A comparison of the DWARF debugging information produced by LLVM and GCC
by
Keith Walker

A lightning talk at
the LLVM Conference on 7th November 2013
Summary

- Debug information is mostly comparable to GCC.
- LLVM is more aggressive at doing function inlining resulting in larger debug tables at –O2.

Areas for improvement:
- Remove duplication of inlined function parameter information.
- Enumerators missing in pubnames.
- System included files in line table if referenced by symbols.
- Generate stack information for ARM.
- Generate stack information for AARCH64 function epilogues.
- `debug_types` can provide significant improvements in reducing size of the debug information.

- No sibling entries reduce debug info sizes at the cost of loading speed (change can be around 10%).
What was used in the comparison

- Comparison performed on a C program of about 70,000 lines of source in 58 source files and 16 header files
- No library code linked in ... all debug information from the compiled sources
- Uses the same header files
- GCC 4.8.2 using:
  arm-none-eabi-gcc –g –gpubnames –gstrict-dwarf
- LLVM tip of trunk using:
  clang –target=arm-none-eabi -g
DWARF debug section sizes

Effect of function inlining

- LLVM -O0
- LLVM -O1
- LLVM -O2
- GCC -O0
- GCC -O1
- GCC -O2
DWARF Tags (symbols)

- formal_parameter
- inlined_subprogram
- lexical_block
- variable
- unspecified_parameters
- label

Legend:
- LLVM -O0
- LLVM -O1
- LLVM -O2
- GCC -O0
- GCC -O1
- GCC -O2
Due to inlined function parameters

Due to inlined functions
Are siblings attributes worth the space?

- LLVM deliberately does not generate entries the DW_AT_sibling attribute due to the size the occupied in the .debug_info section.

- GCC does generate DW_AT_sibling entries:
  - Take ~4% of the .debug_info section size (in this example).
  - Contribute no actual debug information.
  - Implemented using 4-byte entries.
  - ... But of 1508 entries only 96 have a value greater than 255.

- What is the effect of not having sibling entries in a debugger?
  - ... Can affect loading speed by about 10%.
Line Table Differences

- LLVM does not include system header files/directories in the line table.
  - But symbols defined in the system header files have a file/line reference .... The file entry pointing at the current source file.

- LLVM generates “End of Function Prologue” entries.
  - Needs support in debugger but will help to remove the need for heuristics used by debuggers to find the prologue end.

- LLVM (sometimes) marks the function prologue instruction as a place not to put a source level breakpoint.

- Neither compiler generates “Start of Function Epilogue” entries.
Stack Frame Differences

- LLVM does not generate `.debug_frame` information for the selected arm-none-eabi target.
- GCC generates the changes that occur in a function epilogue, LLVM does not (comparing AARCH64).

The effect of this is that when instruction stepping the function epilogue in LLVM generated code, register usage in the called function may be displayed incorrectly.
The .debug_types section was added in DWARF 4 in order to help common up duplicated type information. This removal of duplicate information can result in smaller file sizes and as a consequence means debuggers have less debugging information to process and load.

I am looking forward to the completion of the current development effort to implement support for the .debug_types section.

GCC already has support for .debug_types, so we can see the possible benefits of adding this support.
DWARF Debug section sizes (GCC)

Main sections that change

GCC -O2
GCC -O2 (types)

abbrev
ranges
frame
info
line
loc
pubnames
pubtypes
ranges
str
types
Questions?
DWARF debug section sizes

![Graph showing DWARF debug section sizes for different compilers and optimization levels.](image-url)
DWARF Attributes (symbol properties)

- abstract_origin
- accessibility
- byte_size
- call_file
- call_line
- comp_dir
- const_value
- data_member_location
- decl_file
- decl_line
- declaration
- encoding
- entry_pc
- external
- frame_base
- high_pc
- inline
- language
- location
- low_pc
- name
- producer
- prototyped
- ranges
- sibling
- stmt_list
- type
- upper_bound

- LLVM -O0
- LLVM -O1
- LLVM -O2
- GCC -O0
- GCC -O1
- GCC -O2
Thank You