>assembler</td>
<td>256</td>
<td>55317</td>
<td>10451</td>
<td>19.2%</td>
<td>5351</td>
<td>57918</td>
</tr>
<tr>
<td>yml</td>
<td>75</td>
<td>16506</td>
<td>292</td>
<td>0.8%</td>
<td>1634</td>
<td>19473</td>
</tr>
<tr>
<td>autoconf</td>
<td>989</td>
<td>10814</td>
<td>3063</td>
<td>15.1%</td>
<td>8145</td>
<td>41174</td>
</tr>
<tr>
<td>javascript</td>
<td>82</td>
<td>15585</td>
<td>2181</td>
<td>13.9%</td>
<td>1326</td>
<td>10026</td>
</tr>
<tr>
<td>tcl</td>
<td>2</td>
<td>11000</td>
<td>3176</td>
<td>34.8%</td>
<td>1431</td>
<td>18269</td>
</tr>
<tr>
<td>automake</td>
<td>67</td>
<td>9460</td>
<td>1058</td>
<td>9.9%</td>
<td>1545</td>
<td>11954</td>
</tr>
<tr>
<td>perl</td>
<td>58</td>
<td>6845</td>
<td>1358</td>
<td>2.8%</td>
<td>837</td>
<td>9398</td>
</tr>
<tr>
<td>php</td>
<td>221</td>
<td>6034</td>
<td>1375</td>
<td>19.7%</td>
<td>957</td>
<td>10925</td>
</tr>
<tr>
<td>xml</td>
<td>31</td>
<td>8086</td>
<td>346</td>
<td>4.3%</td>
<td>581</td>
<td>6864</td>
</tr>
<tr>
<td>awk</td>
<td>13</td>
<td>1745</td>
<td>395</td>
<td>18.5%</td>
<td>251</td>
<td>2393</td>
</tr>
<tr>
<td>python</td>
<td>12</td>
<td>7293</td>
<td>322</td>
<td>4.3%</td>
<td>581</td>
<td>6864</td>
</tr>
<tr>
<td>lua</td>
<td>24</td>
<td>1569</td>
<td>143</td>
<td>9.8%</td>
<td>332</td>
<td>2564</td>
</tr>
<tr>
<td>go</td>
<td>8</td>
<td>1064</td>
<td>141</td>
<td>11.9%</td>
<td>218</td>
<td>1083</td>
</tr>
<tr>
<td>sharp</td>
<td>9</td>
<td>879</td>
<td>596</td>
<td>38.6%</td>
<td>226</td>
<td>1035</td>
</tr>
<tr>
<td>cli</td>
<td>2</td>
<td>402</td>
<td>84</td>
<td>17.3%</td>
<td>13</td>
<td>309</td>
</tr>
<tr>
<td>ril</td>
<td>52</td>
<td>848</td>
<td>113</td>
<td>25.2%</td>
<td>72</td>
<td>1877</td>
</tr>
<tr>
<td>javascript</td>
<td>4</td>
<td>341</td>
<td>87</td>
<td>20.3%</td>
<td>35</td>
<td>463</td>
</tr>
<tr>
<td>bash</td>
<td>6</td>
<td>165</td>
<td>0</td>
<td>0.0%</td>
<td>12</td>
<td>177</td>
</tr>
<tr>
<td>csh</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0.0%</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>total</td>
<td>56448</td>
<td>1844010</td>
<td>606918</td>
<td>24.3%</td>
<td>1338286</td>
<td>3858178</td>
</tr>
</tbody>
</table>

### Language Files Code Comment Comment % Blank Total

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Comment %</th>
<th>Blank</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>13296</td>
<td>1254253</td>
<td>282658</td>
<td>18.4%</td>
<td>283766</td>
<td>1820691</td>
</tr>
<tr>
<td>ada</td>
<td>4196</td>
<td>630878</td>
<td>294321</td>
<td>31.6%</td>
<td>217401</td>
<td>1148598</td>
</tr>
<tr>
<td>g++</td>
<td>7418</td>
<td>184186</td>
<td>52163</td>
<td>22.1%</td>
<td>54048</td>
<td>290397</td>
</tr>
<tr>
<td>fortran fixed</td>
<td>2086</td>
<td>67988</td>
<td>1521</td>
<td>2.2%</td>
<td>9079</td>
<td>78588</td>
</tr>
<tr>
<td>assembler</td>
<td>132</td>
<td>31092</td>
<td>7243</td>
<td>18.9%</td>
<td>4770</td>
<td>43105</td>
</tr>
<tr>
<td>autoconf</td>
<td>3</td>
<td>26996</td>
<td>0</td>
<td>0.0%</td>
<td>3383</td>
<td>30389</td>
</tr>
<tr>
<td>fortran free</td>
<td>652</td>
<td>10898</td>
<td>2376</td>
<td>17.9%</td>
<td>1314</td>
<td>14588</td>
</tr>
<tr>
<td>objective c</td>
<td>391</td>
<td>10155</td>
<td>1654</td>
<td>14.0%</td>
<td>2830</td>
<td>14639</td>
</tr>
<tr>
<td>make</td>
<td>3</td>
<td>5340</td>
<td>1027</td>
<td>16.1%</td>
<td>814</td>
<td>7181</td>
</tr>
<tr>
<td>scheme</td>
<td>1</td>
<td>2775</td>
<td>153</td>
<td>5.2%</td>
<td>328</td>
<td>3258</td>
</tr>
<tr>
<td>ocaml</td>
<td>5</td>
<td>2482</td>
<td>538</td>
<td>17.8%</td>
<td>328</td>
<td>3348</td>
</tr>
<tr>
<td>shell</td>
<td>16</td>
<td>2256</td>
<td>712</td>
<td>24.0%</td>
<td>374</td>
<td>3342</td>
</tr>
<tr>
<td>awk</td>
<td>7</td>
<td>1022</td>
<td>251</td>
<td>19.7%</td>
<td>187</td>
<td>1460</td>
</tr>
<tr>
<td>perl</td>
<td>1</td>
<td>772</td>
<td>205</td>
<td>21.0%</td>
<td>137</td>
<td>1114</td>
</tr>
<tr>
<td>haskell</td>
<td>48</td>
<td>149</td>
<td>0</td>
<td>0.0%</td>
<td>16</td>
<td>165</td>
</tr>
<tr>
<td>matlab</td>
<td>2</td>
<td>57</td>
<td>0</td>
<td>0.0%</td>
<td>7</td>
<td>64</td>
</tr>
<tr>
<td>total</td>
<td>28258</td>
<td>224278</td>
<td>647591</td>
<td>22.4%</td>
<td>579484</td>
<td>3469813</td>
</tr>
</tbody>
</table>

### Line Count of gcc-4.5.0/gcc

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
<th>Comment</th>
<th>Comment %</th>
<th>Blank</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>14565</td>
<td>1368937</td>
<td>300284</td>
<td>18.0%</td>
<td>305671</td>
<td>1974892</td>
</tr>
<tr>
<td>ada</td>
<td>4402</td>
<td>645691</td>
<td>301666</td>
<td>31.8%</td>
<td>221882</td>
<td>1169239</td>
</tr>
<tr>
<td>g++</td>
<td>7094</td>
<td>197798</td>
<td>54719</td>
<td>21.7%</td>
<td>57312</td>
<td>309289</td>
</tr>
<tr>
<td>fortran fixed</td>
<td>2453</td>
<td>80403</td>
<td>1768</td>
<td>2.2%</td>
<td>11008</td>
<td>93179</td>
</tr>
<tr>
<td>assembler</td>
<td>136</td>
<td>31802</td>
<td>743</td>
<td>1.8%</td>
<td>4864</td>
<td>44097</td>
</tr>
<tr>
<td>autoconf</td>
<td>3</td>
<td>27317</td>
<td>10</td>
<td>0.0%</td>
<td>3876</td>
<td>31203</td>
</tr>
<tr>
<td>scheme</td>
<td>7</td>
<td>13725</td>
<td>1192</td>
<td>8.0%</td>
<td>1524</td>
<td>16441</td>
</tr>
<tr>
<td>fortran free</td>
<td>722</td>
<td>12201</td>
<td>2683</td>
<td>18.3%</td>
<td>1446</td>
<td>16130</td>
</tr>
<tr>
<td>objective c</td>
<td>392</td>
<td>10375</td>
<td>1721</td>
<td>14.2%</td>
<td>2838</td>
<td>14934</td>
</tr>
<tr>
<td>make</td>
<td>3</td>
<td>5886</td>
<td>1039</td>
<td>15.0%</td>
<td>854</td>
<td>7779</td>
</tr>
<tr>
<td>ocaml</td>
<td>5</td>
<td>2515</td>
<td>540</td>
<td>17.7%</td>
<td>328</td>
<td>3383</td>
</tr>
<tr>
<td>shell</td>
<td>14</td>
<td>2101</td>
<td>642</td>
<td>23.4%</td>
<td>347</td>
<td>3090</td>
</tr>
<tr>
<td>awk</td>
<td>8</td>
<td>1247</td>
<td>299</td>
<td>19.3%</td>
<td>212</td>
<td>1758</td>
</tr>
<tr>
<td>perl</td>
<td>2</td>
<td>805</td>
<td>206</td>
<td>20.4%</td>
<td>144</td>
<td>1155</td>
</tr>
<tr>
<td>haskell</td>
<td>49</td>
<td>153</td>
<td>0</td>
<td>0.0%</td>
<td>17</td>
<td>170</td>
</tr>
<tr>
<td>matlab</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>total</td>
<td>30747</td>
<td>240620</td>
<td>677035</td>
<td>22.0%</td>
<td>613025</td>
<td>3696262</td>
</tr>
</tbody>
</table>
### Why is Understanding GCC Difficult?

#### Deeper technical reasons

- GCC is not a compiler but a *compiler generation framework*
  - Two distinct gaps that need to be bridged
    - Input-output of the generation framework
      - The target specification and the generated compiler
    - Input-output of the generated compiler
      - A source program and the generated assembly program
  - GCC generated compiler uses a derivative of the Davidson-Fraser model of compilation
    - Early instruction selection
    - Machine dependent intermediate representation
    - Simplistic instruction selection and retargatibility mechanism
The Architecture of GCC

An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

  ```c
  static bool gate_tree_loop_distribution (void) {
    return flag_tree_loop_distribution != 0;
  }
  ```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!

- It is described in `common.opt` as follows

  ```
  ftree-loop-distribution
  Common Report Var(flag_tree_loop_distribution) Optimization
  Enable loop distribution on trees
  ```

- The required C statements are generated during the build

Another Example of The Generation Related Gap

- Locating the main function in `gcc-4.6.2/gcc` using `cscope -R` 7359 occurrences!

- Number of main functions in the entire tarball 11777!

- What if we do not search recursively?

Another Example of The Generation Related Gap

Locating the main function in the directory `gcc-4.6.2/gcc` using `cscope`

<table>
<thead>
<tr>
<th>File</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 collect2.c</td>
<td>1076 main (int argc, char **argv)</td>
</tr>
<tr>
<td>01 fp-test.c</td>
<td>85 main (void )</td>
</tr>
<tr>
<td>02 gcc.c</td>
<td>6092 main (int argc, char **argv)</td>
</tr>
<tr>
<td>03 gcov-dump.c</td>
<td>76 main (int argc ATTRIBUTE_UNUSED, char **argv)</td>
</tr>
<tr>
<td>04 gcov-iov.c</td>
<td>29 main (int argc, char **argv)</td>
</tr>
<tr>
<td>05 gcov.c</td>
<td>360 main (int argc, char **argv)</td>
</tr>
<tr>
<td>06 genattr.c</td>
<td>164 main (int argc, char **argv)</td>
</tr>
<tr>
<td>07 genattrtab.c</td>
<td>4820 main (int argc, char **argv)</td>
</tr>
<tr>
<td>08 genautomata.c</td>
<td>9459 main (int argc, char **argv)</td>
</tr>
<tr>
<td>09 genchecksum.c</td>
<td>97 main (int argc, char ** argv)</td>
</tr>
<tr>
<td>a gencodes.c</td>
<td>51 main (int argc, char **argv)</td>
</tr>
<tr>
<td>b genconditions.c</td>
<td>209 main (int argc, char **argv)</td>
</tr>
<tr>
<td>c genconfig.c</td>
<td>261 main (int argc, char **argv)</td>
</tr>
<tr>
<td>d genconstants.c</td>
<td>79 main (int argc, char **argv)</td>
</tr>
<tr>
<td>e genemit.c</td>
<td>830 main (int argc, char **argv)</td>
</tr>
<tr>
<td>f genenums.c</td>
<td>48 main (int argc, char **argv)</td>
</tr>
</tbody>
</table>
Another Example of The Generation Related Gap
Locating the main function in the directory gcc-4.6.2/gcc using cscope

g genextract.c 402 main (int argc, char **argv)
h genflags.c 251 main (int argc, char **argv)
i gengenrtl.c 282 main (void )
j genetype.c 4825 main (int argc, char **argv)
k genhooks.c 335 main (int argc, char **argv)
l genmdeps.c 43 main (int argc, char **argv)
m genmodes.c 1376 main (int argc, char **argv)
n genopinit.c 473 main (int argc, char **argv)
o genoutput.c 999 main (int argc, char **argv)
p genpeep.c 353 main (int argc, char **argv)
q genpreds.c 1388 main (int argc, char **argv)
r genrecog.c 2691 main (int argc, char **argv)
s lto-wraper.c 628 main (int argc, char *argv[]) 
t main.c 34 main (int argc, char **argv)
u mips-tdump.c 1393 main (int argc, char **argv)
v mips-tfile.c 655 main (void )
w mips-tfile.c 4693 main (int argc, char **argv)
x tlink.c 64 const char *main;

The GCC Challenge: Poor Retargetability Mechanism

Symptoms:
- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
- Size in terms of line counts in gcc-4.6.2 (counted using wc -l)

<table>
<thead>
<tr>
<th>Files</th>
<th>i386</th>
<th>mips</th>
<th>arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.md</td>
<td>38851</td>
<td>15534</td>
<td>30951</td>
</tr>
<tr>
<td>*.c</td>
<td>39780</td>
<td>16793</td>
<td>26165</td>
</tr>
<tr>
<td>*.h</td>
<td>17879</td>
<td>5667</td>
<td>18713</td>
</tr>
<tr>
<td>Total</td>
<td>96510</td>
<td>37996</td>
<td>75929</td>
</tr>
</tbody>
</table>
- Machine descriptions are difficult to construct, understand, debug, and enhance
Workshop Coverage

Compiler Specifications

<table>
<thead>
<tr>
<th>External View</th>
<th>Internal View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine descriptions</td>
<td>Front end hooks</td>
</tr>
<tr>
<td>Configuration and building</td>
<td>Retargetability mechanism</td>
</tr>
<tr>
<td>Gray box probing</td>
<td>Pass structure</td>
</tr>
<tr>
<td>Pass structure and IR</td>
<td>Control flow</td>
</tr>
<tr>
<td>Data Flow Analysis</td>
<td>Static and dynamic plugin mechanisms</td>
</tr>
<tr>
<td>Parallelization, Vectorization</td>
<td></td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay