Towards Incremental Compilation

Lawrence Crowl
Diego Novillo

Google™

GCC Summit 2010
Ottawa
Problem

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.

Original at http://xkcd.com/303/
Solution

- Speed up incremental compilation
- Ultimate goal: <100ms event horizon
- Intermediate goal: order of magnitude
Builds at Google

- Build environment
  - Massively parallel *make*
  - Incr. build times limited by link and longest compile
- Code base
  - C++ heavy, 60-80% compile time in FE
  - Many TUs transitively include 1,000 headers
  - Include-What-You-Use rule. No `<world.h>`
  - Continuous modification of ~entire code base
Pre-Compiled Headers

• **Approach**
  • Automatic PCH header for each TU
  • Use `-include=file_PCH.h`

• **Results**
  • Rate of change low for the most part, but some headers change often
  • Changes evenly distributed
  • All-or-nothing granularity meant low re-use rates
  • Storage costs were immense, even w/ compression
Pre-Tokenized Headers

• **Approach**
  • Factor representation so it can be used in pieces
  • Separate tokens into hunks between `#include`
  • Save `cp_token` tokens in a file cache
  • Track and validate *relevant* preprocessor symbols

• **Results**
  • 5-15% improvement even without optimizations
  • Limited by 20% of time taken by preprocessing
  • Used as basis for the next step of development
Pre-Parsed Headers

• **Approach**
  • Keep token cache partitioned into namespace-scope declarations
  • Associate a decl cache with those partitions
  • Cache in memory compile server

• **Concern**
  • Decl smashing requires cache copy-in,copy-out and swizzling
  • Overhead may exceed gain
Analysis Before Implementation

• **Approach**
  • Instrument compiler to emit “exposed” symbols and their dependences
  • Choose representative “large” TUs
  • Determine header change rate from build logs
  • Distribute header changes to decls in proportion to number of tokens in each declaration
  • Assign costs to recompile a decl in proportion to its number of actual tokens and “invisible” tokens
### Analysis Results (YMMV)

#### Change rates

<table>
<thead>
<tr>
<th>decls</th>
<th>avg change rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>98%</td>
<td>0.1%</td>
</tr>
<tr>
<td>1%</td>
<td>0.2%</td>
</tr>
<tr>
<td>1%</td>
<td>1.3%</td>
</tr>
</tbody>
</table>

#### Decl coverage

<table>
<thead>
<tr>
<th>decls</th>
<th>avg max coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>98%</td>
<td>6%</td>
</tr>
<tr>
<td>1%</td>
<td>15%</td>
</tr>
<tr>
<td>1%</td>
<td>51%</td>
</tr>
</tbody>
</table>

#### Expected speedups vs declarations changed

<table>
<thead>
<tr>
<th>decls changed</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.0</td>
</tr>
<tr>
<td>5</td>
<td>4.9</td>
</tr>
<tr>
<td>20</td>
<td>3.5</td>
</tr>
<tr>
<td>100</td>
<td>1.3</td>
</tr>
<tr>
<td>500</td>
<td>1.1</td>
</tr>
</tbody>
</table>

#### Decl coverage by class

<table>
<thead>
<tr>
<th>class</th>
<th>max coverage</th>
<th>change rate</th>
<th>change rate (flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>73%</td>
<td>0.0%</td>
<td>24%</td>
</tr>
<tr>
<td>3rd party</td>
<td>2%</td>
<td>0.2%</td>
<td>4%</td>
</tr>
<tr>
<td>base</td>
<td>64%</td>
<td>1.6%</td>
<td>12%</td>
</tr>
<tr>
<td>core</td>
<td>49%</td>
<td>3.3%</td>
<td>10%</td>
</tr>
<tr>
<td>application</td>
<td>11%</td>
<td>94.9%</td>
<td>50%</td>
</tr>
</tbody>
</table>