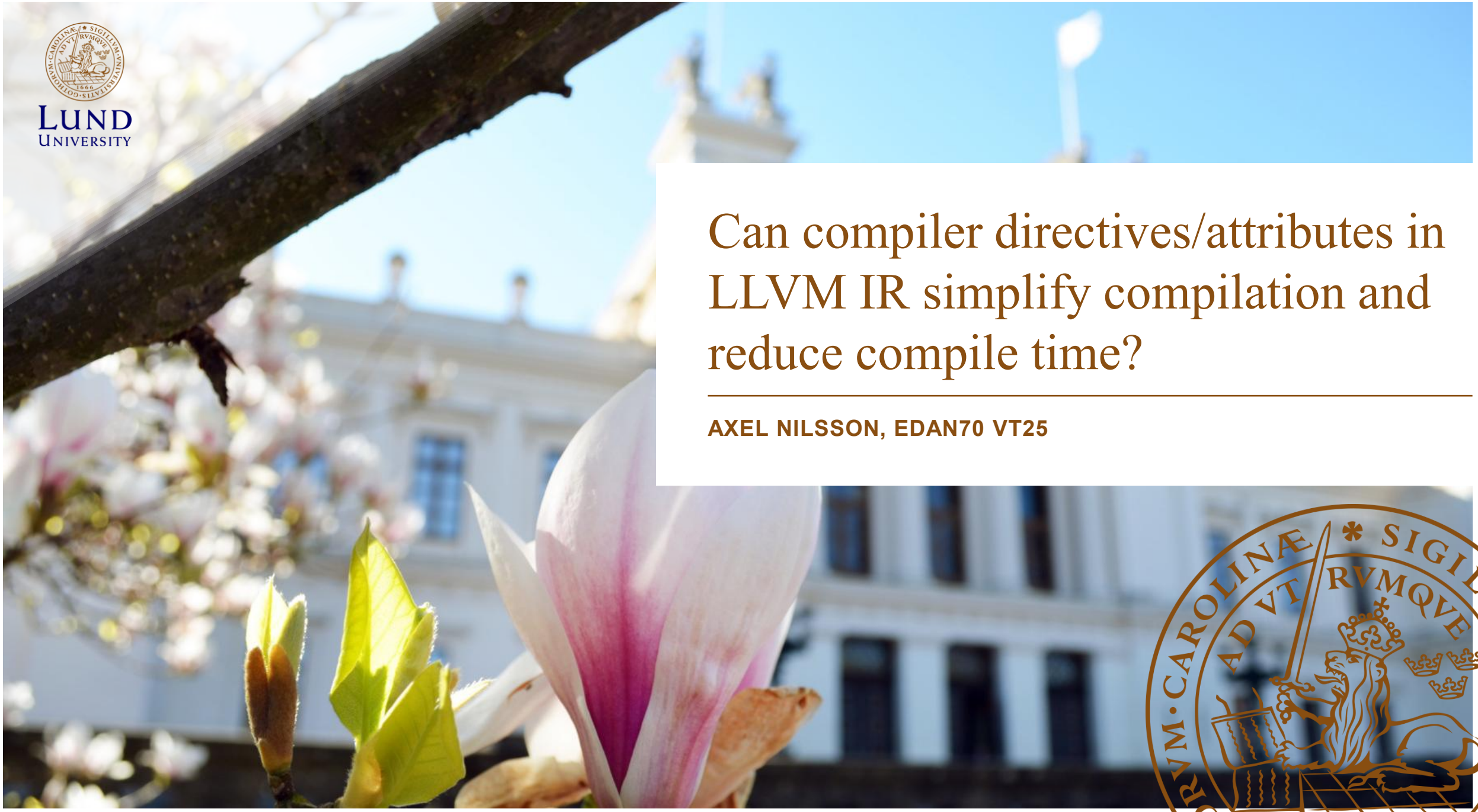

Can compiler directives/attributes in LLVM IR simplify compilation and reduce compile time?

AXEL NILSSON, EDAN70 VT25

Modelon and Modelon Impact

- System modeling and simulation
- Used in:
  - Aerospace
  - Energy
  - Automotive

LUNDS UNIVERSITET

Modelon

LLVM

- Compiler toolchain
- Used in:
  - Clang
  - Rust
- Objectoriented
- Easily extended
- LLVM-IR

Attributes

Attribute	Average (ms)	Change (%)	Std. Dev. (ms)
Baseline	1144.9	0.00	2.20
FramePointer: All	1160.3	1.35	4.07
FramePointer: None	1148.5	0.32	1.55
NoImplicitFloat	1155.4	0.92	5.99
noInline	1151.9	0.61	1.80
optnone	1184.1	3.43	9.03

Table 2: Results of 100 test runs with specific attributes added to all functions.

- Used to give hints to compiler
- No improvement to the Compilationtime

Jacobian Matrix

```
define void @dae_block_44_jacobian_structA21_col(i32** %0) {
func_entry:
  %1 = load i32*, i32** %0, align 8
  %2 = getelementptr i32, i32* %1, i32 0
  store i32 0, i32* %2, align 4
  %3 = getelementptr i32, i32* %1, i32 0
  store i32 0, i32* %3, align 4
  %4 = getelementptr i32, i32* %1, i32 0
  store i32 0, i32* %4, align 4
  %5 = getelementptr i32, i32* %1, i32 0
  store i32 0, i32* %5, align 4
  %6 = getelementptr i32, i32* %1, i32 0
  store i32 0, i32* %6, align 4
```

```
; Function Attrs: mustprogress nofree norecurse nosync nounwind willreturn memory(argmem: write)
define void @dae_block_44_jacobian_structA21_col(ptr writeonly %0) local_unnamed_addr #2 {
func_entry:
  %1 = getelementptr i32, ptr %0, i64 2
  store <4 x i32> <i32 7, i32 1, i32 9, i32 9>, ptr %1, align 4, !noalias !0
  %2 = getelementptr i32, ptr %0, i64 6
  store <4 x i32> <i32 0, i32 7, i32 4, i32 2>, ptr %2, align 4, !noalias !0
  %3 = getelementptr i32, ptr %0, i64 10
  store <4 x i32> <i32 5, i32 1, i32 2, i32 2>, ptr %3, align 4, !noalias !0
  %4 = getelementptr i32, ptr %0, i64 14
  store <4 x i32> <i32 3, i32 3, i32 4, i32 4>, ptr %4, align 4, !noalias !0
  %5 = getelementptr i32, ptr %0, i64 18
  store <4 x i32> <i32 0, i32 2, i32 8, i32 8>, ptr %5, align 4, !noalias !0
  %6 = getelementptr i32, ptr %0, i64 22
  store <4 x i32> <i32 1, i32 3, i32 7, i32 3>, ptr %6, align 4, !noalias !0
```

LUNDS UNIVERSITET

Modelon

Jacobian Matrix

- Good performance on the baseline
- Surprisingly bad performanced when preoptimized
- Tips to frontend developers from LLVM is to not use compound types at all when emitting LLVM-IR

Attribute	Average (ms)	Change (%)	Std. Dev. (ms)
Baseline	366.6	0.00	29.8
No alias	497.4	35.7	44.4
Optimized by clang	664.5	81.3	32.9
Single pointer	448.5	22.4	38.4

Table 7: Results of 100 test runs with 100 000 elements in the array.

LUNDS UNIVERSITET

Modelon

Inline Functions

```
define dso_local void @jmi_set_record_member_real_local(ptr noundef %0, i64 noundef %1, double noundef %2) #6 {
  %4 = getelementptr inbounds %struct._jmi_record_t, ptr %0, i64 0, i32 1
  %5 = load ptr, ptr %4, align 8, !tbaa !30
  %6 = getelementptr inbounds %struct.jmi_record_member_t, ptr %5, i64 %1
  store i32 1, ptr %6, align 8, !tbaa !35
  %7 = getelementptr inbounds %struct.jmi_record_member_t, ptr %5, i64 %1, i32 1
  store double %2, ptr %7, align 8, !tbaa !26
  ret void
}
```

More tests

Attribute	Average (ms)	Change (%)	Std. Dev. (ms)
Baseline	1131.9	0.00	1.80
Optimized by Clang	1354.7	19.7	36.3
Always inline record	1127.4	-0.40	1.71
Old compiler	3673.4	224.5	27.0
Optimized old Compiler	3287.8	190.5	26.7

Table 5: Results for 100 test runs for the other tests

- Even on big test preotpimzed code is slower
- Small improvement when forcing inlining on small methods
- New code is much better than the old code

LUNDS UNIVERSITET

Modelon

Key takeaways

- Compiler directives is probably not the way to go to get better compilation times
- Better to give LLVM simple IR and let LLVM handle optimization
- Try to emulate the IR emited by Clang frontend to get the best performance out of LLVM
- Try to avoid using calls to libraries and instead inline functions when the functions are small.

Modelon

LUNDS
UNIVERSITET

Modelon