PlayStation

Improving debug locations for variables in memory in optimized code

Orlando Cazalet-Hyams

sn systems

Source code

```
int Local;
useAddress(&Local);
```

IR

```
%Local = alloca i32, align 4
call void @useAddress(i32* %Local)
```

The problem

In a debugger, sometimes we see...

1. Variable “optimized out” when it is still live in memory

2. Variable is available, but the value is noncurrent (stale, early, rubbish)

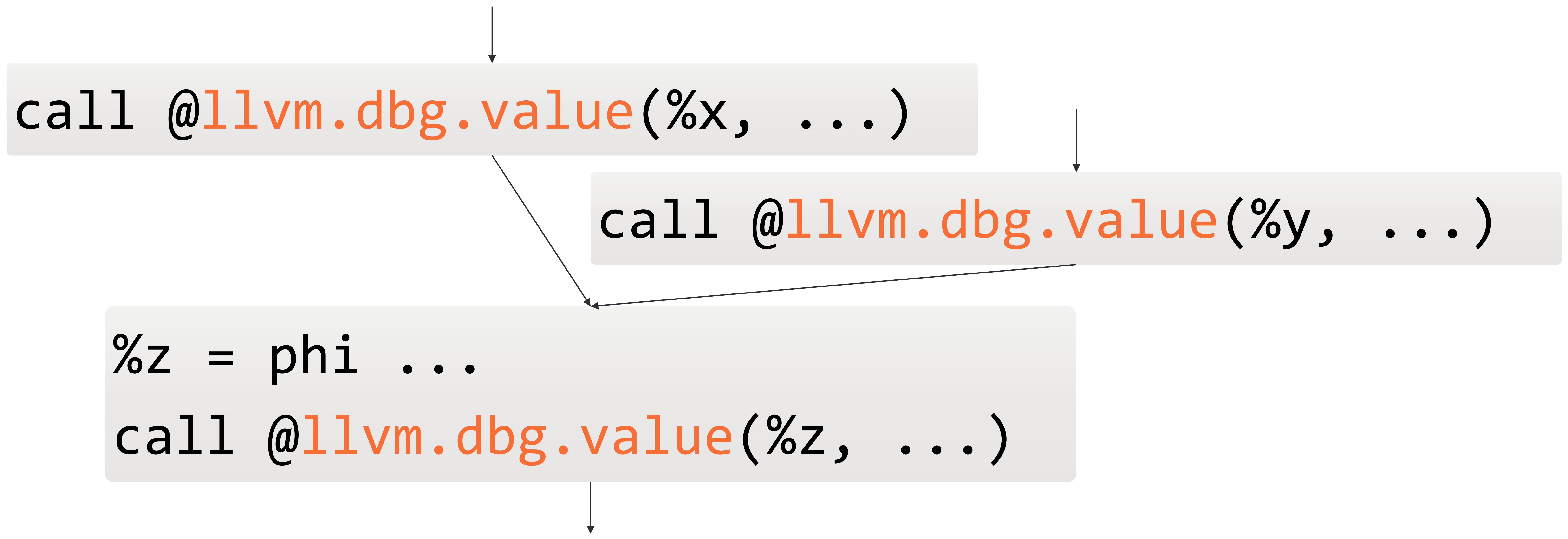
Core difficulty

	Value after statement		
Source	Var	Mem	Optimized IR
`int Local;`	?	?	`%Local = alloca i32, align 4`
`Local = x;`	x	?	`; store / load eliminated`
`useValue(Local);`	x	?	`call void @useValue(i32 %x)`
`Local = y;`	y	y	`store i32 %y, i32* %Local`
`useAddress(&Local);`	?	?	`call void @useAddress(i32* %Local)`

Debug intrinsics

@llvm.dbg.value

- –Many per variable
- –SSA value at point
- –Control-flow dependent



```
call @llvm.dbg.value(%x, ...)
```

```
call @llvm.dbg.value(%y, ...)
```

```
%z = phi ...
call @llvm.dbg.value(%z, ...)
```

@llvm.dbg.declare

- –One per variable
- –Memory location for scope

```
entry:
  %Local = alloca i32, align 4
  call @llvm.dbg.declare(%Local, ...)
```

The cause of the problem

Location tracking method chosen early in the optimisation pipeline

–Before most optimisations occur

`@llvm.dbg.declare(...)`

- Ignorant of changes to memory operations
- Always available, sometimes incorrect

`@llvm.dbg.value(...)`

- Avoids memory locations
- Correct locations with short lifetimes

A solution (prototype)

@llvm.dbg.assign

- –Many per variable
- –Value *and* memory location at point (2 locations!)
- –Choose best location later
- –Control-flow dependent

```
store float %f, float* %f.addr, align 4
call void @llvm.dbg.assign(%f, %f.addr, ...)
```

- –One more problem...

A solution (prototype)

```
store float %f, float* %f.addr, align 4
call void @llvm.dbg.assign(%f, %f.addr, ...)
```

A solution (prototype)

```
~~store float %f, float* %f.addr, align 4~~
call void @llvm.dbg.assign(%f, %f.addr, ...)
```

A solution (prototype)

```
store float %f, float* %f.addr, align 4, !DIAssignID !1
call void @llvm.dbg.assign(%f, %f.addr, ..., metadata !1)
```

A solution (prototype)

```
~~store float %f, float* %f.addr, align 4, !DIAssignID !1~~
call void @llvm.dbg.assign(%f, %f.addr, ..., metadata !1)
```

A solution (prototype): dataflow

```
%Local = alloca i32, align 4, !DIAssignID !1
call @llvm.dbg.assign(undef, %Local, !1)
; store / load eliminated
call @llvm.dbg.assign(%x, %Local, !2)
call void @useValue(i32 %x)
store i32 %y, i32* %Local, !DIAssignID !3
call @llvm.dbg.assign(%y, %Local, !3)
call void @useAddress(i32* %Local)
```

Variable assignment	Memory assignment	Resultant Location
-	!1	-
!1	!1	memory
!2	!1	%x

What would this mean for you?

Preserve DIAssignID metadata attachments

```
store float %f, float* %f.addr, align 4, !DIAssignID !1
```

Split @llvm.dbg.assign when stores are split/shortened

Deleting (whole) stores

Action

Do nothing

Example

```
~~store float %f, float* %f.addr, align 4, !DIAssignID !1~~
call void @llvm.dbg.assign(..., metadata !1)
```

Code

```
Store->eraseFromParent()
```

Moving stores

Action

Preserve the DIAssignID metadata attached to the store

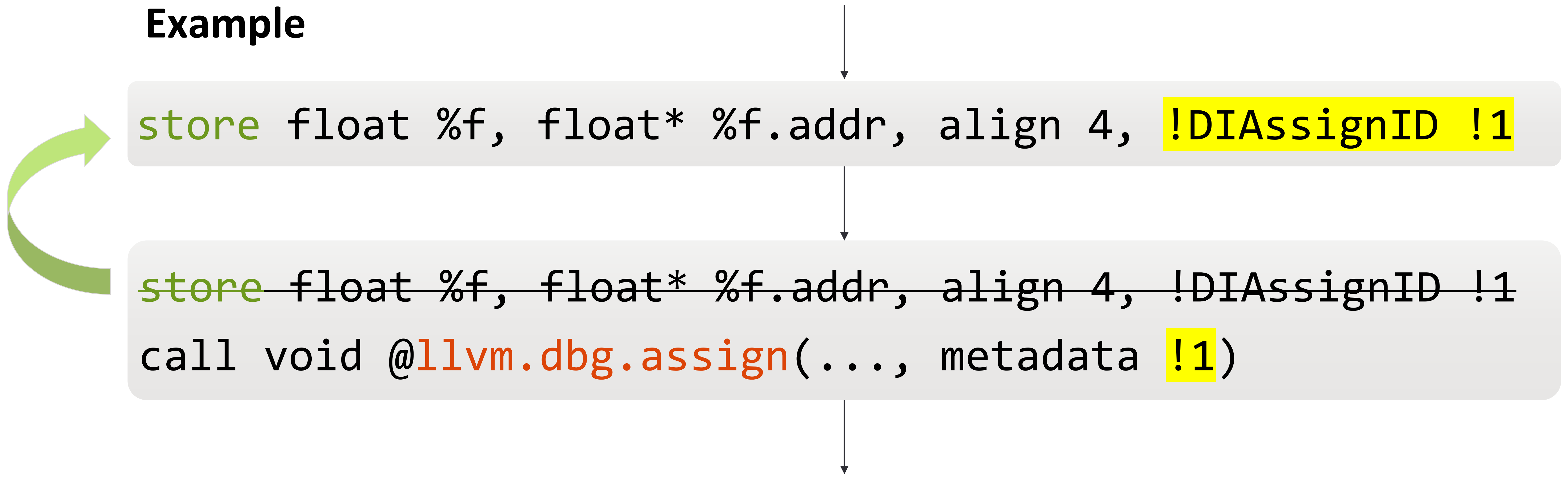
Code

moveBefore, moveAfter, and clone preserve DIAssignID automatically

Replacing the store (e.g. IRBuilder):

```
NewStore->copyMetadata(
    OldStore,
    LLVMContext::MD_DIAssignID);
```

Example



`store float %f, float* %f.addr, align 4, !DIAssignID !1`

~~store float %f, float* %f.addr, align 4, !DIAssignID !1~~

`call void @llvm.dbg.assign(..., metadata !1)`

Aggregating contiguous stores

Action

Merge the DIAssignID metadata attachments

Code option #1

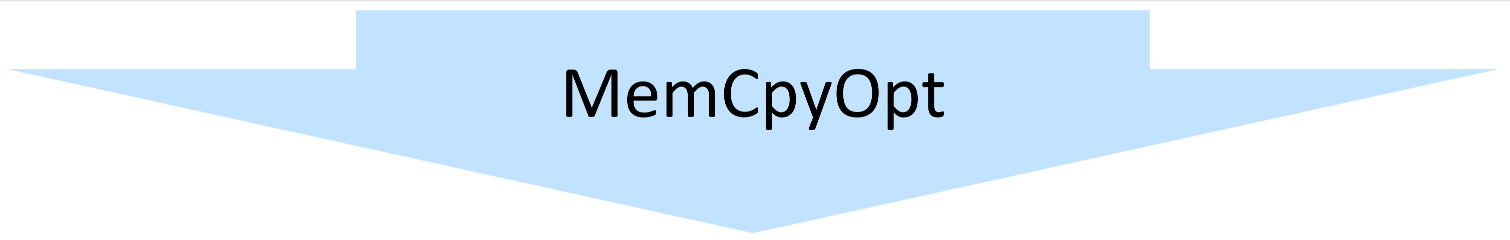
```
for (auto *Store : Stores)
    combineMetadata(
        NewStore, Store);
```

Code option #2

```
NewStore->mergeDIAssignID(Stores)
```

Example

```
%arrayidx0 = getelementptr ..., %Array, i64 0, i64 0
store i32 0, i32* %arrayidx0, !DIAssignID !1
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment,  0, 32), !1)
%arrayidx1 = getelementptr, ... %Array, i64 0, i64 1
store i32 0, i32* %arrayidx1, !DIAssignID !2
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 32, 32), !2)
; Repeat for index 2 & 3
```



```
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment,  0, 32), !5)
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 32, 32), !5)
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 64, 32), !5)
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 96, 32), !5)
call void @llvm.memset(i8* %Array, i8 0, i64 16, ...), !DIAssignID !5
```

Merging stores

Action

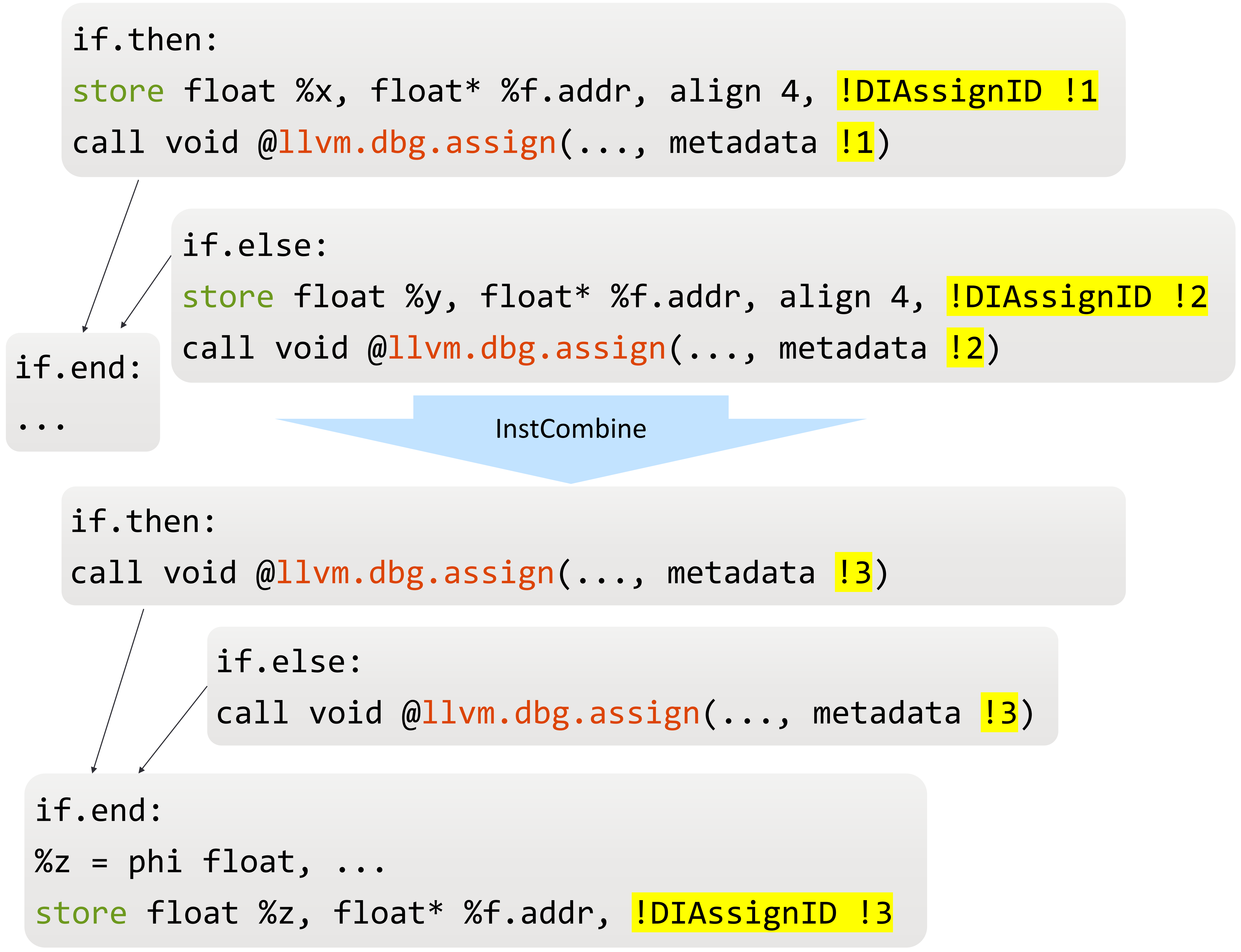
Merge the DIAssignID metadata attachments

Code option #1

```
for (auto *Store : Stores)
    combineMetadata(
        NewStore, Store);
```

Code option #2

```
NewStore->mergeDIAssignID(Stores)
```



```
if.then:
store float %x, float* %f.addr, align 4, !DIAssignID !1
call void @llvm.dbg.assign(..., metadata !1)
```

```
if.else:
store float %y, float* %f.addr, align 4, !DIAssignID !2
call void @llvm.dbg.assign(..., metadata !2)
```

```
if.end:
...
```

```
if.then:
call void @llvm.dbg.assign(..., metadata !3)
```

```
if.else:
call void @llvm.dbg.assign(..., metadata !3)
```

```
if.end:
%z = phi float, ...
store float %z, float* %f.addr, !DIAssignID !3
```

Splitting and shortening stores (SROA, DSE)

Action

Ensure affected variable fragments are represented

Example

```
; memset(dest, /*value=*/0,  /*size=*/40)
call void @llvm.memset.p0i8.i64(i8* %dest, i8 0, i64 40, i1 false), !DIAssignID !1
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 0, 320), !1)
; memset(dest, /*value=*/8,  /*size=*/16)
call void @llvm.memset.p0i8.i64(i8* %dest, i8 8, i64 16, i1 false), !DIAssignID !2
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 0, 128), !2)
```

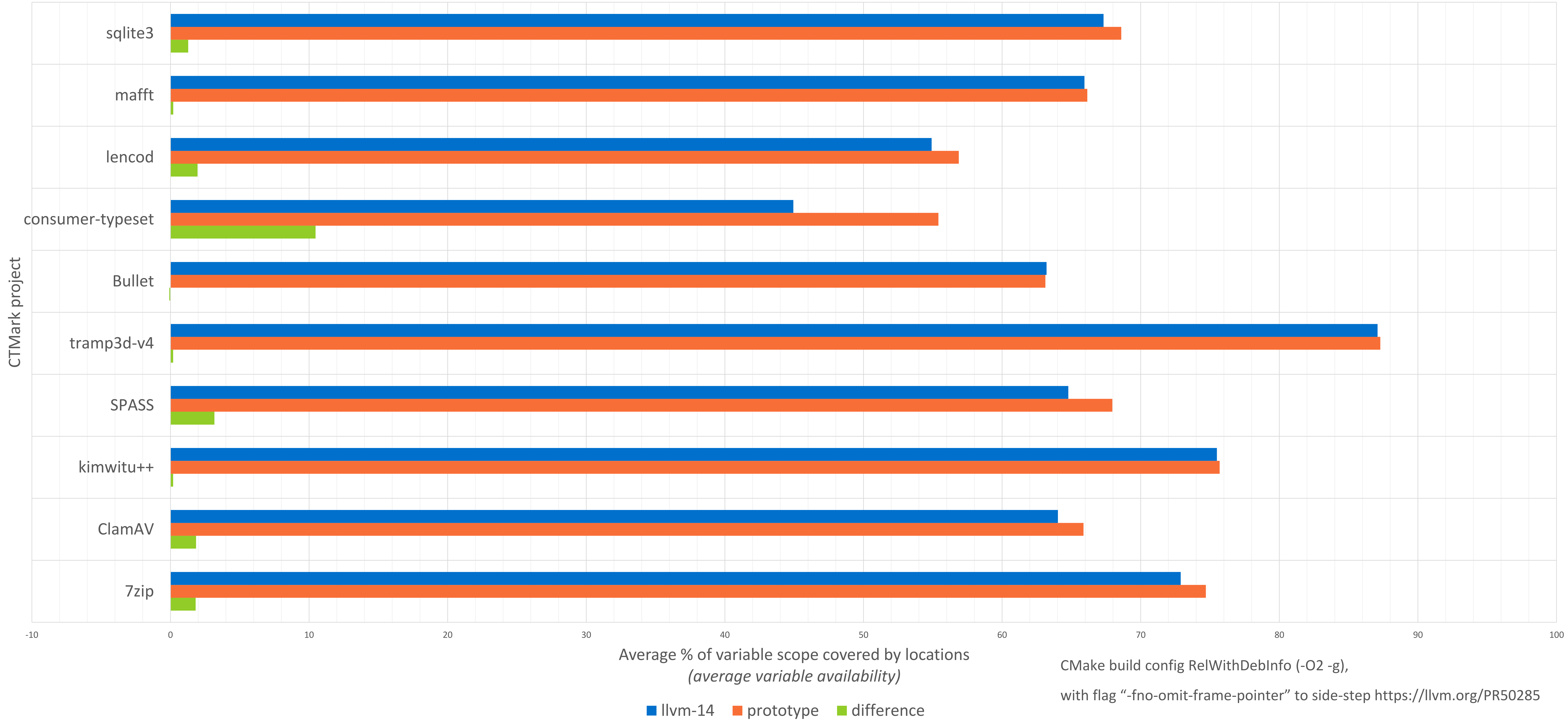
DSE

memset shortened

Sentinel indicates memory loc is invalid for fragment

```
; memset(dest + 16, /*value=*/0,  /*size=*/24)
call void @llvm.memset.p0i8.i64(i8* %offset, i8 0, i64 24, i1 false), !DIAssignID !1
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 0, 320), metadata !1)
call void @llvm.dbg.assign(..., mem=undef, (DW_OP_LLVM_fragment, 0, 128), !1)
; memset(dest, /*value=*/8,  /*size=*/16)
call void @llvm.memset.p0i8.i64(i8* %dest, i8 8, i64 16, i1 false), !DIAssignID !2
call void @llvm.dbg.assign(..., (DW_OP_LLVM_fragment, 0, 128), !2)
```

Comparing variable location coverage of CTMark projects compiled with llvm-14 and the assignment tracking prototype



CMake build config RelWithDebInfo (-O2 -g),

with flag "-fno-omit-frame-pointer" to side-step https://llvm.org/PR50285

Thanks for listening

Discourse post:

[RFC] Assignment tracking: A better way of specify variable locations in IR

https://discourse.llvm.org/t/rfc-assignment-tracking-a-better-way-of-specifying-variable-locations-in-ir/62367

- –More details
- –Limitations
- –Examples
- –Next steps