# Improving LLVM DebugInfo to Recover Optimized-out Function Parameters

Ananth Sowda(Cisco), Djordje Todorovic(RT-RK/Cisco),
Nikola Prica(RT-RK/Cisco), Ivan Baev(Cisco)

# Overview of talk

- Debugging software release products
- Finding parameter values in parent frame
- DWARF 5 extensions
- Implementation
- IR Metadata approach
- Very late MIR approach
- Results
- Future work

# Debugging software release products

- Production software is built with -O2/-O3 and -g
- Customer issues cannot be recreated with debug build
- Bug-fixing relies on core-file analysis of optimized code
- Problem: Large number of function parameters are optimized-out in backtrace - Where are my input values?

# Finding parameter values in caller frame

- For parameters, there is a natural back-up: parent frame
- Outgoing parameters may be available: constants, stack, callee-saved registers
- If user can find them manually, debugger with help from compiler could automate parameter recovery
- @Entry values for parameters in backtrace
- Special case: Unmodified parameters

# Backtrace with parameters optimized-out

```
Core was generated by `./n.x'.
Program terminated with signal SIGABRT, Aborted.
#0  0x0000003cae432495 in raise () from /lib64/libc.so.6

(gdb) bt
#0  0x0000003cae432495 in raise () from /lib64/libc.so.6
#1  0x0000003cae433c75 in abort () from /lib64/libc.so.6
#2  0x0000000000400544 in fn2 (val=<optimized out>) at ex.c:9
#3  0x0000000000400509 in fn1 (x=<optimized out>, y=<optimized out>) at n.c:17
#4  0x0000000000400520 in main () at n.c:27
```

Compiler：LLVM 8.0.0, compile options: -g -O

# Backtrace with @entry value for parameters

Core was generated by `./n.x.entry-val'.
Program terminated with signal SIGABRT, Aborted.
#0 0x0000003cae432495 in raise () from /lib64/libc.so.6

(gdb) bt
#0 0x0000003cae432495 in raise () from /lib64/libc.so.6
#1 0x0000003cae433c75 in abort () from /lib64/libc.so.6
#2 0x0000000000400544 in fn2 (val=val@entry=111) at ex.c:9
#3 0x0000000000400509 in fn1 (x=x@entry=40, y=y@entry=70) at n.c:17
#4 0x0000000000400520 in main () at n.c:27

Compiler: LLVM 8.0.0, compile options: –femit-param-entry-values –g -O

# DWARF 5 Extensions

- Callee parameter entry-value information: DW_OP_entry_value used in .debug_loc or DW_AT_location block
- Call site information:
  - DWARF tags: DW_TAG_call_site, DW_TAG_call_site_parameter
  - Along with DWARF attributes: DW_AT_call_pc, DW_AT_call_origin, DW_AT_call_target, DW_AT_call_value
  - Some more attributes for tail call detection
- Jakub Jelinek, “Improving debug info for optimized away parameters”, 2010

# Implementation in LLVM

# Implementation

- Two approaches
  - Using IR Metadata
  - Using very late MIR analysis

- By implementing this feature in LLVM/Clang we improve backtrace quality in terms of the number of optimized-out values by 27%

# IR Metadata approach

- Parts of the implementation
  - Front-end part
  - IR (middle-end) part
  - Instruction selection (SelectionDAG phase)
  - Backend representation
  - LiveDebugValues (using ‘ArgNotModified’ info generated by front-end)
  - AsmPrinter

# IR Metadata approach

## Front-end part
## IR part

- DIMetadata → DWARF Tags
- DICallSite and DICallSiteParam

```
extern int foo(int);
int baa(int a)  {
  if (a > 0)
    return foo(a  + 3);
  return 0;
}
```

```
…
if.then:                                          ; preds = %entry
  %1 = load i32, i32* %a.addr, align 4, !dbg !28, !tbaa !20
  %add = add nsw i32 %1, 3, !dbg !29
  %call = call i32 @foo(i32 %add), !dbg !30, !call_site !12
  store i32 %call, i32* %retval, align 4, !dbg !31
  br label %return, !dbg !31
…
```

```
!12 = !DICallSite(scope: !13, file: !1, parameters: !14, line: 4, calledSubprogram:  !16)
!14 = !{!15}
!15 = !DICallSiteParam(argno:  1, variable:  !11, expr: !DIExpression(DW_OP_lit3, DW_OP_plus))
!16 = !DISubprogram(name: "foo", scope: !1, file: !1, line: 1, isLocal: false, isDefinition: false, flags:
DIFlagPrototyped,  isOptimized: true,  elements: !2)
```

# IR Metadata approach

## Back-end representation

- DBG_CALLSITE and DBG_CALLSITEPARAM

```
extern int foo(int ,int ,
int);
extern int gaa(int);
int baa(int a, int b) {
  int c = gaa(c);
  int d = gaa(d);
  return foo(c, d + 3, -31);
}
```

```
…
%EAX<def> = KILL %EAX, %RAX;
%ESI<def> = LEA64_32r %RAX<kill>, 1, %noreg, 3, %noreg;
%EDX<def> = MOV32ri -31;
%EDI<def> = MOV32rr %EBX<kill>;
%RBX<def> = POP64r %RSP<imp-def>, %RSP<imp-use>; flags: FrameDestroy
TAILJMPd64 <ga:@foo>, <regmask …>, %RSP, %RSP, %EDI, %ESI, %EDX;
DBG_CALLSITE 1, %noreg, <!22>;
  * DBG_CALLSITEPARAM %EDI, "c", %EBX, %noreg;
  * DBG_CALLSITEPARAM %EDX, !DIExpression(DW_OP_lit31, DW_OP_neg) ,
4294967265, %noreg;
  * DBG_CALLSITEPARAM %ESI, "d" !DIExpression(DW_OP_lit3, DW_OP_plus) ,
                      %RAX, 0, <!DIExpression(DW_OP_constu, 3,
DW_OP_plus)>;
…
```

# IR Metadata approach

## LiveDebugValues
## AsmPrinter

- Emitting of DBG_VALUEs with DW_OP_entry_value in LiveDebugValues for function arguments that have unmodified value throughout the function
- Using ‘DIFlagArgumentNotModified’ flag from argument’s DILocalVariable
- Dumping DWARF info in AsmPrinter

# IR Metadata approach
## Measurements

- The improvement of debug user experience is 16%
- Dwarfy-stats
- GDB 7.11 as benchmark
  - With no ‘femit-param-entry-values’

```
==================================
20 backtraces with 804 parameters
<optimized out> parameters 26%
@entry values 0.0%
==================================
```

  - With ‘femit-param-entry-values’

```
==================================
20 backtraces with 804 parameters
<optimized out> parameters 22%
@entry values 27%
==================================
```

# IR Metadata approach

- Advantages
  - More readable IR and MIR
  - No target specific parts (almost)

- Disadvantages
  - Hard for maintaining
  - LLVM passes can break call site debug info
  - Handling new kind of meta instructions through the whole pipeline

# Very late MIR approach

# Very late MIR approach

- Reuse from previous IR approach
  - LiveDebugValues part
  - AsmPrinter part
- CallLowering phase
  - MachineInst and call site information
  - MIR dump representation

# Very late MIR approach

## MIR example

```
callSites:
 - { bb: 0, offset: 2, fwdArgRegs:
     - { arg: 0, reg: '$edi' }
     - { arg: 1, reg: '$esi' } }
body:
 bb.0.entry:
   liveins: $edi, $esi

   renamable $esi = nsw ADD32rr killed renamable $esi, renamable $edi, implicit-def dead $eflags
   TAILJMPd64 @foo, csr_64, implicit $rsp, implicit $ssp, implicit $rsp, implicit $ssp, implicit $edi, implicit $esi
```

# Very late MIR approach
## Interpretation

- Interpretation analysis
  - describeLoadedValue
  - DW_OP_entry_value
  - Loading register in parent block

```
$esi = MOV32ri 4
$edi = MOV32rr $r14d
$rdx = LEA64r $rsp, 1, $noreg, 8, $noreg
CALL64pcrel32 @foo
```

# Very late MIR approach
## Interpretation

- Target Instruction Interface function

  *describeLoadedValue (const MachineInstruction &MI,*
  *const MachineOperand *&Op,*
  *DIExpression **Expr) const;*

  - Target independent part
    - Loading of immediate values
    - Register to Register moves
    - Stack Loading
    - Memory Loading
  - Target independent part
    - X86 – LEA interpretation

# Very late MIR approach
## Measurements

| | CLANG | 483.xalancbmk | GDB-7.11 |
|---|---|---|---|
| # of call sites | 688,400 | 23,858 | 92,985 |
| # of call site parameters | 875,694(48%) | 38,513(83%) | 103,274(72%) |
| % of dbg size increase | 1% | 1% | 11% |

# Very late MIR approach
## Measurements

- The improvement of debug user experience is 27%
- GDB 7.11 as benchmark
  - With no ‘femit-param-entry-values’

```
==================================
20 backtraces with 804 parameters
<optimized out> parameters 26%
@entry values 0.0%
==================================
```

  - With ‘femit-param-entry-values’

```
==================================
20 backtraces with 804 parameters
<optimized out> parameters 19%
@entry values 33%
==================================
```

# Very late MIR approach

- Advantages
  - The latest approach easy for maintaining
  - Backbone for further improvements
  - Better debug user experience
- Disadvantages
  - Target specific solution

# Future work

- Upstreaming Very late MIR approach
- Improving interpretation analysis for more instructions and parent basic blocks
- Adding support for other types of argument forwarding
- Adding support for other architectures

Thanks to all reviewers!