JEAN-MICHEL GORIUS, TOBIAS WICKY, TOBIAS GROSSER, AND TOBIAS GYSI

A Compiler Intermediate Representation for Stencils

"Climate change is now affecting every country on every continent. It is disrupting national economies and affecting lives, costing people, communities and countries dearly today and even more tomorrow. Weather patterns are changing, sea levels are rising, weather events are becoming more extreme and greenhouse gas emissions are now at their highest levels in history." - **United Nations, Sustainable Development Goals**

Open Climate Compiler Initiative

MeteoSwiss

ETH zürich

VULCAN

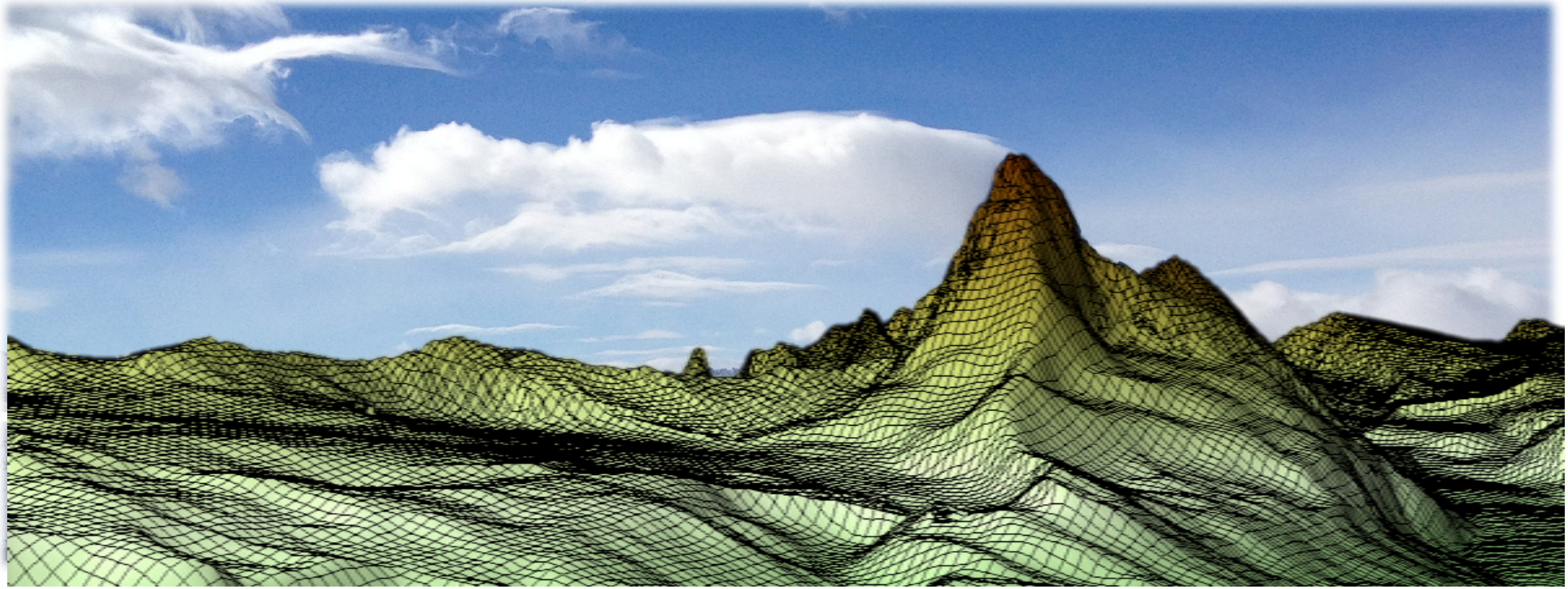
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

COSMO Atmospheric Model

- Regional atmospheric model used by 7 national weather services
- Implements many different stencil programs

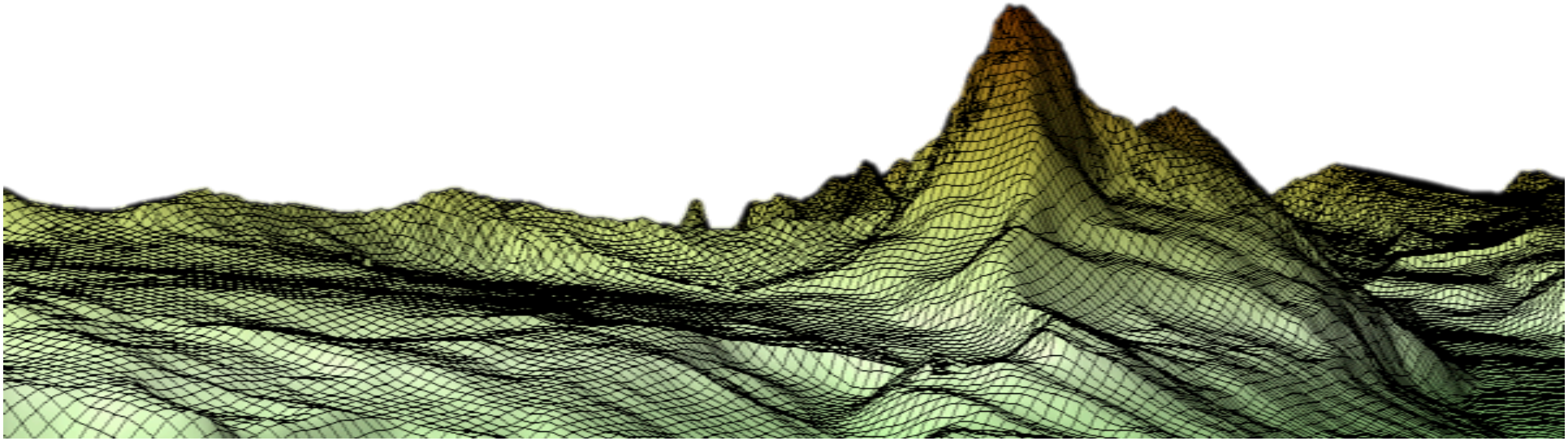
Resolution (35m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



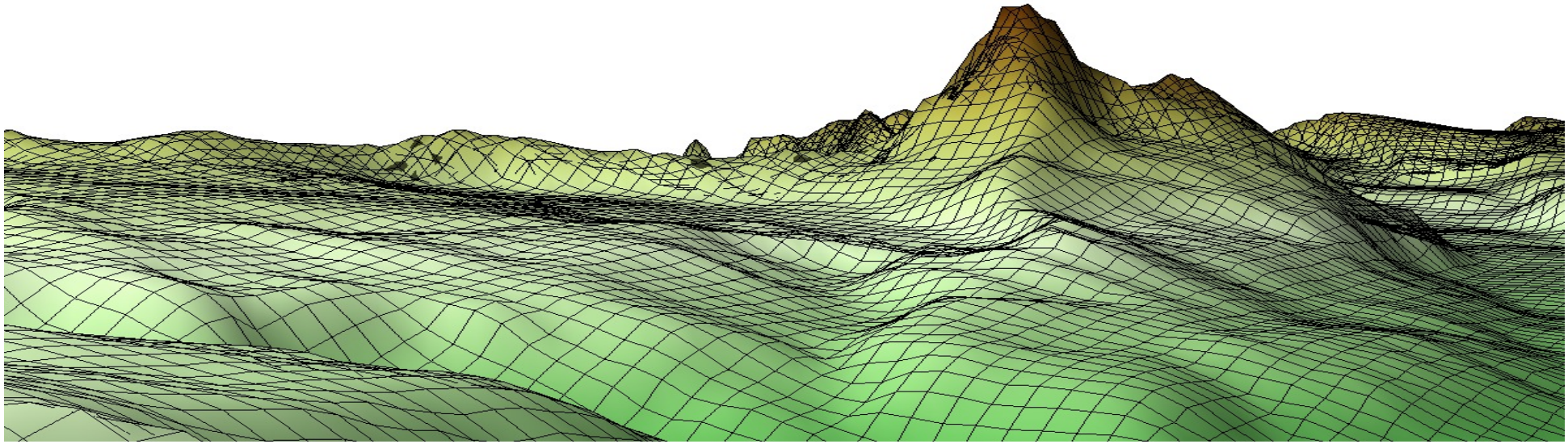
Resolution (35m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



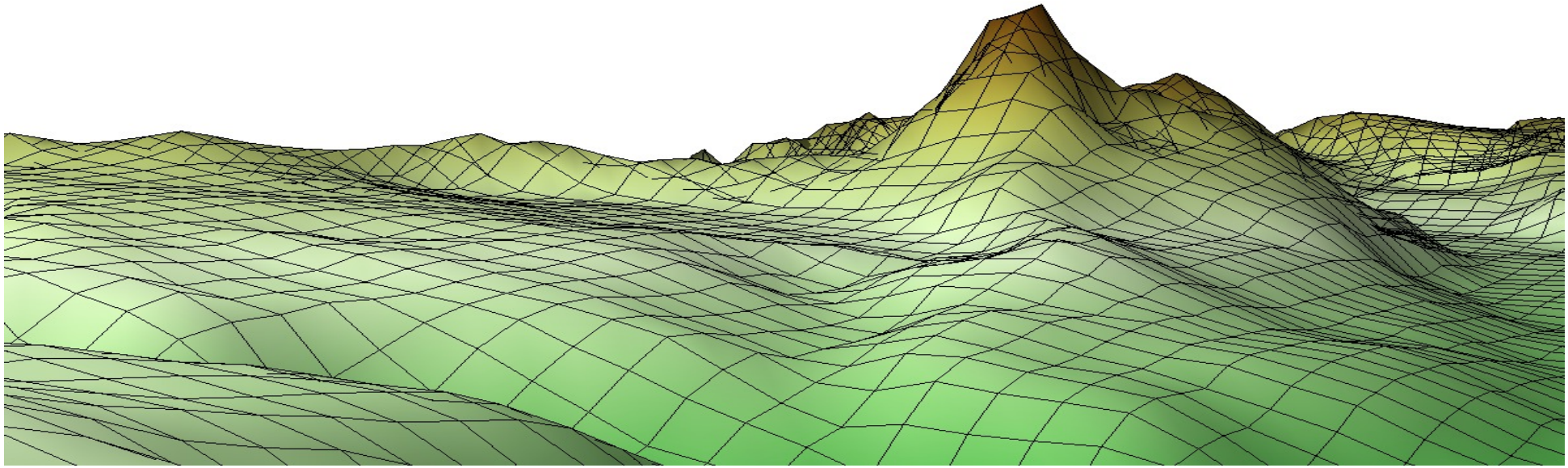
Resolution (70m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



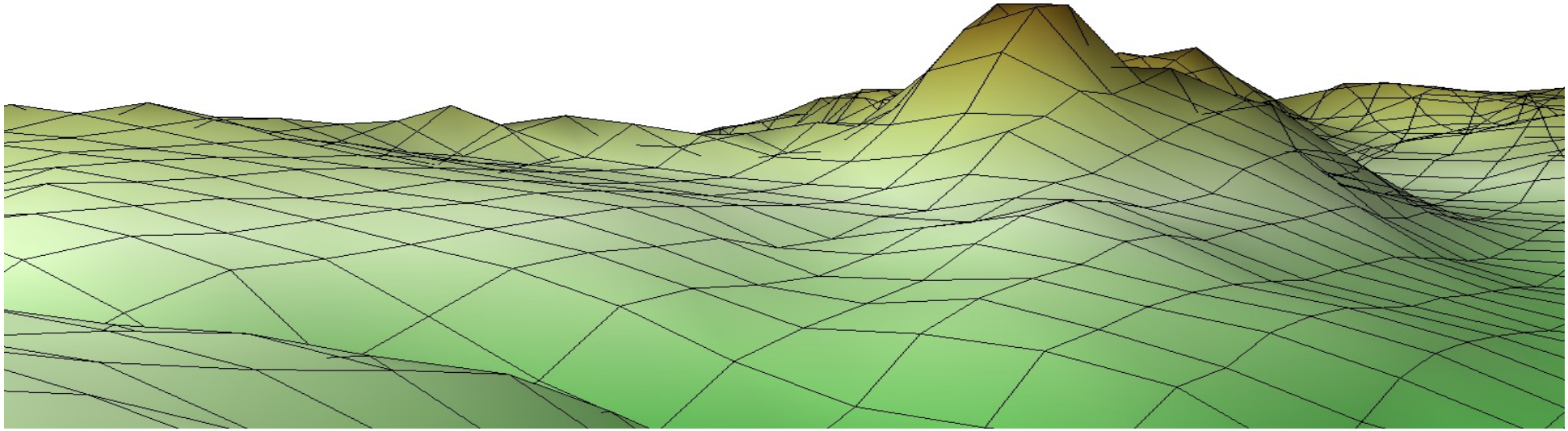
Resolution (140m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



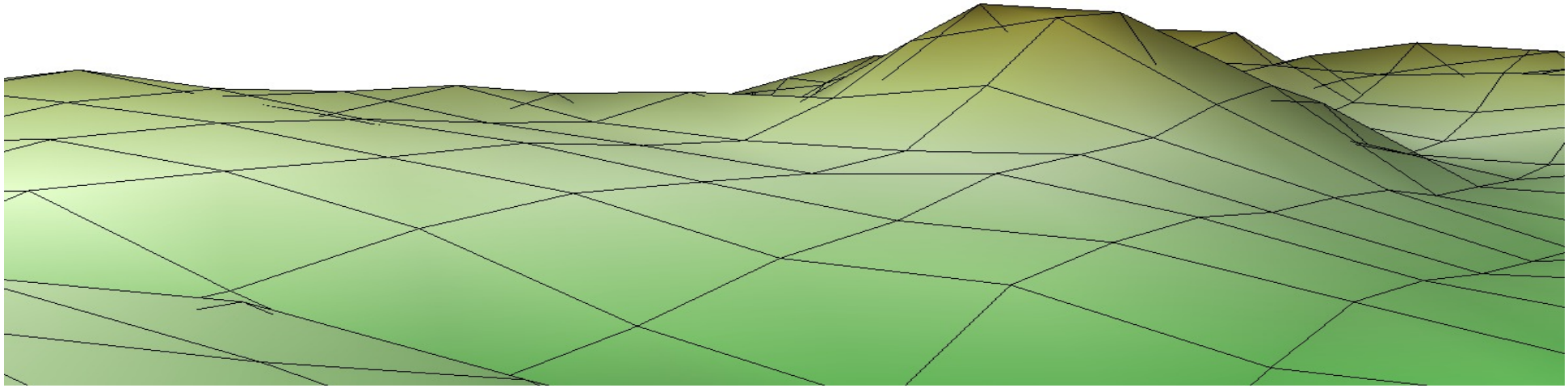
Resolution (280m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



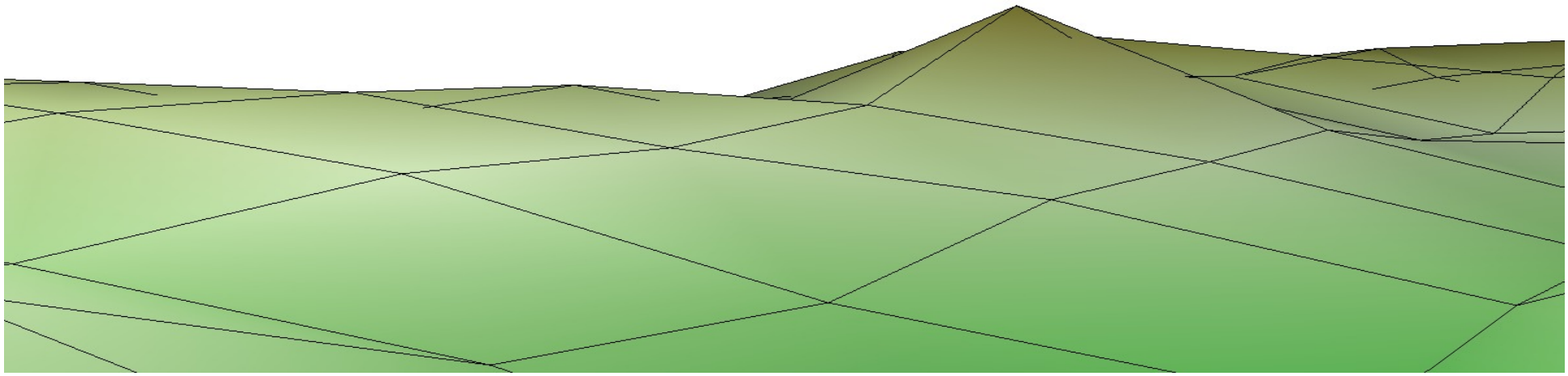
Resolution (560m)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



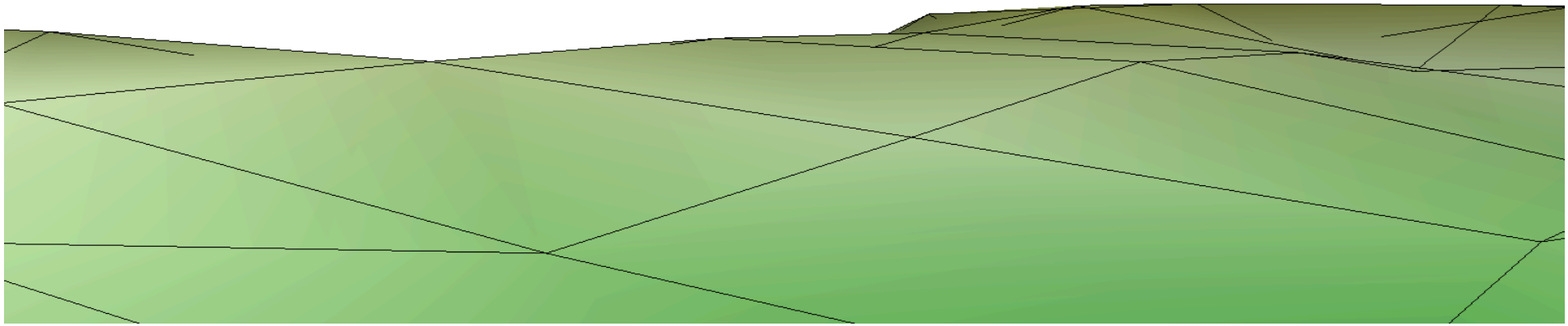
Resolution (1.1km – Weather Forecast Today)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



Resolution (2.2km – Weather Forecast 2015)

What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?



Achieving High-Performance, Portability, and Productivity

COSMO

Stella/GridTools

1st GPU model running in production

Dawn (GTClang)

1998 — 2010 — **2015** — 2017

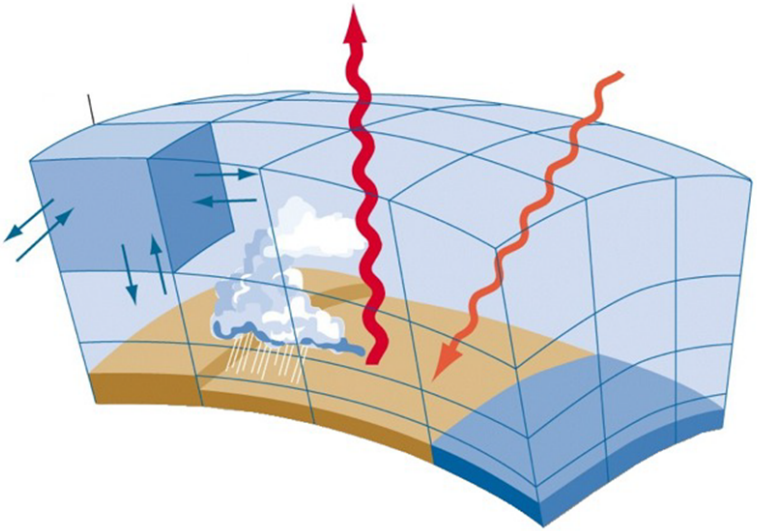
- Fortran code
- optimized for vector machines

- DSL embedded in C++
- GPU and CPU support
- performance & portability

- domain-specific compiler
- front end language agnostic
- powerful analysis and optimization passes
- productivity

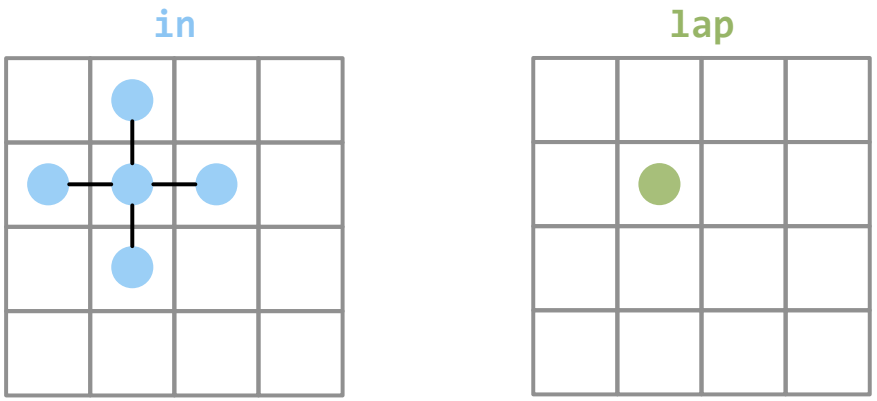
Domain-Science vs Computer-Science

- solve PDE
- finite differences
- structured grid

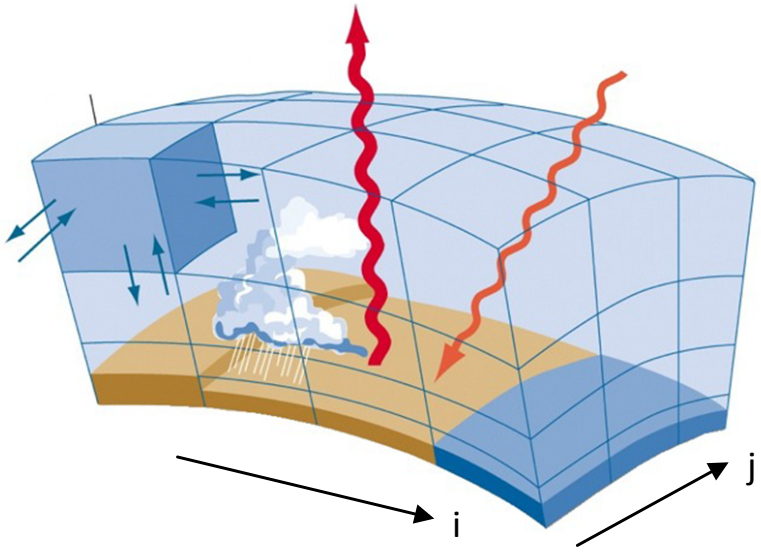


- element-wise computation
- fixed neighborhood

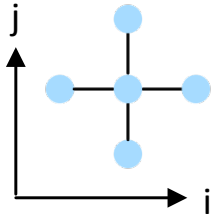
```
lap(i,j) = -4.0 * in(i,j) +
  in(i-1,j) + in(i+1,j) +
  in(i,j-1) + in(i,j+1)
```



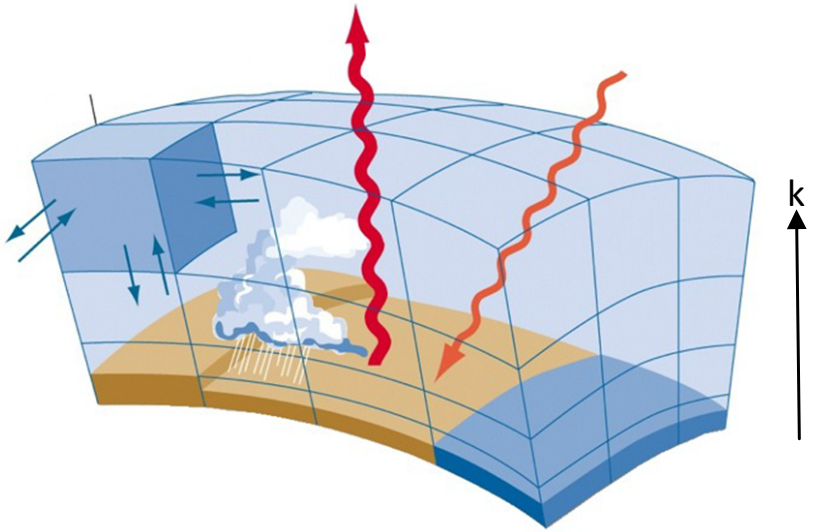
Algorithmic Motifs – Finite Differences



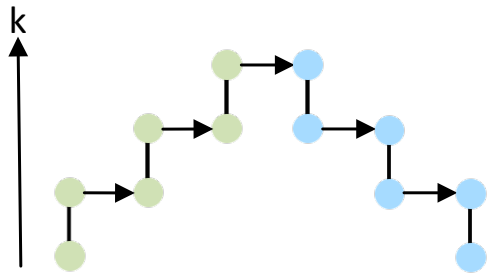
- stencils (no loop carried dependencies)
- mostly horizontal dependencies



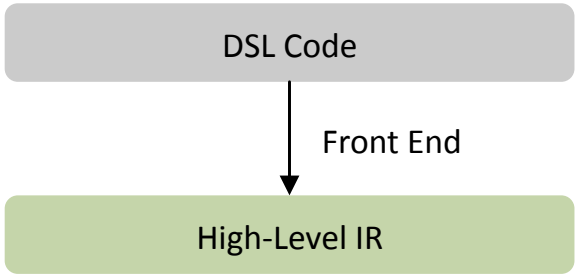
Algorithmic Motifs – Tridiagonal Systems



- vertical dependencies
- loop carried dependencies



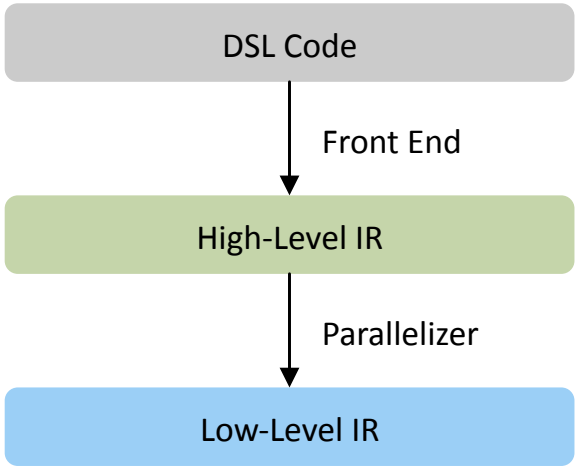
Architecture of the Dawn Compiler



Front End

```
stencil average {
  storage in, out;
  Do {
    vertical_region(kstart, kend) {
      out[i,j,k] = 0.5 * (in[i-1,j,k] + in[i+1,j,k])
    }
  }
};
```

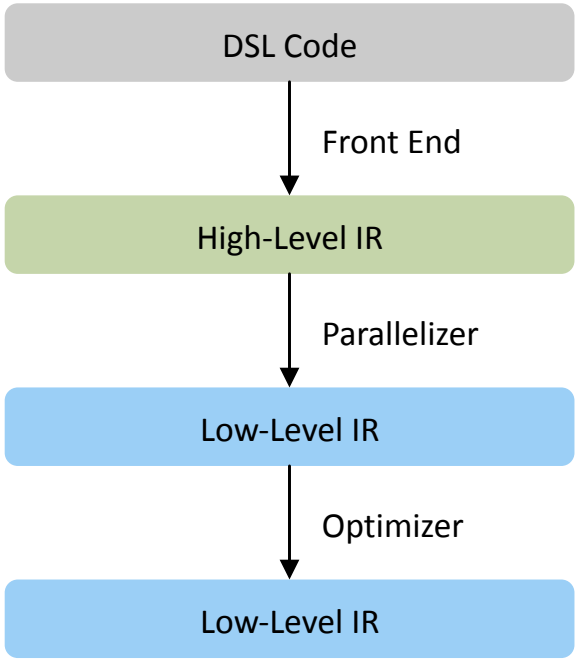
Architecture of the Dawn Compiler



Parallelizer

- add synchronization
- solve data races
- safety checks

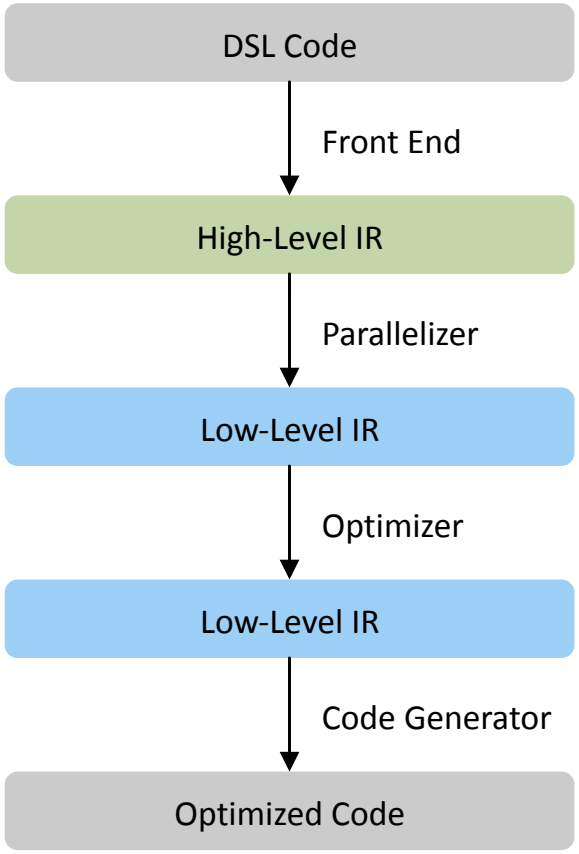
Architecture of the Dawn Compiler



Optimizer

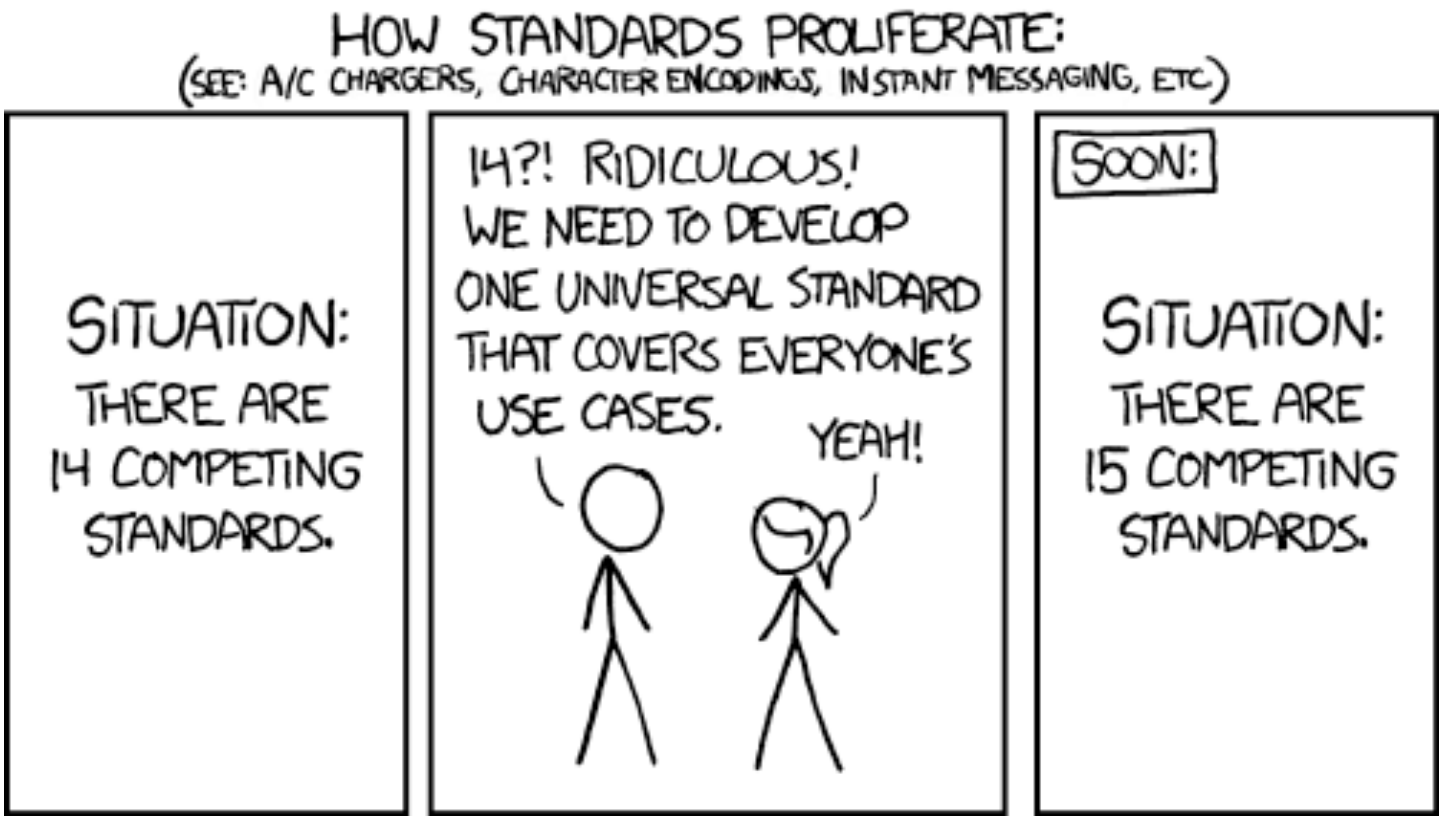
- data-locality
- caching
- memory footprint

Architecture of the Dawn Compiler



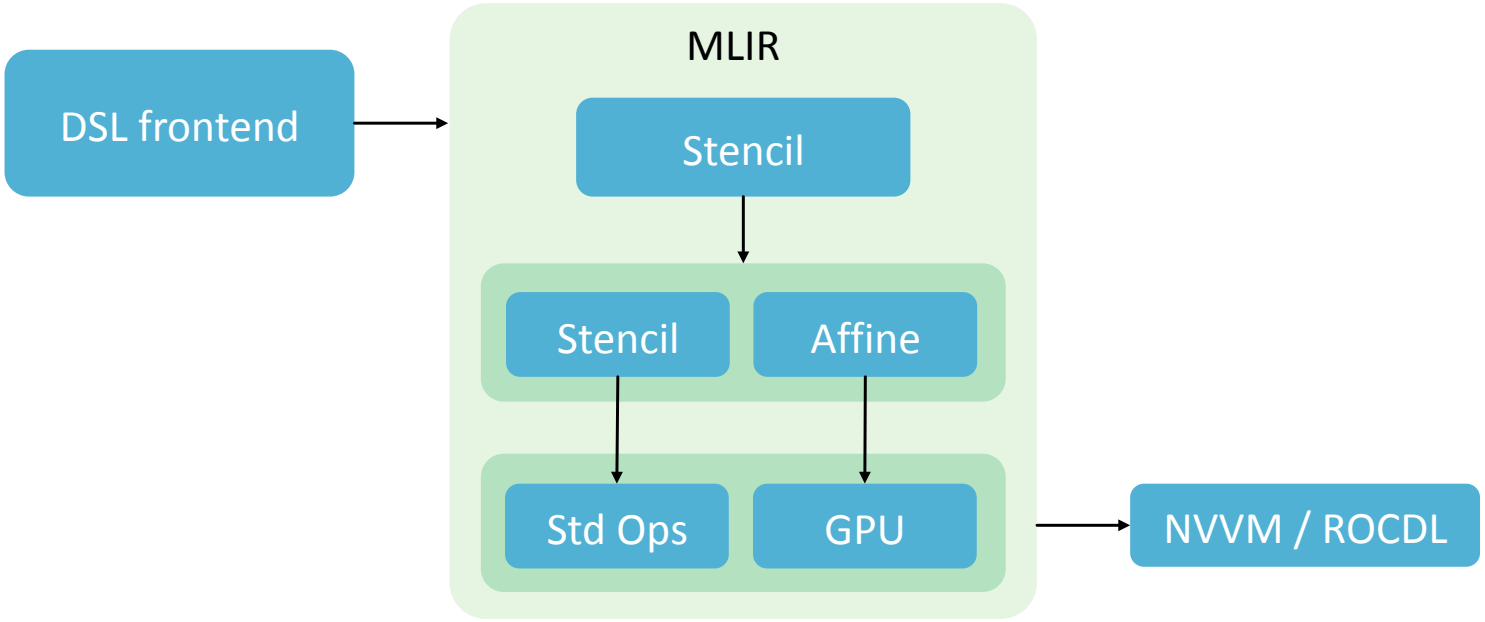
Code Generator

- CUDA
- GridTools
- Debug


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.


ice at record
nd shrinking
TensorFlow
Save our Planet

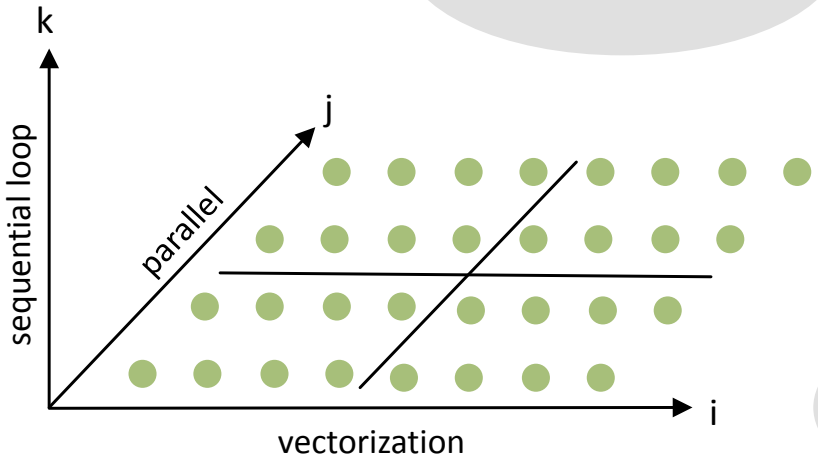
Climate Stencil Compilation with MLIR



GPU Execution Model and Optimizations

loop shifting & fusion
sliding window in registers

overlapped tiling
shared memory

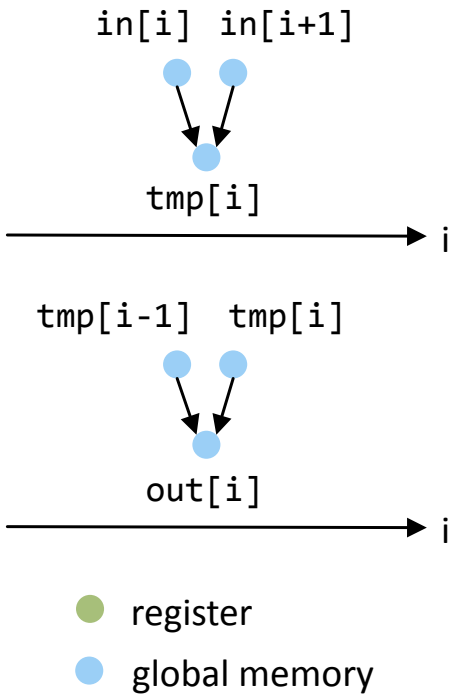


loop tiling
stencil inlining

threads (grouped in blocks)

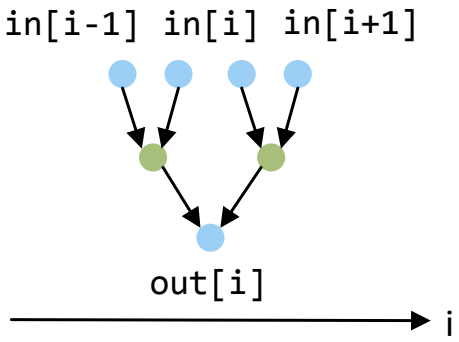
Stencil Inlining

```
for(int i = IB; i < IE; i++)
  tmp[i] = in[i] + in[i+1];

for(int i = IB; i < IE; i++)
  out[i] = tmp[i] + tmp[i-1];
```

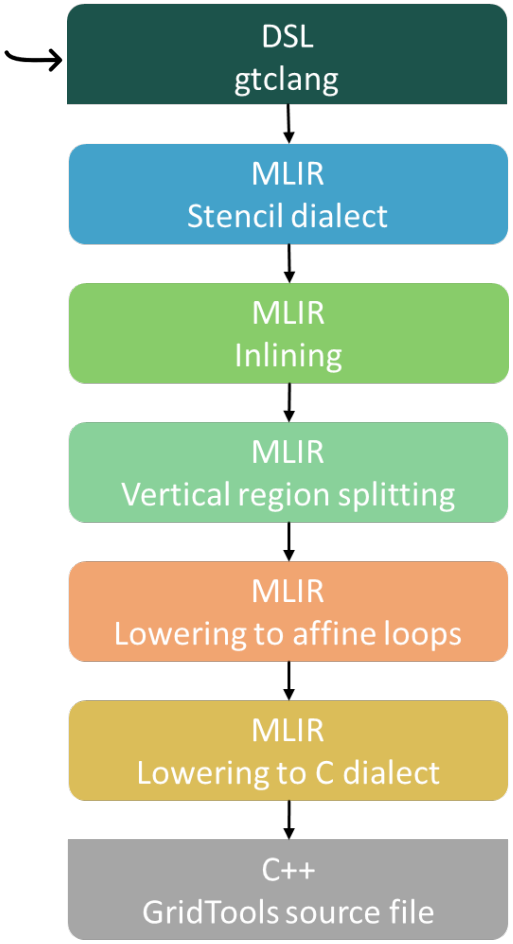


Stencil Inlining

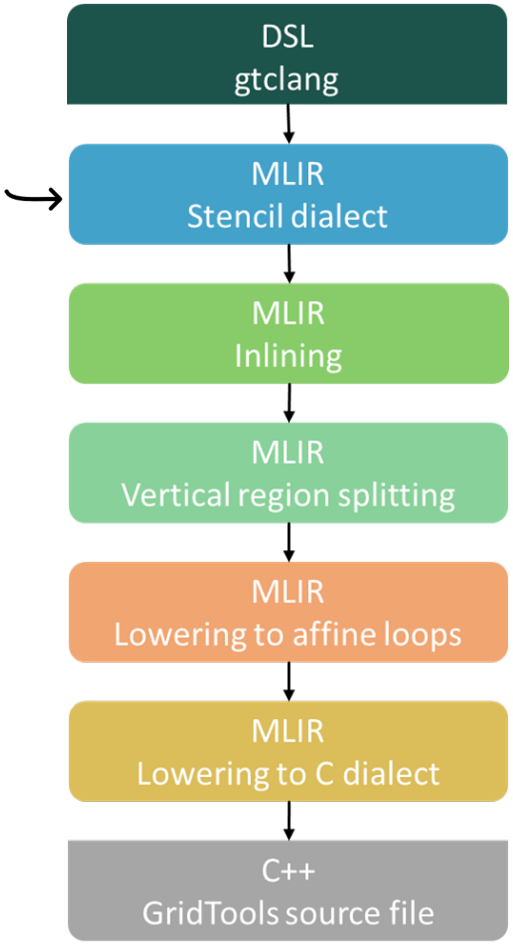
```
for(int i = IB; i < IE; i++)
  tmp[i] = in[i] + in[i+1];

for(int i = IB; i < IE; i++)
  out[i] = tmp[i] + tmp[i-1];
```



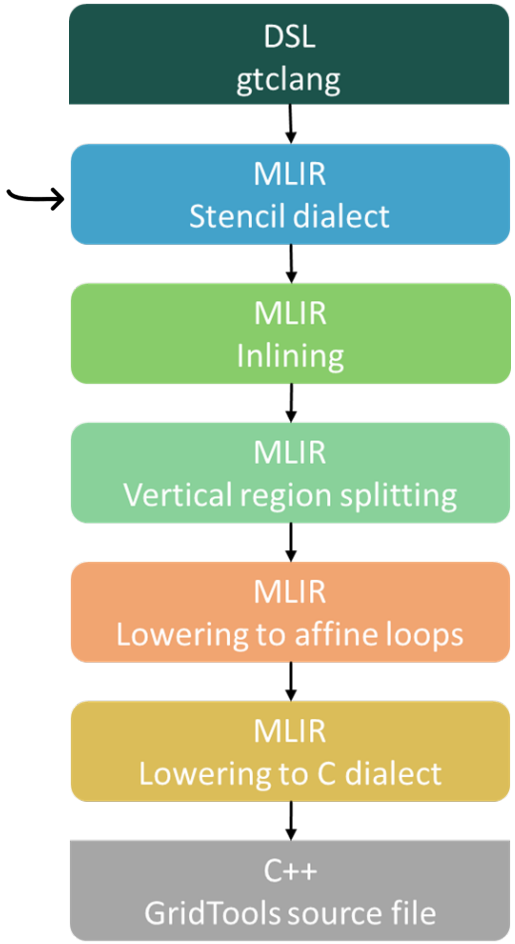

```
for(int i = IB; i < IE; i++)
  out[i] =
    (in[i] + in[i+1]) +
    (in[i-1] + in[i]);
```



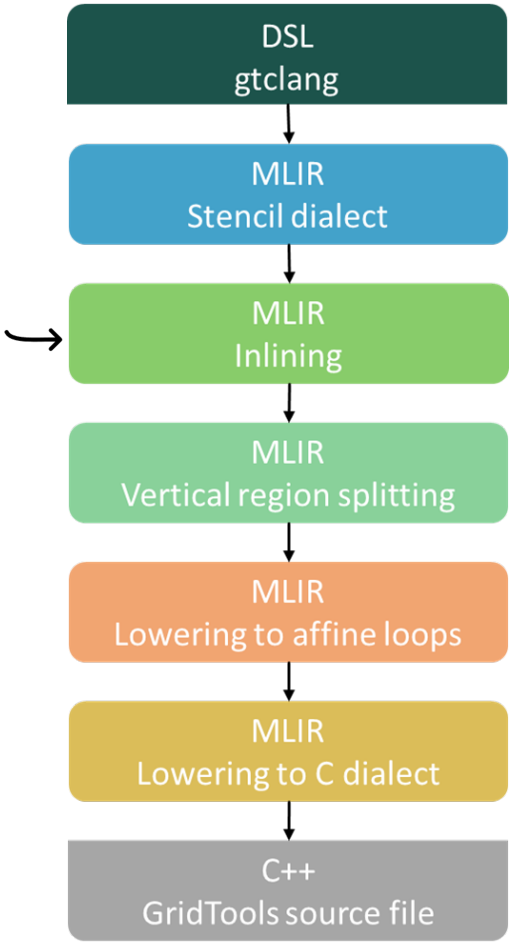

```
stencil_function laplacian {
  storage phi;

  Do {
    return phi(i + 1) + phi(i - 1) +
           phi(j + 1) + phi(j - 1) - 4.0 * phi;
  }
};

stencil hori_diff_stencil {
  storage u, out;
  var lap;
  Do {
    vertical_region(k_start, k_end) {
      lap = laplacian(u);
      out = laplacian(lap);
    }
  }
};
```



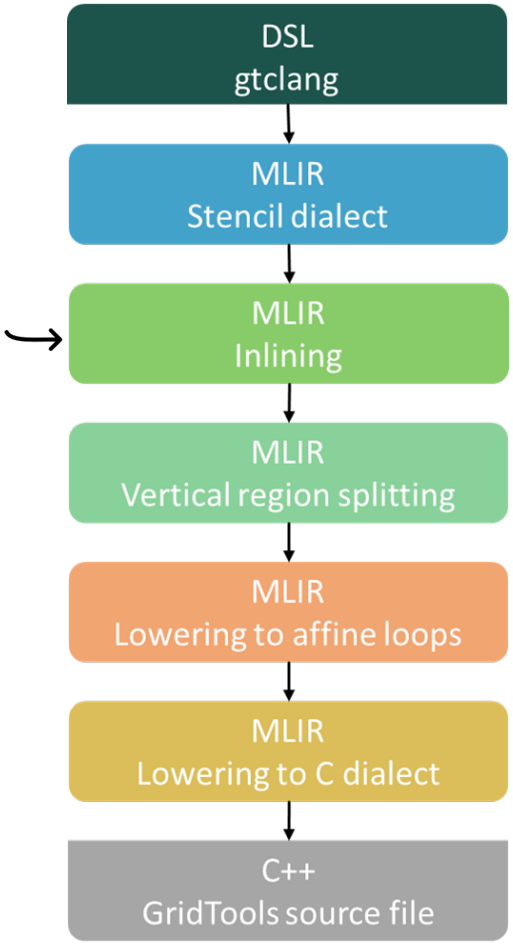

```
func @laplacian(%arg0: !stencil<"field:f64">) -> f64
  attributes  {stencil.function} {
  %0 = stencil.constant_offset 1 0 0
  %1 = stencil.read(%arg0, %0) : f64
  // ...
  %cst = constant 4.000000e+00 : f64
  %11 = stencil.constant_offset 0 0 0
  %12 = stencil.read(%arg0, %11) : f64
  %13 = stencil.mul(%cst, %12) : f64
  %14 = stencil.sub(%10, %13) : f64
  return %14 : f64
}
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  %0 = stencil.temp : !stencil<"field:f64">
  %1 = stencil.context "kstart" : index
  %2 = stencil.context "kend" : index
  stencil.vertical_region(%1, %2) {
    // ...
    %6 = stencil.lambda @laplacian(%0) : (!stencil<"field:f64">) -> f64
    %7 = stencil.constant_offset 0 0 0
    %8 = stencil.read(%6, %7) : f64
    stencil.write(%arg1, %8) : f64
  }
  return
}
```



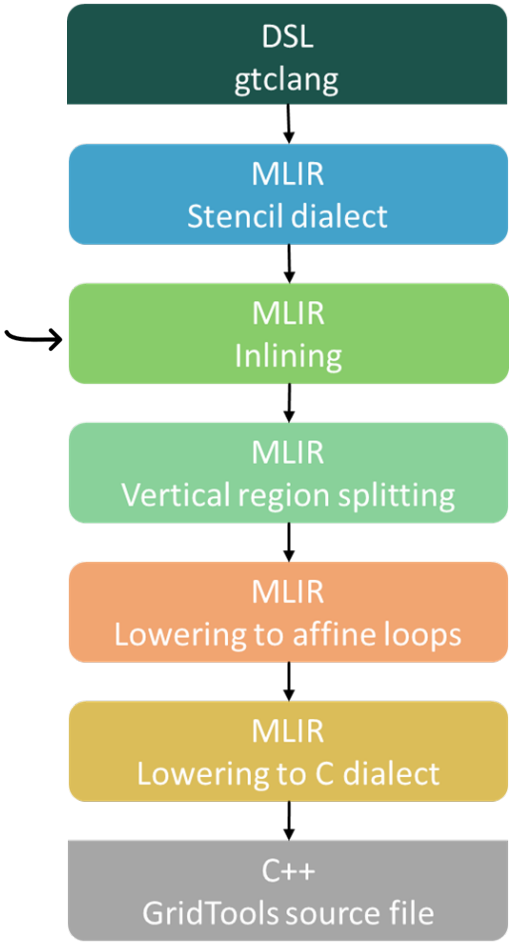

```
func @laplacian(%arg0: !stencil<"field:f64">) -> f64
  attributes  {stencil.function} {
  %0 = stencil.constant_offset 1 0 0
  %1 = stencil.read(%arg0, %0) : f64
  // ...
  %cst = constant 4.000000e+00 : f64
  %11 = stencil.constant_offset 0 0 0
  %12 = stencil.read(%arg0, %11) : f64
  %13 = stencil.mul(%cst, %12) : f64
  %14 = stencil.sub(%10, %13) : f64
  return %14 : f64
}
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  %0 = stencil.temp : !stencil<"field:f64">
  %1 = stencil.context "kstart" : index
  %2 = stencil.context "kend" : index
  stencil.vertical_region(%1, %2) {
    // ...
    %6 = stencil.lambda @laplacian(%0) : (!stencil<"field:f64">) -> f64
    %7 = stencil.constant_offset 0 0 0
    %8 = stencil.read(%6, %7) : f64
    stencil.write(%arg1, %8) : f64
  }
  return
}
```



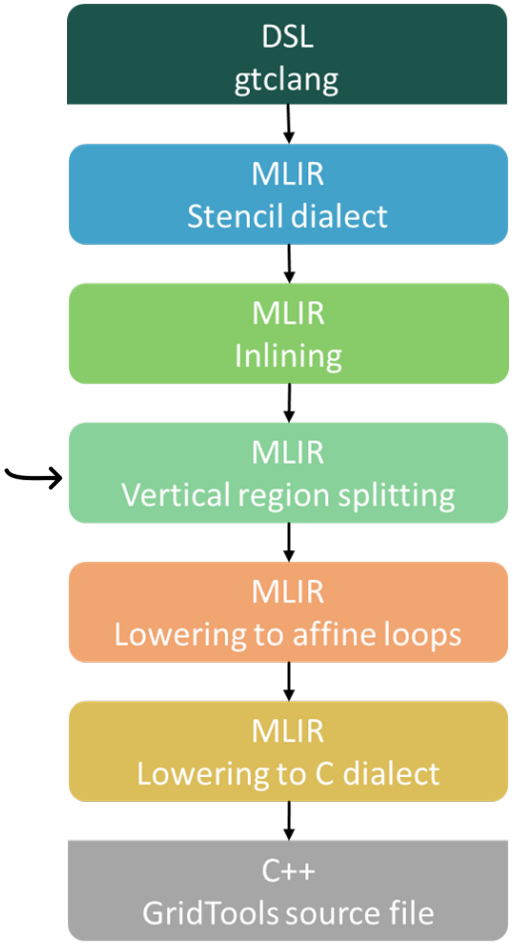

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%1, %2) {
    // ...
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
    // ...
  }
  return
}
```



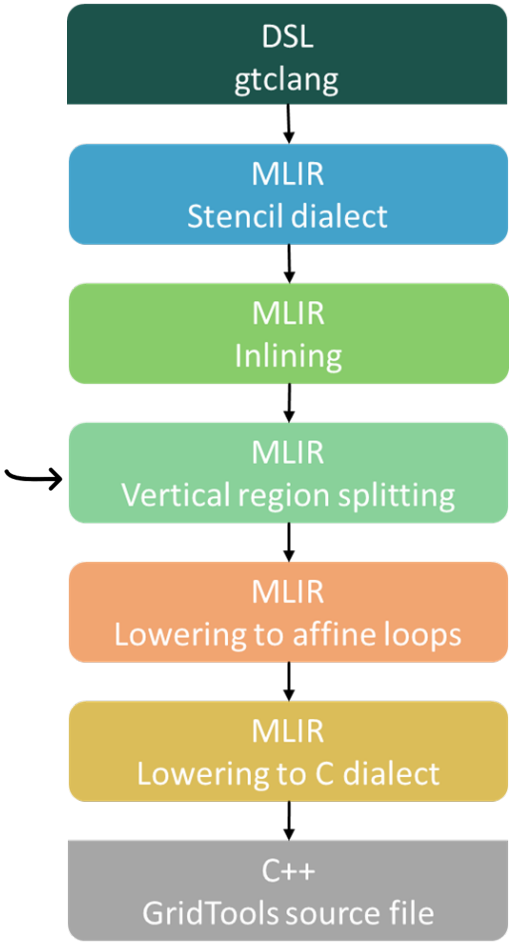

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%1, %2) {
    // ...
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
    // ...
  }
  return
}
```



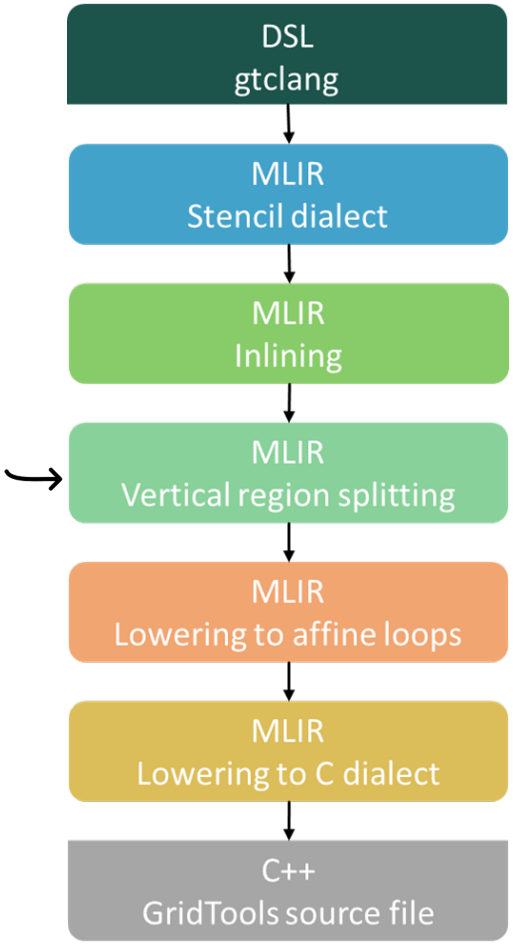

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%1, %2) {
    // ...
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
    // ...
  }
  return
}
```



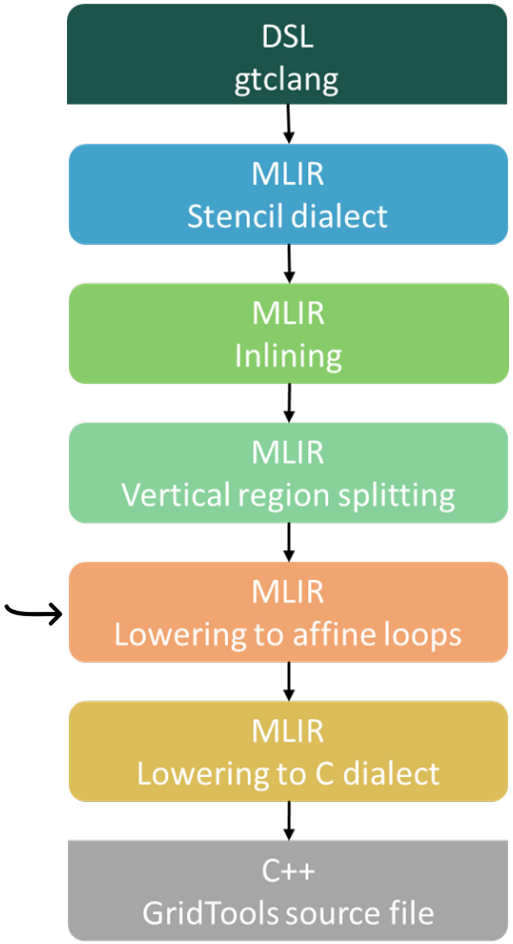

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%1, %2) {
    // ...
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
    // ...
  }
  return
}
```



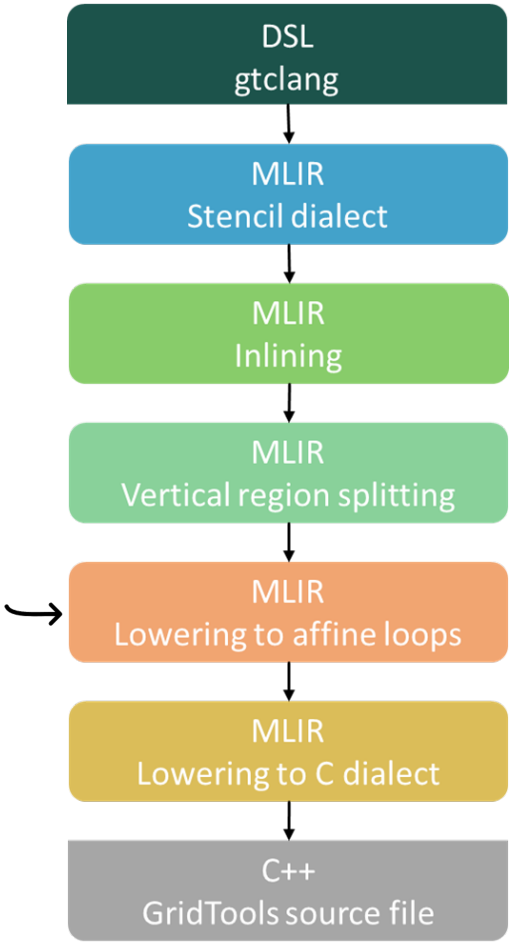

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%3, %4) {
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
  }
  stencil.vertical_region(%3, %4) {
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
  }
  return
}
```



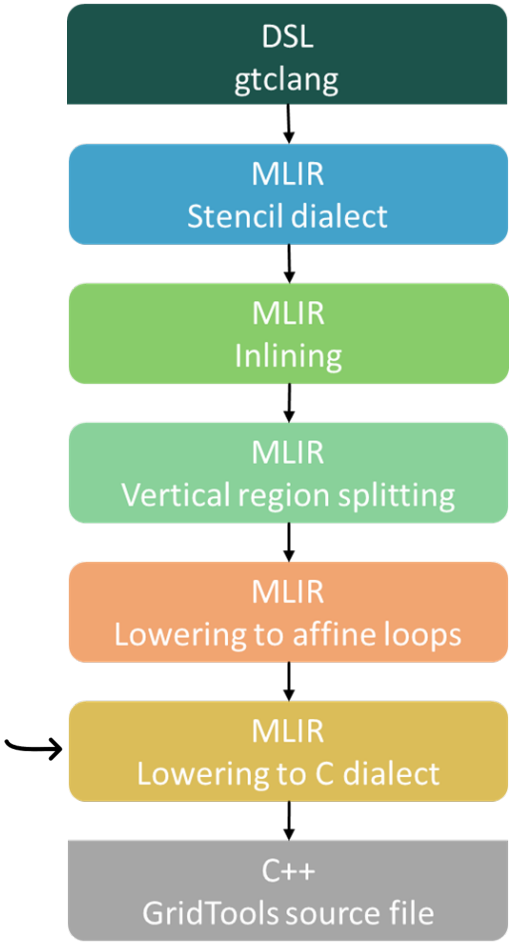

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  stencil.vertical_region(%3, %4) {
    %22 = stencil.constant_offset 1 0 0
    %23 = stencil.read(%2, %22) : f64
    %24 = stencil.constant_offset -1 0 0
    %25 = stencil.read(%2, %24) : f64
    %26 = stencil.add(%23, %25) : f64
    // ...
    %cst_0 = constant 4.000000e+00 : f64
    %33 = stencil.constant_offset 0 0 0
    %34 = stencil.read(%2, %33) : f64
    %35 = stencil.mul(%cst_0, %34) : f64
    %36 = stencil.sub(%32, %35) : f64
    stencil.write(%0, %36) : f64
  }
  stencil.vertical_region(%3, %4) {
    %37 = stencil.constant_offset 0 0 0
    %38 = stencil.read(%0, %37) : f64
    stencil.write(%arg1, %38) : f64
  }
  return
}
```



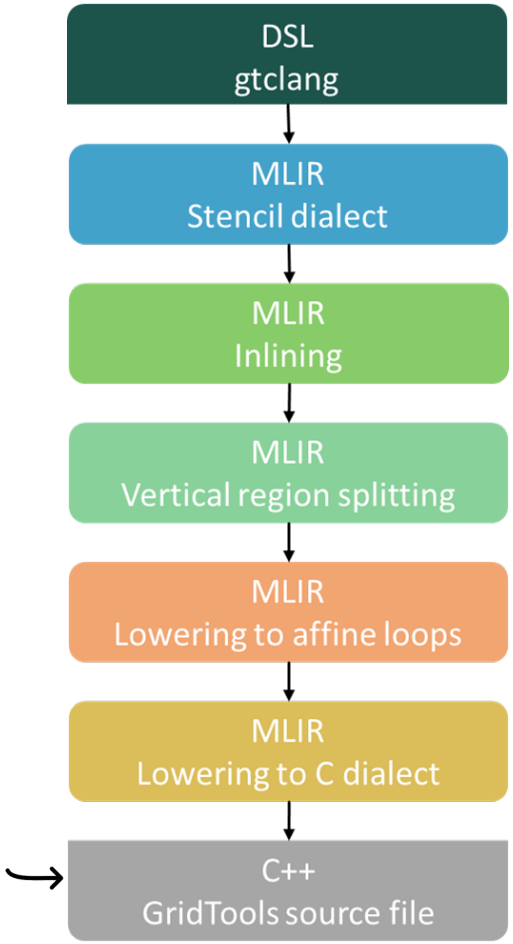

```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  %49 = stencil.context "istart" : index
  %50 = stencil.context "iend" : index
  %51 = stencil.context "jstart" : index
  %52 = stencil.context "jend" : index
  affine.for %i9 = #map2(%3) to #map3(%4) {
    stencil.induction_var "K" %i9 : index
    affine.for %i10 = #map2(%49) to #map3(%50) {
      stencil.induction_var "I" %i10 : index
      affine.for %i11 = #map2(%51) to #map3(%52) {
        stencil.induction_var "J" %i11 : index
        %53 = stencil.constant_offset 0 0 0
        %54 = stencil.read(%0, %53) : f64
        stencil.write(%arg1, %54) : f64
      }
    }
  }
  return
}
```




```
func @hori_diff_stencil(%arg0: !stencil<"field:f64">,
                        %arg1: !stencil<"field:f64">) {
  // ...
  %49 = stencil.context "istart" : index
  %50 = stencil.context "iend" : index
  %51 = stencil.context "jstart" : index
  %52 = stencil.context "jend" : index
  affine.for %i9 = #map2(%3) to #map3(%4) {
    stencil.induction_var "K" %i9 : index
    affine.for %i10 = #map2(%49) to #map3(%50) {
      stencil.induction_var "I" %i10 : index
      affine.for %i11 = #map2(%51) to #map3(%52) {
        stencil.induction_var "J" %i11 : index
        %53 = stencil.constant_offset 0 0 0
        %54 = stencil.read(%0, %53) : f64
        stencil.write(%arg1, %54) : f64
      }
    }
  }
  return
}
```




```
func @hori_diff_stencil(%arg0: !C.voidptr, %arg1: !C.voidptr) {
  // ...
  %62 = call @istart() : () -> index
  %63 = call @iend() : () -> index
  %64 = call @jstart() : () -> index
  %65 = call @jend() : () -> index
  %c1_11 = constant 1 : index
  %66 = C.addi(%4, %c1_11) : index
  C.for (%i9 = %3 to %66) {
    stencil.induction_var "K" %i9 : index
    %c1_12 = constant 1 : index
    %67 = C.addi(%63, %c1_12) : index
    C.for (%i10 = %62 to %67) {
      stencil.induction_var "I" %i10 : index
      %c1_13 = constant 1 : index
      %68 = C.addi(%65, %c1_13) : index
      C.for (%i11 = %64 to %68) {
        stencil.induction_var "J" %i11 : index
        %69 = stencil.constant_offset 0 0 0
        %70 = call @readTemp(%0, %i10, %i11, %i9, %69) : // ...
        call @write(%arg1, %70, %i10, %i11, %i9) : // ...
  }}}
  return
}
```

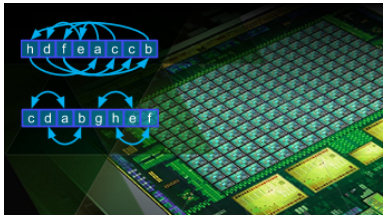



```
// gridtools boilerplate

void hori_diff_stencil(void *v_0, void *v_1) {
  // ...
  int32_t v_69 = istart();
  int32_t v_70 = iend();
  int32_t v_71 = jstart();
  int32_t v_72 = jend();
  for (int32_t i_0 = v_5; i_0 < v_12; i_0++) {
    int32_t v_73 = v_70 + v_11;
    for (int32_t i_1 = v_69; i_1 < v_73; i_1++) {
      int32_t v_74 = v_72 + v_11;
      for (int32_t i_2 = v_71; i_2 < v_74; i_2++) {
        int32_t v_75[] = {0, 0, 0};
        double v_76 = readTemp(v_2, i_1, i_2, i_0, v_75);
        write(v_1, v_76, i_1, i_2, i_0);
      }
    }
  }
  return
}
```

Low-level Dialect

```
stencil.iir {
  stencil.stencil(%arg0: !stencil<"field:f64">, %arg1: !stencil<"field:f64">) {
    stencil.multi_stage "Parallel" {
      stencil.stage {
        stencil.do_method [0, 0, 60, 0] {
          %0 = stencil.field_access %arg1 [0, 0, 0] : !stencil<"ptr:f64">
          %1 = stencil.field_access %arg0 [0, 0, 0] : !stencil<"ptr:f64">
          %2 = stencil.get_value %0 : f64
          %3 = stencil.get_value %1 : f64
          %4 = addf %2, %3 : f64
          %cst = constant 4.000000e+00 : f64
          %5 = mulf %4, %cst
          stencil.write %0, %5 : f64
        }
      }
    }
  }
}
```

GPU Dialect Extensions



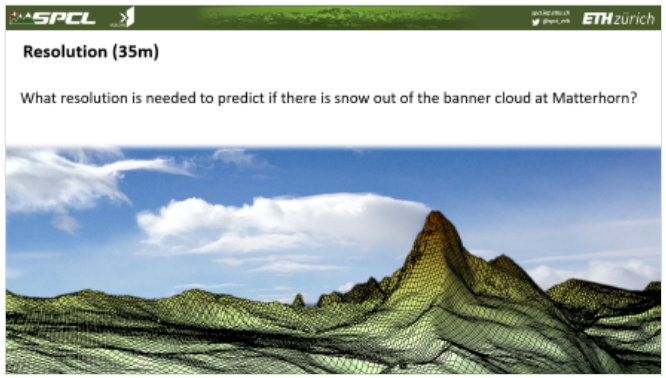
SUPPORT FOR WARP-LEVEL PRIMITIVES, SUCH AS SHUFFLING

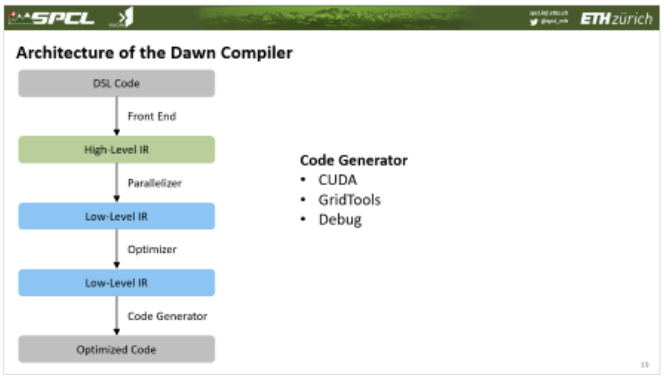
ACCESS TO SHARED MEMORY

SUPPORT FOR PARALLEL KERNEL EXECUTION

INTER-NODE COMMUNICATION FOR DISTRIBUTED GPU APPLICATIONS

Conclusion


Resolution (35m)
What resolution is needed to predict if there is snow out of the banner cloud at Matterhorn?


Architecture of the Dawn Compiler
DSL Code
Front End
High-Level IR
Parallelizer
Low-Level IR
Optimizer
Low-Level IR
Code Generator
Optimized Code
Code Generator
CUDA
GridTools
Debug


TensorFlow


Low-level Dialect
stencil.iir {
stencil.stencil(%arg0: !stencil<"field:f64">, %arg1: !stencil<"field:f64">) {
stencil.multi_stage "Parallel" {
stencil.stage {
stencil.do_method [0, 0, 60, 0] {
%0 = stencil.field_access %arg1 [0, 0, 0] : !stencil<"ptr:f64">
%1 = stencil.field_access %arg0 [0, 0, 0] : !stencil<"ptr:f64">
%2 = stencil.get_value %0 : f64
%3 = stencil.get_value %1 : f64
%4 = addf %2, %3 : f64
%cst = constant 4.000000e+00 : f64
%5 = mulf %4, %cst
stencil.write %0, %5 : f64
}
}
}
}
}