

Improving optimized-code line table quality

Orlando Cazalet-Hyams (speaker)

Stephen Livermore-Tozer

Jeremy Morse

Outline

- LLVM source location **defect finder** [almost done]
- Improved `is_stmt` placement for **better interactive debugging** [WIP]

Basics - Line table what and why

```
C++ source #1
1 #include <cstring>
2
3 void do_something();
4 const char *getStr();
5
6 void fun() {
7     const char *S = getStr();
8     if (!std::strcmp("-h", S))
9         do_something();
10 }
11
12
13
14
15

x86-64 clang 19.1.0 (Editor #1)
x86-64 clang 19.1.0 -O3 -gmlt
Output... Filter... Libraries Overrides + Add new... Add tool...
1 fun():
2     push    rax
3     call   getStr()@PLT
4     movzx  ecx, byte ptr [rax]
5     sub    ecx, 45
6     jne    .LBB0_3
7     movzx  ecx, byte ptr [rax + 1]
8     sub    ecx, 104
9     jne    .LBB0_3
10    movzx  ecx, byte ptr [rax + 2]
11 .LBB0_3:
12    neg    ecx
13    test   ecx, ecx
14    je     .LBB0_5
15    pop    rax
16    ret
17 .LBB0_5:
18    pop    rax
19    jmp   do_something()@PLT
```

Basics – LLVM IR

The image shows a side-by-side comparison of C++ source code and its LLVM IR. On the left, the C++ source code is shown in a text editor. On the right, the LLVM IR is displayed in a viewer window. An orange callout box highlights the !dbg !14 instruction in the IR, with a text label: '!DILocation metadata attached to instructions'.

```
#include <cstring>
void do_something();
const char *getStr();

void fun() {
    const char *S = getStr();
    if (!std::strcmp("-h", S))
        do_something();
}
```

```
30  %11 = phi i32 [ %2, %entry ], [ %6, %sub_1 ], [ %10, %sub_2 ]
31  %tobool.not = icmp eq i32 %11, 0, !dbg !14
32  br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
33
34  if.then:
35  tail call void @do_something(), !dbg !16
36  br label %if.end, !dbg !16
37
38  if.end:
39  ret void, !dbg !17
40 }
41
42 declare !dbg !18 noundef pt
43
44 declare !dbg !19 void @do_s
45
46 !llvm.dbg.cu = !{!0}
47 !llvm.module.flags = !{!12, !13, !14, !15, !16, !17}
48 !llvm.ident = !{!18}
49
50 !0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang version 19.1.0 (https://github.com/llvm/llvm-project.git a4bf6cd7cfb1a1421ba92bca9d017b49936c55e4)")
51 !1 = !DIFile(filename: "/app/example.cpp", directory: "/app")
52 !2 = !{i32 7, !"Dwarf Version", i32 4}
53 !3 = !{i32 2, !"Debug Info Version", i32 3}
54 !4 = !{i32 1, !"wchar_size", i32 4}
55 !5 = !{i32 8, !"PIC Level", i32 2}
56 !6 = !{i32 7, !"PIE Level", i32 2}
57 !7 = !{i32 7, !"uwtable", i32 2}
58 !8 = !{"clang version 19.1.0 (https://github.com/llvm/llvm-project.git a4bf6cd7cfb1a1421ba92bca9d017b49936c55e4)"}
59 !9 = distinct !DISubprogram(name: "fun", scope: !10, file: !10, line: 6, type: !11, scopeLine: 6, flags: DIFlagPrototyped)
60 !10 = !DIFile(filename: "example.cpp", directory: "/app")
61 !11 = !DISubroutineType(types: !12)
62 !12 = !{}
63 !13 = !DILocation(line: 1, column: 19, scope: !9)
64 !14 = !DILocation(line: 8, column: 8, scope: !9)
65 !15 = !DILocation(line: 8, column: 7, scope: !9)
66 !16 = !DILocation(line: 9, column: 5, scope: !9)
```

Basics – LLVM IR

The image shows a side-by-side comparison of C++ source code and its corresponding LLVM IR. On the left, the C++ source code for a function `fun()` is shown, with the `if` statement and its body highlighted in green. On the right, the LLVM IR for the same function is displayed, with the `!dbg !14` metadata attached to the `br` instruction in the `if.then` block highlighted in green. An orange callout box with a red arrow points to the `!dbg !14` metadata in the IR, containing the text: **!DILocation metadata attached to instructions**. Below the source code, a terminal window shows the LLVM command-line interface, with the `setDebugLoc` method call in the `MachineSink.cpp` source code highlighted in yellow. The IR viewer also shows a list of `!DILocation` metadata entries at the bottom, with `!14 = !DILocation(line: 8, column: 8, scope: !9)` highlighted in red.

```
1 #include <cstring>
2
3 void do_something();
4 const char *getStr();
5
6 void fun() {
7     const char *S = getStr();
8     if (!std::strcmp("-h", S))
9         do_something();
10 }
11
12
13
```

```
30 %11 = phi i32 [ %2, %entry ], [ %6, %sub_1 ], [ %10, %sub_2 ]
31 %tobool.not = icmp eq i32 %11, 0, !dbg !14
32 br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
33
34 if.then:
35     tail call void @do_something(), !dbg !16
36     br label %if.end, !dbg !16
37
38 if.end:
39     ret void, !dbg !17
40 }
41
42 declare !dbg !18 noundef ptr @do_something()
43
44 declare !dbg !19 void @do_something()
45
46 !llvm.dbg.cu = !{!0}
47 !llvm.module.flags = !{!12, !13, !14, !15, !16, !17}
48 !llvm.ident = !{!18}
49
50 !0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang version 19.1.0 (https://github.com/llvm/llvm-project.git a4bf6cd7c7b1a1421ba92bca9d017b49936c55e4)")
51 !1 = !DIFile(filename: "/app/example.cpp", directory: "/app")
52 !2 = !{i32 7, !"Dwarf Version", i32 4}
53 !3 = !{i32 2, !"Debug Info Version", i32 3}
54 !4 = !{i32 1, !"wchar_size", i32 4}
55 !5 = !{i32 8, !"PIC Level", i32 2}
56 !6 = !{i32 7, !"PIE Level", i32 2}
57 !7 = !{i32 7, !"uwtable", i32 2}
58 !8 = !{"clang version 19.1.0 (https://github.com/llvm/llvm-project.git a4bf6cd7c7b1a1421ba92bca9d017b49936c55e4)"}
59 !9 = distinct !DISubprogram(name: "fun", scope: !10, file: !10, line: 6, type: !11, scopeLine: 6, flags: DIFlagPrototyped)
60 !10 = !DIFile(filename: "example.cpp", directory: "/app")
61 !11 = !DISubroutineType(types: !12)
62 !12 = !{}
63 !13 = !DILocation(line: 1, column: 19, scope: !9)
64 !14 = !DILocation(line: 8, column: 8, scope: !9)
65 !15 = !DILocation(line: 8, column: 7, scope: !9)
66 !16 = !DILocation(line: 9, column: 5, scope: !9)
```

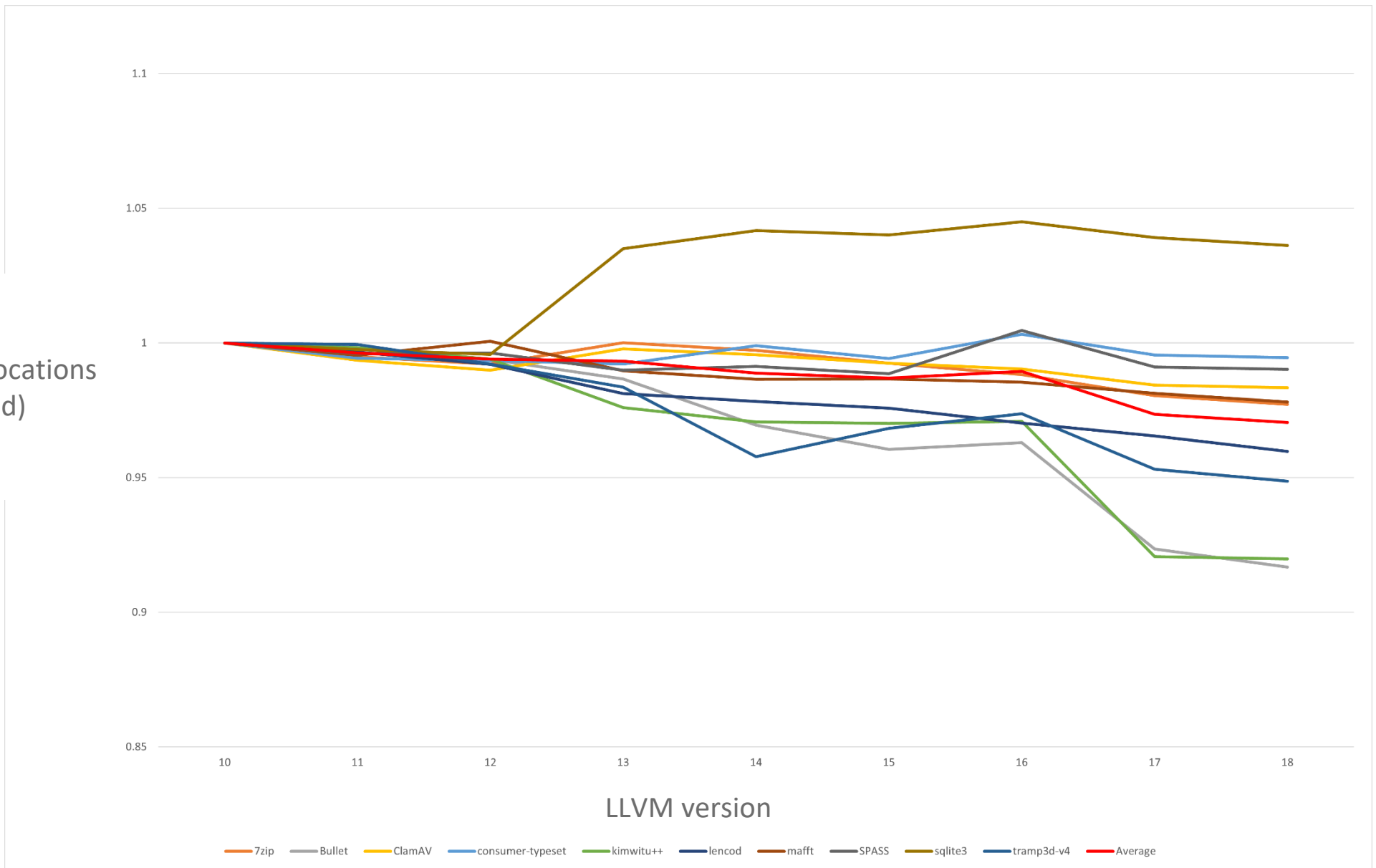
LLVM Source Code

```
MachineSink.cpp X
llvm > lib > CodeGen > MachineSink.cpp
365 bool MachineSink::PerformSinkAndFold(MachineInstr &MI,
514     for (auto &[SinkDst, MaybeAM] : SinkInto) {
518         if (SinkDst->isCopy()) {
529             // Sink a copy of the instruction, replacing a COPY instruction.
530             MachineBasicBlock::iterator InsertPt = SinkDst->getIterator();
531             Register DstReg = SinkDst->getOperand(0).getReg();
532             TII->reMaterialize(*SinkDst->getParent(), InsertPt, DstReg, 0, MI, *TRI);
533             New = &*std::prev(InsertPt);
534             if (!New->getDebugLoc())
535                 New->setDebugLoc(SinkDst->getDebugLoc());
536
```

Source location defect finder

<https://discourse.llvm.org/t/rfc-proposed-update-to-handling-debug-locations-in-llvm>

Unique source locations
(normalized)



Misattribution

```
C++ source #1
Save/Load + Add new... Vim CppInsights Quick-bench C++
1 #include <cstring>
2
3 void do_something();
4 const char *getStr();
5
6 void fun() {
7     const char *S = getStr();
8     if (!std::strcmp("-h", S))
9         do_something();
10 }
11
12
13
14
15
```

```
x86-64 clang 19.1.0 (Editor #1)
x86-64 clang 19.1.0 -O3 -gmlt
Output... Filter... Libraries Overrides + Add new... Add tool...
1 fun():
2     push    rax
3     call   getStr()@PLT
4     movzx  ecx, byte ptr [rax]
5     sub    ecx, 45
6     jne    .LBB0_3
7     movzx  ecx, byte ptr [rax + 1]
8     sub    ecx, 104
9     jne    .LBB0_3
10    movzx  ecx, byte ptr [rax + 2]
11 .LBB0_3:
12    neg    ecx
13    test   ecx, ecx
14    je     .LBB0_5
15    pop    rax
16    ret
17 .LBB0_5:
18    pop    rax
19    jmp   do_something()@PLT
```


Misattribution

```
C++ source #1
Save/Load + Add new... Vim CppInsights Quick-bench C++
1 #include <cstring>
2
3 void do_something();
4 const char *getStr();
5
6 void fun() {
7     const char *S = getStr();
8     if (!std::strcmp("-h", S))
9         do_something();
10 }
11
12
13
14
15
```

```
x86-64 clang 19.1.0 (Editor #1)
x86-64 clang 19.1.0 -O3 -gmlt
Output... Filter... Libraries Overrides + Add new... Add tool...
1 fun():
2     push    rax
3     call   getStr()@PLT
4     movzx  ecx, byte ptr [rax]
5     sub    ecx, 45
6     jne    .LBB0_3
7     movzx  ecx, byte ptr [rax + 1]
8     sub    ecx, 104
9     jne    .LBB0_3
10    movzx  ecx, byte ptr [rax + 2]
11.LBB0_3:
12    neg    ecx
13    test   ecx, ecx
14    je     .LBB0_5
15    pop    rax
16    ret
17.LBB0_5:
18    pop    rax
19    jmp   do_something()@PLT
```

Misattribution

```
C++ source #1
Save/Load + Add new... Vim CppInsights Quick-bench C++
1 #include <cstring>
2
3 void do_something();
4 const char *getStr();
5
6 void fun() {
7     const char *S = getStr();
8     if (!std::strcmp("-h", S))
9         do_something();
10 }
11
12
13
14
15
```

```
x86-64 clang 19.1.0 (Editor #1)
x86-64 clang 19.1.0 -O3 -gmlt
Output... Filter... Libraries Overrides + Add new... Add tool...
1 fun():
2     push    rax
3     call   getStr()@PLT
4     movzx  ecx, byte ptr [rax]
5     sub    ecx, 45
6     jne    .LBB0_3
7     movzx  ecx, byte ptr [rax + 1]
8     sub    ecx, 104
9     jne    .LBB0_3
10    movzx  ecx, byte ptr [rax + 2]
11 .LBB0_3:
12    neg    ecx
13    test   ecx, ecx
14    je     .LBB0_5
15    pop    rax
16    ret
17 .LBB0_5:
18    pop    rax
19    jmp   do_something()@PLT
```

Other issues

- Missing/incorrect location in crash trace
- Bonus coverage (ctrl-flow confusion)
- Cover unreachable code

```

1 define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2 entry:
3   %call = call noundef ptr @getStr(), !dbg !13
4-  %call1 = call i32 @strcmp(ptr noundef nonnull dereferenceable(3) @.str, ptr noundef nonnull dereferenceable(1) %call) #3, !dbg !14
5-  %tobool.not = icmp eq i32 %call1, 0, !dbg !14
6   br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
7
8-if.then:                                ; preds = %entry
9   call void @do_something(), !dbg !16
10  br label %if.end, !dbg !16
11
12-if.end:                                  ; preds = %if.then, %entry
13  ret void, !dbg !17
14 }
    
```

```

1 define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2 entry:
3   %call = call noundef ptr @getStr(), !dbg !13
4+  br label %sub_0, !dbg !14
5+
6+sub_0:                                    ; preds = %entry
7+  %0 = load i8, ptr %call, align 1
8+  %1 = zext i8 %0 to i32
9+  %2 = sub i32 45, %1
10+ %3 = icmp ne i32 %2, 0
11+ br i1 %3, label %ne, label %sub_1
12+
13+sub_1:                                    ; preds = %sub_0
14+ %4 = getelementptr inbounds i8, ptr %call, i64 1
15+ %5 = load i8, ptr %4, align 1
16+ %6 = zext i8 %5 to i32
17+ %7 = sub i32 104, %6
18+ %8 = icmp ne i32 %7, 0
19+ br i1 %8, label %ne, label %sub_2
20+
21+sub_2:                                    ; preds = %sub_1
22+ %9 = getelementptr inbounds i8, ptr %call, i64 2
23+ %10 = load i8, ptr %9, align 1
24+ %11 = zext i8 %10 to i32
25+ %12 = sub i32 0, %11
26+ br label %ne
27+
28+ne:                                        ; preds = %sub_2, %sub_1, %sub_0
29+ %13 = phi i32 [ %2, %sub_0 ], [ %7, %sub_1 ], [ %12, %sub_2 ]
30+ br label %entry.tail
31+
32+entry.tail:                              ; preds = %ne
33+ %tobool.not = icmp eq i32 %13, 0, !dbg !14
34  br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
35
36+if.then:                                  ; preds = %entry.tail
37  call void @do_something(), !dbg !16
38  br label %if.end, !dbg !16
39
40+if.end:                                  ; preds = %if.then, %entry.tail
41  ret void, !dbg !17
42 }
    
```

```

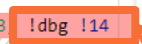
1 define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2 entry:
3   %call = call noundef ptr @getStr(), !dbg !13
4-  %call1 = call i32 @strcmp(ptr noundef nonnull dereferenceable(3) @.str, ptr noundef nonnull dereferenceable(1) %call) #3 !dbg !14
5-  %tobool.not = icmp eq i32 %call1, 0, !dbg !14
6   br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
7
8-if.then:                                ; preds = %entry
9   call void @do_something(), !dbg !16
10  br label %if.end, !dbg !16
11
12-if.end:                                ; preds = %if.then, %entry
13  ret void, !dbg !17
14 }

```

```

1 define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2 entry:
3   %call = call noundef ptr @getStr(), !dbg !13
4+  br label %sub_0, !dbg !14
5
6-sub_0:                                  ; preds = %entry
7+  %0 = load i8, ptr %call, align 1
8+  %1 = zext i8 %0 to i32
9+  %2 = sub i32 45, %1
10+ %3 = icmp ne i32 %2, 0
11+ br i1 %3, label %ne, label %sub_1
12
13+sub_1:                                  ; preds = %sub_0
14+ %4 = getelementptr inbounds i8, ptr %call, i64 1
15+ %5 = load i8, ptr %4, align 1
16+ %6 = zext i8 %5 to i32
17+ %7 = sub i32 104, %6
18+ %8 = icmp ne i32 %7, 0
19+ br i1 %8, label %ne, label %sub_2
20
21+sub_2:                                  ; preds = %sub_1
22+ %9 = getelementptr inbounds i8, ptr %call, i64 2
23+ %10 = load i8, ptr %9, align 1
24+ %11 = zext i8 %10 to i32
25+ %12 = sub i32 0, %11
26+ br label %ne
27
28+ne:                                      ; preds = %sub_2, %sub_1, %sub_0
29+ %13 = phi i32 [ %2, %sub_0 ], [ %7, %sub_1 ], [ %12, %sub_2 ]
30+ br label %entry.tail
31
32+entry.tail:                              ; preds = %ne
33+ %tobool.not = icmp eq i32 %13, 0, !dbg !14
34  br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
35
36+if.then:                                ; preds = %entry.tail
37  call void @do_something(), !dbg !16
38  br label %if.end, !dbg !16
39
40+if.end:                                ; preds = %if.then, %entry.tail
41  ret void, !dbg !17
42 }

```



Existing tooling - Debugify

/suite/MultiSource/Benchmarks/Bullet/btSoftBody.cpp	SLPVectorizerPass	insertelement	_ZN1006SoftBody10updatePosE
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btSphereTriangleCollisionAlgorithm.cpp	SimplifyCFGPass	br	_ZN34btSphereTriangleCollisionAlgorithm16processCollisionEP17btCollisionObjectS1_RK16btDispatcherInfoP16btManifoldResult
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btSubSimplexConvexCast.cpp	SLPVectorizerPass	insertelement	_ZN22btSubsimplexConvexCast16calcTimeOfImpactERK11btTransformS2_S2_S2_RN12btConvexCast10CastResultE
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btSubSimplexConvexCast.cpp	SLPVectorizerPass	shufflevector	_ZN22btSubsimplexConvexCast16calcTimeOfImpactERK11btTransformS2_S2_S2_RN12btConvexCast10CastResultE
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btSubSimplexConvexCast.cpp	SLPVectorizerPass	shufflevector	_ZN22btSubsimplexConvexCast16calcTimeOfImpactERK11btTransformS2_S2_S2_RN12btConvexCast10CastResultE
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btSubSimplexConvexCast.cpp	SLPVectorizerPass	insertelement	_ZN22btSubsimplexConvexCast16calcTimeOfImpactERK11btTransformS2_S2_S2_RN12btConvexCast10CastResultE
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btTriangleMesh.cpp	IPSCCPass	unreachable	_ZN14btTriangleMesh15findOrAddVertexERK9btVector3b
/home/gbtozers/dev/llvm-test-suite/MultiSource/Benchmarks/Bullet/btVoronoiSimplexSolver.cpp	JumpThreadingPass	freeze	_ZN22btVoronoiSimplexSolver9inSimplexERK9btVector3

Summary of Location Bugs

LLVM Pass Name	Number of bugs
CorrelatedValuePropagationPass	1
GlobalOptPass	5
IPSCCPass	1
InstCombinePass	4
JumpThreadingPass	3
LoopUnrollPass	4
ReassociatePass	11
SLPVectorizerPass	17
SROAPass	9
SimplifyCFGPass	38
TailCallElimPass	3

Generate HTML report of **dropped-locations per optimisation pass**

Contains false positives ☹️

-verify-debuginfo-preserve

-verify-di-preserve-export=sample.json

llvm-original-di-preservation.py

Solution

Declare intent using new API

```
967 967 void Instruction::dropLocation() {
968 968     const DebugLoc &DL = getDebugLoc();
969 969     if (!DL)
970 970         return;
971 971
972 972     // If this isn't a call, drop the location to allow a location from a
973 973     // preceding instruction to propagate.
974 974     bool MayLowerToCall = false;
975 975     if (isa<CallBase>(this)) {
976 976         auto *II = dyn_cast<IntrinsicInst>(this);
977 977         MayLowerToCall =
978 978             !II || IntrinsicInst::mayLowerToFunctionCall(II->getIntrinsicID());
979 979     }
980 980
981 981     if (!MayLowerToCall) {
982 982 -   setDebugLoc(DebugLoc());
982 982 +   setDebugLoc(DebugLoc::getLineZero());
983 983     return;
984 984     }
985 985 }
```

DebugLoc API

13

33

```
34 + #if ENABLE_DEBUGLOC_COVERAGE_TRACKING
35 +   DILocAndCoverageTracking::DILocAndCoverageTracking(const DILocation *L)
36 +       : TrackingMDNodeRef(const_cast<DILocation *>(L)), DbgLocOrigin(!L),
37 +       Kind(DebugLocKind::Normal) {}
38 +
39 + DebugLoc DebugLoc::getTemporary() { return DebugLoc(DebugLocKind::Temporary); }
40 + DebugLoc DebugLoc::getUnknown() { return DebugLoc(DebugLocKind::Unknown); }
41 + DebugLoc DebugLoc::getLineZero() { return DebugLoc(DebugLocKind::LineZero); }
42 +
43 + #else
44 +
45 + DebugLoc DebugLoc::getTemporary() { return DebugLoc(); }
46 + DebugLoc DebugLoc::getUnknown() { return DebugLoc(); }
47 + DebugLoc DebugLoc::getLineZero() { return DebugLoc(); }
48 + #endif // ENABLE_DEBUGLOC_COVERAGE_TRACKING
49 +
```

Does nothing by default

14

```
50 //====-----
```

15

```
51 // DebugLoc Implementation
```

16

```
52 //====-----
```

```

Opt Pipeline Viewer x86-64 clang 19.1.0 (Editor #1, Compiler #1)
A Options Filters Function: fun()
Passes:
Filter passes:
SROAPass on fun()
GlobalOpt Pass on (module)
InstCombinePass on fun()
AggressiveInstCombinePass on fun()
TailCallElimPass on fun()
SimplifyCFGPass on fun()
InstCombinePass on fun()
CodeGenPrepare (codegen prepare)
X86 DAG->DAG Instruction
1 define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2   entry:
3     %call1 = call noundef ptr @getStr(), !dbg !13
4     %call11 = call @132 @strcmp(ptr noundef nonnull dereferenceable(3) @.str, ptr noundef nonnull dereferenceable(1) %call1) #!dbg !14
5     %tobool.not = icmp eq i32 %call11, 0, !dbg !14
6     br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
7
8     if.then:
9       call void @do_something(), !dbg !16
10      br label %if.end, !dbg !16
11
12     if.end:
13      ret void, !dbg !17
14 }
1   define dso_local void @fun()() local_unnamed_addr #0 !dbg !9 {
2   entry:
3     %call = call noundef ptr @getStr(), !dbg !13
4     br label %sub_0, !dbg !14
5
6     sub_0:
7     %1 = load i8, ptr %call, align 1, !dbg !14; preds = %entry
8     %1 = zext i8 %1 to i32, !dbg !14
9     %2 = sub i32 45, %1, !dbg !14
10    %3 = icmp ne i32 %2, 0, !dbg !14
11    br i1 %3, label %ne, label %sub_1, !dbg !14
12
13    sub_1:
14    %4 = getelementptr inbounds i8, ptr %call, i64 1, !dbg !14; preds = %sub_0
15    %5 = load i8, ptr %4, align 1, !dbg !14
16    %6 = zext i8 %5 to i32, !dbg !14
17    %7 = sub i32 104, %6, !dbg !14
18    %8 = icmp ne i32 %7, 0, !dbg !14
19    br i1 %8, label %ne, label %sub_2, !dbg !14
20
21    sub_2:
22    %9 = getelementptr inbounds i8, ptr %call, i64 2, !dbg !14; preds = %sub_1
23    %10 = load i8, ptr %9, align 1, !dbg !14
24    %11 = zext i8 %10 to i32, !dbg !14
25    %12 = sub i32 0, %11, !dbg !14
26    br label %ne, !dbg !14
27
28    ne:
29    %13 = phi i32 [ %2, %sub_0 ], [ %7, %sub_1 ], [ %12, %sub_2 ], !dbg !14; preds = %sub_2, %sub_1, %sub_0
30    br label %entry.tail, !dbg !14
31
32    entry.tail:
33    %tobool.not = icmp eq i32 %13, 0, !dbg !14; preds = %ne
34    br i1 %tobool.not, label %if.then, label %if.end, !dbg !15
35
36    if.then:
37    call void @do_something(), !dbg !16; preds = %entry.tail
38    br label %if.end, !dbg !16
39
40    if.end:
41    ret void, !dbg !17; preds = %if.then, %entry.tail
42 }

```


▼ View Origin StackTrace

Stack Trace 0 (--opt-bisect-limit=528):

```
#0 0x000055cb2aeca5d5 llvm::DbgLocOrigin::DbgLocOrigin(bool) /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/lib/IR/DebugLoc.cpp:100:3
#1 0x000055cb2cac410f DILocAndCoverageTracking /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/include/llvm/IR/DebugLoc.h:100:3
0x000055cb2cac410f DebugLoc /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/include/llvm/IR/DebugLoc.h:127:5
0x000055cb2cac410f IRBuilderBase /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/include/llvm/IR/IRBuilder.h:143:3
0x000055cb2cac410f IRBuilder /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/include/llvm/IR/IRBuilder.h:2708:9
0x000055cb2cac410f (anonymous namespace)::StrNCmpInliner::inlineCompare llvm::Value*, llvm::StringRef, unsigned long, bool)
#2 0x000055cb2cabece0 foldLibCalls /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/lib/Transforms/AggressiveInstCombine/AggressiveInstCombine.cpp:100:3
0x000055cb2cabece0 foldUnusualPatterns(llvm::Function&, llvm::DominatorTree&, llvm::TargetTransformInfo&, llvm::TargetLibraryInfo&)
#3 0x000055cb2cabca08 runImpl /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/lib/Transforms/AggressiveInstCombine/AggressiveInstCombine.cpp:100:3
0x000055cb2cabca08 llvm::AggressiveInstCombinePass::run(llvm::Function&, llvm::AnalysisManager&) /home/gbtozers/dev/llvm-line-instrument-stage1/llvm/lib/Transforms/AggressiveInstCombine/AggressiveInstCombine.cpp:100:3
#4 0x000055cb2c881f1d llvm::detail::PassModel>
#5 0x000055cb2af8419c llvm::PassManager>::run(
#6 0x000055cb2937b25d llvm::detail::PassModel>
#7 0x000055cb2a6ba9bf llvm::CGSCCToFunctionPass
#8 0x000055cb2939653d llvm::detail::PassModel,
#9 0x000055cb2a6b665d llvm::PassManager, llvm:
#10 0x000055cb2c8764fd llvm::detail::PassModel,
#11 0x000055cb2a6b94d1 llvm::DevirtSCCRepeatedPass
#12 0x000055cb2c88da1d llvm::detail::PassModel,
#13 0x000055cb2a6b7cde llvm::ModuleToPostOrderCFG
#14 0x000055cb2c87679d llvm::detail::PassModel>
#15 0x000055cb2af8349c llvm::PassManager>::run(
```

The screenshot shows the LLVM Pipeline Viewer interface. The main window displays assembly code for a function. The code includes instructions like `define dso_local void @fun() local_unnamed_addr #0 ldbg 19 {`, `entry:`, `%call1 = call noundef ptr @getStr(), ldbg 113`, `%call11 = call @strcmp(ptr noundef nonnull dereferenceable(1) %str, ptr noundef nonnull dereferenceable(1) %call1) #8 ldbg 114`, and various arithmetic and branch instructions. Red arrows point from the `%call11` instruction in the assembly to the corresponding entry in the stack trace above. The interface also shows a sidebar with various optimization passes and filters.

Defect finder summary

- New defect-finder based on debugify, no false positives
 - DLLVM_ENABLE_DEBUGLOC_COVERAGE_TRACKING = COVERAGE_AND_ORIGIN**
- Pass authors encode intent with new API
- We're fixing existing issues it found
- Then let's put this on a buildbot

Key Instructions

<https://discourse.llvm.org/t/rfc-improving-is-stmt-placement-for-better-interactive-debugging>

Optimised code debugging

-O2 -g

```
1  int g1 = 1, g2 = 2, g3 = 3;
2
3  struct point { double x, y, z; };
4
5  [[clang::optnone]]
6  point fun(const point& p) { return p; }
7
8  [[clang::optnone]]
9  void f2(double) {}
10
11 int main() {
12     double y = (double)g2;
13     double z = (double)g3;
14     double x = (double)g1;
15     point p = { x * 2, y / 3, y / 3 + z + x * 2 };
16     point p2 = { p.x, p.y, p.z };
17     f2(p.z);
18     f2(p.x);
19     fun(p2);
20     p = fun(p);
21     return p.x + p.y + p.z;
22 }
23
```

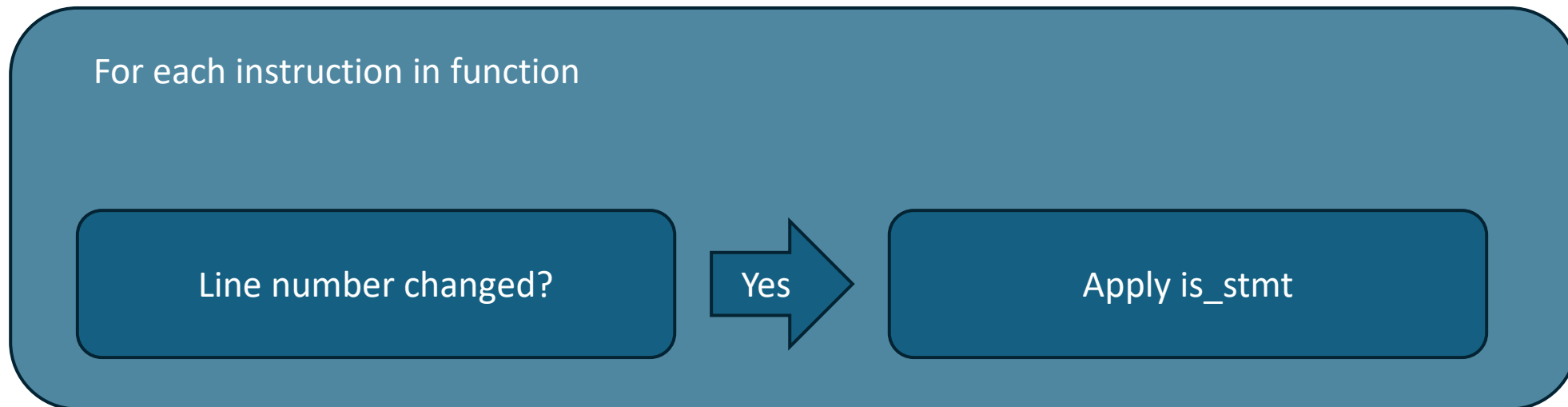
-O2 -g -???

```
1  int g1 = 1, g2 = 2, g3 = 3;
2
3  struct point { double x, y, z; };
4
5  [[clang::optnone]]
6  point fun(const point& p) { return p; }
7
8  [[clang::optnone]]
9  void f2(double) {}
10
11 int main() {
12     double y = (double)g2;
13     double z = (double)g3;
14     double x = (double)g1;
15     point p = { x * 2, y / 3, y / 3 + z + x * 2 };
16     point p2 = { p.x, p.y, p.z };
17     f2(p.z);
18     f2(p.x);
19     fun(p2);
20     p = fun(p);
21     return p.x + p.y + p.z;
22 }
23
```

DWARF line table

Address	Line	Column	File	ISA	Discriminator	OpIndex	Flags
0x000000000000001130	5	0	0	0	0	0	is_stmt
0x000000000000001134	6	6	0	0	0	0	is_stmt prologue_end
0x00000000000000113a	7	1	0	0	0	0	is_stmt epilogue_begin
0x000000000000001140	9	0	0	0	0	0	is_stmt
0x000000000000001144	10	6	0	0	0	0	is_stmt prologue_end
0x00000000000000114e	11	1	0	0	0	0	is_stmt epilogue_begin
0x000000000000001150	13	0	0	0	0	0	is_stmt
0x000000000000001154	14	3	0	0	0	0	is_stmt prologue_end
0x000000000000001159	10	6	0	0	0	0	is_stmt
0x000000000000001170	16	12	0	0	0	0	is_stmt
0x00000000000000117f	16	15	0	0	0	0	is_stmt
0x000000000000001181	16	3	0	0	0	0	is_stmt
0x000000000000001183	17	5	0	0	0	0	is_stmt
0x000000000000001188	0	5	0	0	0	0	is_stmt
0x00000000000000118a	18	3	0	0	0	0	is_stmt
0x00000000000000118c	18	3	0	0	0	0	epilogue_begin
0x00000000000000118e	18	3	0	0	0	0	end_sequence

LLVM's `is_stmt` strategy



This “works” but results in excessively jumpy stepping

```
3 void do_something(int);
4
5 int fun(int a) {
6     if (a * 2 > 5)
7         do_something(a);
8     return a * 2;
9 }
```


Diff before(-) and after (+) early-cse

```
1 1 define dso_local noundef i32 @_Z3funi(i32 noundef %a) local_unnamed_addr #0 !dbg
2 2 entry:
3 3     %mul = mul nsw i32 %a, 2, !dbg !12 ; line 6
4 4     %cmp = icmp sgt i32 %mul, 5, !dbg !12 ; line 6
5 5     br i1 %cmp, label %if.then, label %if.end, !dbg !12 ; line 6
6 6
7 7 if.then: ; preds = %entry
8 8     call void @_Z12do_somethingb(i32 noundef %a), !dbg !13 ; line 7
9 9     br label %if.end, !dbg !13 ; line 7
10 10
11 11 if.end: ; preds = %if.then, %entry
12 - %mul1 = mul nsw i32 %a, 2, !dbg !14 ; line 8
13 - ret i32 %mul1, !dbg !14 ; line 8
12+ ret i32 %mul, !dbg !14 ; line 8
14 13 }
15 14
16 15 ...
17 16 !12 = !DILocation(line: 6, scope: !9)
18 17 !13 = !DILocation(line: 7, scope: !9)
19 18 !14 = !DILocation(line: 8, scope: !9)
20 19
```

```
3 void do_something(int);
4
5 int fun(int a) {
6     if (a * 2 > 5)
7         do_something(a);
8     return a * 2;
9 }
```

CSE: %mul1 RAUW %mul

Diff before(-) and after (+) incstcombine

```
1 1 define dso_local noundef i32 @_Z3funi(i32 noundef %a) local_unnamed_addr #0 !dbg
2 2 entry:
3 - %mul = mul nsw i32 %a, 2, !dbg !12 ; line 6
4 - %cmp = icmp sgt i32 %mul, 5, !dbg !12 ; line 6
3+ %cmp = icmp sgt i32 %a, 2, !dbg !12 ; line 6
5 4 br i1 %cmp, label %if.then, label %if.end, !dbg !12 ; line 6
6 5
7 6 if.then: ; preds = %entry
8 7 call void @_Z12do_somethingb(i32 noundef %a), !dbg !13 ; line 7
9 8 br label %if.end, !dbg !13 ; line 7
10 9
11 10 if.end: ; preds = %if.then, %entry
11+ %mul = shl nsw i32 %a, 1, !dbg !12 ; line 6
12 12 ret i32 %mul, !dbg !14 ; line 8
13 13 }
14 14
15 15 ...
16 16 !12 = !DILocation(line: 6, scope: !9)
17 17 !13 = !DILocation(line: 7, scope: !9)
18 18 !14 = !DILocation(line: 8, scope: !9)
19 19
```

```
3 void do_something(int);
4
5 int fun(int a) {
6     if (a * 2 > 5)
7         do_something(a);
8     return a * 2;
9 }
```

Simplify br condition

Sink %mul

For each instruction in function

Line number changed?

Yes

Apply is_stmt

```
1  define dso_local noundef i32 @_Z3funi(i32 noundef %a) local_unnamed_addr #0 !dbg !9 {
2  entry:
3      %cmp = icmp sgt i32 %a, 2, !dbg !12                ; line 6
4      br i1 %cmp, label %if.then, label %if.end, !dbg !12 ; line 6
5
6  if.then:                                             ; preds = %entry
7      call void @_Z12do_somethingb(i32 noundef %a), !dbg !13 ; line 7
8      br label %if.end, !dbg !13                       ; line 7
9
10 if.end:                                             ; preds = %if.then, %entry
11     %mul = shl nsw i32 %a, 1, !dbg !12                ; line 6
12     ret i32 %mul, !dbg !14                            ; line 8
13 }
14
```

Accurate attribution but undesirable step

Solution – Key Instructions

The image shows a code editor with two panes. The left pane displays C++ source code for a function `fun` that takes two floats `x` and `y`. The right pane displays the corresponding LLVM IR, which includes instructions for loading `x` and `y`, performing a multiplication `%mul1 = fmul float %2, %3`, and a call to `@llvm.fmuladd.f32` to compute the final result.

```
1 float fun(float x, float y) {  
2     float z = x * x  
3         + y * y;  
4  
5     return z * z - x * y;  
6 }  
7
```

```
6     store float %x, ptr %x.addr, align 4  
7     store float %y, ptr %y.addr, align 4  
8     %0 = load float, ptr %x.addr, align 4, !dbg !14  
9     %1 = load float, ptr %x.addr, align 4, !dbg !15  
10    %2 = load float, ptr %y.addr, align 4, !dbg !16  
11    %3 = load float, ptr %y.addr, align 4, !dbg !17  
12    %mul1 = fmul float %2, %3, !dbg !18  
13    %4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19  
14    store float %4, ptr %z, align 4, !dbg !20  
15    %5 = load float, ptr %z, align 4, !dbg !21  
16    %6 = load float, ptr %z, align 4, !dbg !22  
17    %7 = load float, ptr %x.addr, align 4, !dbg !23  
18    %8 = load float, ptr %y.addr, align 4, !dbg !24  
19    %mul2 = fmul float %7, %8, !dbg !25  
20    %neg = fneg float %mul2, !dbg !26  
21    %9 = call float @llvm.fmuladd.f32(float %5, float %6, float %neg), !dbg !26  
22    ret float %9, !dbg !27  
23 }  
24  
25 declare float @llvm.fmuladd.f32(float, float, float) #1  
26
```

Prior work + inspiration

- **Key Instructions: Solving the Code Location Problem for Optimized Code** (C. Tice, S. L. Graham, 2000)
- **Debugging Optimized Code: Concepts and Implementation on DIGITAL Alpha Systems** (R. F. Brender et al)

Key Instructions overview

```
1 float fun(float x, float y) {
2   float z = x * x
3     + y * y;
4
5   return z * z - x * y;
6 }
7
```

Source atoms

```
6   store float %x, ptr %x.addr, align 4
7   store float %y, ptr %y.addr, align 4
8   %0 = load float, ptr %x.addr, align 4, !dbg !14
9   %1 = load float, ptr %x.addr, align 4, !dbg !15
10  %2 = load float, ptr %y.addr, align 4, !dbg !16
11  %3 = load float, ptr %y.addr, align 4, !dbg !17
12  %mul1 = fmul float %2, %3, !dbg !18
13  %4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19
14  store float %4, ptr %z, align 4, !dbg !20
15  %5 = load float, ptr %z, align 4, !dbg !21
16  %6 = load float, ptr %z, align 4, !dbg !22
17  %7 = load float, ptr %x.addr, align 4, !dbg !23
18  %8 = load float, ptr %y.addr, align 4, !dbg !24
19  %mul2 = fmul float %7, %8, !dbg !25
20  %neg = fneg float %mul2, !dbg !26
21  %9 = call float @llvm.fmuladd.f32(float %5, float %6, float %neg), !dbg !26
22  ret float %9, !dbg !27
23 }
24
25 declare float @llvm.fmuladd.f32(float, float, float) #1
26
```

Core ideas

- Source is made up of interesting **atoms** (ctrl-flow, assignments, calls)

Key Instructions overview

```
1 float fun(float x, float y) {  
2   float z = x * x;  
3     + y * y;  
4  
5   return z - x * y;  
6 }  
7
```

```
6   store float %x, ptr %x.addr, align 4  
7   store float %y, ptr %y.addr, align 4  
8   %0 = load float, ptr %x.addr, align 4, !dbg !14  
9   %1 = load float, ptr %x.addr, align 4, !dbg !15  
10  %2 = load float, ptr %y.addr, align 4, !dbg !16  
11  %3 = load float, ptr %y.addr, align 4, !dbg !17  
12  %mul1 = fmul float %2, %3, !dbg !18  
13  %4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19  
14  store float %4, ptr %z, align 4, !dbg !20  
15  %5 = load float, ptr %z, align 4, !dbg !21  
16  %6 = load float, ptr %z, align 4, !dbg !22  
17  %7 = load float, ptr %x.addr, align 4, !dbg !23  
18  %8 = load float, ptr %y.addr, align 4, !dbg !24  
19  %mul2 = fmul float %7, %8, !dbg !25  
20  %neg = fneg float %mul2, !dbg !26  
21  %9 = call float @llvm.fmuladd.f32(float %5, float %6, float %neg), !dbg !26  
22  ret float %9, !dbg !27  
23 }  
24  
25 declare float @llvm.fmuladd.f32(float, float, float) #1  
26
```

Core ideas

- Source is made up of interesting **atoms** (ctrl-flow, assignments, calls)
- Atoms typically have one “**key instruction**”

Key Instructions overview

The screenshot shows a code editor with two panes. The left pane displays C++ source code for a function `float fun(float x, float y)`. The right pane displays the corresponding LLVM IR. Orange callouts connect source code elements to IR instructions:

- Source atoms:** Points to the expression `x * y` in the assignment `float z = x * y;` and the expression `x * y` in the return statement `return z - x * y;`.
- Key instructions:** Points to the IR instructions `store float %4, ptr %z, align 4, !dbg !20` and `ret float %9, !dbg !27`.

Core ideas

- Source is made up of interesting **atoms** (ctrl-flow, assignments, calls...)
- Atoms typically have one “**key instruction**”
- Only apply `is_stmt` to key instructions

Prototype implementation

The image shows a side-by-side comparison of C++ source code and its LLVM IR. On the left, the C++ code is: `float fun(float x, float y) { float z = x * x + y * y; return z * z - x * y; }`. On the right, the LLVM IR is: `store float %x, ptr %x.addr, align 4; store float %y, ptr %y.addr, align 4; %0 = load float, ptr %x.addr, align 4, !dbg !14; %1 = load float, ptr %x.addr, align 4, !dbg !15; %2 = load float, ptr %y.addr, align 4, !dbg !16; %3 = load float, ptr %y.addr, align 4, !dbg !17; %mul1 = fmul float %2, %3, !dbg !18; %4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19; store float %4, ptr %z, align 4, !dbg !20; %5 = load float, ptr %z, align 4, !dbg !21; %6 = load float, ptr %z, align 4, !dbg !22; %7 = load float, ptr %x.addr, align 4, !dbg !23; %8 = load float, ptr %y.addr, align 4, !dbg !24; %mul2 = fmul float %7, %8, !dbg !25; %neg = fneg float %mul2, !dbg !26; %9 = call float @llvm.fmuladd.f32(float %5, float %6, float %neg), !dbg !26; ret float %9, !dbg !27; }`. Annotations include orange boxes labeled 'Source atoms' pointing to the C++ expressions and 'Key instructions' pointing to the corresponding IR instructions. A function signature `!DILocations(..., atomGroup: num, atomRank: num)` is shown with arrows pointing to specific IR instructions. Three callout boxes provide details: `DILocation(line: 2..., atomGroup: 1, atomRank: 2)`, `DILocation(line: 2..., atomGroup: 1, atomRank: 1)`, and `DILocation(line: 5..., atomGroup: 2, atomRank: 1)`.

Prototype implementation

The image shows a side-by-side comparison of C++ source code and its LLVM IR. On the left, the C++ code is: `float fun(float x, float y) { float z = x * x + y * y; return z * z - x * y; }`. On the right, the LLVM IR is: `store float %x, ptr %x.addr, align 4; store float %y, ptr %y.addr, align 4; %0 = load float, ptr %x.addr, align 4, !dbg !14; %1 = load float, ptr %x.addr, align 4, !dbg !15; %2 = load float, ptr %y.addr, align 4, !dbg !16; %3 = load float, ptr %y.addr, align 4, !dbg !17; %mul1 = fmul float %2, %3, !dbg !18; %4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19; store float %4, ptr %z, align 4, !dbg !20; %5 = load float, ptr %z, align 4, !dbg !21; %6 = load float, ptr %z, align 4, !dbg !22; %7 = load float, ptr %x.addr, align 4, !dbg !23; %8 = load float, ptr %y.addr, align 4, !dbg !24; %mul2 = fmul float %7, %8, !dbg !25; %neg = fneg float %mul2, !dbg !26; %9 = call float @llvm.fmuladd.f32(float %5, float %6, float %neg), !dbg !26; ret float %9, !dbg !27;`. Annotations include orange boxes labeled 'Source atoms' pointing to the C++ expressions `x * x`, `y * y`, and `z * z - x * y`. Another orange box labeled 'Key instructions' points to the IR instructions `store float %4, ptr %z, align 4, !dbg !20` and `ret float %9, !dbg !27`. A red box highlights `!dbg !19` in the IR, which is linked to a DILocation in the bottom right.

`!DILocations(..., atomGroup: num, atomRank: num)`

An atomGroup is roughly a Source Atom

`DILocation(line: 2..., atomGroup: 1, atomRank: 2)`

`DILocation(line: 2..., atomGroup: 1, atomRank: 1)`

`DILocation(line: 5..., atomGroup: 2, atomRank: 1)`

Prototype implementation

The image shows a side-by-side comparison of C++ source code and its LLVM IR. On the left, the C++ code is: `float fun(float x, float y) { float z = x * x + y * y; return z * z - x * y; }`. On the right, the LLVM IR is shown, with instructions corresponding to the C++ code. Annotations include orange boxes labeled 'Source atoms' pointing to the C++ expressions and 'Key instructions' pointing to the corresponding IR instructions. A call to `!dbg !19` in the IR is also highlighted.

`!DILocations(..., atomGroup: num, atomRank: num)`

An atomGroup is roughly a Source Atom

atomRank specifies is_stmt precedence

Three orange boxes containing DILocation annotations are shown, with arrows pointing from the IR to them. The annotations are: `DILocation(line: 2..., atomGroup: 1, atomRank: 2)`, `DILocation(line: 2..., atomGroup: 1, atomRank: 1)`, and `DILocation(line: 5..., atomGroup: 2, atomRank: 1)`. The numbers 1, 2, and 2 in the atomGroup and atomRank fields are circled in black.

Prototype implementation

The image shows a side-by-side comparison of C++ source code and its LLVM IR. On the left, the C++ code is: `float fun(float x, float y) { float z = x * x + y * y; return z * z - x * y; }`. On the right, the LLVM IR is shown with instructions like `store float %x, ptr %x.addr, align 4`, `%0 = load float, ptr %x.addr, align 4, !dbg !14`, `%4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19`, and `ret float %9, !dbg !27`. Annotations include orange boxes labeled 'Source atoms' pointing to `float z = x * x` and `+ y * y;` in the C++ code, and 'Key instructions' pointing to `store float %4, ptr %z, align 4, !dbg !20` and `ret float %9, !dbg !27` in the IR. A call to `@llvm.fmuladd.f32` in the IR is also annotated with `!dbg !19`.

`!DILocations(..., atomGroup: num, atomRank: num)`

An atomGroup is roughly a Source Atom

atomRank specifies is_stmt precedence

Converted to is_stmt at DWARF emission

Three orange boxes show DILocation annotations with arrows pointing to specific IR instructions: `DILocation(line: 2..., atomGroup: 1, atomRank: 2)` points to `store float %4, ptr %z, align 4, !dbg !20`; `DILocation(line: 2..., atomGroup: 1, atomRank: 1)` points to `ret float %9, !dbg !27`; and `DILocation(line: 5..., atomGroup: 2, atomRank: 1)` points to `%4 = call float @llvm.fmuladd.f32(float %0, float %1, float %mul1), !dbg !19`. The numbers 1, 2, and 2 in the atomGroup and atomRank fields are circled in black.

Costs (prototype)

stage1-ReleaseLTO-g:

Benchmark	Old	New
kimwitu++	59076M	59540M (+0.79%)
sqlite3	61163M	61616M (+0.74%)
consumer-typeset	54620M	55142M (+0.96%)
Bullet	111784M	112269M (+0.43%)
tramp3d-v4	176555M	177689M (+0.64%)
mafft	38854M	39137M (+0.73%)
ClamAV	85975M	86536M (+0.65%)
lencod	118636M	119331M (+0.59%)
SPASS	77932M	78433M (+0.64%)
7zip	272642M	273848M (+0.44%)
geomean	89451M	90042M (+0.66%)

stage1-00-g:

Benchmark	Old	New
kimwitu++	24932M	25167M (+0.94%)
sqlite3	4823M	4938M (+2.39%)
consumer-typeset	12635M	13114M (+3.79%)
Bullet	65029M	65445M (+0.64%)
tramp3d-v4	21231M	21477M (+1.16%)
mafft	6772M	6921M (+2.20%)
ClamAV	13722M	14000M (+2.02%)
lencod	12818M	13025M (+1.61%)
SPASS	14455M	14647M (+1.33%)
7zip	142937M	143449M (+0.36%)
geomean	18676M	18982M (+1.64%)

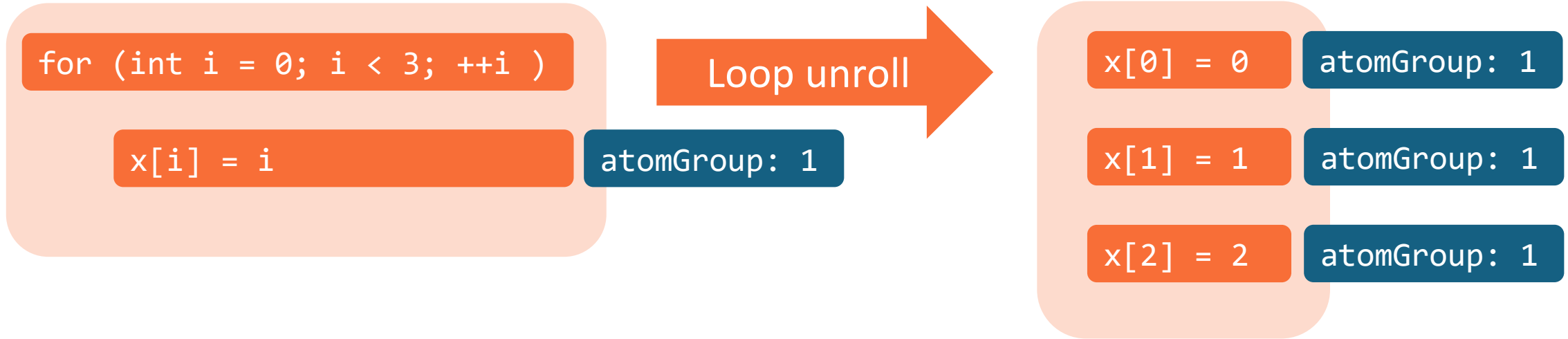
Prototype compile-time cost:

- *compile-time-tracker, CTMark, instructions:u*
- +0.66% to LTO builds
- +1.64% to unoptimized builds

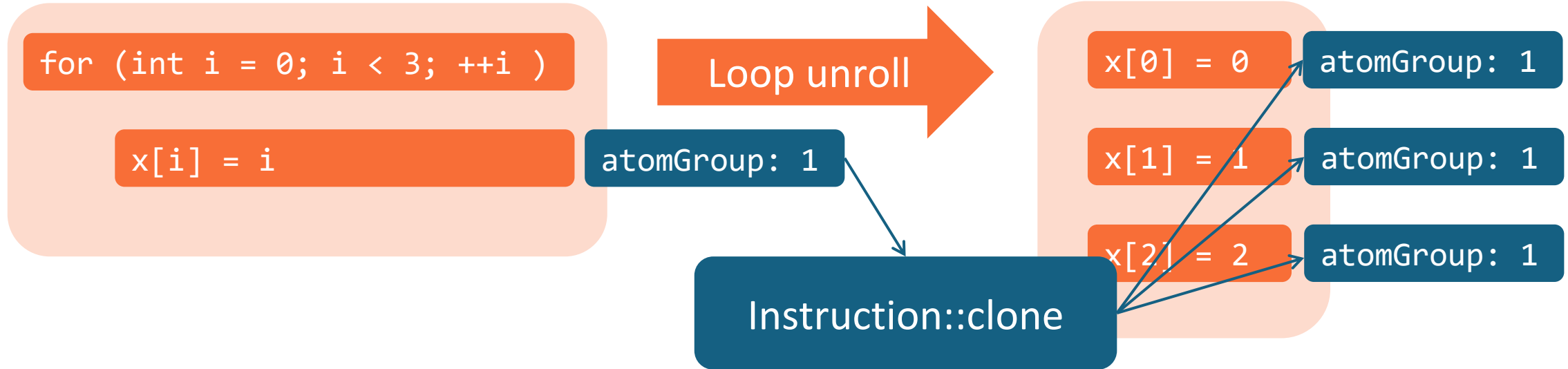
Dev-time cost:

- More info for some optimisations

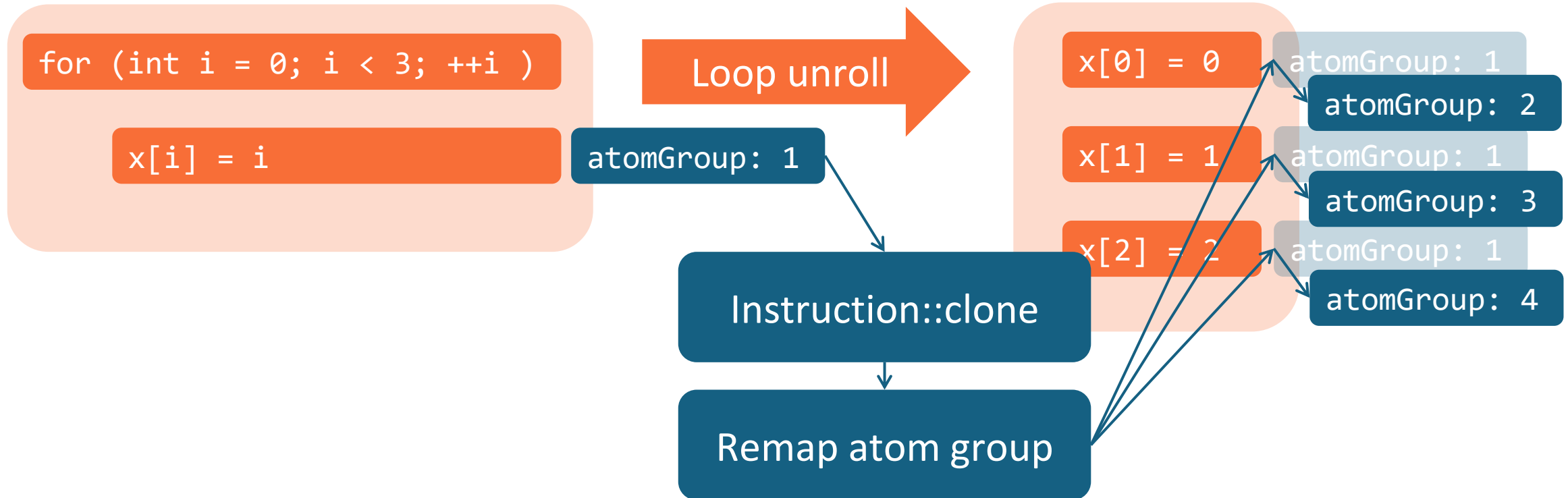
Costs - middle end work



Costs - middle end work



Remap atoms for duplicated control flow



Remap atoms for duplicated control flow

```
diff --git a/llvm/lib/Transforms/Utils/LoopUnroll.cpp b/llvm/lib/Transforms/Utils/LoopUnroll.cpp
index 20978cf2e748..b0fd24653bb5 100644
--- a/llvm/lib/Transforms/Utils/LoopUnroll.cpp
+++ b/llvm/lib/Transforms/Utils/LoopUnroll.cpp
@@ -731,6 +731,13 @@ llvm::UnrollLoop(Loop *L, UnrollLoopOptions ULO, LoopInfo *LI,
     NewPHI->eraseFromParent();
   }

+   // Remap source location atom instance. Do this now, rather than
+   // when we remap instructions, because remap is called once we've
+   // cloned all blocks (all the clones would get the same instance
+   // number!).
+   for (Instruction &I : *New)
+     RemapSourceAtom(&I, VMap);
+
   // Update our running map of newest clones
   LastValueMap[*BB] = New;
   for (ValueToValueMapTy::iterator VI = VMap.begin(), VE = VMap.end();
```

llvm/lib/Transforms/IPO/IROutliner.cpp

llvm/lib/Transforms/Scalar/JumpThreading.cpp

llvm/lib/Transforms/Scalar/SimpleLoopUnswitch.cpp

llvm/lib/Transforms/Utils/BreakCriticalEdges.cpp

llvm/lib/Transforms/Utils/CloneFunction.cpp

llvm/lib/Transforms/Utils/InlineFunction.cpp

llvm/lib/Transforms/Utils/LoopRotationUtils.cpp

llvm/lib/Transforms/Utils/LoopUnroll.cpp

llvm/lib/Transforms/Utils/SimplifyCFG.cpp

Risks / evaluation



Risks / evaluation

-O2 -g

```
1  int g1 = 1, g2 = 2, g3 = 3;
2
3  struct point { double x, y, z; };
4
5  [[clang::optnone]]
6  point fun(const point& p) { return p; }
7
8  [[clang::optnone]]
9  void f2(double) {}
10
11  int main() {
12      double y = (double)g2;
13      double z = (double)g3;
14      double x = (double)g1;
15      point p = { x * 2, y / 3, y / 3 + z + x * 2 };
16      point p2 = { p.x, p.y, p.z };
17      f2(p.z);
18      f2(p.x);
19      fun(p2);
20      p = fun(p);
21      return p.x + p.y + p.z;
22  }
23
```

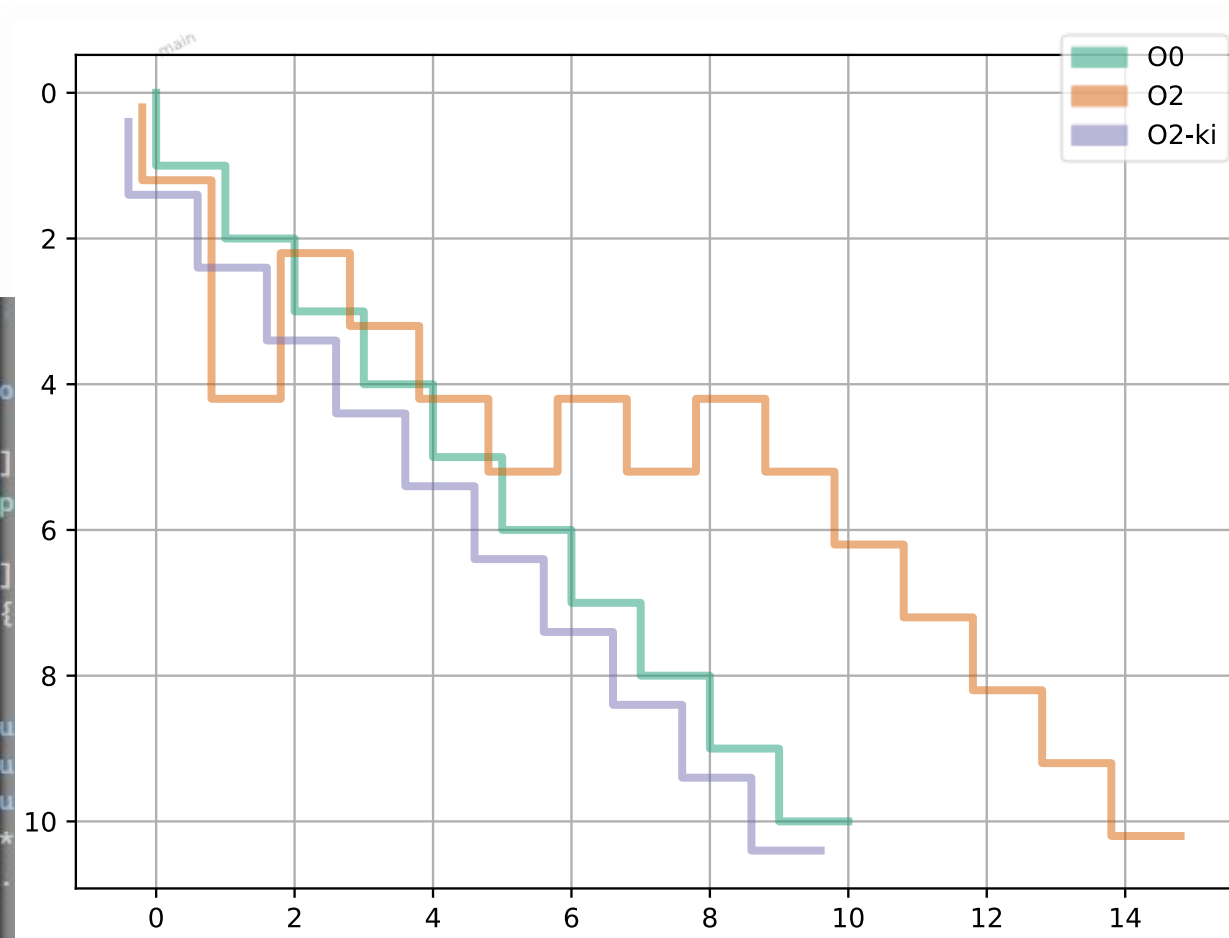
-O2 -g -fkey-instructions

```
1  int g1 = 1, g2 = 2, g3 = 3;
2
3  struct point { double x, y, z; };
4
5  [[clang::optnone]]
6  point fun(const point& p) { return p; }
7
8  [[clang::optnone]]
9  void f2(double) {}
10
11  int main() {
12      double y = (double)g2;
13      double z = (double)g3;
14      double x = (double)g1;
15      point p = { x * 2, y / 3, y / 3 + z + x * 2 };
16      point p2 = { p.x, p.y, p.z };
17      f2(p.z);
18      f2(p.x);
19      fun(p2);
20      p = fun(p);
21      return p.x + p.y + p.z;
22  }
23
```

```

1  int g1 = 1, g2 =
2
3  struct point { do
4
5  [[clang::optnone]
6  point fun(const p
7
8  [[clang::optnone]
9  void f2(double) {
10
11  int main() {
12  double y = (dou
13  double z = (dou
14  double x = (dou
15  point p = { x *
16  point p2 = { p.
17  f2(p.z);
18  f2(p.x);
19  fun(p2);
20  p = fun(p);
21  return p.x + p.y + p.z;
22  }
23

```

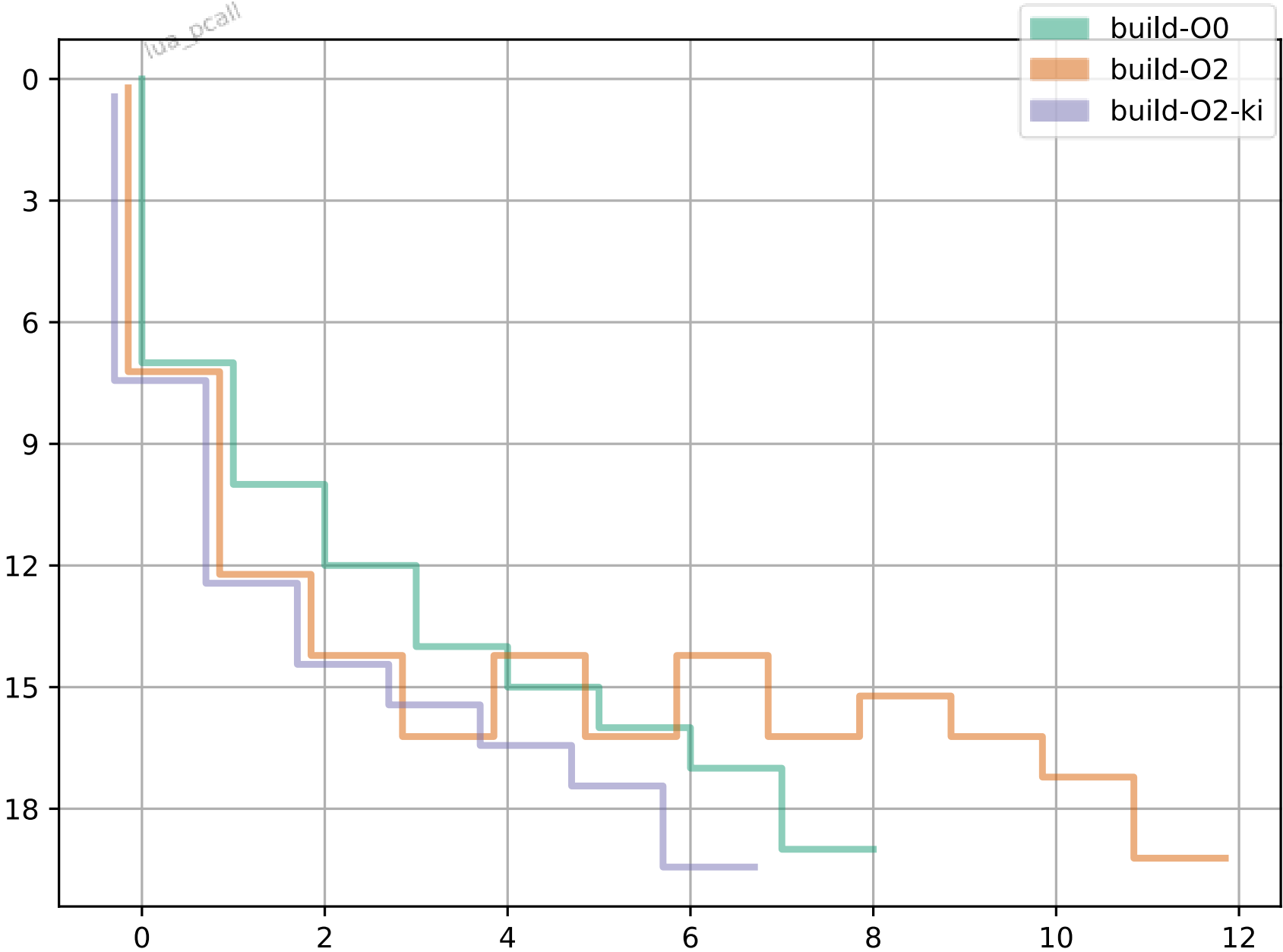


```

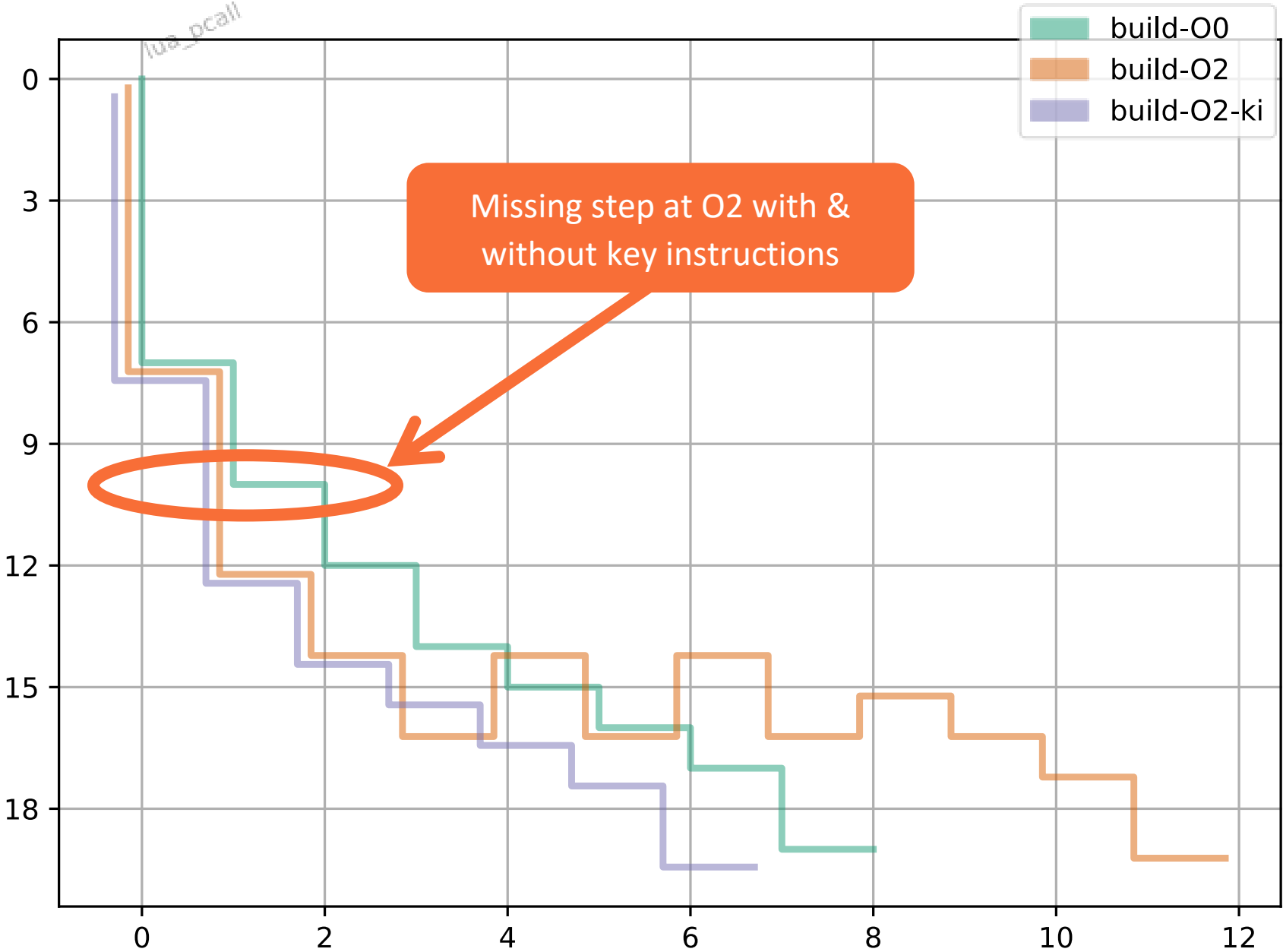
3 = 3;
x, y, z; };
&p) { return p; }
g2;
g3;
g1;
y / 3, y / 3 + z + x * 2 };
.y, p.z };
18  f2(p.x);
19  fun(p2);
20  p = fun(p);
21  return p.x + p.y + p.z;
22  }
23

```

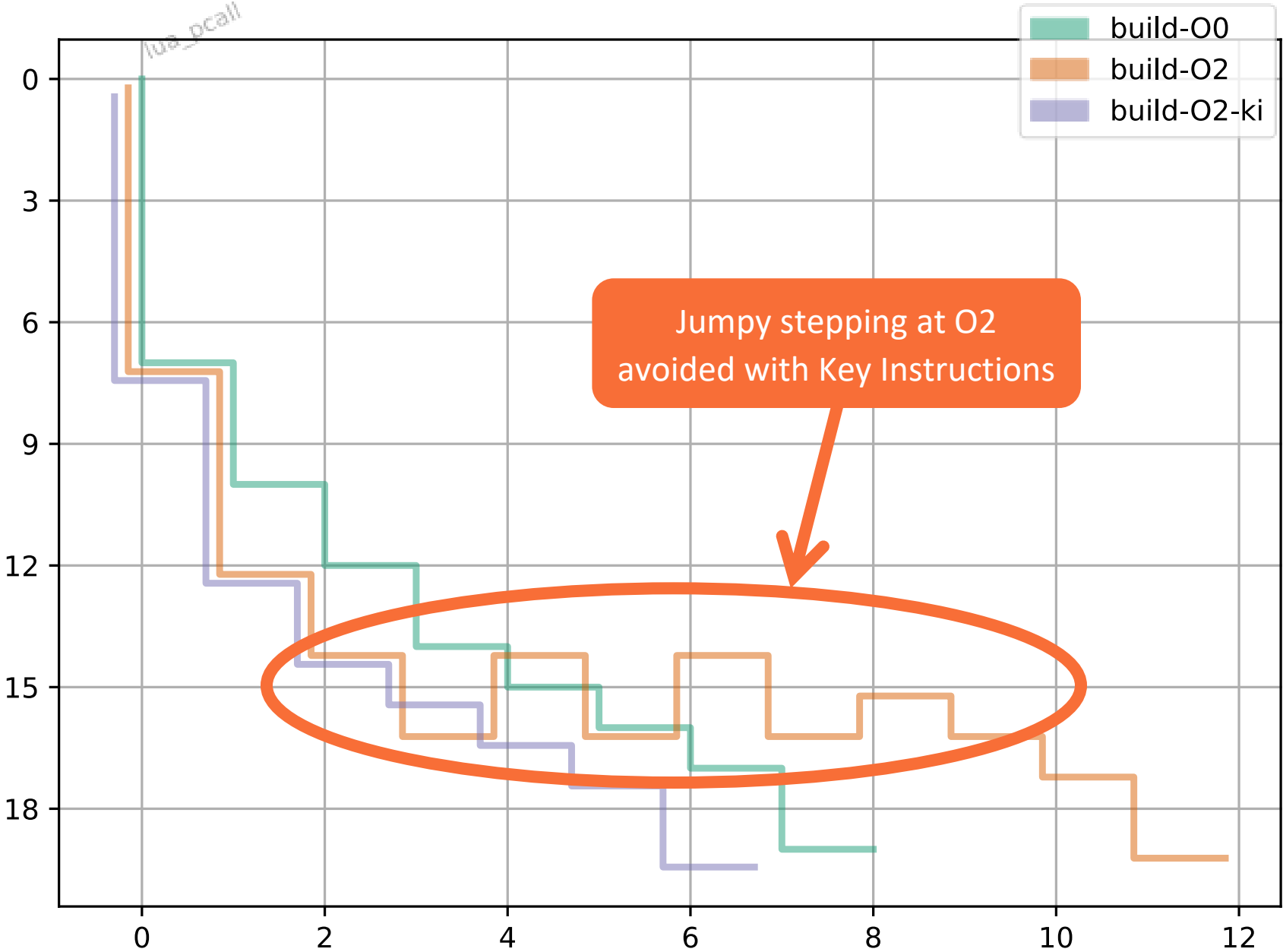
A debugging trace from Lua (bytecode interpreter, C)

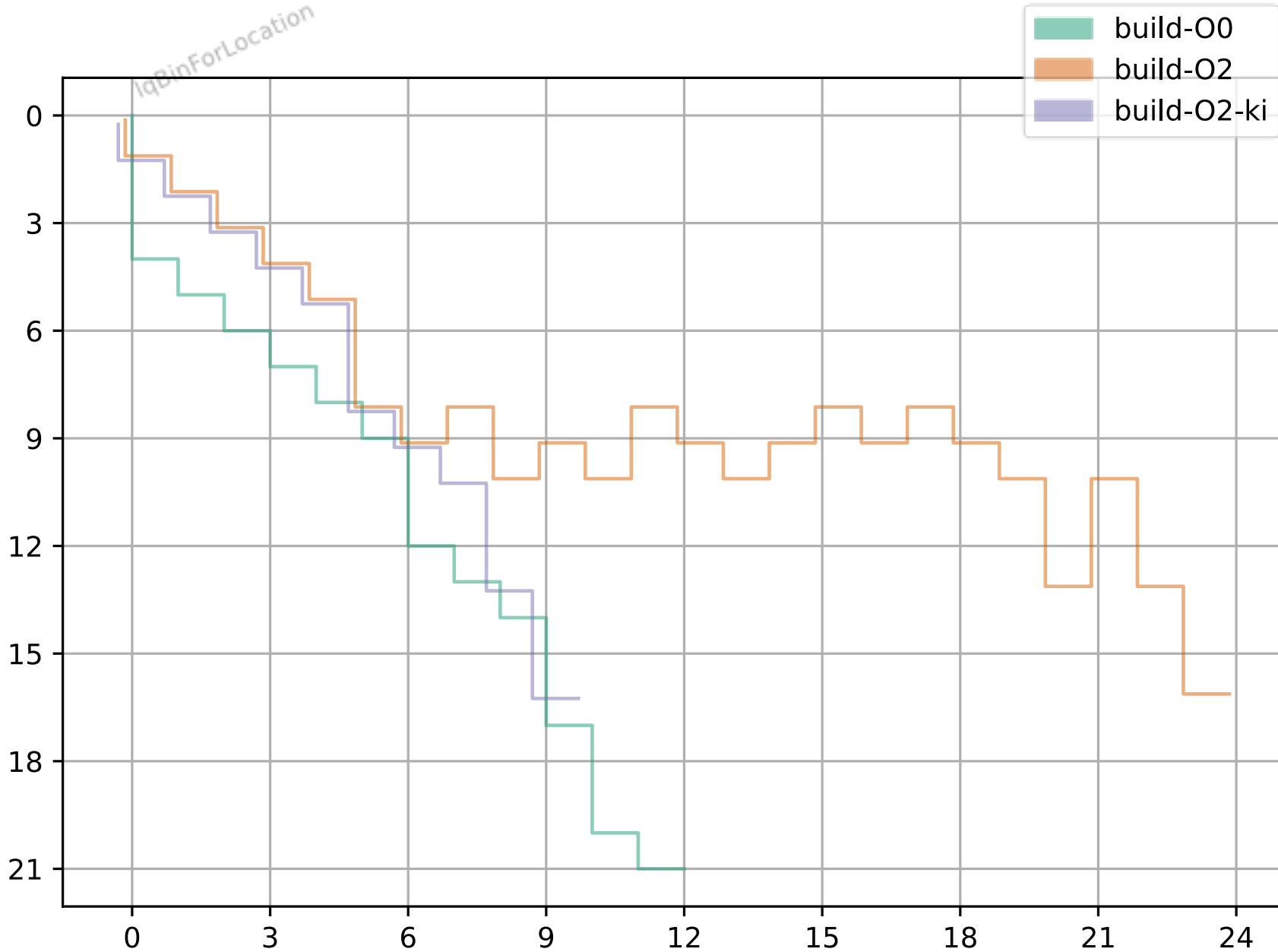


A debugging trace from Lua (bytecode interpreter, C)



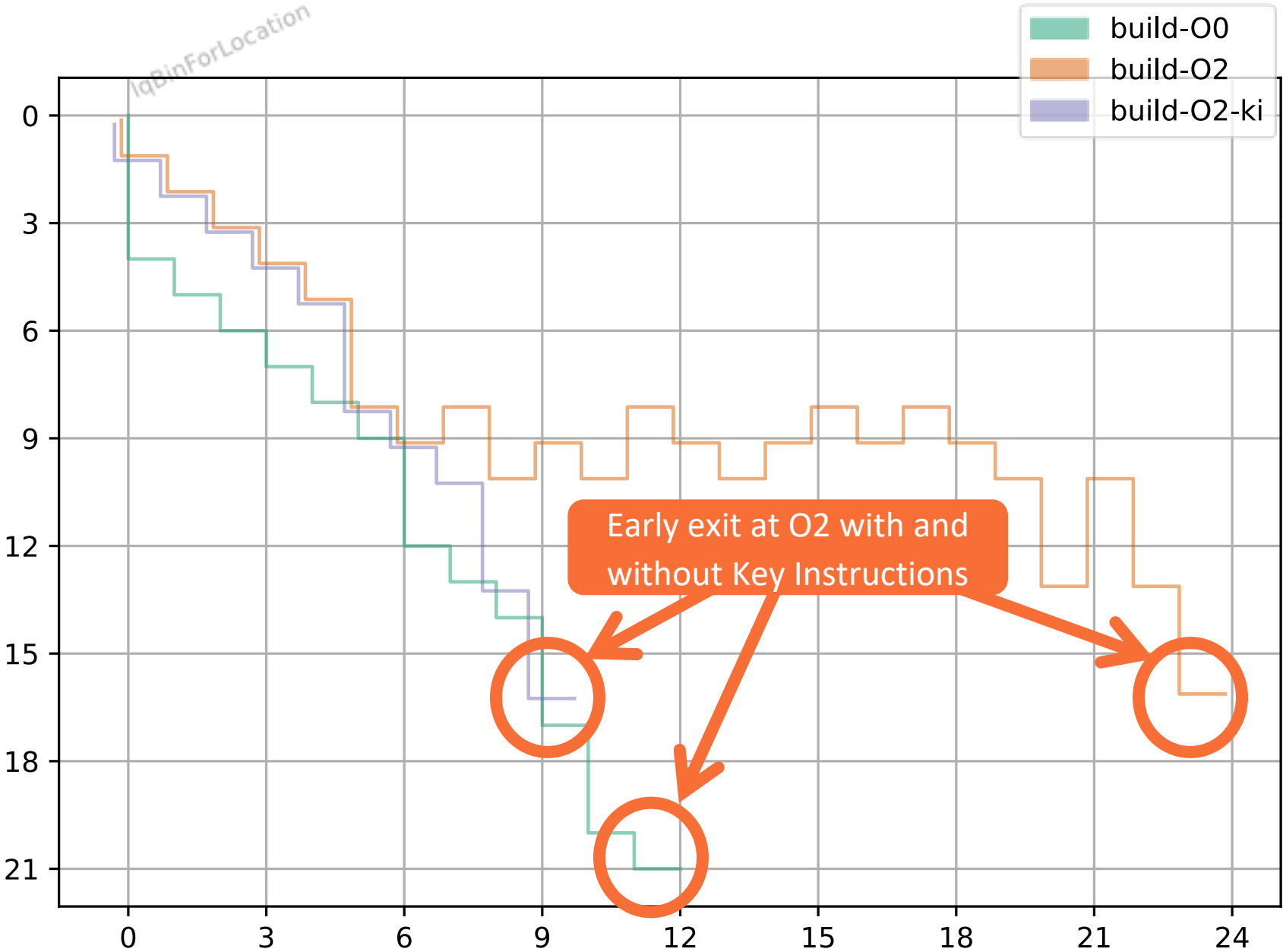
A debugging trace from Lua (bytecode interpreter, C)

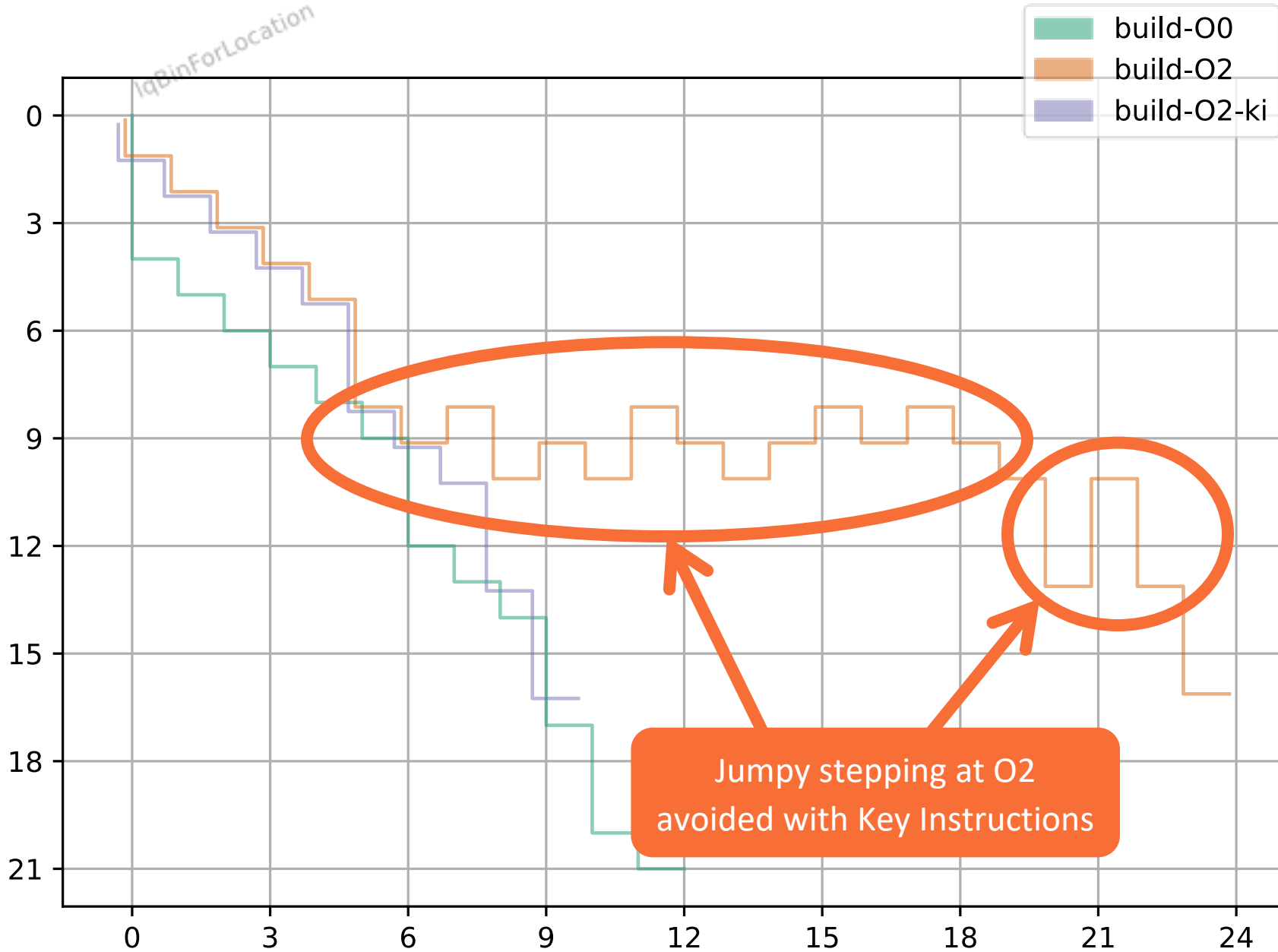




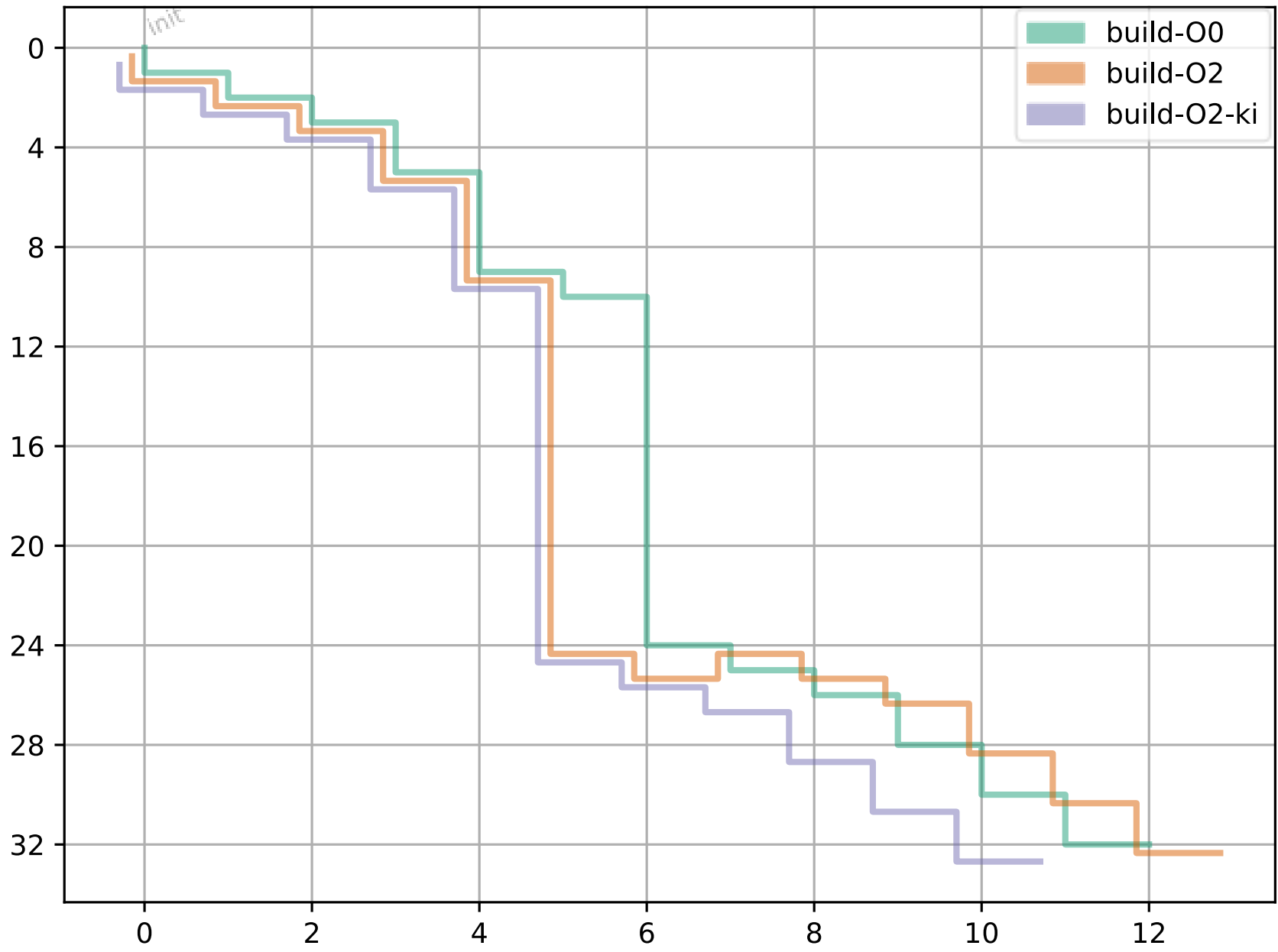
A debugging trace from OpenSteer (agent steering library, C++)

A debugging trace from OpenSteer (agent steering library, C++)

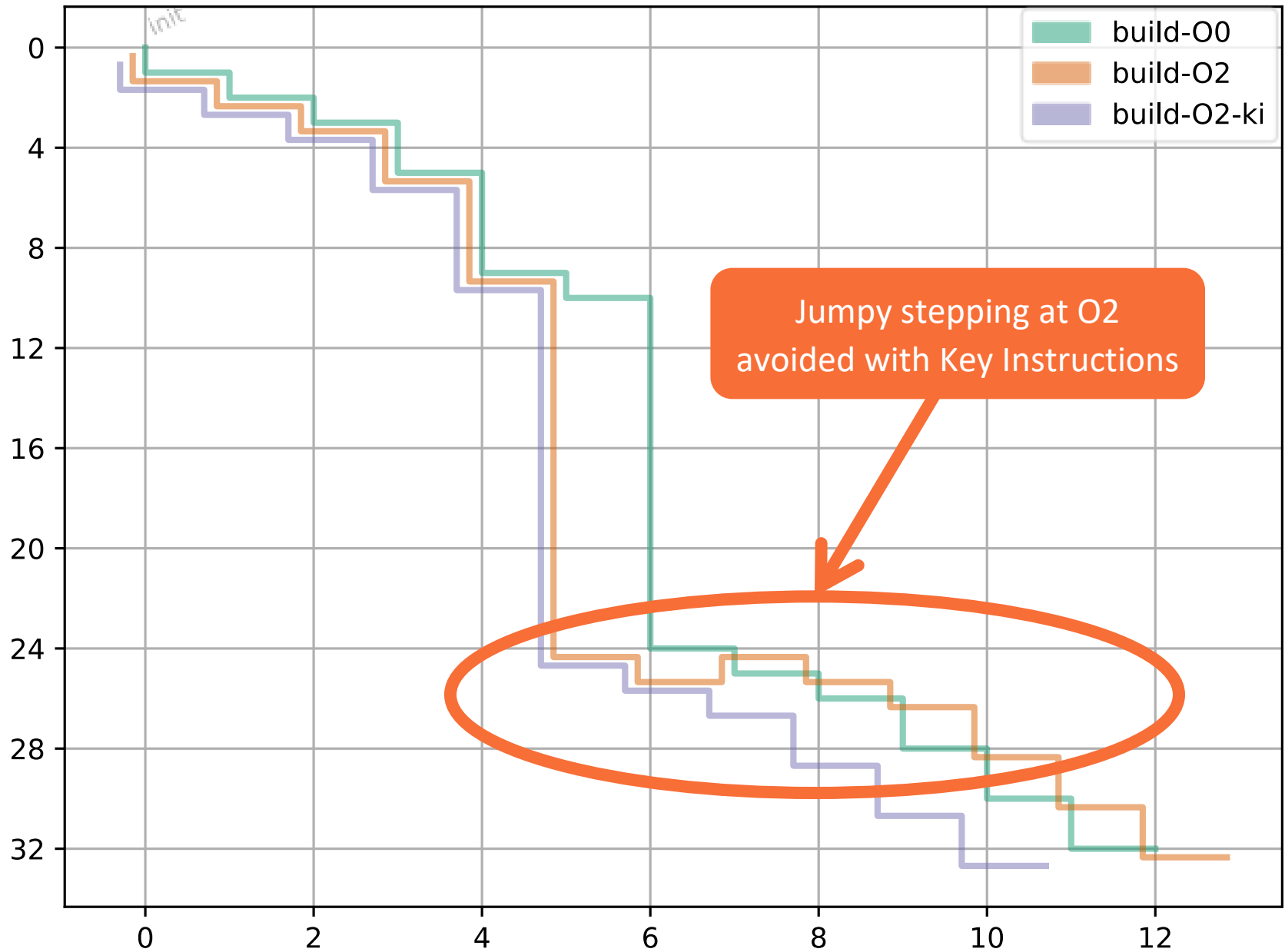




A debugging trace from OpenSteer (agent steering library, C++)



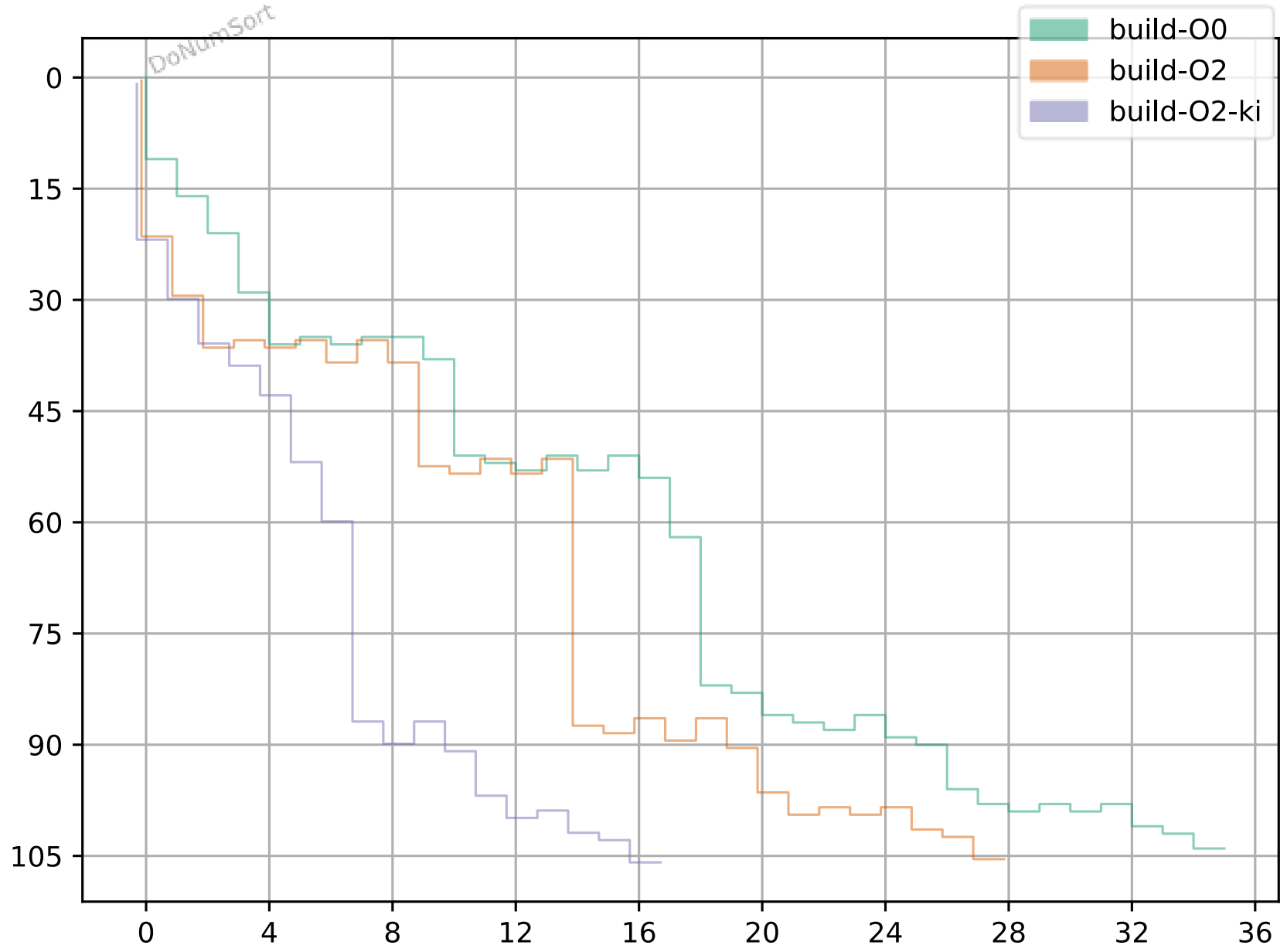
A debugging trace
 from AGG
 aa_demo
 (vector graphics
 library, C++)



A debugging trace from AGG aa_demo (vector graphics library, C++)

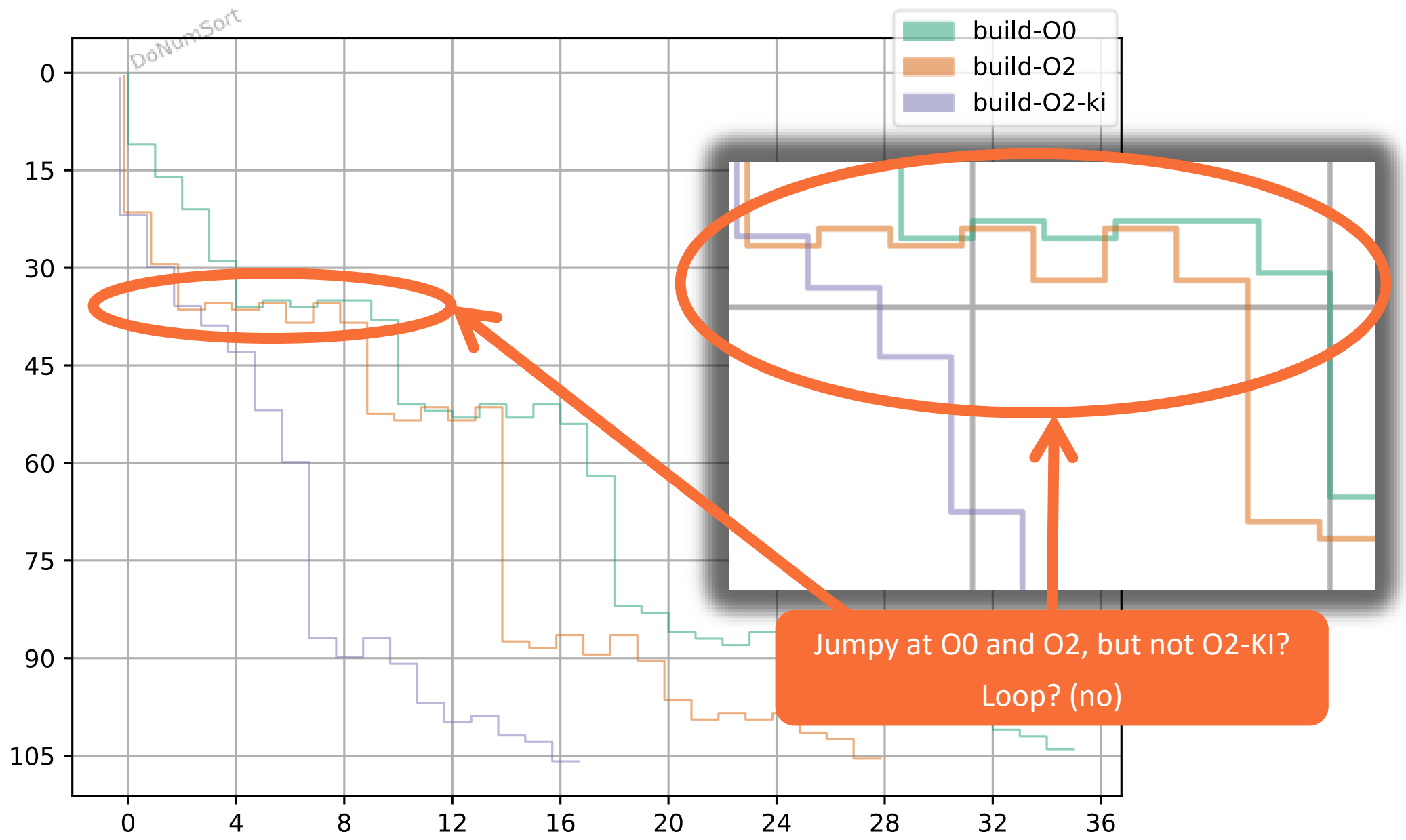
Jumpy stepping at O2 avoided with Key Instructions

DoNumSort



A debugging trace from BYTEmark (benchmarking, C)

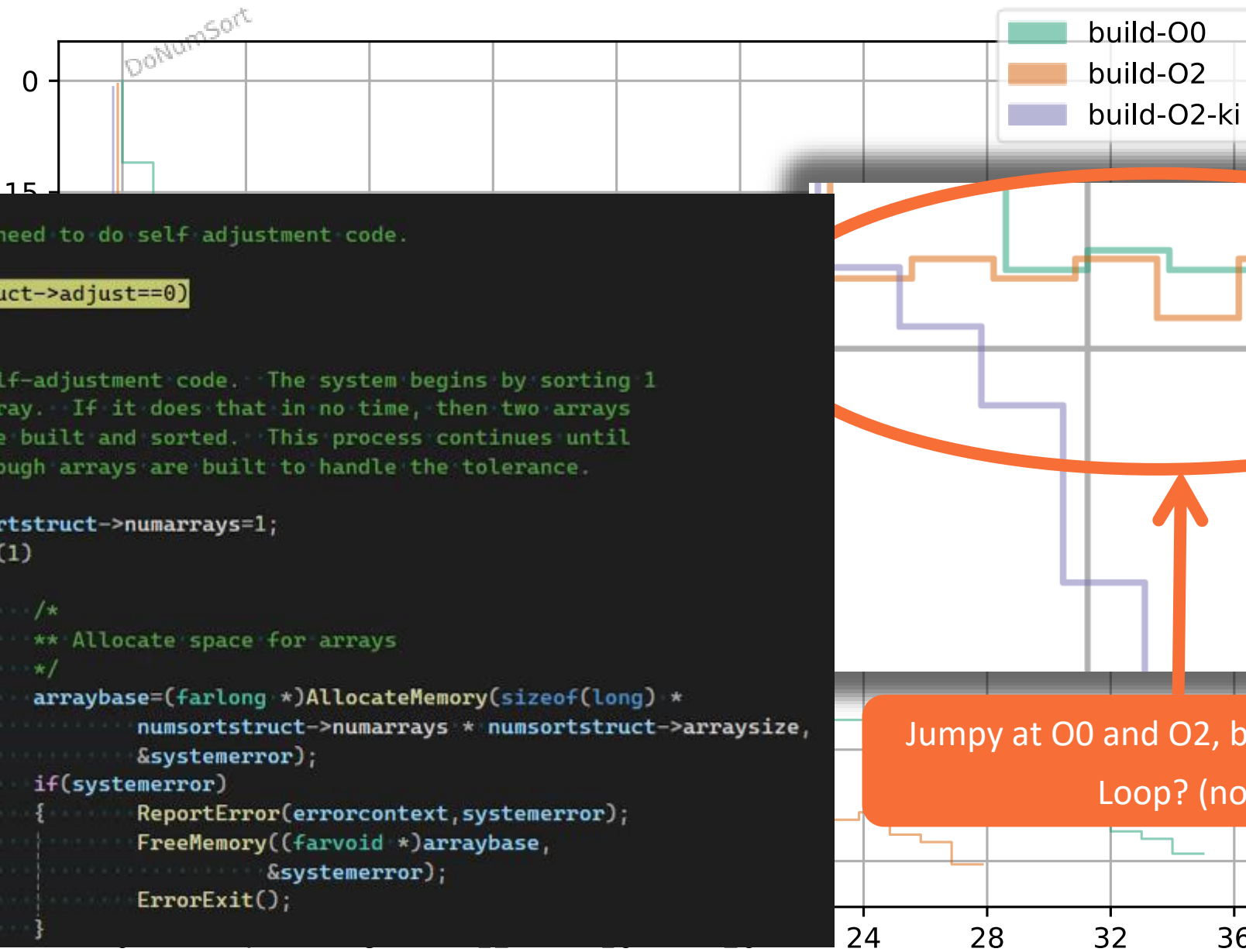
A debugging trace from BYTEmark (benchmarking, C)

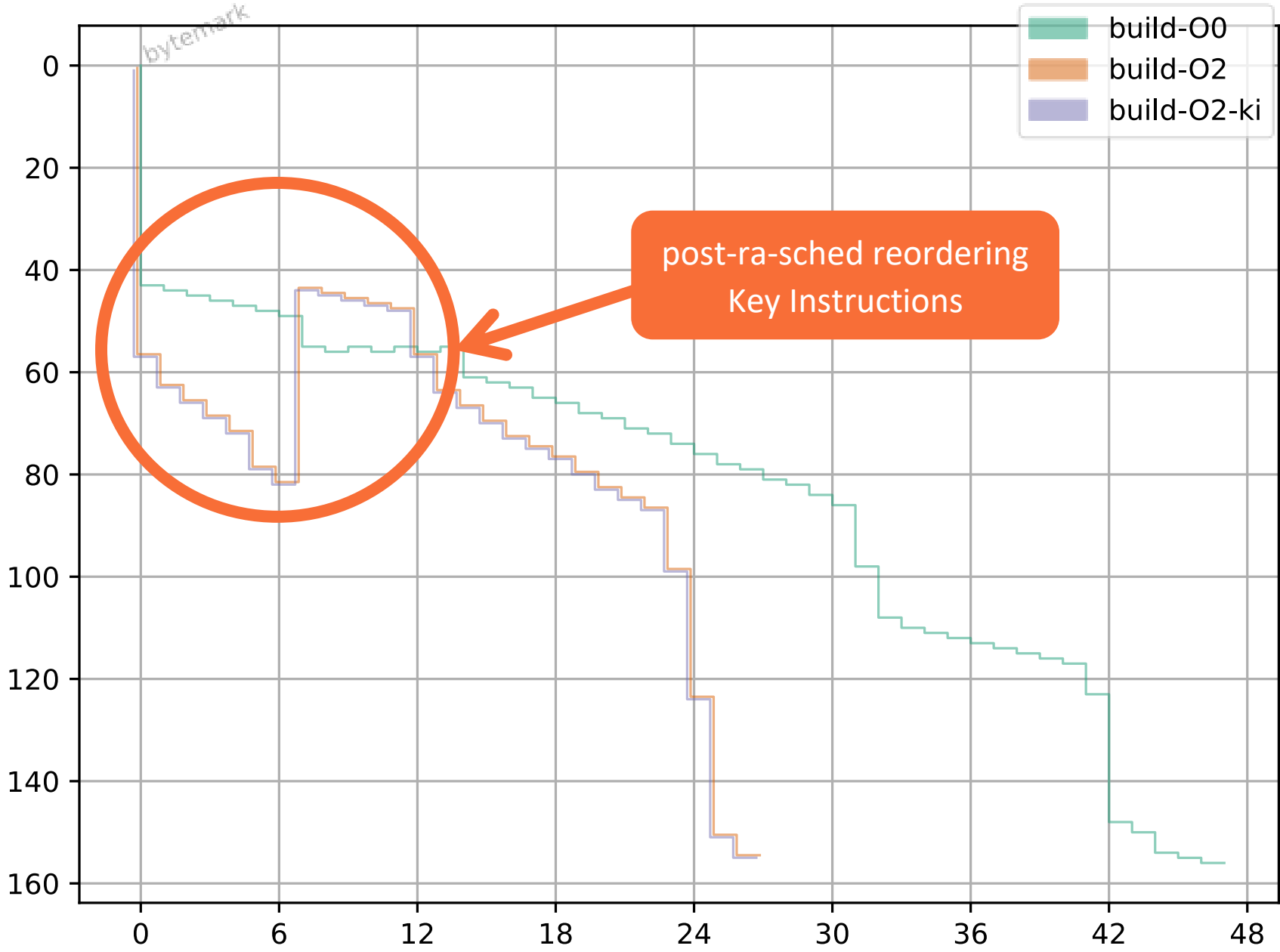


Single step through

BYTEmark (nbench)

```
290 /*
291 ** See if we need to do self adjustment code.
292 */
293 if(numsortstruct->adjust==0)
294 {
295     /*
296     ** Self-adjustment code. The system begins by sorting 1
297     ** array. If it does that in no time, then two arrays
298     ** are built and sorted. This process continues until
299     ** enough arrays are built to handle the tolerance.
300     */
301     numsortstruct->numarrays=1;
302     while(1)
303     {
304         /*
305         ** Allocate space for arrays
306         */
307         arraybase=(farlong *)AllocateMemory(sizeof(long) *
308         numsortstruct->numarrays * numsortstruct->arraysize,
309         &systemerror);
310         if(systemerror)
311         { ReportError(errorcontext,systemerror);
312         FreeMemory((farvoid *)arraybase,
313         &systemerror);
314         ErrorExit();
315     }
```





A debugging trace from BYTEmark (benchmarking, C)

Key Instructions summary

- Code motion and scheduling causes a lot of jumpiness
- Smarter `is_stmt` placement can greatly reduce impact
- Pass authors encode intent with new API
- See RFC for open question(s)

Thanks for listening

- LLVM's optimized-code line tables can be improved
- Use new APIs to encode info about optimisations for better debug info handling
- Source location defect finder
 - Improves attribution
 - <https://discourse.llvm.org/t/rfc-proposed-update-to-handling-debug-locations-in-llvm>
- Key instructions
 - Improves stepping
 - <https://discourse.llvm.org/t/rfc-improving-is-stmt-placement-for-better-interactive-debugging>

A debugging trace from BYTEmark (benchmarking, C)

