Improving debug info for optimized away parameters

Jakub Jelínek
Red Hat, Inc.

Debug improvements through VTA

- `dwarflint --check=locstats \`
  `  --locstats:tabulate=0.0:10,99`

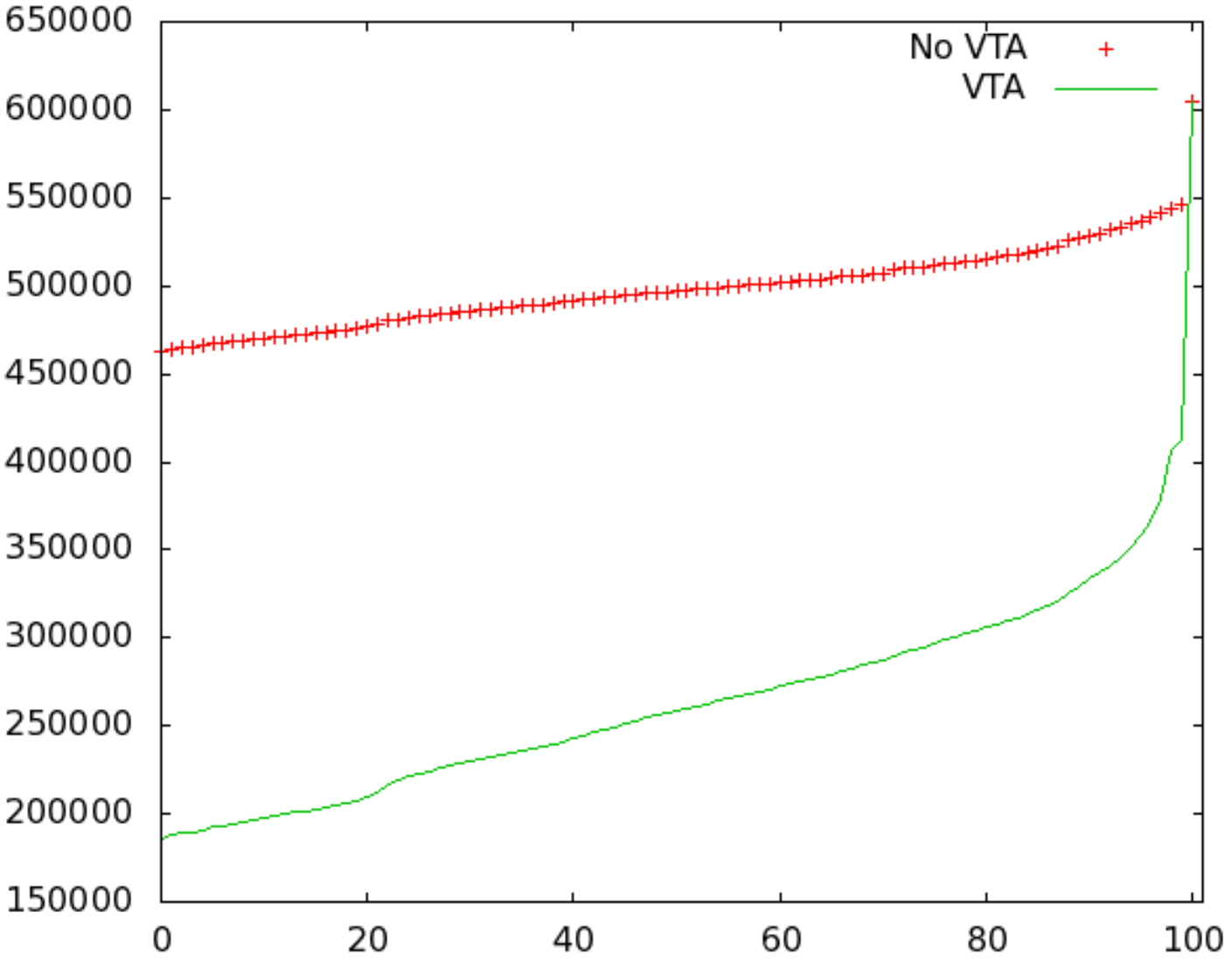
-fno-var-tracking-assignments

cov%	samples	cumul
0.0	462200/76%	462200/76%
0..10	7932/1%	470132/77%
11..20	6782/1%	476914/78%
21..30	8727/1%	485641/80%
31..40	5814/0%	491455/81%
41..50	5695/0%	497150/82%
51..60	4712/0%	501862/82%
61..70	5436/0%	507298/83%
71..80	7909/1%	515207/85%
81..90	13326/2%	528533/87%
91..99	18189/3%	546722/90%
100	58482/9%	605204/100%

-fvar-tracking-assignments

cov%	samples	cumul
0.0	185779/30%	185779/30%
0..10	11683/1%	197462/32%
11..20	12419/2%	209881/34%
21..30	20102/3%	229983/38%
31..40	12891/2%	242874/40%
41..50	16269/2%	259143/42%
51..60	13859/2%	273002/45%
61..70	14223/2%	287225/47%
71..80	18975/3%	306200/50%
81..90	27019/4%	333219/55%
91..99	79826/13%	413045/68%
100	192161/31%	605206/100%

Debug improvements through VTA



Optimized away parameters

```
void foo (int a, int b)
{
  int c = a;
  a = a + 4;
  bar (b, a);
  baz (6);  // Evaluate a, b and c here.
}

int test (int a, int b)
{
  foo (a, a + 1);
  return a;
}
```


Optimized away parameters

- Variables a, b and c have no location during/after call to bar
- The caller has the value saved
- The user can look the value up manually
- Or better, let the compiler and debugger cooperate to look the value up for the user automatically

Callee side information

- Could be just a flag whether parameter is unmodified
  - That allows to express just parameter b, but not a and c
- Need something usable in location expressions
  - A variable can have value of a parameter only in certain range of instructions
  - Its value can be a complex expression based on the parameter value
- Should be usable both for looking up the value in the caller and for non-interactive debugging

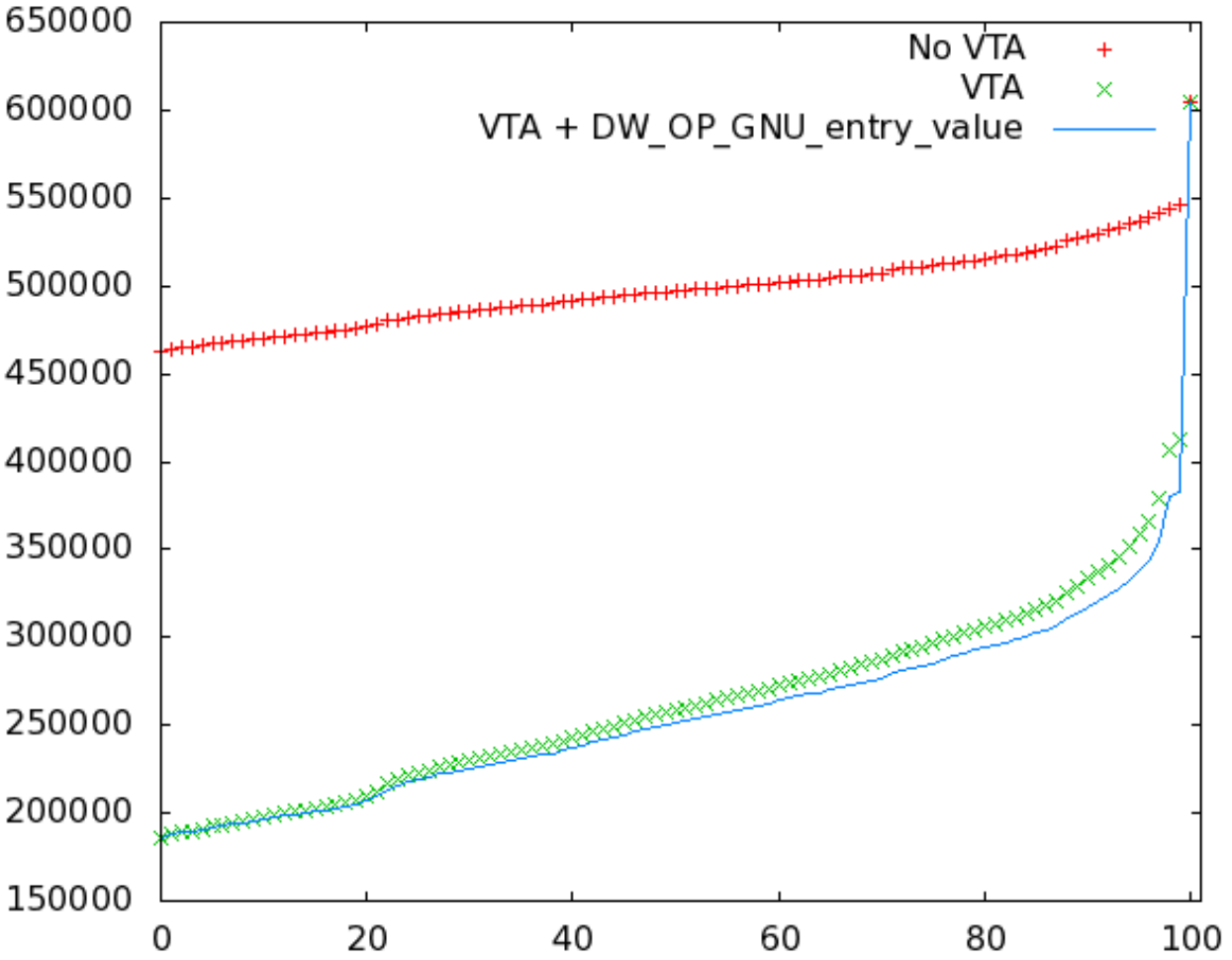
Callee side information

- DW_OP_GNU_entry_value
- operands:
  - length of block (uleb128)
  - block
    - simple register
    - DWARF expression
- DW_OP_GNU_entry_value <1, DW_OP_reg4>
- DW_OP_GNU_entry_value <3,
  DW_OP_breg4 <16>
  DW_OP_deref>

Callee side information

- Implemented in GCC just by changing var-tracking and dwarf2out
- In var-tracking inject new RTL expression – **ENTRY_VALUE** – to cselib locations of **VALUE** each parameter has upon entry (and optionally also **VALUE** pointed by it)
- In dwarf2out just expand the new RTL into corresponding DW_OP_GNU_entry_value op with arguments
- In the future could be also usable for unused parameters that are never touched in the code

Callee side information



Call site information

- Call insn address
- What callee is called
- For each parameter
  - Some identification of the parameter
  - DWARF expression how it can be computed, using only objects not clobbered by the call
- Should be extensible, -g3 could give more detailed info, e.g. for for static call graph analysis, etc.

Call site information

- DW_TAG_GNU_call_site
  - DW_AT_low_pc .LVL*NN*
  - DW_AT_abstract_origin <reference to callee DIE>
  - DW_TAG_GNU_call_site_parameter
    - DW_AT_location DW_OP_reg5
    - DW_AT_GNU_call_site_value DW_OP_lit7
  - DW_TAG_GNU_call_site_parameter
    - DW_AT_location DW_OP_reg4
    - DW_AT_GNU_call_site_value DW_OP_breg3 16 DW_OP_lit3 DW_OP_shl

  ...

Tail calls

- Tail calls don't show up in the unwind info ⇒ can't use call site parameter values
  - DW_TAG_GNU_call_site has DW_AT_abstract_origin or DW_AT_GNU_call_site_target attribute to allow checking the caller
- If current function could tail call to itself, directly or indirectly, call site parameter values can't be used reliably
  - Dynamic test whether a function can tail call itself, using DW_AT_GNU_tail_call flags on tail calls and DW_AT_GNU_all_tail_calls flag on the current function DIE

Tail calls

- Occassionally it is possible to reconstruct tail call backtraces, including parameter values
  - If callee doesn't match target in call site, but there is just one possible sequence of tail calls leading to it
  - Sometimes just the first frame or a few frames
  - Implemented already in GDB

Parameters passed by reference

- In addition to the parameter value itself it is sometimes interesting to know what it points to
  - Fortran scalar parameters when not using %VAL
  - C++ parameters with type reference to scalar
- DW_AT_GNU_call_site_data_value attribute on DW_TAG_GNU_call_site_parameter DIE

Call site implementation in GCC

- Again just in var-tracking and dwarf2out
- In var-tracking
  - Ensure registers used for parameter passing are tracked
  - Before processing a call insn with cselib remember **VALUE**s of each of the parameter (cselib invalidates them)
  - During note emission create special **NOTE_INSN_CALL_ARG_LOCATION**, avoid using location cache in that case
- In dwarf2out find out the right DIE parent for each call site and emit it there

Example backtrace

```
#0  ggc_internal_alloc_stat (size=<value optimized out>)
    at ../../gcc/ggc-page.c:1152
#1  0x0000000000833ba6 in ggc_internal_cleared_alloc_stat
    (size=40) at ../../gcc/ggc-common.c:205
#2  0x0000000000b44272 in ggc_internal_zone_cleared_alloc_stat
    (s=40, z=<value optimized out>) at ../../gcc/ggc.h:316
#3  ggc_alloc_zone_cleared_tree_node_stat (s=40, z=<value
    optimized out>) at ../../gcc/ggc.h:346
#4  make_tree_vec_stat (len=<value optimized out>)
    at ../../gcc/tree.c:1641
#5  0x0000000000b4d37b in build_int_cst_wide
    (type=0x7ffff17a0738, low=<value optimized out>, hi=<value
     optimized out>) at ../../gcc/tree.c:1231
#6  0x00000000008537bf in gimplify_expr (expr_p=0x7ffff1784dc0,
    pre_p=0x7fffffffdcb0, post_p=0x7fffffffda48,
    gimple_test_f=0x849a40 <is_gimple_reg_rhs_or_call>,
    fallback=1) at ../../gcc/gimplify.c:6814
```


Example backtrace

```
#0  ggc_internal_alloc_stat (size=40)
    at ../../gcc/ggc-page.c:1152
#1  0x0000000000833ba6 in ggc_internal_cleared_alloc_stat
    (size=40) at ../../gcc/ggc-common.c:205
#2  0x0000000000b44272 in ggc_internal_zone_cleared_alloc_stat
    (s=40, z=<value optimized out>) at ../../gcc/ggc.h:316
#3  ggc_alloc_zone_cleared_tree_node_stat (s=40, z=<value
    optimized out>) at ../../gcc/ggc.h:346
#4  make_tree_vec_stat (len=1)
    at ../../gcc/tree.c:1641
#5  0x0000000000b4d37b in build_int_cst_wide
    (type=0x7ffff17a0738, low=0, hi=0) at ../../gcc/tree.c:1231
#6  0x0000000000b4d9f7 in build_int_cst (type=<value optimized
    out>, low=0) at ../../gcc/tree.c:1039
#7  0x00000000008537bf in gimplify_expr (expr_p=0x7ffff1784dc0,
    pre_p=0x7fffffffdcb0, post_p=0x7fffffffda48,
    gimple_test_f=0x849a40 <is_gimple_reg_rhs_or_call>,
    fallback=1) at ../../gcc/gimplify.c:6814
```


Example backtrace

```
#0  ggc_internal_alloc_stat (size=40)
    at ../../gcc/ggc-page.c:1152
#1  0x0000000000833ba6 in ggc_internal_cleared_alloc_stat
    (size=40) at ../../gcc/ggc-common.c:205
#2  0x0000000000b44272 in ggc_internal_zone_cleared_alloc_stat
    (s=40, z=<value optimized out>) at ../../gcc/ggc.h:316
#3  ggc_alloc_zone_cleared_tree_node_stat (s=40, z=<value
    optimized out>) at ../../gcc/ggc.h:346
#4  make_tree_vec_stat (len=1)
    at ../../gcc/tree.c:1641
#5  0x0000000000b4d37b in build_int_cst_wide
    (type=0x7ffff17a0738, low=0, hi=0) at ../../gcc/tree.c:1231
#6  0x0000000000b4d9f7 in build_int_cst (type=<value optimized
    out>, low=0) at ../../gcc/tree.c:1039
#7  0x00000000008537bf in gimplify_expr (expr_p=0x7ffff1784dc0,
    pre_p=0x7fffffffdcb0, post_p=0x7fffffffda48,
    gimple_test_f=0x849a40 <is_gimple_reg_rhs_or_call>,
    fallback=1) at ../../gcc/gimplify.c:6814
```


Conditionally optimized away

- In

```
extern GTY(()) tree integer_types[itk_none];
#define integer_type_node integer_types[itk_int]
tree build_int_cst (tree type, HOST_WIDE_INT low)
{
  if (!type)
    type = integer_type_node;
  return build_int_cst_wide (type, low,
                             low < 0 ? -1 : 0);
}
```

  type has no location when inside of the tail call, because integer_type_node might be clobbered by the call

- If type wasn't NULL upon entry, we know its value

Other uses of call site information

- Static callgraph analysis
- Enhanced value printing in backtraces

```
void foo (char *p)
{
  /* some code * /
  p = strchr (p, 0);
  bar ();
  /* some further code */
}
```


Statistics

- For million backtraces in cc1plus 2194055 entry value lookups
- 960286 (43.8%) of them resulted in successful value retrival
- .debug_info growth – 23.19MB to 34.53MB
- .debug_loc growth – 43.69MB to 46.26MB

Completely optimized away parameters

- Parameter y is not passed at all in:

```
__attribute__((noinline)) static int
foo (int x, int y, int z)
{
  return x + z;
}


int bar (int x, int y, int z)
{
  return foo (x, y - 4, z)
         + foo (x + 8, y, z - 8);
}
```


Completely optimized away parameters

- Can only happen if compiler can adjust all callers, so debug info for both sides can be adjusted too
- Could use something like DW_OP_GNU_parameter_ref
  - 1 operand – reference to the optimized away parameter DIE
- In call site information can use DW_AT_abstract_origin instead of DW_AT_location for such parameter and value in DW_AT_GNU_call_site_value

Floating point expressions

- Currently no locations are created for floating point expressions, or for long long expressions on 32-bit targets
- Representable with current DWARF expressions and using DW_OP_piece, but most of the floating point arithmetics DWARF subroutines would be huge
- Debuggers already handle floating point for user expressions and inferior calls

Floating point expressions

- Change the internal DWARF stack representation from stack of integral words to stack of pairs of type and union of various representations (word, double word, IEEE 754 float/double/extended/quad, fixed point and decimal types, etc.)
- Either use DW_OP* prefixes for the kind of operation, or derive the type of operation from types of arguments. Have DW_OP* ops for conversion between types or reinterpretation of one type as another type (like VIEW_CONVERT_EXPR)

Questions?