



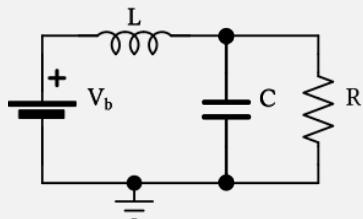
MARCO

Modelica Advanced Research CCompiler

Michele Scuttari
LLVM Developers' Meeting – October 28th 2025

Introduction & Motivation

Modelica



```
connector Pin  
  Voltage v;  
  flow Current i;  
end Pin;
```

```
model Capacitor  
  parameter Real C;  
  Voltage u;  
  Pin pin_p, pin_n;  
equation  
  0 = pin_p.i + pin_n.i;  
  u = pin_p.v - pin_n.v;  
  C * der(u) = pin_p.i;  
end Capacitor;
```

du / dt

Equations, not assignments!

```
for i in 1:100 loop  
  C * der(u[i]) = pin_p.i[i];  
end for;
```

Not obvious

3 lines of code
100 equations!



Why a new compiler?

- Array and loop declarations are typically unrolled

```
for i in 1:100 loop  
    C * der(u[i]) = pin_p.i[i];  
end for;
```



```
C * der(u[1]) = pin_p.i[1];  
C * der(u[2]) = pin_p.i[2];  
C * der(u[3]) = pin_p.i[3];  
...
```

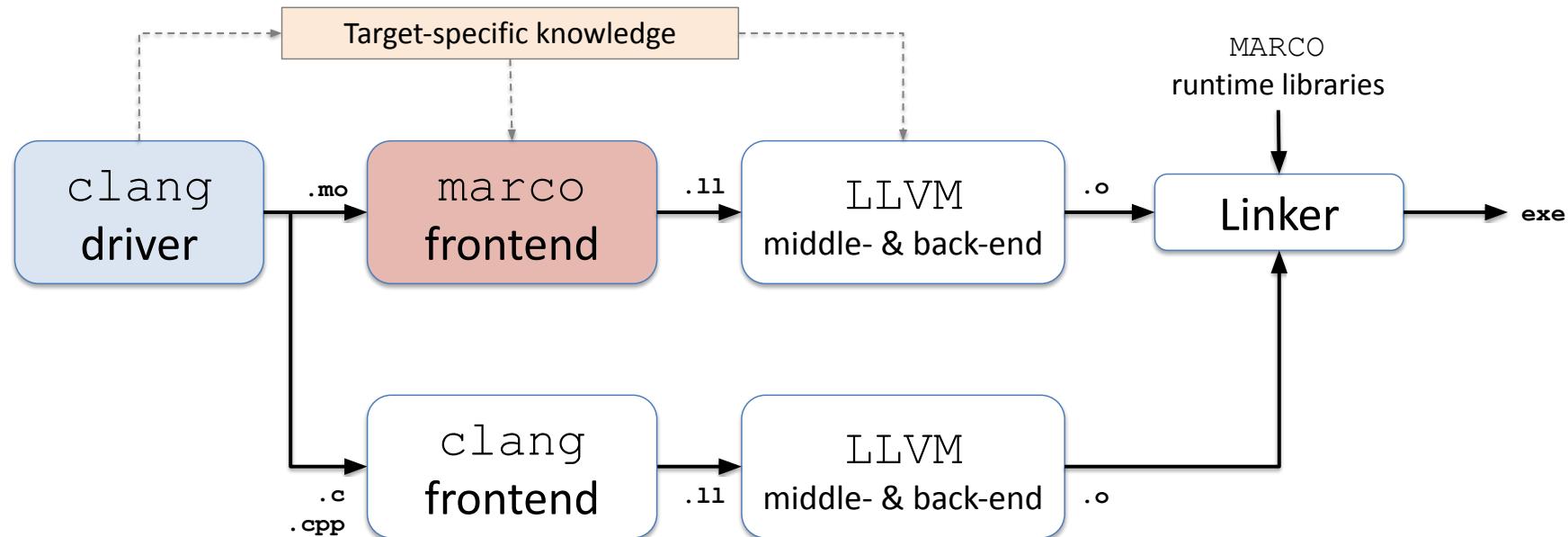
- High compilation times and binary sizes
- Development of new array-aware algorithms for the DAE domain
 - Index reduction, Matching, Variables & Equations pruning, Detection SCCs, Scheduling, etc.

Why MLIR?

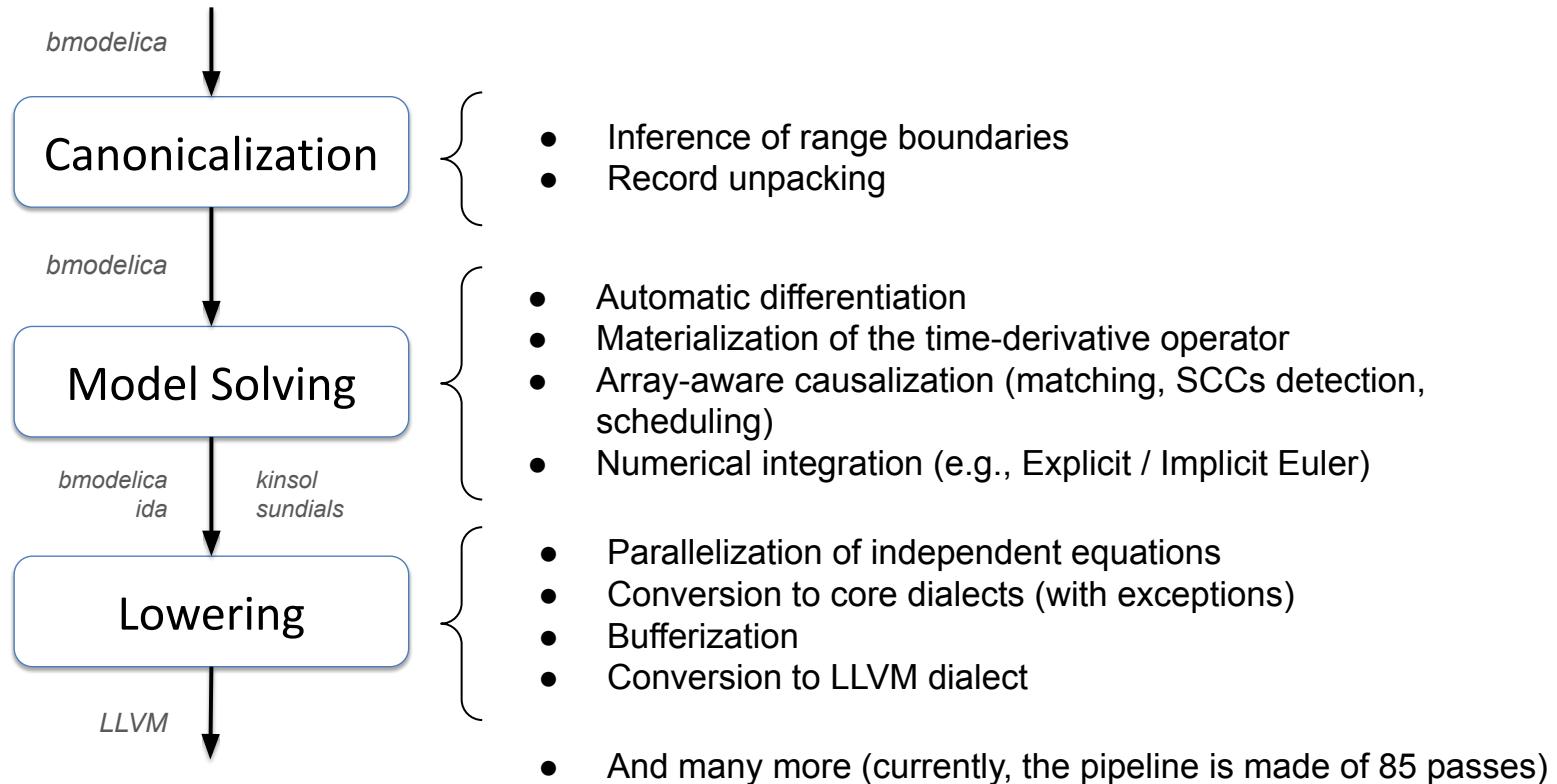
- No need to define yet another IR infrastructure
 - Example: OpenModelica is a bootstrapping compiler extending Modelica to MetaModelica
- Reuse of concepts and optimizations shared across different domains
- Tighter integration with compilers' backends
 - Example: OpenModelica & Dymola generate C code which is then given to GCC / Clang
- Easier code generation for custom architectures
- Better debuggability

Architecture & IR

Software Architecture



Compilation Pipeline



bmodelica dialect

- Purpose: **Representing Differential-Algebraic Equation (DAE) systems**
- Main concepts:
 - **Model**

```
bmodelica.model @Test {  
    // Model description  
    // ...  
}
```
 - **Variables**
 - **Equations**

bmodelica: Variables

- Design goal: **preserve interoperability with arbitrary dialects**
- What if the operations are *isolated-from-above*?
- Solution: adopt Symbol semantics
 - Dedicated operations to bridge Symbol and SSA semantics: `variable_get` & `variable_set`

```
bmodelica.variable @x : !bmodelica.variable<10xf64>

mydialect.isolated_op {
    %x = bmodelica.variable_get @x : tensor<10xf64>
    ...
    %new_x = ... : tensor<10xf64>
    bmodelica.variable_set @x, %new_x : tensor<10xf64>
}
```

bmodelica: Variables

- Lowering strategy:
 - In the case of models, they will become global symbols

```
memref.global @x : memref<1000xf64>
mydialect.isolated_op {
    %x = memref.get_global @x : memref<1000xf64>
    ...
    %new_x = ... : memref<1000xf64>
    memref.copy %new_x to %x : memref<1000xf64> to memref<1000xf64>
}
```

- In the case of functions, they will become `allocs` / `allocas` when converting to CFG

```
func.func @foo {
    %x = memref.alloc() : memref<1000xf64>
    ...
}
```

bmodelica: Equations

- Design goal: **avoid code duplication since the beginning**
- Why? Throughout compilation, the value of attribute will depend on the indices of the equations
- How? Separation between the equation structure and its actual indices

```
bmodelica.model @Test {  
    %t0 = bmodelica.equation_template inductions = [%i] {  
        %0 = ... : f64  
        ...  
        %lhs = bmodelica.equation_side %0 : tuple<f64>  
        %rhs = bmodelica.equation_side %1 : tuple<f64>  
        bmodelica.equation_sides %lhs, %rhs : tuple<f64>, tuple<f64>  
    }  
    bmodelica.dynamic {  
        bmodelica.equation_instance %t0 {  
            indices = {[1,1000]},  
            match = <@x, {[0,999]}>  
        }  
    }  
}
```

Parametric on the iteration space

We control the IR structure -> SSA semantics

Iteration space, backed by an R-Tree implementation

ida, kinsol, sundials **dialects**

- Purpose: **Integration of the IDA and KINSOL external solvers**
- Used during the model solving stage:
 - IDA: BDF numerical integration method
 - KINSOL: Newton iterations for algebraic cycles

ida, kinsol, sundials dialects: Declaring Variables

```
ida.instance @ida

func.func @init() {
    ida.init @ida

    %x = ida.add_differential_variable @ida {
        dimensions = [1000],
        differentialGetter = @x_getter,
        differentialSetter = @x_setter,
        derivativeGetter = @der_x_getter,
        derivativeSetter = @der_x_setter
    } : !ida.variable

    func.return
}
```

```
sundials.getter @x_getter(%i: index) -> f64 {
    %0 = bmodelica.variable_get @Test::@x : tensor<1000xf64>
    %1 = tensor.extract %0[%i] : tensor<1000xf64>
    ida.return %1 : f64
}

sundials.setter @x_setter(%v: f64, %i : index) {
    %0 = bmodelica.variable_get @Test::@x : tensor<1000xf64>
    %1 = tensor.insert %v into %0[%i] : tensor<1000xf64>
    bmodelica.variable_set @x, %1 : tensor<1000xf64>
    ida.return
}
```

ida, kinsol, sundials dialects: Declaring Equations

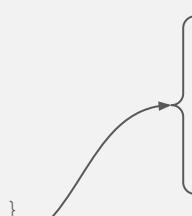
- Need to compute residual values and Jacobian matrix of the system
- Each equation is partially differentiated (in-house forward automatic differentiation)
 - Interface-based infrastructure

```
ida.instance @ida

func.func @init() {
    %x = ... : !ida.variable

    %eq0 = ida.add_equation @ida indices = {[1, 1000]}
    ida.set_residual_function @ida, %eq0, @eq0_res
    ida.add_pder_function @ida, %eq0, %x, @eq0_pder_x

func.return
}
```



```
ida.residual_function @eq0_res(%time: f64, %e_i: index, %v_i: index) -> f64 {
    %0 = ... : f64
    ida.return %0 : f64
}
```



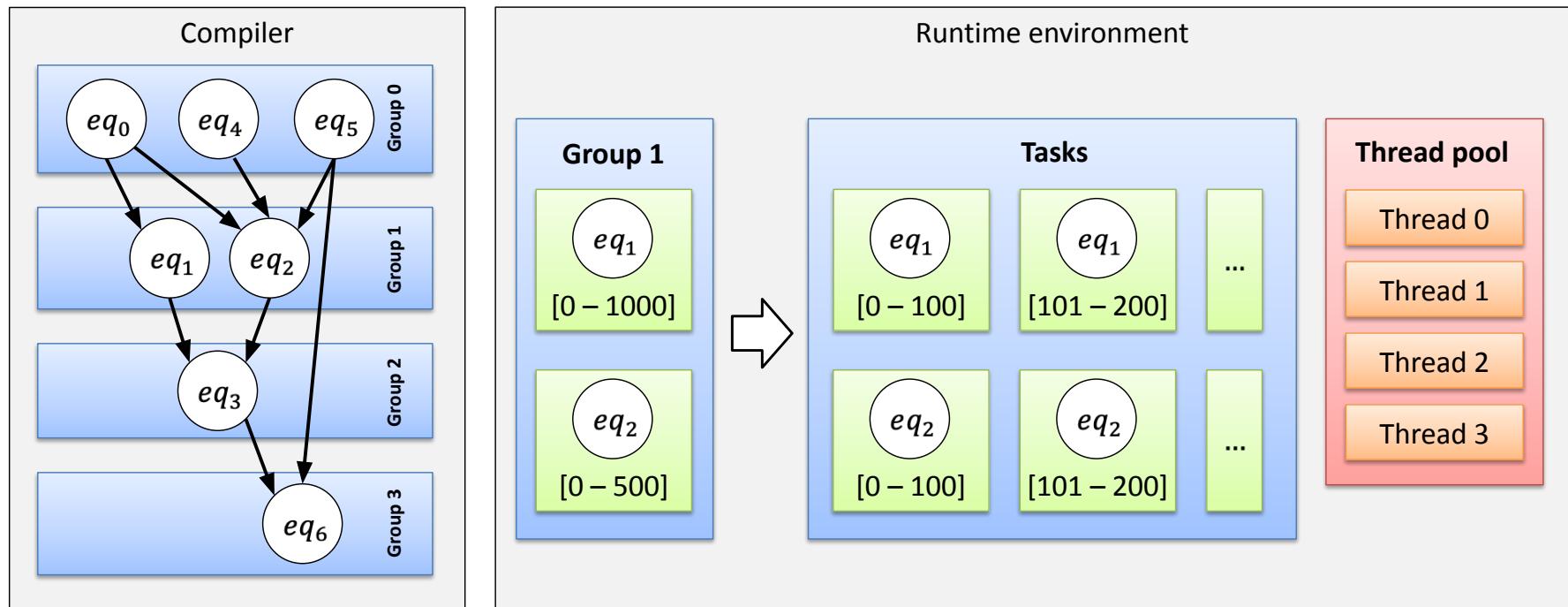
```
ida.pder_function @eq0_pder_x(%time: f64, %e_i: index, %v_i: index, %alpha: f64) -> f64 {
    %0 = ... : f64
    ida.return %0 : f64
}
```

runtime dialect

- Purpose: **Interfacing with the runtime environment**
- Support functions provided by the runtime environment (e.g., sin, cos)
 - This includes stateful workflows, such as the configuration of dynamic schedulers
- Callback functions provided by the compiled model
 - Some functions are always generated (e.g., setTime, getVariableValue)
 - Some other functions depend on the chosen numerical integration algorithm

Optimization Example

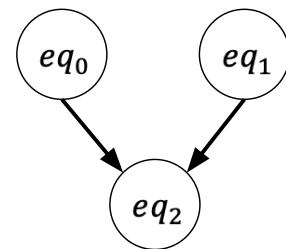
Parallelization of Equations: Idea



Parallelization of Equations: Starting IR

Opaque computational blocks indicate whether they may grouped together with others.

```
bmodelica.model {  
    bmodelica.dynamic {  
        bmodelica.schedule_block written = [@x] read = [] {  
            bmodelica.equation_call @eq0  
        } {parallelizable = true}  
  
        bmodelica.schedule_block written = [@y] read = [] {  
            bmodelica.equation_call @eq1  
        } {parallelizable = true}  
  
        bmodelica.schedule_block written = [@z] read = [@x, @y] {  
            bmodelica.equation_call @eq2  
        } {parallelizable = true}  
    }  
}
```

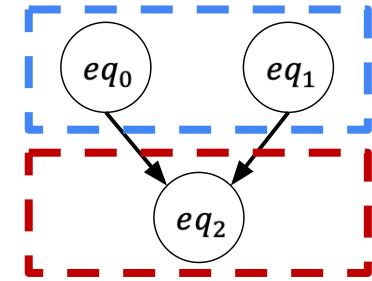


Can be merged in a group with adjacent parallelizable blocks

Parallelization of Equations: Dependency Analysis

Computationally independent and parallelizable opaque blocks are grouped together.

```
bmodelica.model {
    bmodelica.dynamic {
        bmodelica.parallel_schedule_blocks {
            bmodelica.schedule_block written = [@x] read = [] {
                bmodelica.equation_call @eq0
            } {parallelizable = true}
            bmodelica.schedule_block written = [@y] read = [] {
                bmodelica.equation_call @eq1
            } {parallelizable = true}
        }
        bmodelica.parallel_schedule_blocks {
            bmodelica.schedule_block written = [@z] read = [@x, @y] {
                bmodelica.equation_call @eq2
            } {parallelizable = true}
        }
    }
}
```



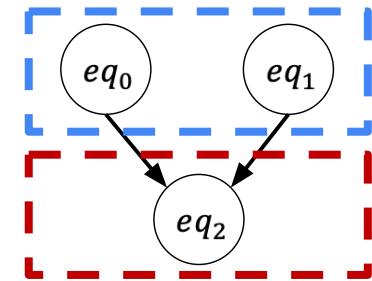
Parallelization of Equations: Runtime Scheduling

The runtime environment is informed about the computation blocks.

```
runtime.scheduler @scheduler0
runtime.scheduler @scheduler1

bmodelica.model {
    bmodelica.dynamic {
        runtime.scheduler_run @scheduler0
        runtime.scheduler_run @scheduler1
    }
}

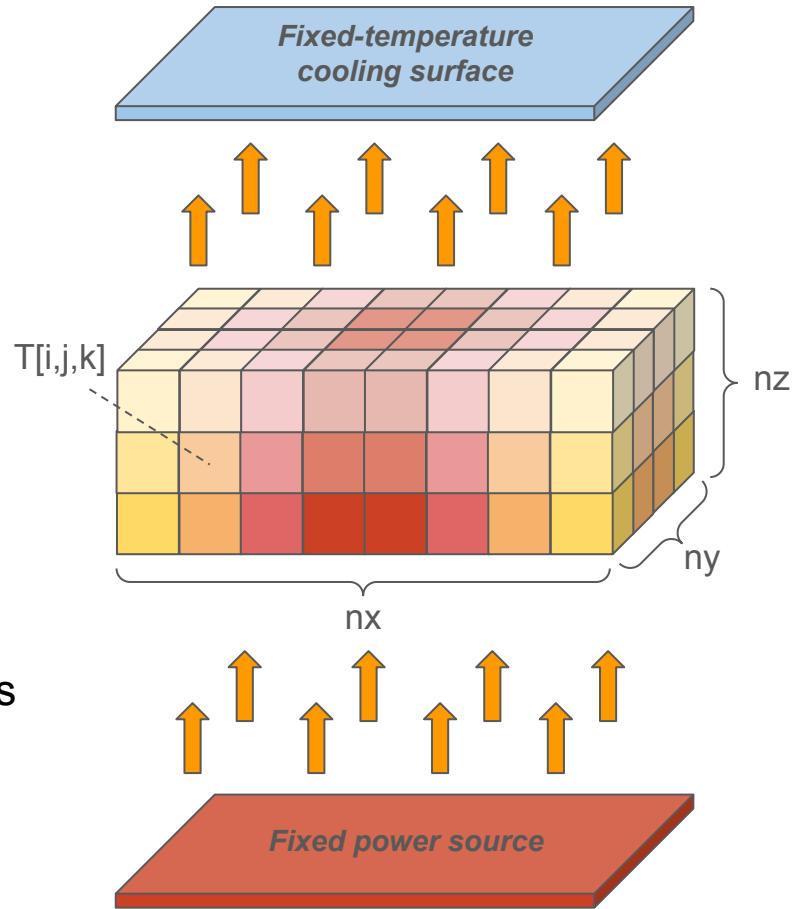
func.func @dynamic_init() {
    runtime.scheduler_add_equation @scheduler0, @eq0
    runtime.scheduler_add_equation @scheduler0, @eq1
    runtime.scheduler_add_equation @scheduler1, @eq2
}
```



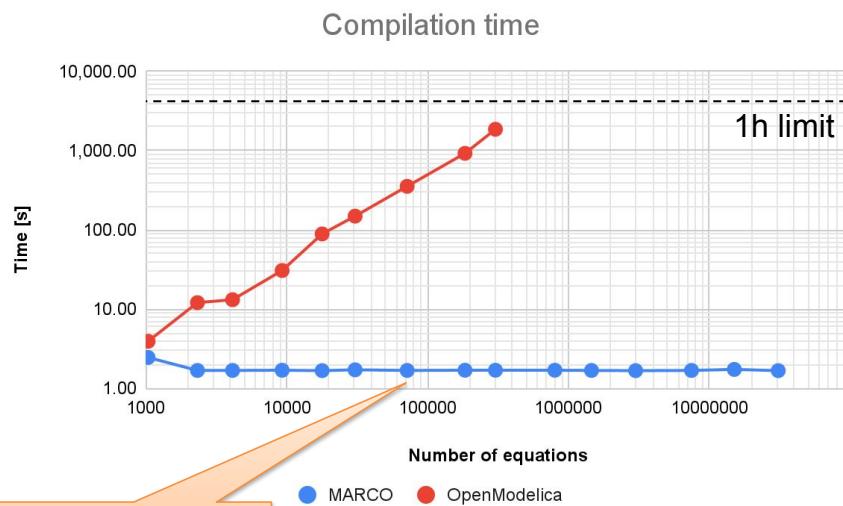
Application Example

Thermal Model of a Silicon Chip

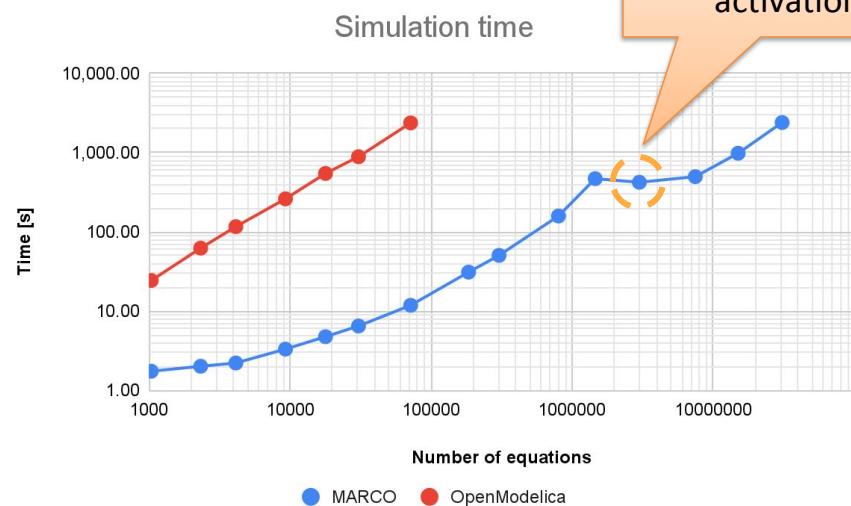
- 3D silicon chip thermal model
 - Discretized in volumes
 - Parametrized number of volumes
- Explicit Euler method
 - Stop time: 0.4s
 - Time step: 1.5 μ s (~ 270 k steps)
- Interest in both compilation and simulation times
 - MARCO
 - OpenModelica compiler



Performance Evaluation



Array-aware
DAE algorithms



Automatic MT
activation

Conclusions & Future Works

Current Status & Future Directions

Language coverage

- ✓ Ordinary Differential Equations (ODE)
- ✓ Differential-Algebraic Equations (DAE)
- ⚠ High-Index DAE systems
- ⚠ Hybrid systems (time & state events)
- ⚠ Model instantiation & flattening

Toolchain features

- ⚠ Functional Mockup Interface (FMI)
- ⚠ Multiple architectures

Other

- ⚠ Benchmark suite
- ⚠ Other modeling languages

Difficulties & Possible Improvements

- **Integration of out-of-tree projects with the clang driver**
 - Not easy to understand how to plug into the infrastructure.
 - A considerable amount of headers are private, and mandate copy-pasting into out-of-tree frontends.
 - Looking inside the Flang project was very helpful!
 - Still not entirely clear how to handle 1:N source modifications.
- **MLIR's Symbol Tables**
 - Constructed and updated manually.
 - Optionally cached through `mlir::SymbolTableCollection`.
 - However, caching is rarely used. A few passes (Bufferization and ToLLVM conversion) were scaling quadratically. They have been fixed, but...
 - *Should the Symbol Table trait automatically attach a property, to be automatically updated by the core infrastructure?*

Difficulties & Possible Improvements

- **Hardware cost model**

- For now, parallelization of equations does not take into consideration the runtime computational costs
- A cost model is needed
- Is there already something in LLVM? Can we leverage it?
- Should an interface / analysis be developed within upstream MLIR?
- What should be the level of detail?
- How do we allow for arbitrary user-defined architectures?

- **Early exits**

- (Base) Modelica allows for early exits within its functions
- The `bmodelica` dialect has its own “`scf.if`”, “`scf.while`”, “`scf.for`”

Thanks for your attention!



<https://github.com/marco-compiler/marco>
michele.scuttari@polimi.it