


GCC – 3.4 and Beyond

1st June 2004
Ottawa Canada LUG
Nathan Sidwell
nathan@codesourcery.com

 www.codesourcery.com


GCC 3.4

- Released 18 April 2004
- Unit-at-a-time compilation mode
- Precompiled Headers
- New C++ Parser
- C++ standard library optimizations

 www.codesourcery.com


Unit at a Time

- Allows inter-procedural optimizations
 - Better inlining
 - Removal of unreachable functions and variables
 - Optimized parameter passing for local functions
 - Uses registers on i86

 www.codesourcery.com


Compilers 101

- Parse the source code into a high-level tree-like representation called AST
 - Semantic level similar to source language
 - Can optimize at this level
- Convert trees to low-level assembly-like representation called RTL
 - Semantic level similar to target language
 - Can optimize at this level too

 www.codesourcery.com


GCC's Development

- GCC 1 and Pre-EGCS GCC-2 parsed each statement and then converted that to RTL
 - RTL was then optimized
- GCC 2.95 had function-at-a-time translation
 - Parse whole function to AST
 - Do function inlining at this level
 - Then convert that to RTL
 - Optimize the RTL

 www.codesourcery.com

Unit at a time

- Parse whole file into AST
- Determine call graph
- Inline as appropriate
- Remove unreachable functions
- Convert to RTL
- Optimize the RTL

 www.codesourcery.com

User Impact

- On conforming code there is no functionality impact
- Noticeable in system code
 - Functions and variables are reordered
 - `-fno-unit-at-a-time` (will go away)
- Program growth limited (`-param inline-unit-growth`)
 - Inliner can consider whole file, rather than local function



www.codesourcery.com

Forcing inlining

- 'inline' will not force inlining
 - Use `__attribute__((always_inline))`
- Not 'inline' will not force not inlining
 - Use `__attribute__((noinline))`



www.codesourcery.com

Zero Initialization

- `static int i = 5;` → .data section
- `static int j;` → .bss section
- `static int k = 0;` → .bss section
 - Used to go in .data and take up disk space
 - `-fno-zero-initialized-in-bss` to inhibit



www.codesourcery.com

Coverage

- Program coverage improved
 - Merges information at runtime
 - Allows profile directed optimization
 - Static branch prediction, loop unrolling
 - Easy to use `-fprofile-generate` and `-fprofile-use` flags



www.codesourcery.com

Variable Use Splitter

- ```
int t;
t = baz(); ... foo (t); // use 1
t = baz(); ... foo (t); // use 2
```
- Create two instances of `t` and allocate them independently
  - Needs DWARF3 to be debuggable



www.codesourcery.com

## Pre Compiled Headers

- Works when all your source #includes the same single giant header file
  - `#include <Carbon/Carbon.h>`
  - Creates `Carbon/Carbon.pch`
- Undocumented version-specific format
  - Essentially serializes memory to disk using structure walking routines
- Can give 2x speedup



www.codesourcery.com

## New C++ Parser

- Hand written recursive descent parser with backtracking
  - C++'s grammar is not LALR(1)
  - Much better standard conformance
  - Implements two-stage lookup

CodeSourcery

www.codesourcery.com

## Two Stage Lookup

- In a template definition, names can be bound to objects/functions
  - At parse time
  - At instantiation time
- GCC used to do it all at instantiation time
  - Correct programs can tell the difference

CodeSourcery

www.codesourcery.com

## Two Stage Lookup

- Lookup at parse time
  - If the meaning of the name does not depend on a template parameter
- Lookup at instantiation time
  - If the meaning of the name depends on a template parameter

CodeSourcery

www.codesourcery.com

## Example

- `void Foo (int); // #1`
- `template<typename T>`  
`void Baz (T t) {`  
`Foo (1.5f); // non-dependent`  
`Foo (t); // dependent`  
`}`
- `void Foo (float);`
- `Foo (1.5f);`

CodeSourcery

www.codesourcery.com

## C++ Std Library

- Optimizations
  - Streamlined streambuf, filebuf
  - Cached locale information
  - Memory/speed optimizations
  - Uses GCC builtins
  - String optimizations

CodeSourcery

www.codesourcery.com

## Gone

- Cast as lvalue – `(char) i = 5`
  - Gone in C++, will go in C
- Conditional as lvalue – `(a ? b : c) = 5`
  - Will go in C, legal in C++
- Compound as lvalue – `(a, b) = 5`
  - Will go in C, legal in C++
- `-fwritable-strings`, `-fvolatile`
  - Deprecated/removed

CodeSourcery

www.codesourcery.com

## Gone in G++

- Named return value
  - Automatically optimized
- Default values in function types
  - Will go
- ARM style name-injection of friends
  - Will go

CodeSourcery

www.codesourcery.com

## Backends

- More backends use the DFA scheduler
  - Accurately models a CPU's pipeline
- I86 MMX and SSE vector operands passed in registers
  - An ABI change

CodeSourcery

www.codesourcery.com

## ... and Beyond

- Significant change to internal optimizers
- Optimize at tree level
  - More semantic information (like types)
- Use Static Single Assignment
  - Powerful representation used by modern compilers

CodeSourcery

www.codesourcery.com

## Static Single Assignment

- Representation where variables are only assigned to in one location
  - Subsequent assignments create new instances
  - Makes data flow analysis much simpler
- $i = 1; \dots i_{1\dots}; i = 2; \dots i_{2\dots};$ 
  - $i_1 = 1; \dots i_{1\dots}; i_2 = 2; \dots i_{2\dots};$
  - Much like the splitter pass

CodeSourcery

www.codesourcery.com

## Ifs

- ```
if (cond)
  j = 5;
else
  j = 6;
k = j; // which j?
```
- ```
if (cond)
 j1 = 5;
else
 j2 = 6;
k1 = PHI (j1, j2);
```

CodeSourcery

www.codesourcery.com

## Optimizations

- The RTL optimizers struggle to perform
  - They have no type information
  - Global optimizations are awkward
- The tree structure is hard to optimize
  - It is too complicated
- Use a simplified tree structure, GIMPLE
  - Trees complexity is restricted, but retains the high level information

CodeSourcery

www.codesourcery.com

## Optimizers

---

- Many of the RTL optimizers will be replaced by tree-SSA optimizers
  - Use `-ftree-dump-all` to see the C-like intermediate representations
  - Optimization at SSA can be faster than at RTL

## 3.5 or 4.0?

---

- Tree-SSA is
  - Not a major functionality improvement
  - A major technology improvement
  - An enabler for better optimizations
- Technical considerations → 3.5
- Marketing considerations → 4.0