
PlayStation

# Save Our Source-Locations

## Or: Introspecting LLVM to fix bugs to improve debug info and SPGO outcomes

Stephen Livermore-Tozer

©2024 Sony Interactive Entertainment

# Why do we care about source locations?

©2024 Sony Interactive Entertainment

# Why do we care about source locations?

```
PHINode *PN = PHINode::Create(Ty: LoadI->getType(), NumReservedValues: pred_
PN->insertBefore(InsertPos: LoadBB->begin());
PN->takeName(V: LoadI);
PN->setDebugLoc(Loc: LoadI->getDebugLoc());

// Insert new entries into the PHI for each predecessor.  A single block ma
// have multiple entries here.
for (BasicBlock *P : predecessors(BB: LoadBB)) {
  AvailablePredsTy::iterator I =
      llvm::lower_bound(&Range: AvailablePreds, Value: std::make_pair(&x: P,

  assert(I != AvailablePreds.end() && I->first == P &&
         "Didn't find entry for predecessor!");

  // If we have an available predecessor but it requires casting, insert th
  // cast in the predecessor and use the cast. Note that we have to update
  // AvailablePreds vector as we go so that all of the PHI entries for this
  // predecessor use the same bitcast.
  Value *&PredV = I->second;
  if (PredV->getType() != LoadI->getType()) {
    PredV = CastInst::CreateBitOrPointerCast(
        S: PredV, Ty: LoadI->getType(), Name: "", InsertBefore: P->getTermina
    // The new cast is producing the value used to replace the load
    // instruction, so uses the load's debug location. If P does not always
    // branch to the load BB however then the debug location must be droppe
```

```
0x55555B36A843    48 85 d2                  testq   %rdx, %rdx
0x55555B36A846    7e 25                     jle     0x55555b36a86d
0x55555B36A848    48 d1 ea                  shrq    %rdx
0x55555B36A84B    48 89 d7                  movq    %rdx, %rdi
0x55555B36A84E    48 c1 e7 04               shlq    $0x4, %rdi
0x55555B36A852    4d 39 7c 3d 00            cmpq    %r15, (%r13,%rdi
0x55555B36A857    73 e7                     jae     0x55555b36a840
0x55555B36A859    48 f7 d2                  notq    %rdx
0x55555B36A85C    48 01 f2                  addq    %rsi, %rdx
0x55555B36A85F    49 01 fd                  addq    %rdi, %r13
0x55555B36A862    49 83 c5 10               addq    $0x10, %r13
0x55555B36A866    eb d8                     jmp     0x55555b36a840
0x55555B36A868    31 c9                     xorl    %ecx, %ecx
0x55555B36A86A    49 89 c5                  movq    %rax, %r13
0x55555B36A86D    48 c1 e1 04               shlq    $0x4, %rcx
0x55555B36A871    48 01 c8                  addq    %rcx, %rax
0x55555B36A874    49 39 c5                  cmpq    %rax, %r13
0x55555B36A877    0f 84 7f 03 00 00         je      0x55555b36abfc
0x55555B36A87D    4d 39 7d 00               cmpq    %r15, (%r13)
0x55555B36A881    0f 85 75 03 00 00         jne     0x55555b36abfc
0x55555B36A887    49 8b 75 08               movq    0x8(%r13), %rsi
0x55555B36A88B    48 8b 43 08               movq    0x8(%rbx), %rax
0x55555B36A88F    48 39 46 08               cmpq    %rax, 0x8(%rsi)
0x55555B36A893    0f 84 f1 01 00 00         je      0x55555b36aa8a
0x55555B36A899    66 c7 44 24 78 01 01      movw    $0x101, 0x78(%rs
0x55555B36A8A0    49 8b 4f 30               movq    0x30(%r15), %rcx
0x55555B36A8A4    49 83 c7 30               addq    $0x30, %r15
```

*Debugging large programs is significantly harder without accurate source locations.*

©2024 Sony Interactive Entertainment

# Why do we care about source locations?

- SPGO improves LLVM optimization decisions by using a trace of the program's execution.
- Source locations are required to map profiles back to source code.
- Performance gains are dependent on completeness and correctness of source locations.
- "Every missing source location is a missed optimization."

©2024 Sony Interactive Entertainment

# Why do we lose source locations?

©2024 Sony Interactive Entertainment

# Why do we lose source locations?

```
define i1 @test(i1 %in) !dbg !9 {
entry:
  %call = call i1 @make_condition(), !dbg !12
  br i1 %call, label %if.then, label %if.end, !dbg !13

if.then:
  %0 = xor i1 %in, true, !dbg !14
  br label %if.end, !dbg !15

if.end
  %cond.0.in = phi i1 [ %0, %if.then ], [ %call, %entry ]
  ret i1 %cond.0.in, !dbg !16
}
```

Hoisted, now unconditional

```
define i1 @test(i1 %in) !dbg !9 {
entry:
  %call = call i1 @make_condition(), !dbg !12
; Speculatively executed xor instruction.
; Now executes unconditionally!
  %0 = xor i1 %in, true, !dbg !14
; Original branch replaced with simple select.
  %spec.select = select i1 %call, i1 %0, i1 %call, !dbg !13
  ret i1 %spec.select, !dbg !16
}
```

*Movement of instructions across basic blocks may require source locations to be removed to prevent misleading users and profilers.*

©2024 Sony Interactive Entertainment

# Why do we lose source locations?

```
define i32 @foo(i1 %c1) !dbg !5 {
entry:
  %baz = alloca i32
  br i1 %c1, label %lhs, label %rhs, !dbg !15

lhs:
  store i32 1, ptr %baz, !dbg !16
  br label %cleanup, !dbg !17

rhs:
  store i32 2, ptr %baz, !dbg !18
  br label %cleanup, !dbg !19

cleanup:
  %baz.val = load i32, ptr %baz
  %ret.val = call i32 @escape(i32 %baz.val), !dbg !20
  ret i32 %ret.val, !dbg !21
}
```

Merged

Replaced

```
define i32 @foo(i1 %c1) !dbg !5 {
entry:
  br i1 %c1, label %lhs, label %rhs, !dbg !15

lhs:
  br label %cleanup, !dbg !17

rhs:
  br label %cleanup, !dbg !19

cleanup:
; Stores+loads replaced with a PHI with line number = 0.
  %storemerge = phi i32 [ 2, %rhs ], [ 1, %lhs ], !dbg !22
  %ret.val = call i32 @escape(i32 %storemerge), !dbg !20
  ret i32 %ret.val, !dbg !21
}
```

*Merged instructions without a common source location cannot be correctly attributed to a single location.*

©2024 Sony Interactive Entertainment

# Why do we lose source locations?

```
966         // The global is initialized when the store to it occurs. If the stored
967         // value is null value, the global bool is set to false, otherwise true.
968 -       new StoreInst(ConstantInt::getBool(
969 -                         GV->getContext(),
970 -                         !isa<ConstantPointerNull>(SI->getValueOperand())),
971 -                     InitBool, false, Align(1), SI->getOrdering(),
972 -                     SI->getSyncScopeID(), SI->getIterator());
973         SI->eraseFromParent();
974         continue;
```

```
966         // The global is initialized when the store to it occurs. If the stored
967         // value is null value, the global bool is set to false, otherwise true.
968 +       auto *NewSI = new StoreInst(
969 +           ConstantInt::getBool(GV->getContext(), !isa<ConstantPointerNull>(
970 +                                                    SI->getValueOperand())),
971 +           InitBool, false, Align(1), SI->getOrdering(), SI->getSyncScopeID(),
972 +           SI->getIterator());
973 +       NewSI->setDebugLoc(SI->getDebugLoc());
974         SI->eraseFromParent();
975         continue;
```

```
1519        Constant *C = Ty->isIntOrIntVectorTy() ?
1520          ConstantInt::get(Ty, NumFound) : ConstantFP::get(Ty, NumFound);
1521        Instruction *Mul = CreateMul(TheOp, C, "factor", I->getIterator(), I);
1522
1523        // Now that we have inserted a multiply, optimize it. This allows us to
1524        // handle cases that require multiple factoring steps, such as this:
```

```
1519        Constant *C = Ty->isIntOrIntVectorTy() ?
1520          ConstantInt::get(Ty, NumFound) : ConstantFP::get(Ty, NumFound);
1521        Instruction *Mul = CreateMul(TheOp, C, "factor", I->getIterator(), I);
1522 +      Mul->setDebugLoc(I->getDebugLoc());
1523
1524        // Now that we have inserted a multiply, optimize it. This allows us to
1525        // handle cases that require multiple factoring steps, such as this:
```

```
350     auto *T = Rem->getType();
351     auto *N = Rem->getOperand(0), *D = Rem->getOperand(1);
352     ICmpInst *ICmp = new ICmpInst(Rem->getIterator(), ICmpInst::ICMP_EQ, N, D);
353     SelectInst *Sel =
354         SelectInst::Create(ICmp, ConstantInt::get(T, 0), N, "iv.rem", Rem-
    >getIterator());
355     Rem->replaceAllUsesWith(Sel);
```

```
350     auto *T = Rem->getType();
351     auto *N = Rem->getOperand(0), *D = Rem->getOperand(1);
352     ICmpInst *ICmp = new ICmpInst(Rem->getIterator(), ICmpInst::ICMP_EQ, N, D);
353 +   ICmp->setDebugLoc(Rem->getDebugLoc());
354     SelectInst *Sel =
355         SelectInst::Create(ICmp, ConstantInt::get(T, 0), N, "iv.rem", Rem-
    >getIterator());
356     Rem->replaceAllUsesWith(Sel);
```

*Setting source locations for new instructions is an optional step and is sometimes simply overlooked.*

©2024 Sony Interactive Entertainment

# Can’t we just use tests?

©2024 Sony Interactive Entertainment

# Can’t we just use tests?

```
; NOTE: Assertions have been autogenerated by utils/update_test_checks.py UTC_ARGS: --version 5
; RUN: opt < %s -p=indvars -S | FileCheck %s

;; When that when IndVarSimplify simplifies the rem to a cmp and select, we
;; propagate the rem's source location to both the new instructions.

define i32 @widget() !dbg !5 {
; CHECK-LABEL: define i32 @widget(
; CHECK-SAME: ) !dbg [[DBG5:![0-9]+]] {
; CHECK-NEXT:  [[BB:.*:]]
; CHECK-NEXT:    br label %[[BB1:.*]]
; CHECK:       [[BB1_LOOPEXIT:.*]]:
; CHECK-NEXT:    br label %[[BB1]]
; CHECK:       [[BB1]]:
; CHECK-NEXT:    br label %[[BB2:.*]]
; CHECK:       [[BB2]]:
; CHECK-NEXT:    [[PHI:%.*]] = phi i32 [ 0, %[[BB1]] ], [ [[ADD:%.*]], %[[BB2]] ]
; CHECK-NEXT:    [[ADD]] = add nuw nsw i32 1, [[PHI]]
; CHECK-NEXT:    [[TMP0:%.*]] = icmp eq i32 [[ADD]], 3, !dbg [[DBG8:![0-9]+]]
; CHECK-NEXT:    [[IV_REM:%.*]] = select i1 [[TMP0]], i32 0, i32 [[ADD]], !dbg [[DBG8]]
; CHECK-NEXT:    [[ZEXT:%.*]] = zext i32 [[IV_REM]] to i64
; CHECK-NEXT:    br i1 false, label %[[BB2]], label %[[BB1_LOOPEXIT]]
;
bb:
  br label %bb1

bb1:                                              ; preds = %bb2, %bb
  br label %bb2

bb2:                                              ; preds = %bb2, %bb1
  %phi = phi i32 [ 0, %bb1 ], [ %add, %bb2 ]
  %add = add i32 1, %phi
  %urem = urem i32 %add, 3, !dbg !8
  %zext = zext i32 %urem to i64
  %icmp = icmp ult i32 %phi, 2
  br i1 %icmp, label %bb2, label %bb1
}

!llvm.dbg.cu = !{!0}
!llvm.debugify = !{!2, !3}
!llvm.module.flags = !{!4}

!0 = distinct !DICompileUnit(language: DW_LANG_C, file: !1, producer: "debugify", isOptimized: true, runtimeVersion: 0, emissionKind: Ful
!1 = !DIFile(filename: "llvm/test/Transforms/IndVarSimplify/debugloc-rem-subst.ll", directory: "/")
!2 = !{i32 8}
!3 = !{i32 0}
!4 = !{i32 2, !"Debug Info Version", i32 3}
!5 = distinct !DISubprogram(name: "widget", linkageName: "widget", scope: null, file: !1, line: 1, type: !6, scopeLine: 1, spFlags: DISPF
!6 = !DISubroutineType(types: !7)
!7 = !{}
!8 = !DILocation(line: 1, column: 1, scope: !5)
;.
; CHECK: [[META0:![0-9]+]] = distinct !DICompileUnit(language: DW_LANG_C, file: [[META1:![0-9]+]], producer: "debugify", isOptimized: tru
; CHECK: [[META1]] = !DIFile(filename: "llvm/test/Transforms/IndVarSimplify/debugloc-rem-subst.ll", directory: {{.*}})
; CHECK: [[DBG5]] = distinct !DISubprogram(name: "widget", linkageName: "widget", scope: null, file: [[META1]], line: 1, type: [[META6:![
; CHECK: [[META6]] = !DISubroutineType(types: [[META7:![0-9]+]])
; CHECK: [[META7]] = !{}
; CHECK: [[DBG8]] = !DILocation(line: 1, column: 1, scope: [[DBG5]])
```

Adds test coverage for…

```
// (i+1) % n  -->  (i+1)==n?0:(i+1)  if i is in [0,n).
void SimplifyIndvar::replaceRemWithNumeratorOrZero(Binar
  auto *T: Type * = Rem->getType();
  auto *N: Value * = Rem->getOperand(i_nocapture: 0), *D
  ICmpInst *ICmp = new ICmpInst(InsertBefore: Rem->getIt
  ICmp->setDebugLoc(Loc: Rem->getDebugLoc());
  SelectInst *Sel =
      SelectInst::Create(C: ICmp, S1: ConstantInt::get(Ty
  Rem->replaceAllUsesWith(V: Sel);
  Sel->setDebugLoc(Loc: Rem->getDebugLoc());
  LLVM_DEBUG(dbgs() << "INDVARS: Simplified rem: " << *R
  ++NumElimRem;
  Changed = true;
  DeadInsts.emplace_back(Rem);
}
```

*Currently unrealistic for lit or unit tests to cover every single callsite in LLVM.*

©2024 Sony Interactive Entertainment

# Can’t we just use tests?

```
define void @foo(i1 %b, i1 %c, i1 %d) !dbg !5 {
entry:
  %p = alloca i32, align 8
  br i1 %b, label %left, label %right

left:
  call void @llvm.memset.p0.i64(ptr %p, i8 0, i64 8, i1 false)
  br label %end

right:
  br i1 %c, label %right.left, label %right.right

right.left:
  call void @llvm.memset.p0.i64(ptr %p, i8 0, i64 8, i1 false)
  br i1 %d, label %end, label %exit

right.right:
  br label %end

end:
  %0 = load ptr, ptr %p, align 8, !dbg !8
  %isnull = icmp eq ptr %0, null
  br i1 %isnull, label %exit, label %notnull

notnull:
  br label %exit

exit:
  ret void
}
```

Copy !dbg

Drop !dbg

```
define void @foo(i1 %b, i1 %c, i1 %d) !dbg !5 {
entry:
  %p = alloca i32, align 8
  br i1 %b, label %left, label %right

left:
  call void @llvm.memset.p0.i64(ptr %p, i8 0, i64 8, i1 false)
  %0 = inttoptr i64 0 to ptr, !dbg !8
  br label %end

right:
  br i1 %c, label %right.left, label %endthread-pre-split

right.left:
  call void @llvm.memset.p0.i64(ptr %p, i8 0, i64 8, i1 false)
  %1 = inttoptr i64 0 to ptr
  br i1 %d, label %end, label %exit

endthread-pre-split:
  %.pr = load ptr, ptr %p, align 8, !dbg !8
  br label %end, !dbg !8

end:
  %2 = phi ptr [ %.pr, %endthread-pre-split ], [ %1, %right.left ], [ %0, %left ], !dbg !8
  %isnull = icmp eq ptr %2, null
  br i1 %isnull, label %exit, label %exit

exit:
  ret void
}
```

*Not all cases are trivial to determine from looking at the source or IR, especially when unfamiliar with debug info.*

©2024 Sony Interactive Entertainment

# The solution

©2024 Sony Interactive Entertainment

# The solution

```
enum class DebugLocKind : uint8_t {
  // Non-annotated location.
  Normal,
  // The instruction is not associated with any line in the user source.
  CompilerGenerated,
  // The instruction has had its location intentionally removed.
  Dropped,
  // The instruction is transient and will not be emitted to assembly.
  Temporary,
  // The instruction has a location we cannot or do not know how to
  // represent.
  Unknown
};
```

```
        SelectInst *SI =
            SelectInst::Create(RetKnownPN, RetPN, RI->getOperand(0),
                               "current.ret.tr", RI->getIterator());
        SI->setDebugLoc(DebugLoc::getCompilerGenerated());
```

New “annotations” encode the intent of empty source locations.

©2024 Sony Interactive Entertainment

# The solution

- In normal LLVM builds, annotative locations have no effect or cost.
- CMake with `-DLLVM_ENABLE_DEBUGLOC_COVERAGE_TRACKING=Coverage`:

```
static inline DebugLoc getCompilerGenerated() {
  return DebugLoc(DebugLocKind::CompilerGenerated);
}
```

- All source locations validated after every pass, using Debugify:

```
clang++ -O2 -gmlt                                                  # Build with optimizations and line numbers
  -Xclang -fverify-debuginfo-preserve                              # Run Debugify checks after each pass
  -Xclang -fverify-debuginfo-preserve-export=debugify-report.json  # Print Debugify results to debugify-report.json
  -mllvm -debugify-level=locations                                 # Check source locations only (no variables)
  -o ArchiveCommandLine.cpp.o -c CTMark/7zip/CPP/7zip/UI/Common/ArchiveCommandLine.cpp
```

- Any instruction without either a **source location** or an **annotation** is a bug.

©2024 Sony Interactive Entertainment

# The solution

Following build(s), reports can be displayed as HTML or YAML:

```
$ llvm/utils/llvm-original-di-preservation.py debugify-report.json --report-html-file report.html
```

**Location Bugs found by the Debugify**

| File | LLVM Pass Name | LLVM IR Instruction | Function Name | Basic Block Name | Action |
|---|---|---|---|---|---|
| /home/gbtozers/dev/llvm-test-suite/CTMark/7zip/CPP/7zip/UI/Common/ArchiveCommandLine.cpp | SROAPass | br | _ZN25CArchiveCommandLineParser6Parse2ER26CArchiveCommandLineOptions | lpad.i.i1362 | drop |

```
$ llvm/utils/llvm-original-di-preservation.py debugify-report.json --acceptance-test
DILocation Bugs:
  /home/gbtozers/dev/llvm-test-suite/CTMark/7zip/CPP/7zip/UI/Common/ArchiveCommandLine.cpp:
    SROAPass:
    - action: drop
      bb_name: lpad.i.i1362
      fn_name: _ZN25CArchiveCommandLineParser6Parse2ER26CArchiveCommandLineOptions
      instr: br

Errors detected for: debugify-report.json
```

©2024 Sony Interactive Entertainment

# The solution

```
DILocation Bugs:
  /home/gbtozers/dev/llvm-test-suite/CTMark/7zip/CPP/7zip/UI/Common/ArchiveCommandLine.cpp:
    InstCombinePass:
    - action: not-generate
      bb_name: for.body.lr.ph.i.i
      fn_name: _ZN25CArchiveCommandLineParser6Parse2ER26CArchiveCommandLineOptions
      instr: icmp
      origin: |
        #0 0x00005dc71e005465 llvm::DbgLocOrigin::DbgLocOrigin(bool) /home/gbtozers/dev/upstream-llvm/llvm/lib/IR/DebugLoc.cpp:21:9
        #1 0x00005dc71e0540de DILocAndCoverageTracking /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/DebugLoc.h:86:11
        #2 0x00005dc71e0540de DebugLoc /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/DebugLoc.h:129:5
        #3 0x00005dc71e0540de llvm::Instruction::Instruction(llvm::Type*, unsigned int, llvm::User::AllocInfo, llvm::InsertPosition) /home/gbtozers/dev/upstream-llvm/llvm/lib/IR/Instruction.cpp:47:14
        #4 0x00005dc71e0699d4 op_begin /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/OperandTraits.h:35:38
        #5 0x00005dc71e0699d4 OpFrom<0, llvm::CmpInst> /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/User.h:193:9
        #6 0x00005dc71e0699d4 Op<0> /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/InstrTypes.h:1005:1
        #7 0x00005dc71e0699d4 llvm::CmpInst::CmpInst(llvm::Type*, llvm::Instruction::OtherOps, llvm::CmpInst::Predicate, llvm::Value*, llvm::Value*, llvm::Twine const&, llvm::InsertPosition, llvm::Instru
        #8 0x00005dc71bfd9545 llvm::ICmpInst::ICmpInst(llvm::CmpInst::Predicate, llvm::Value*, llvm::Value*, llvm::Twine const&) /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/Instructions.h:1222
        #9 0x00005dc71e3200d4 canonicalizeCmpWithConstant /home/gbtozers/dev/upstream-llvm/llvm/lib/Transforms/InstCombine/InstCombineCompares.cpp:0:14
        #10 0x00005dc71e3200d4 llvm::InstCombinerImpl::visitICmpInst(llvm::ICmpInst&) /home/gbtozers/dev/upstream-llvm/llvm/lib/Transforms/InstCombine/InstCombineCompares.cpp:7612:26
        #11 0x00005dc71e286422 llvm::InstCombinerImpl::run() /home/gbtozers/dev/upstream-llvm/llvm/lib/Transforms/InstCombine/InstructionCombining.cpp:5628:22
        #12 0x00005dc71e289644 combineInstructionsOverFunction(llvm::Function&, llvm::InstructionWorklist&, llvm::AAResults*, llvm::AssumptionCache&, llvm::TargetLibraryInfo&, llvm::TargetTransformInfo&,
        #13 0x00005dc71e2889a9 llvm::InstCombinePass::run(llvm::Function&, llvm::AnalysisManager<llvm::Function>&) /home/gbtozers/dev/upstream-llvm/llvm/lib/Transforms/InstCombine/InstructionCombining.cp
        #14 0x00005dc71ed77ecd llvm::detail::PassModel<llvm::Function, llvm::InstCombinePass, llvm::AnalysisManager<llvm::Function>>::run(llvm::Function&, llvm::AnalysisManager<llvm::Function>&) /home/gb
        #15 0x00005dc71e0c6857 llvm::PassManager<llvm::Function, llvm::AnalysisManager<llvm::Function>>::run(llvm::Function&, llvm::AnalysisManager<llvm::Function>&) /home/gbtozers/dev/upstream-llvm/llvm
        #16 0x00005dc71c0a686d llvm::detail::PassModel<llvm::Function, llvm::PassManager<llvm::Function, llvm::AnalysisManager<llvm::Function>>, llvm::AnalysisManager<llvm::Function>>::run(llvm::Functior
        #17 0x00005dc71e0ca4c1 llvm::ModuleToFunctionPassAdaptor::run(llvm::Module&, llvm::AnalysisManager<llvm::Module>&) /home/gbtozers/dev/upstream-llvm/llvm/lib/IR/PassManager.cpp:132:23
        #18 0x00005dc71c0a71fd llvm::detail::PassModel<llvm::Module, llvm::ModuleToFunctionPassAdaptor, llvm::AnalysisManager<llvm::Module>>::run(llvm::Module&, llvm::AnalysisManager<llvm::Module>&) /hor
        #19 0x00005dc71e0c5907 llvm::PassManager<llvm::Module, llvm::AnalysisManager<llvm::Module>>::run(llvm::Module&, llvm::AnalysisManager<llvm::Module>&) /home/gbtozers/dev/upstream-llvm/llvm/include
        #20 0x00005dc71ed73cbc isSmall /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/ADT/SmallPtrSet.h:248:33
        #21 0x00005dc71ed73cbc ~SmallPtrSetImplBase /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/ADT/SmallPtrSet.h:89:10
        #22 0x00005dc71ed73cbc ~PreservedAnalyses /home/gbtozers/dev/upstream-llvm/llvm/include/llvm/IR/Analysis.h:112:7
        #23 0x00005dc71ed73cbc (anonymous namespace)::EmitAssemblyHelper::RunOptimizationPipeline(clang::BackendAction, std::unique_ptr<llvm::raw_pwrite_stream, std::default_delete<llvm::raw_pwrite_strea
        #24 0x00005dc71ed6aa3d emitAssembly /home/gbtozers/dev/upstream-llvm/clang/lib/CodeGen/BackendUtil.cpp:0:3
        #25 0x00005dc71ed6aa3d clang::emitBackendOutput(clang::CompilerInstance&, clang::CodeGenOptions&, llvm::StringRef, llvm::Module*, clang::BackendAction, llvm::IntrusiveRefCntPtr<llvm::vfs::FileSys
```

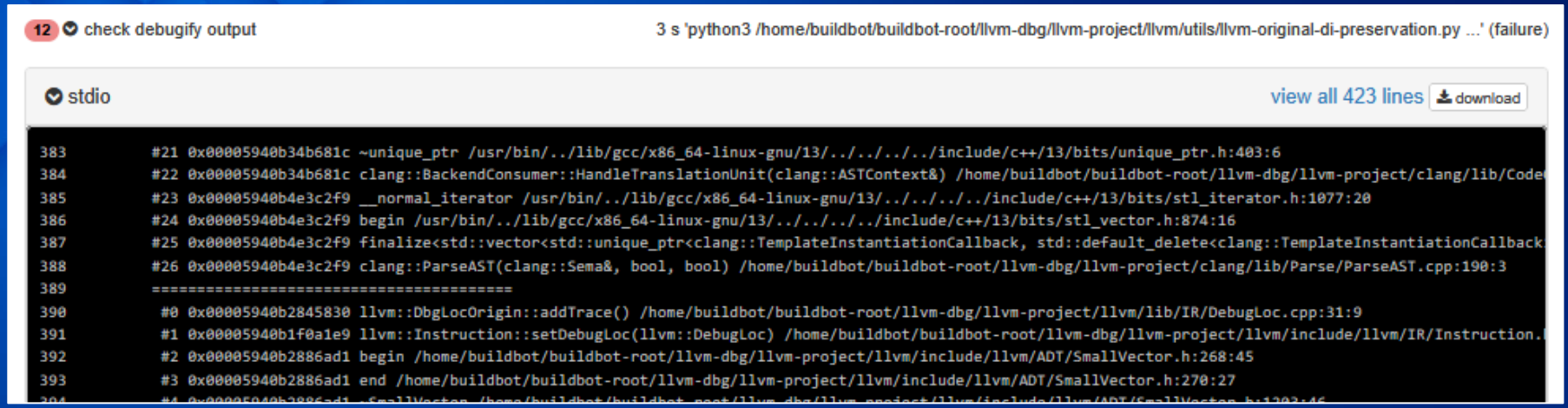
*To get a symbolized stacktrace with every bug, just add:*

`-DLLVM_ENABLE_DEBUGLOC_COVERAGE_TRACKING=COVERAGE_AND_ORIGIN`

©2024 Sony Interactive Entertainment

# The solution

- Buildbot currently running at: lab.llvm.org/staging/#/builders/222
- Configuration: x86_64, origin tracking, building CTMark with `-O2 -g`
- Bugs detected appear as failures in the “check debugify output” step:




©2024 Sony Interactive Entertainment

# Results and what comes next

©2024 Sony Interactive Entertainment

# Results and what comes next

- 19 existing bugs detected and fixed.
- Current coverage includes almost all IR passes, excludes:
  - Instruction selection
  - MIR passes
  - Vectorizers (Loop+SLP)
- Aiming to extend coverage to all of these in future, for full compiler coverage.

Instruction location counts in CTMark, pre-ISel, built with `clang -O2 -g`:

| | | |
|---|---|---|
| Valid locations | 1169793 | 93.266% |
| Line 0 locations | 14318 | 1.142% |
| Missing locations | 495 | 0.039% |
| Compiler Generated | 18388 | 1.466% |
| Dropped | 47513 | 3.788% |
| Temporary | 43 | 0.003% |
| Unknown | 3704 | 0.295% |

©2024 Sony Interactive Entertainment

# Results and what comes next

- Future work: verifying annotations?
- Temporary locations should never be considered for line table emission.
- Conditions for dropping locations can be detected with compiler instrumentation:
  - Huang, Shan & Liang, Jingjing & Su, Ting & Zhang, Qirun. (2025). Robustifying Debug Information Updates in LLVM via Control-Flow Conformance Analysis. Proceedings of the ACM on Programming Languages. 9. 527-549. 10.1145/3729267.
- Potential investigation for other annotation types?

©2024 Sony Interactive Entertainment

# Thank you!

©2024 Sony Interactive Entertainment