Fission
GCC Cauldron
July 9, 2012

Cary Coutant
Doug Evans
Sterling Augustine
Goal: Debug Info All the Time

Linking large apps takes too much time and space
• Debug info expands .o files by 5x.
• For a distributed build, all that data has to be copied to remote server.
• Compressed debug info reduces that by 30–40%.

GDB time-to-main is too long
• Fast-lookup index reduces time-to-main from minutes to seconds.
• Link-time gdb index generation from raw debug info is too slow—need to use pubnames/pubtypes tables.

Solution: Split the bulk of the debug info into separate files and replace with fast-lookup index.
What’s Taking So Much Space?

- Debug relocations
- Debug strings
- Debug info
- Relocations
- Text & Data
- Overhead
Observations

• Relocations for debug info account for more than half of the debug info overhead in .o files.
• Debug strings account for 10% of the total size of the .o files.
• References to strings account for 90% of the debug relocations.
• Line number tables, range lists, location lists, etc., are small relative to debug info. (But with more optimization, location lists and their relocations continue to grow.)
Step 1: Link-Time Generation of GDB Index

• Modify GCC to emit all public names needed for fast lookup:
  • enumerator constants
  • “const int” vs. “int const”, anonymous namespaces, etc. (demangler as canonical form)
• Modify linker to read .debug_pubnames and .debug_pubtypes to build .gdb_index.
• For DWARF-5, consider developing new, improved, fast-lookup tables.
Step 2: Eliminate Relocations to Debug Strings

• Add a new section, `.debug_str_offsets`, which holds offsets to strings in the `.debug_str` section.

  • Even with explicit relocations, the coalescing of multiple references to a single string will reduce the number of string relocations.

  • This section can be implicitly relocated by the linker if desired, allowing the compiler to generate unrelocated offsets.

  • With separate debug info files, no relocations are necessary.

• Add a new `DW_TAG_compile_unit` attribute, `DW_AT_str_base`, whose value points to the base of the compilation unit’s contributions to `.debug_str_offsets`. (Optional if using separate debug info files.)

• Add a new FORM code, `DW_FORM_str_index`, representing an index to a slot in `.debug_str_offsets`, relative to the start of the compilation unit’s base offset.
Step 3: Isolate References to Loadable Segments

• Add a new section, .debug_addr, which holds relocatable addresses referencing locations in a loadable segment.

• Add a new attribute to the compile unit DIE, DW_AT_addr_base, whose value points to the base of the compile unit’s contributions to .debug_addr.

• Add a new attribute to the compile unit DIE, DW_AT_ranges_base, whose value points to the base of the compile unit’s contributions to .debug_ranges.

• Add a new FORM code, DW_FORM_addr_index, representing an index to a slot in .debug_addr, relative to the compilation unit’s base offset.

• Add two new OP codes, DW_OP_addr_index and DW_OP_const_index.

• Modify format of .debug_loc section so that start address is an index to a slot in .debug_addr, and end address is either an index or the length of the range.
Step 4: Split Bulk of Debug Info

• For references to ranges in .debug_ranges, use raw unrellocated offsets, relative to offset given by DW_AT_ranges_base.

• Move the following sections to a separate .dwo file (adding “.dwo” to the section names):
  - .debug_abbrev
  - .debug_info and .debug_types
  - .debug_loc
  - .debug_str and .debug_str_offsets
  - .debug_macinfo or .debug_macro

• Write a skeleton .debug_lines.dwo section to the .dwo file to provide file names needed by .debug_types.dwo sections.
Step 5: Add Index Information

• Write a skeleton compile unit DIE in the .o file with attributes:
  DW_AT_comp_dir
  DW_AT_stmt_list
  DW_AT_low_pc/high_pc or DW_AT_ranges
  DW_AT_dwo_name and DW_AT_dwo_id
  DW_AT_addr_base
  DW_AT_ranges_base

• Keep the following debug sections in the .o file:
  .debug_info and .debug_types (skeletons)
  .debug_abbrev (for skeleton info and types sections)
  .debug_lines
  .debug_ranges
  .debug_addr
  .debug_pubnames and .debug_pubtypes
  .debug_aranges
Step 6: Package .dwo Files

- GDB can use the .dwo files in the build tree during development.
- When releasing a binary for production use, we need to collect the .dwo files into a convenient package.
- While collecting the .dwo files, eliminate duplicate types and merge string tables.
- Optionally, run dwz while collecting .dwo files.
Results (so far)

<table>
<thead>
<tr>
<th>.o (orig)</th>
<th>.o (split)</th>
<th>.dwo</th>
<th>Executable</th>
</tr>
</thead>
</table>

- .o (orig)
- .o (split)
- .dwo
- Executable

Legend:
- .gdb_index
- Debug Relocations
- .debug_pubnames/types
- .debug_ranges
- .debug_line
- .debug_loc
- .debug_addr
- .debug_str
- .debug_str_offsets
- .debug_info/types
- Relocations
- Text & Data
- Overhead
Status

- GCC patches have all been submitted for review; all but the final one have been committed.
- GDB support is in trunk.
- Gold support for building .gdb_index is in trunk (except for new DW_AT_pubnames attribute handling).
- Packaging tool is not yet done (expected by end of July).
- DWARF proposal has been submitted for review for version 5.
Future Improvements

• Coalesce redundant entries in `.debug_addr` (we still see many duplicates due to `.LVL` labels for location lists).
• Replace `.gdb_index` with better fast-lookup tables.
• Eliminate redundancy between `.debug_addr` and `.debug_ranges` tables.
Thank You