Modeling Stencils in a Multi-Level Intermediate Representation

Jean-Michel Gorius

Univ Rennes

Supervisor: Tobias Grosser, SPCL, ETH Zurich

Internship duration: 15/05–26/07/2019

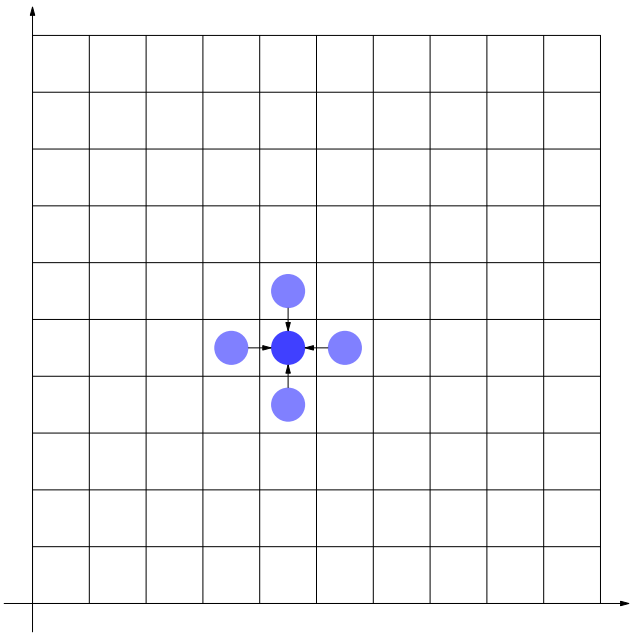
September 3, 2019

About this work

Stencil computations

Stencil kernel

- Iterative computation kernel
- Compute value of cell according to neighbors



Example: Jacobi stencil

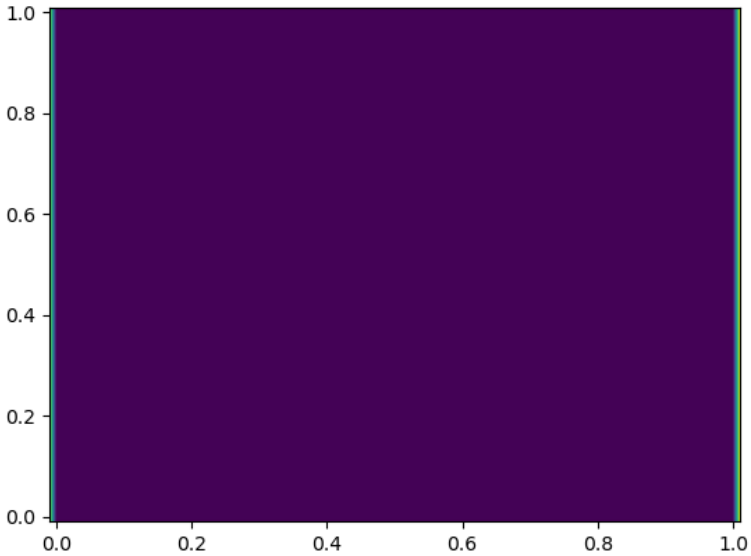
The heat equation

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = 0$$

Boundary conditions

$$\mathscr{D} = [0,1]^2$$

$$\mathscr{D}(i,j) = \begin{cases} 1 & \text{if } i,j \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$



Example: Jacobi stencil

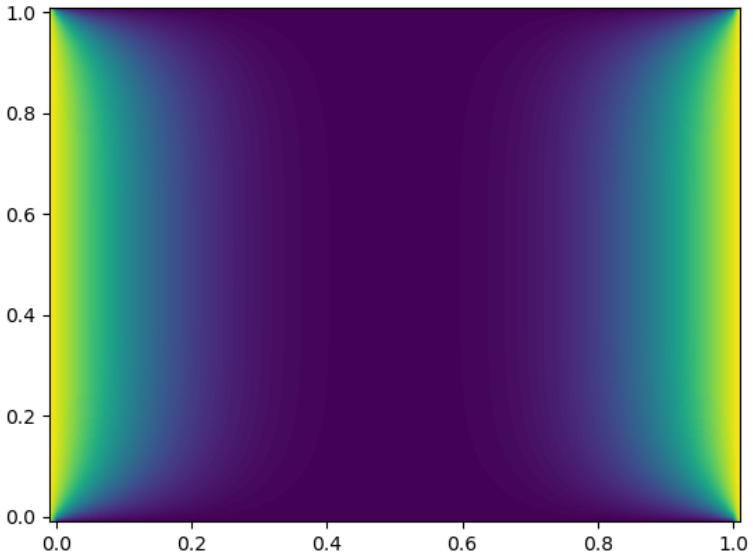
The heat equation

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = 0$$

Boundary conditions

$$\mathscr{D} = [0,1]^2$$

$$\mathscr{D}(i,j) = \begin{cases} 1 & \text{if } i,j \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$



Example: Jacobi stencil

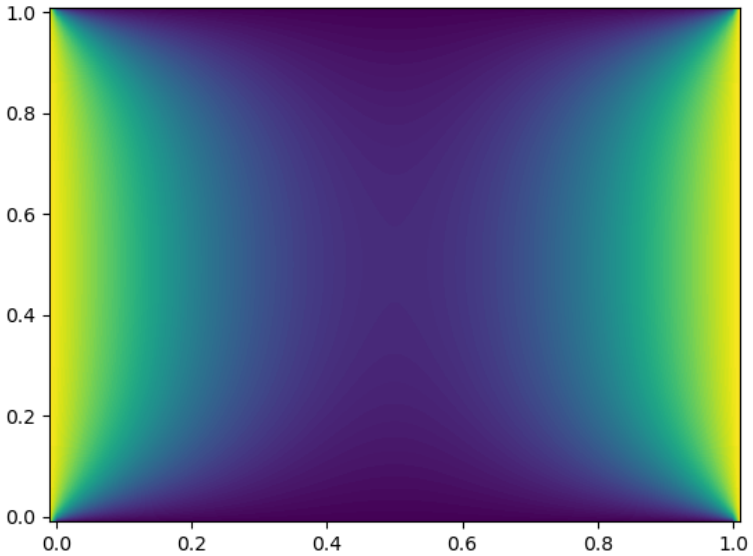
The heat equation

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = 0$$

Boundary conditions

$$\mathscr{D} = [0,1]^2$$

$$\mathscr{D}(i,j) = \begin{cases} 1 & \text{if } i,j \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$



Example: Jacobi stencil

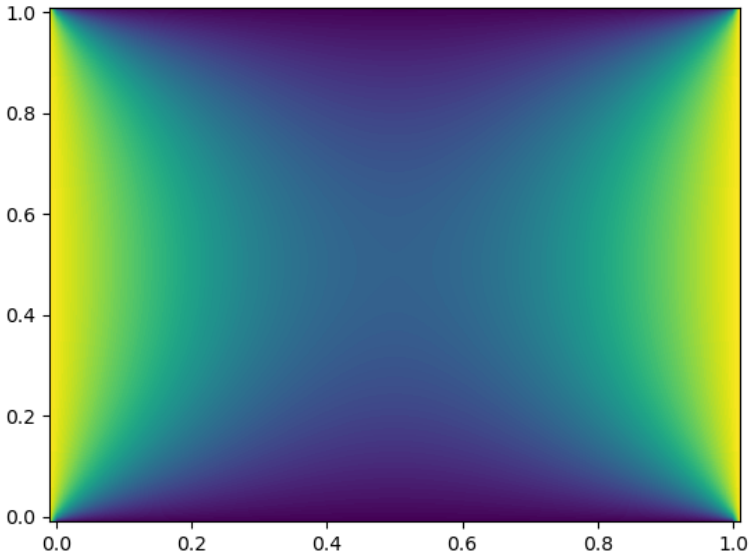
The heat equation

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = 0$$

Boundary conditions

$$\mathscr{D} = [0,1]^2$$

$$\mathscr{D}(i,j) = \begin{cases} 1 & \text{if } i,j \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$



Example: Jacobi stencil

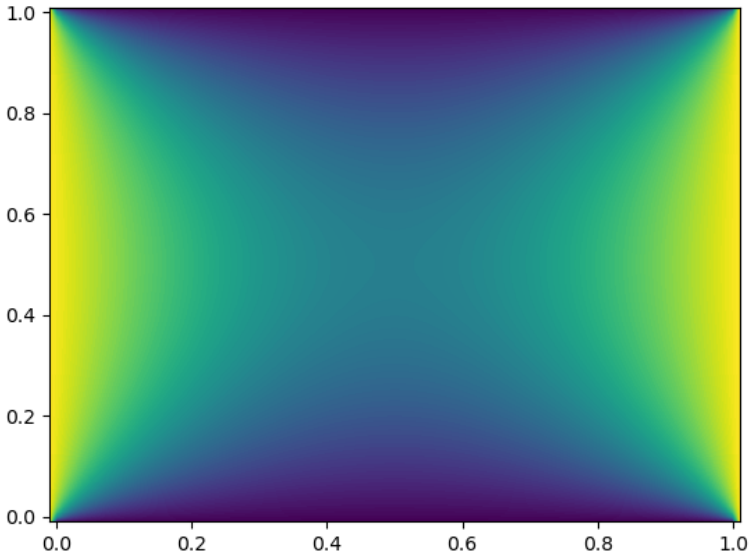
The heat equation

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = 0$$

Boundary conditions

$$\mathscr{D} = [0,1]^2$$

$$\mathscr{D}(i,j) = \begin{cases} 1 & \text{if } i,j \in \{0,1\}, \\ 0 & \text{otherwise.} \end{cases}$$



Several stencil DSLs

- gtclang

Several stencil DSLs

- gtclang
- Stella

Several stencil DSLs

- gtclang
- Stella
- Gridtools

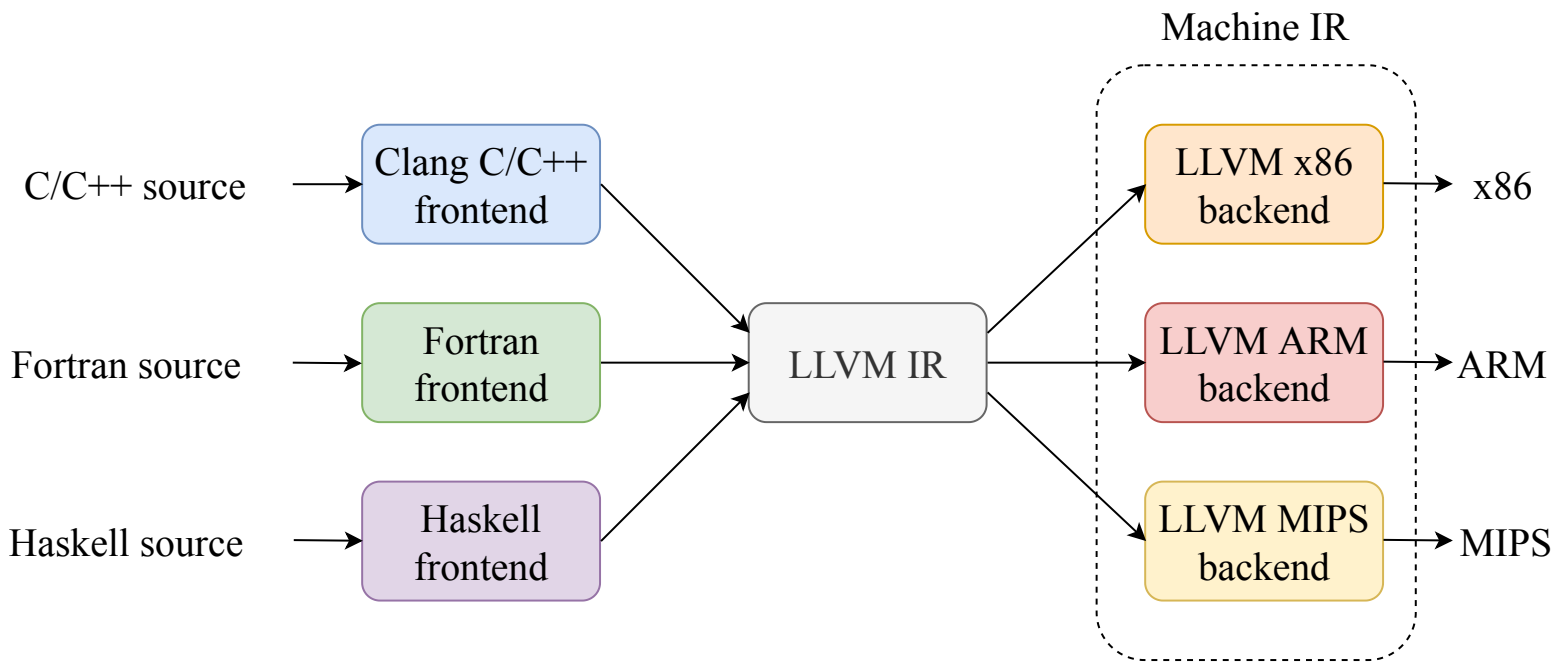
Countless stencil DSLs

- gtclang
- Stella
- Gridtools
- Snowflake
- Simflowny
- Simflowny 2
- PATUS
- Lift
- Exastencil
- Halide
- OpenSBLI
- PADS
- Stincilla
- Multi-Stencil
- SDSL
- Firedrake
- DACE
- Nebo-Wasatch
- Liszt
- Physis
- MSL
- AMRStencil
- NUMA
- ...

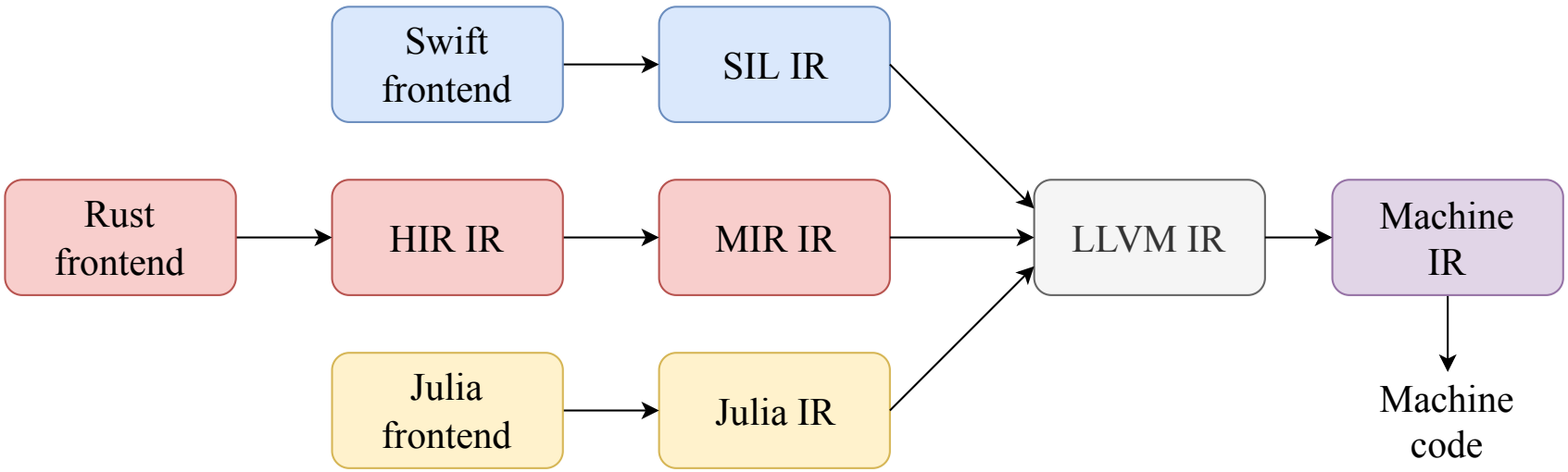
The LLVM project

The LLVM infrastructure

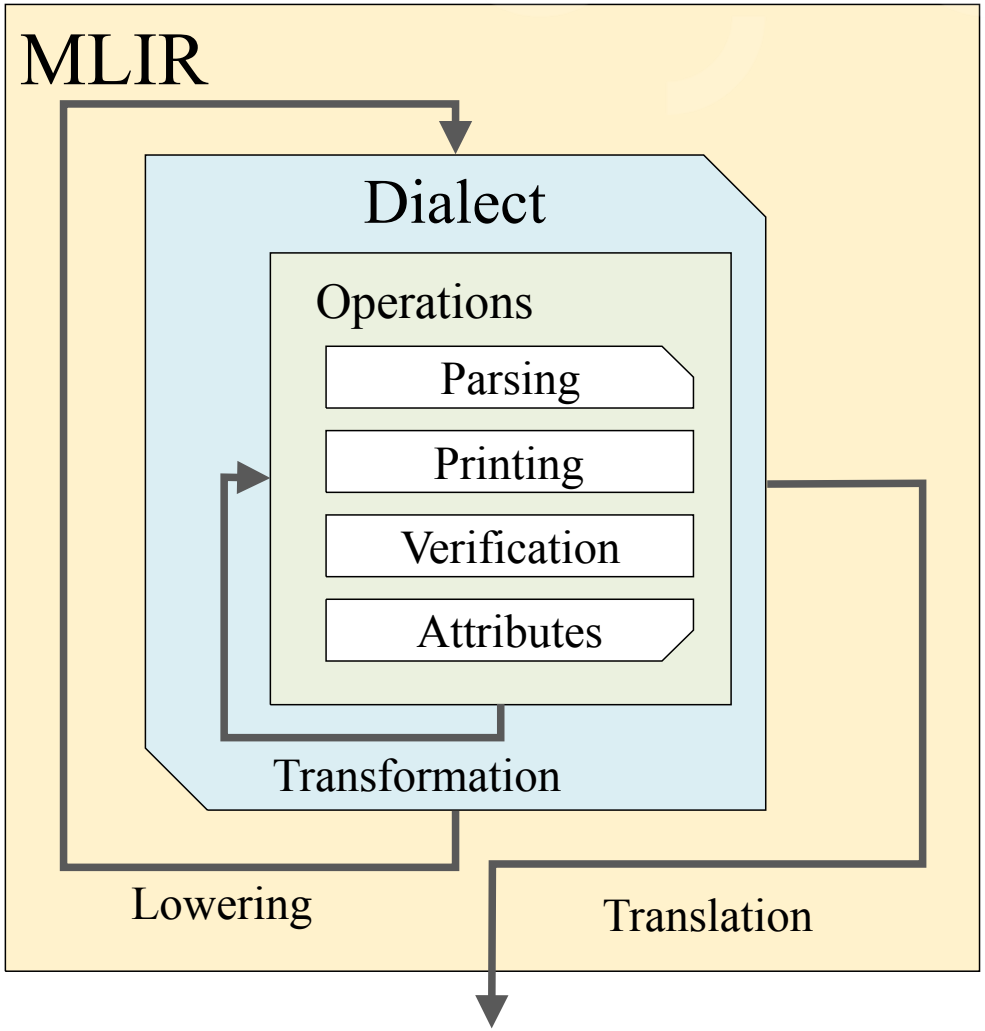
- Complete compiler infrastructure
- Common intermediate representation (LLVM IR)
- Powerful optimizer



Towards high-level intermediate representations



The MLIR infrastructure



Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`
- Vertical regions

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`
- Vertical regions
- Execution context access

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`
- Vertical regions
- Execution context access
- Field operations

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`
- Vertical regions
- Execution context access
- Field operations
- Function calls

Key concepts

Basic building blocks

- Data types: `i1/16/32/64` and `f16/32/64`
- Basic operations: arithmetic (`add`, `sub`, etc.), comparison (`le`, `gt`, etc.) and boolean (`and`, `or`, etc.)
- Control flow: `if`
- Vertical regions
- Execution context access
- Field operations
- Function calls
- Global variables

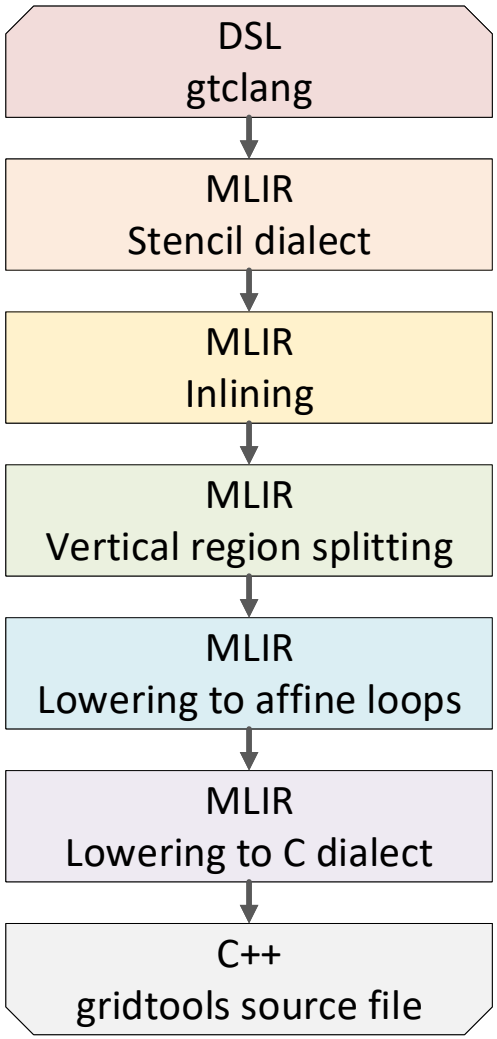
Quick syntax overview (1/2)

```
func @jacobi(%G: !stencil<"field:f64">) -> f64
  attributes {stencil.function} {
    %off0 = stencil.constant_offset 1 0 0
    // ...
    %0 = stencil.read(%G, %off0) : f64
    // ...
    %cst = stencil.constant 0.25 : f64
    %4 = stencil.add(%0, %1) : f64
    %5 = stencil.add(%4, %2) : f64
    %6 = stencil.add(%5, %3) : f64
    %res = stencil.mul(%6, %cst) : f64
    return %res : f64
}
```

Quick syntax overview (2/2)

```
func @jacobi_stencil(%in: !stencil<"field:f64">,
                     %out: !stencil<"field:f64">) {
    %kstart = stencil.context "kstart" : i64
    %kend = stencil.context "kend" : i64
    stencil.vertical_region(%kstart, %kend) {
        %val = stencil.call @jacobi(%G) :
            (!stencil<"field:f64">) -> f64
        stencil.write(%out, %val) : f64
    }
    return
}
```

From a stencil DSL to MLIR



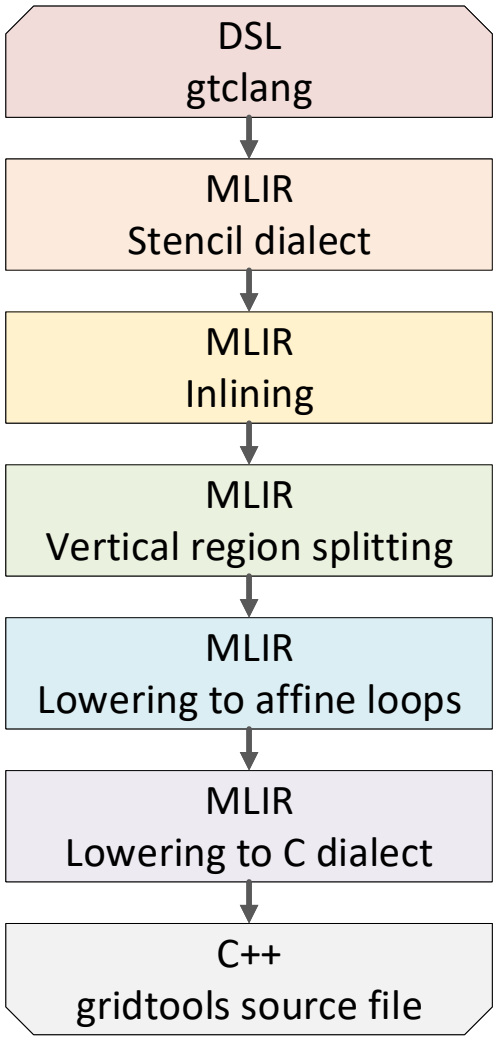
Source DSL

Developed and maintained by MeteoSwiss

`gtclang`

MeteoSwiss frontend, adapted to MLIR

Transformation and lowering



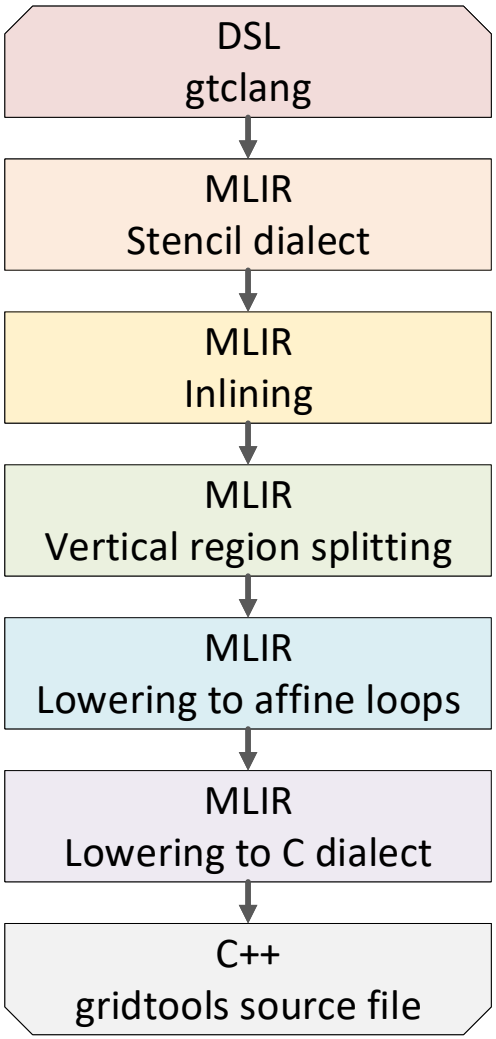
Transformation passes

- Function call inlining
- Avoid data races by splitting vertical regions

Lowering passes

- Lower vertical regions to loops
- Lower to a basic C dialect

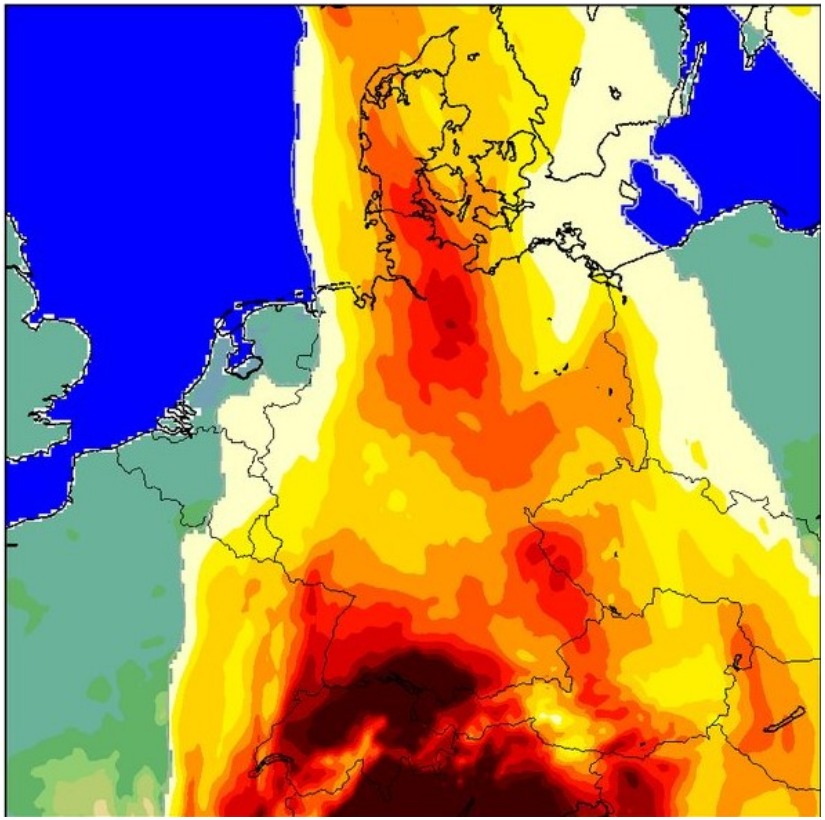
Translating to C



Generating a compilable source file

- Serialize the C dialect to C code
- Integrate into the `gridtools` framework

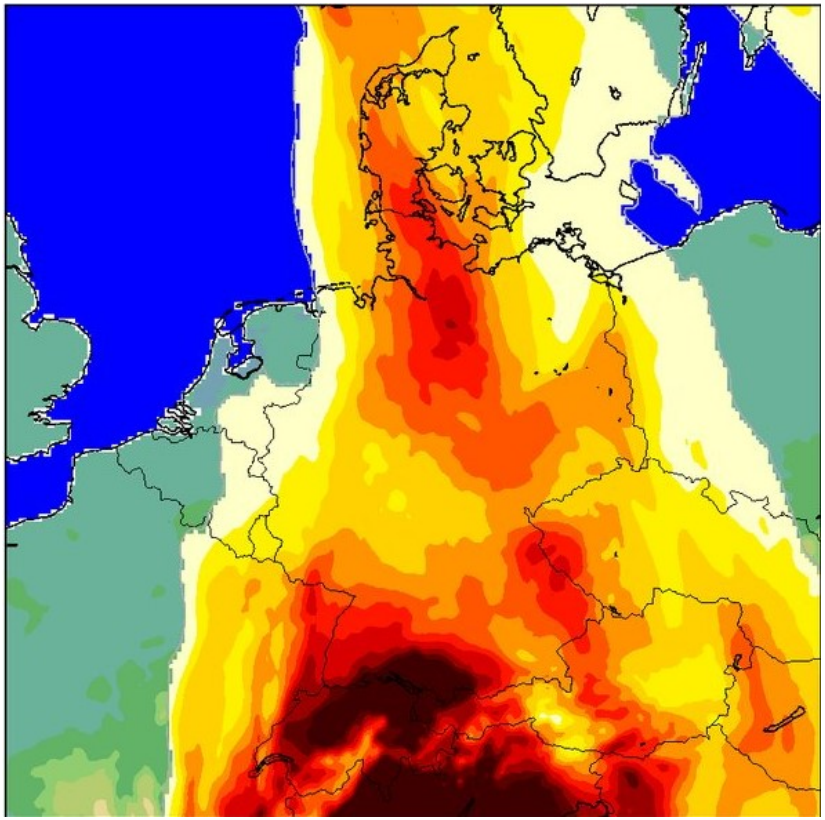
The COSMO dynamic core



The COSMO model

- European model
- Weather forecast
- Environmental simulation

The COSMO dynamic core



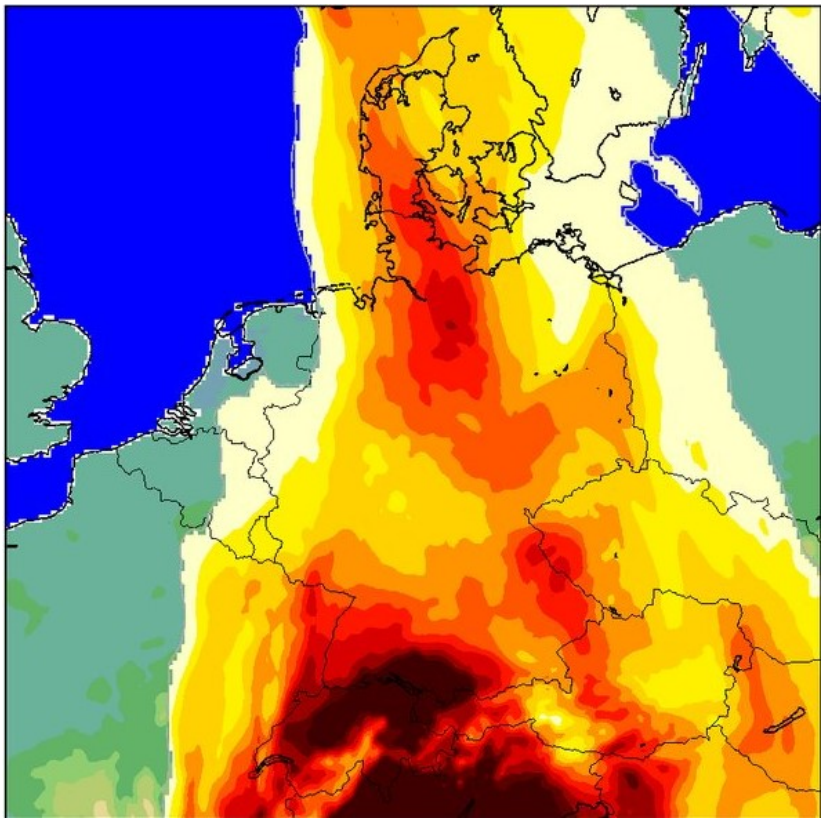
The COSMO model

- European model
- Weather forecast
- Environmental simulation

Feature-completeness

90% feature coverage

The COSMO dynamic core



The COSMO model

- European model
- Weather forecast
- Environmental simulation

Feature-completeness

90% feature coverage

Model validation

Two bugs uncovered

Conclusion

Achievements

- End-to-end compiler infrastructure for a stencil DSL
- Reusable and entirely tested infrastructure
- Near feature-complete implementation

Future work

- Reach feature-completeness
- Add optimization passes
- Generate GPU code
- Support additional frontends
- Make it the standard stencil dialect for MLIR