

Recent and future GCC development

Michael Matz
SuSE Labs



Novell.



Not so recent achievements

User visible

- precompiled headers (“technical preview”)
- much closer to ISO C++
- better performing libstdc++, also closer to ISO C++
- -fvisibility
- character set support in C/C++

internal

- variable tracking
- reorder functions
- superblock formation



precompiled headers

- to generate: `gcc -c -o bla.h.gch bla.h`
- to use: do nothing (for each include “x.h” GCC will try to first load “x.h.gch”)
- consider `-include` option

ELF visibility

Issue

- without support of the binary format, every class member and associated C++ symbols to implement the language (type information, vtables, ...) need to be exported, i.e. globally visible
- many exported symbols, but only used inside the DSO (internal classes for instance), --> relocation time
- ELF can do better

Solution

- hide symbols (-fvisibility for default, attribute() for overriding)



ELF visibility (cont.)

```
class __attribute__((visibility("hidden")))
Foo
{
    int foo1();
    void foo2();
};
```

just hiding all inlines can reduce the symbol count by 40%
and binary size by 10% (-fvisibility-inlines-hidden)



Character set support

supports UTF-8 input files (at least for strings, not yet for identifiers everywhere):

```
char *str = "Schöne Grüße";
```

```
wchar *wstr = L"Schöne Grüße";
```

assuming UTF-8 input and 4 byte wchar:

str is UTF-8 encoded

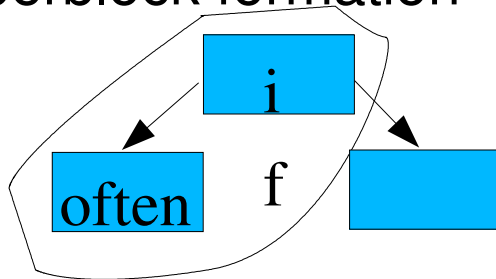
wstr is UCS-4 encoded

Internal non-recent changes

Variable tracking

- debug information where a variables content is placed (memory/register) with location lists; this and unwind sections enable debugging without frame pointer

Superblock formation



unit-at-a-time (1.3% on SPECint2000, Athlon)

- unreferenced static vars/functions get deleted
- discover local functions



Recent changes

realistic code size estimates

improved PGO

- value profiles

$a = b \text{ div } c \rightarrow a = (c == X) ? b \text{ div } X : b \text{ div } c;$

- faster profile merging (speedup of profiled programs)
- usage in loop optimization
- 11% speedup when GCC itself is PGO'ed (-O0), 7.5% at PR8361 (-O2)
- support of multiple processes

Recent development

Tree-SSA

- modern compiler framework based on a tree-like IR
- more than 40 new passes optimizing on this IR
- only slightly slower compiler ;-)

LNO (loop nest optimizer)

- partly merged, will be in 4.0 / 4.1
- loop optimizer based on Tree-SSA framework

Fortran 95 frontend

- newly written frontend, based on GENERIC/GIMPLE (IR of Tree-SSA)



Recent Development 2

Infrastructure

- edges-in-vectors conversion
- CFG-based inliner (prerequisite for PGO'ed inlining)

Future and WIP

integrate rest of LNO + autovect

- autovectorizer (cross-platform!)
 - unknown loop bounds
 - support arbitrary alignment
 - linear loop transforms (changing the order how an array is traversed)
- prefetching mem accesses

IPO + infrastructure

- reduce representation bloat
- intermediate goal: SSA-based inliner
- immediate use integration
- zone garbage collector
- --enable-mapped-location (pch support)



Future and WIP 2

Optimizer improvements

- tree-profiling branch
- alias improvements for structure fields (accesses to different fields of a struct can't alias)
- value range propagation
- escape analysis of variables
- variable expansion optimization
- SMS improvements (overlapping iterations of loops without explicit unrolling)

Restructuring GIMPLE / RTL interaction

- expand most complicated constructs at GIMPLE, not at RTL
- do less work in RTL, reduce optimizers working on RTL
- tree combiner

Future 3

C++

- stabilize C++ API, i.e. libstdc++
- ABI was again changed in 4.0 due to one bug, but we are getting there ;)

Objective-C++

Java

- use/integrate eclipse compiler (design state)
- fix GCJ ABI

Mono?

Questions?

Thank you!