Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

Platform for Advanced Scientific Computing

High-level Domain Specific Language and Abstractions for NWP and Climate Models

COSMO PP Impact

C. Osuna, X. Lapillonne, T. Wicky, O. Fuhrer and the GridTools team, ESCAPE-2, ESiWACE-2 community

2016…

"COSMO first operational NWP model running on a GPU based heterogeneous architecture"

COSMO 1.1km
+ 21 members COSMO 2km

Computational cost = 40x wrt previous operational product

Large software investment

Memory efficiency of dynamical core 67%

What is a DSL for weather and climate codes?

Why are we developing DSLs ?

I : Performance

COSMO-E member (2.2km) (version 15th May 2017)

	F90 CPU double	→C++ CPU double	→C++ GPU double	→C++ GPU single
Dynamics (STELLA)	280s	1.3x	2.8x	4.4x
Physics (OpenACC)	80s	-	2.1x	3.0x
Total	374s	1.2x	2.2x	**3.4x**

CPU setup: 8 Haswell CPUs (Intel Xeon E5-2690, 96 cores in total)
GPU setup: 4 NVIDIA Tesla K80 (8 GK210 GPUs) + 8 Haswell cores

Dynamics: STELLA GPU optimizations

STELLA optimization	Time	Speedup
No optimization	93.7s	
Merge stencils	84.9s	1.10x
+ software managed caches	67.2s	1.39x
+ parallel vertical levels	65.7s	1.42x
+ texture cache	64.5s	1.45x

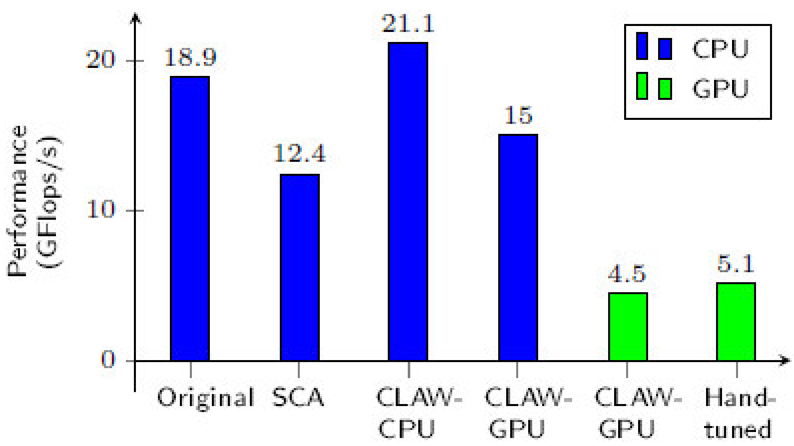
II : Portability

Many examples where in the end, performance portability is not attainable with directives

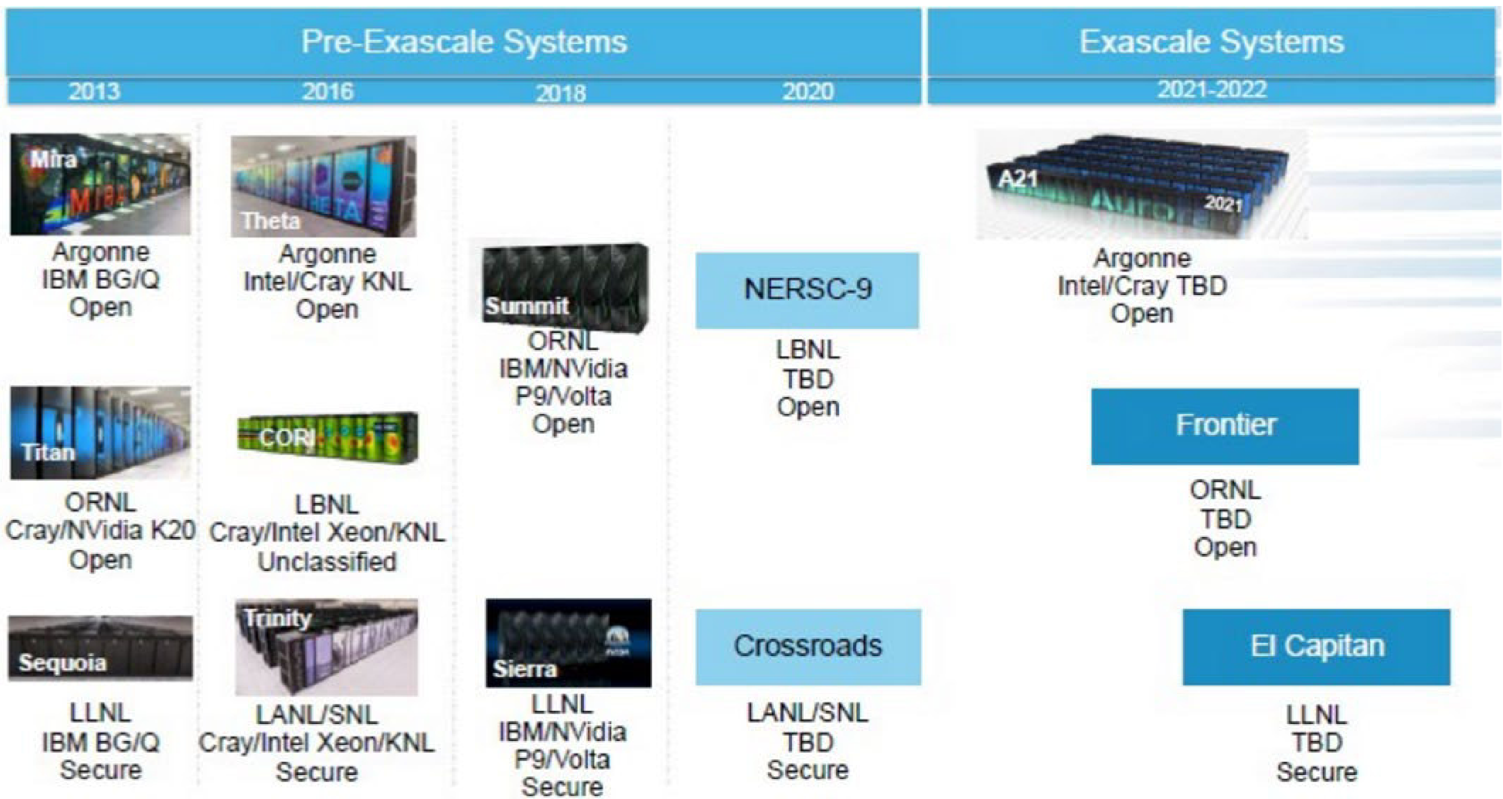
```
!cdir …
!omp parallel for private(…) schedule …
do blocks loop {
  !acc data present(f1, f2,…)
  !acc loop gang vector
  do i,j,k loop {
    …
  }
}
```

Final thoughts:
- **directives are not comments**, they are code, need to be maintained
- we have not solved the software engineering problem with the multiple directive/architecture paradigm: **Testing, maintainance, release mgmt**

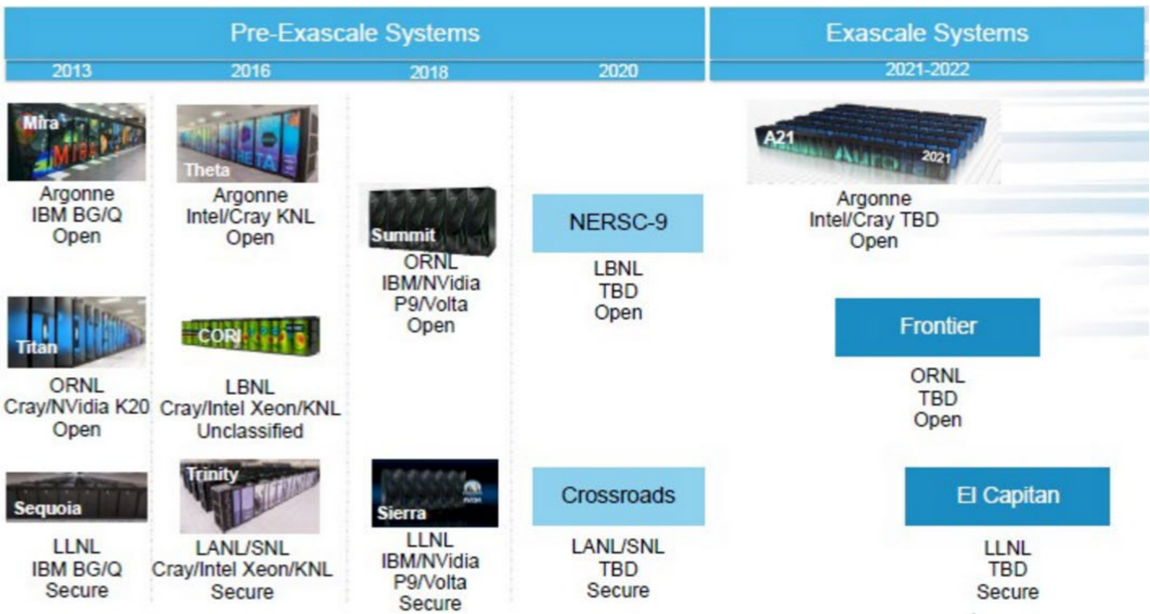
CLAW SCA results of IFS CLOUDSC
(M. Lange @ECMWF PASC 19)



III : Maintainance and support multiple architectures



III : Maintainance and support multiple architectures



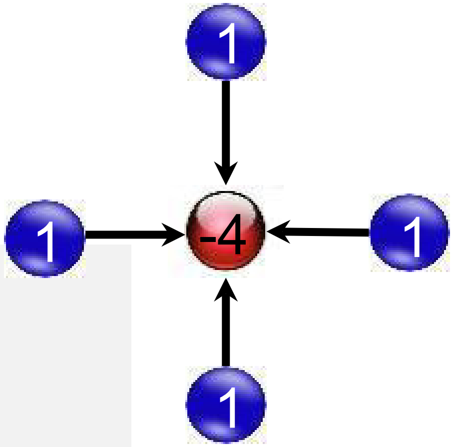
Intel Xe GPUs
OneAPI (incorporates C++ SYCL)

AMD GPUs

OMP, Openacc ?

??

GridTools DSL example



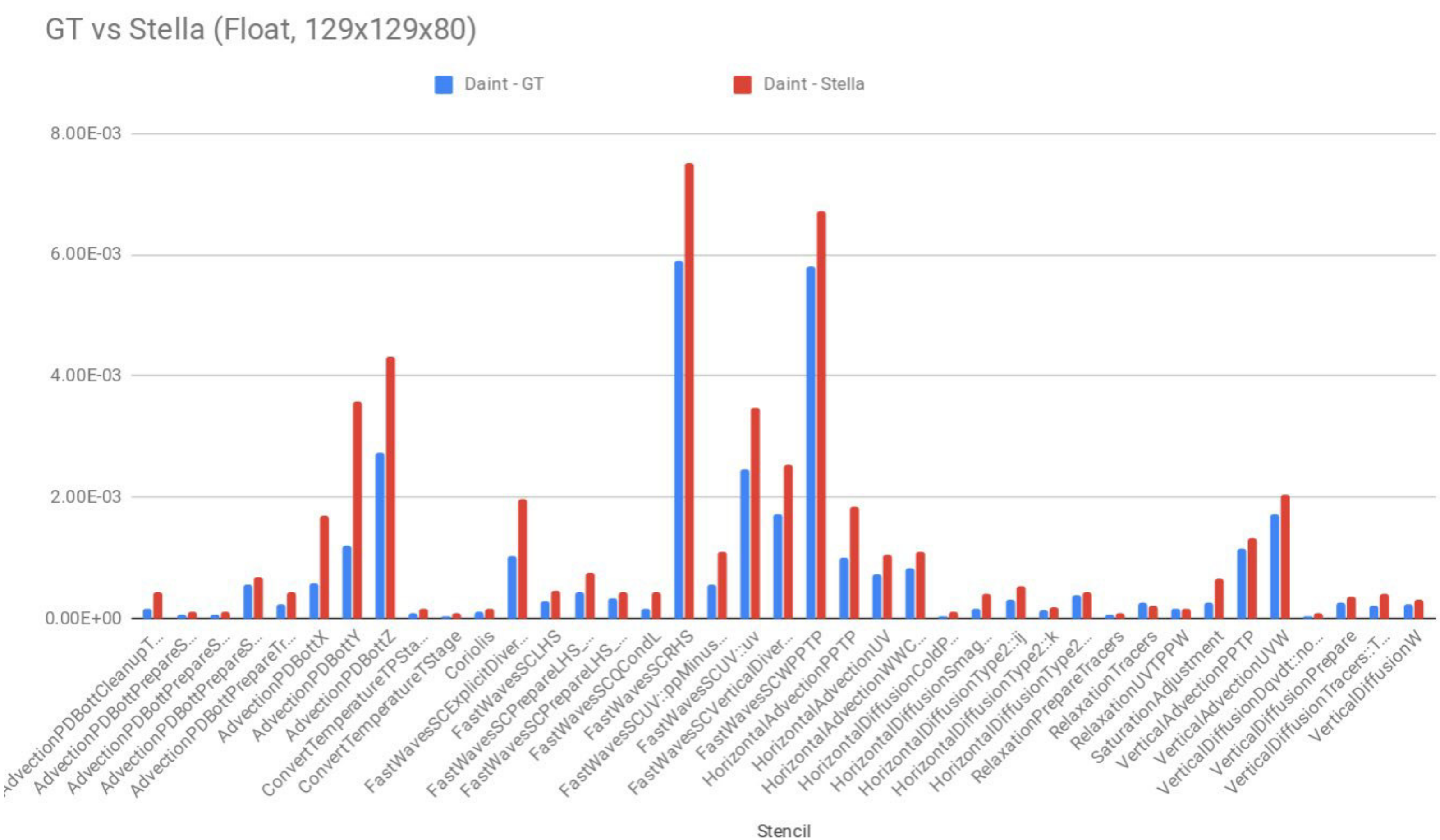
- Syntax for a stencil

```
struct Laplace{
  typedef in_accessor<0, extent<-1,1,-1,1> > u;
  typedef out_accessor<1> lap;

  template <typename Evaluation>
  static void Do(Evaluation eval, domain)
  {
      eval(lap()) = eval(-4*u() + u(i+1) + u(i-1) + u(j+1) + u(j-1));
   }
};
```

- Composing stencils

```
auto hori_diff =
    make_computation<Cuda>(
        domain, grid,
        make_multistage(
            execute<forward>(),

            make_stage<Laplace>(lap(), u(), crlat0()),
            make_stage<Div>(u(), lap(), coeff())
    )
);
hori_diff->run();
```

- The GridTools dycore will replace STELLA in production on the new V100 MeteoSwiss HPC system (arolla)
  In COSMO v6.0 official



Next Generation DSL

High-level DSL toolchain: gtclang/dawn (in development)

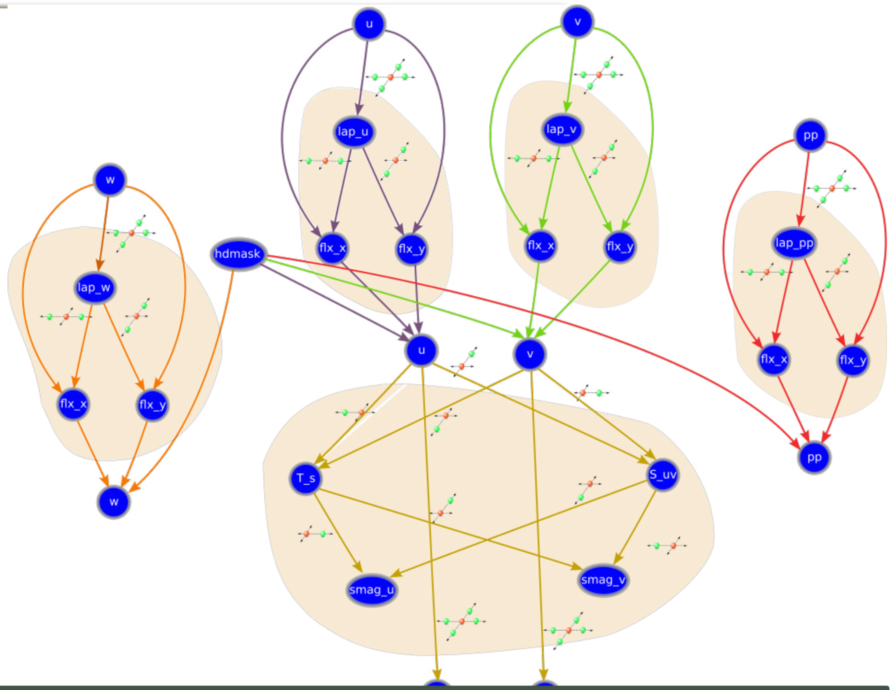
- 85 LOC vs 12 LOC

```
stencil hori_diff {
  storage out, u, coeff;
  var flx, fly, lap;

  Do {
    vertical_region(k_start, k_end)
      lap = 4*u - (u[i+1]+u[i-1]+u[j+1]+u[j-1]);
      flx = lap[i+1] - lap;
      if(flx*(u[i+1] - u) > 0) flx = 0.;
      fly = lap[j+1] - lap;
      if(fly*(u[j+1] - u) > 0) fly = 0.;
      out = u - coeff*(flx - flx[i-1] + fly - fly[j-1]);
} } }
```

gtclang high level DSL: https://github.com/MeteoSwiss-APN/gtclang

Parallelization and optimizations (fusion) are automatized by the DSL

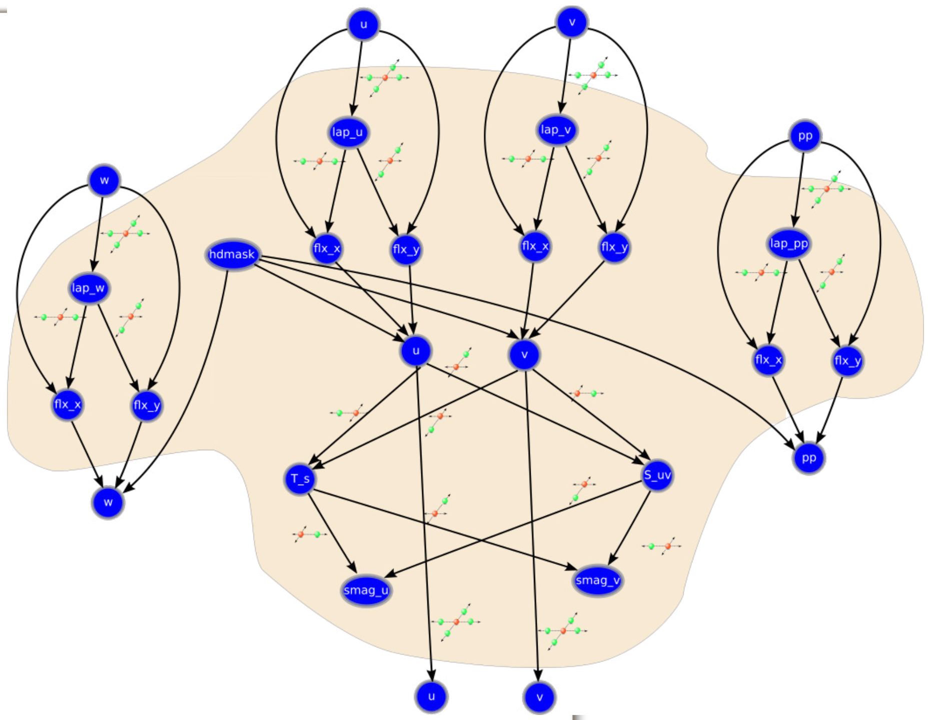
```
stencil_function diffuse {
  storage u, hdmask;
  var flx, fly, lap;
  Do {
    vertical_region(k_start, k_end)
      lap = 4*u - (u[i+1]+u[i-1]+u[j+1]+u[j-1]);
      flx = lap[i+1] - lap;
      if(flx*(u[i+1] - u) > 0) flx = 0.;
      fly = lap[j+1] - lap;
      if(fly*(u[j+1] - u) > 0) fly = 0.;
      u = u - hdmask*(flx - flx[i-1] +
                fly - fly[j-1]);
} } };
stencil hori_diff {
  storage u, hdmask;

  Do {
    diffuse(u,hdmask);
    diffuse(v,hdmask);
    diffuse(w,hdmask);
    diffuse(pp,hdmask);
    smag(u,v,hdmask);
} } };
```



- 4 parallel groups
- edges crossing a group is access to main memory
- edges within group is access to cache memory

Parallelization and optimizations (fusion) are automatized by the DSL

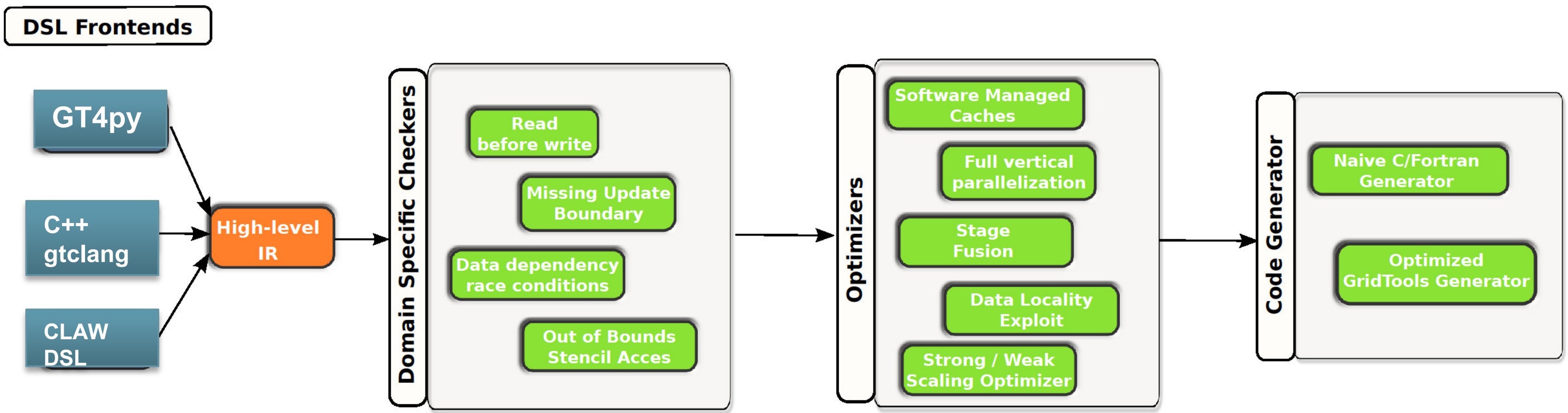
```
stencil_function diffuse {
  storage u, hdmask;
  var flx, fly, lap;
  Do {
    vertical_region(k_start, k_end)
      lap = 4*u - (u[i+1]+u[i-1]+u[j+1]+u[j-1]);
      flx = lap[i+1] - lap;
      if(flx*(u[i+1] - u) > 0) flx = 0.;
      fly = lap[j+1] - lap;
      if(fly*(u[j+1] - u) > 0) fly = 0.;
      u = u - hdmask*(flx - flx[i-1] +
                fly - fly[j-1]);
} } };
stencil hori_diff {
  storage u, hdmask;

  Do {
    diffuse(u,hdmask);
    diffuse(v,hdmask);
    diffuse(w,hdmask);
    diffuse(pp,hdmask);
    smag(u,v,hdmask);
} } };
```



- Fused all operators in 1 group

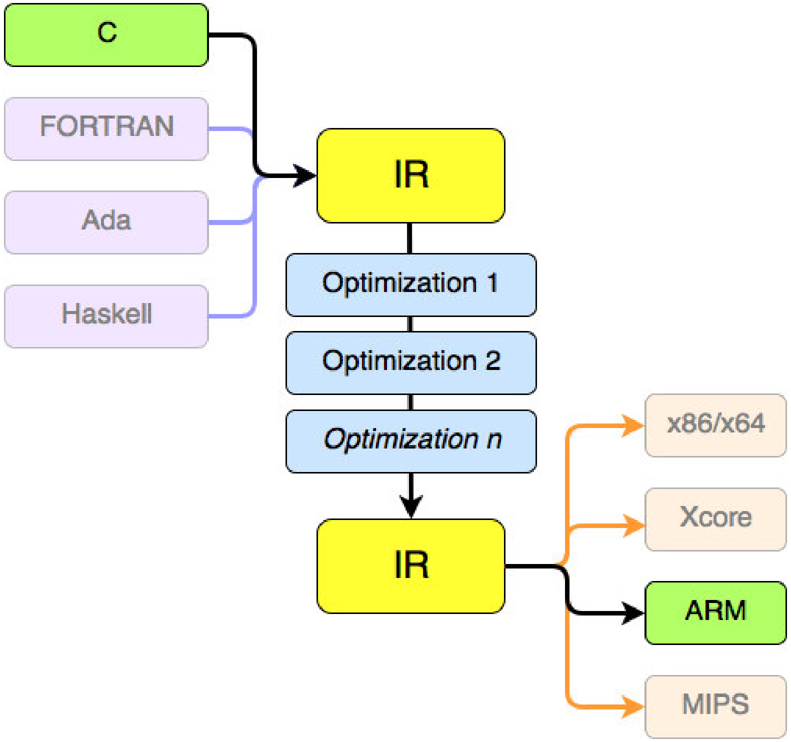
A High-Level DSL design

Code generation (specific programming language/model per architecture)
Naive code generation for scientific debugging
No template metaprogramming
A standard HIR to decouple the frontend from the rest of the toolchain



What is an IR?

llvm IR is a language indenpendent, hardware agnostic Intermediate representation that allows to easily build new languages.



Dead code elimination
Constant folding
Loop unrolling
Tail call elimination
…
http://llvm.org/docs/Passes.html

Building an HIR as a community effort

→ First workshop on DSL/HIR definition held on Jan 2019

LE1. computation: a kernel computation.

contains a domain and a set of statements.

LE2. domain: the 3D domain that define the iteration space where the grid points will be updated with the corresponding computation.

contains a set of dimensions that define the iteration space

LE3. field: identifier that identifies each of the fields used within a computation, e.g. lap.

LE4. field access: a field access to the center of the grid point or a n point (like i+1).

contains a set of dimensions (e.g. i)

a set of integer (neighbor) offsets

```
field u,lap
computation in domain {
    lap = -4*u + u[i+1] + u[i-1] + u[j-1] + u[j+1]
    u = -4*lap + lap[i+1] + lap[i-1] + lap[j-1] +
lap[j+1]
}
```

https://github.com/MeteoSwiss-APN/HIR

ICON roadmap for high-level DSL

- Jan 2020: first basic support for simple operators (div/curl) on staggered fields in an icosahedral grid.
- Toolchain will support unstructured meshes for GPU/CPU (including ICON mesh)
- Python based frontend, first release Oct 2019 (Cartesian grids)
- 2021: DSL tool chain supporting most of irregular grids patterns
- 2022: Full dynamical core (w/o semilagrangian)
- All the code is licensed under permissive open source

Full example of interpolation of two fields at cell centre (from ICON dynamical core) in HIR

```
field [edges,levels] rho_e, vn, vt, theta_v_e
field [cells,levels] rho, theta_v

field [edges,cells] c_lin_e

field [edges] inv_dual_edge_length, inv_primal_edge_length,
tangent_orientation

field [vertices,levels] rho_v, theta_v_v

vertical_region(start_level, end_level) {
  compute_on edges in domain {
    rho_e = sum(cells, c_lin_e * rho) - dtime
       * (vn * inv_dual_edge_length * sum(cells, [-1,1], rho)
           + vt * inv_primal_edge_length * tangent_orientation *
           sum(cells, [-1,1], rho))
    theta_v_e = sum(cells, theta_v) - dtime
       * (vn * inv_dual_edge_length * sum(cells, [-1,1], theta_v)
       + vt * inv_primal_edge_length * tangent_orientation *
       sum(cells, [-1,1], z_theta_v_v))
  }
}
```

Engage!

- Large development team ~10 people:
  MeteoSwiss (ICON), CSCS (FVM), Vulcan (FV3)
- We need ICON model developers to drive the DSL language features, usability, etc (from Jan 2020)
- Possibility to attract funding for HPC efforts in a collaboration model developer institutions: EuroHPC ?

Conclusions

- DSLs and code transformations have proven to provide portability and good level of performance portability for weather and climate applications
- STELLA / GridTools first DSL in production for GPU weather models.
- New generation (code generation) of descriptive / high level language in development for the next production phase

BACKUP SLIDES

HIR

https://github.com/MeteoSwiss-APN/HIR

```
Program {
 GridDimension {[ ncol , nlay , ngpt ]}
 Domain {
  parallel_domain_dim : ncol
  vertical_dim : nlay
  parallel dim : ngpt
 },
 FieldDecl {
  name : u, GridDimension {[ ncol ]}
 },
 VarDecl {
  type: int
  name: ncolbounds
 },
 ScopedProgram {
  Computation {
   dimension : nlay
   dimension : ncol
   BlockStmt {
     vuavg = u(nlay+1) - u
   }
  }
 }
```

- **FieldDecl** [
  dimensions: i,j,k,h
  string: name
]
- **VarDecl** (scalar)
- **Computations**: multidimensional boxed loops over dimensions of the domain
- **StencilAST**: description of the computations within a Computation.
- **For loops**: iterations over loop bounds on scalars (not dimensions of the domain)
- **BoundaryCondition**: specification of computation applied on a boundary to a field

MeteoSwiss

How do we build the domain specific concepts of an HIR?

Example from lat-lon compact stencils of finite different applications

Start with naive impl in existing general purpose programming language (GPPL),
No optimization or implementation details (tiling, fusion, ...)

```
DO k = 1, ke
  DO i = 0, ie
    DO j = 0, je
      lap(i,j,k) = -4*u(i,j,k) + u(i-1,j,k) +
u(i+1,j,k) + u(i,j-1,k) + u(i,j+1,k)
    ENDDO
  ENDDO
  DO i = 1, ie-1
    DO j = 1, je-1
      u(i,j,k) = -4*lap(i,j,k) + lap(i-1,j,k) +
lap(i+1,j,k) + lap(i,j-1,k) + lap(i,j+1,k)
    ENDDO
  ENDDO
ENDDO
```

Simplify in pseudo-code, eliminate semantic of GPPL not really used or redundant
(i.e. loop bounds, general array access)

```
field u,lap
computation in domain {
    lap = -4*u + u[i+1] + u[i-1] + u[j-1] +
u[j+1]
    u = -4*lap + lap[i+1] + lap[i-1] +
lap[j-1] + lap[j+1]
}
```

Vertical regions specializations

```
def horizontal_derivative(Δx, Δs, k_F, compute_domain, φ):
    """Compute the x-derivative, holding z fixed."""
    for i, j, k in compute_domain:
        if k < k_F:
```

$$\left(\frac{\partial \phi}{\partial x}\right)_{i,j,k} \approx \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x}$$

```
        else:
```

$$\left(\frac{\partial \phi}{\partial x}\right)_{i,j,k} \approx \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x} + \frac{\phi_{i,j,k-1} - \phi_{i,j,k+1}}{2\Delta s}\left(\frac{\partial s}{\partial x}\right)_{i,j,k}$$

```
    return
```

$$\frac{\partial \phi}{\partial x}$$

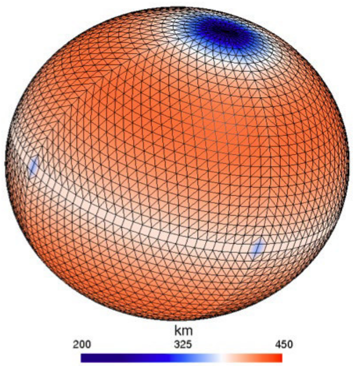
LE7. vertical_region: specifies a region in the vertical where the *computation AST* (LE5) is executed

contains loop bounds [start_level, end_level]

Loop order:

- forward
- backward
- parallel

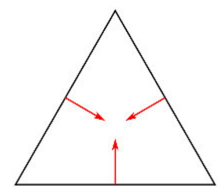
Irregular grids



N24 octahedral Gaussian grid

New elements of computations in irregular grids:

- different locations or function spaces
- No directional operators (few exceptions on edge based operators) but rather homogeneous stencil operators



$$\mathrm{div}(\mathbf{v})_i := \frac{1}{A_i} \sum_{l \in \mathcal{E}(i)} v_{n_l} (\mathbf{N}_l \cdot \mathbf{n}_{i,l}) l$$

LE14. neighbour_reduce: a reduction operation over near neighbours of certain *location type*

contains	neighbour *dimension* (LE6) (that has a *location type* (LE13))
	a reduction arithmetic operation (AST Expression)
	an initial value of the reduction
	computation AST (LE5) applied to each neighbour grid point

Full example of interpolation of two fields at cell centre (from ICON dynamical core) in HIR

```
field [edges,levels] rho_e, vn, vt, theta_v_e
field [cells,levels] rho, theta_v

field [edges,cells] c_lin_e

field [edges] inv_dual_edge_length, inv_primal_edge_length,
tangent_orientation

field [vertices,levels] rho_v, theta_v_v

vertical_region(start_level, end_level) {
  compute_on edges in domain {

    rho_e = neighbour_reduce(cells, op::plus, 0.0, c_lin_e * rho) -
dtime

       * (vn * inv_dual_edge_length * neighbour_reduce(cells, [-1,1],
0.0, rho)

           + vt * inv_primal_edge_length * tangent_orientation *

           neighbour_reduce(cells, [-1,1], 0.0, rho))
    theta_v_e = neighbour_reduce(cells, op::plus, 0.0, theta_v) -
dtime

       * (vn * inv_dual_edge_length * neighbour_reduce(cells, [-1,1],
0.0, theta_v)

       + vt * inv_primal_edge_length * tangent_orientation *

       neighbour_reduce(cells, [-1,1], 0.0, z_theta_v_v))
  }
}
```

This is not a language but an IR:

- complete & minimal semantic
- non redundant
- extensible
- Language independent

HIR prototype

- Current HIR prototype for lat-lon grids.
- If offers an API (java, python, C++) to build HIR instances for the toolchain frontend
- Input of a complete (lat-lon) implementation of an entire DSL toolchain by T. Wicky talk
- Demonstrations by R. Ford from PSyclone

```
hir = makeSIR(«hori_diff.cpp», [
  makeComputation(«hori_diff»,
  [makeField(«in»), makeField(«out»), makeField(«coeff»), makeField(«lap»,
is_temporary=false)],
  makeVerticalRegion( makeInterval(Interval.Start, Interval.End,0,0)),
    makeStmt(«lap = -4*in + in[1,0,0] + in[-1,0,0] + in[0,1,0] + in[0,-
1,0]»),
    makeStmt(«out = -4*lap + lap[1,0,0] + lap[-1,0,0] + lap[0,1,0] + lap[0,-
1,0]»),
```