2nd Dissemination Workshop
Sep 2021

Programming Models & Domain-Specific Languages

J. Behrens (DKRZ), R. Budich (MPI), M. Chiarelli (CMCC), W. Deconinck (ECMWF), I. Epicoco (CMCC), C. Müller (MSWISS), C. Osuna (MSWISS), M. Röthlin (MSWISS), B. Weber(MSWISS) & ESCAPE-2 WP2

FUNDED BY THE EUROPEAN UNION

ESCAPE-2 is funded by the European Commission under the Future and Emerging Technologies – High Performance Computing call for research and innovation actions, grant agreement 800897.

ESCAPE-2 WP2 Is About

- Define, develop, and apply a **domain-specific language** (DSL) toolchain applicable to a comprehensive list of algorithmic motives (dwarfs) in weather and climate prediction.
- **Demonstrate code adaptation** and code generation **via the DSL** toolchain for a number of representative and fundamentally different mathematical algorithms and horizontal discretizations.
- **Develop and promote APIs** and ***standard interfaces*** across the DSL toolchain in order to improve reusability and inter-operability, and leverage code adaptation to emerging HPC architectures.

Matrix transpose in different languages

matlab

```
A=B.'
```

Domain specific language:

domain = matrix operations

Semantic information: matrix transpose

Concise syntax for a problem

Abstracts implementation and hardware dependent details:

- ✔ OMP parallelization
- ✔ GPU cuda implementation
- ✔ linear algebra calls (blas, mkl,…)

C

```
for(int i=0; i < n; ++i) {
   for(int j=0; j < n; ++j) {
      double z=m[i][j];
      m(i,j)=m[j][i];
      m[j][i]=z;
   }
}
```

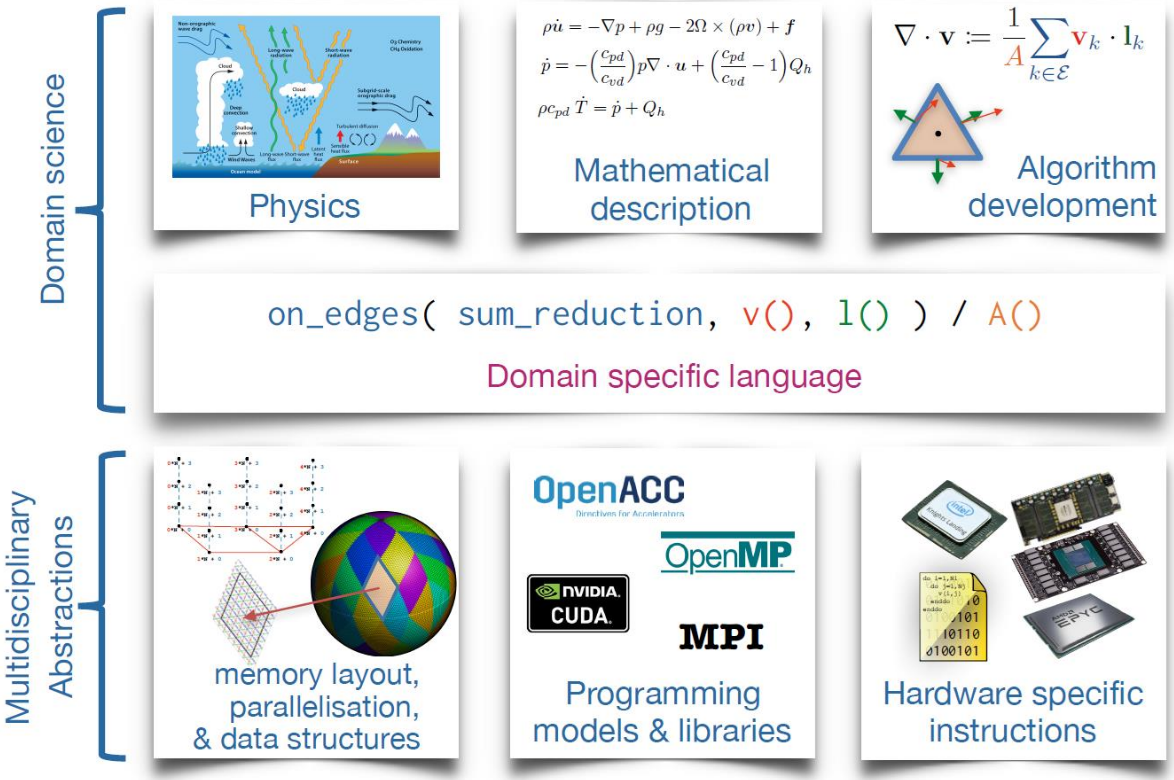
General purpose, it can solve any problem

Semantic information: double nested loop modifying matrix with self-dependencies

Domain Specific Languages for Weather and Climate

- Started 8 years ago pioneer research using DSL for COSMO
- In production at MeteoSwiss since 2016 for GPU based machines
- Growing interest and developments of solutions to portability problem:
  in weather & climate: e.g. GridTools, PSyClone DSL
  in other domains (e.g. AI):
    https://mlir.llvm.org/

ESCAPE 2 aims at developing a high-level DSL (with high-level language elements):

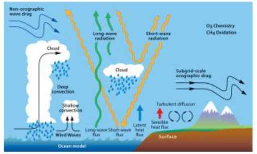


- **Concise language** for solving weather problems
- **High scientific productivity**
- Leverage **high-level semantic** of the problem to apply **domain specific optimizations**.

W. Deconinck (ECMWF)

aims at developing a high-level DSL (with high-level language elements):

Domain science



Physics

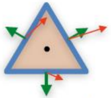
$$\rho\dot{u} = -\nabla p + \rho g - 2\Omega \times (\rho v) + f$$

$$\dot{p} = -\left(\frac{c_{pd}}{c_{vd}}\right) p \nabla \cdot u + \left(\frac{c_{pd}}{c_{vd}} - 1\right) Q_h$$

$$\rho c_{pd}\, \dot{T} = \dot{p} + Q_h$$

Mathematical description

$$\nabla \cdot \mathbf{v} := \frac{1}{A} \sum_{k \in \mathcal{E}} \mathbf{v}_k \cdot \mathbf{l}_k$$

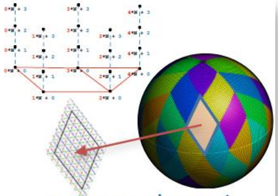


Algorithm development

```
on_edges( sum_reduction, v(), l() ) / A()
```

Domain specific language

Multidisciplinary Abstractions



memory layout, parallelisation, & data structures

OpenACC

OpenMP

NVIDIA CUDA

MPI

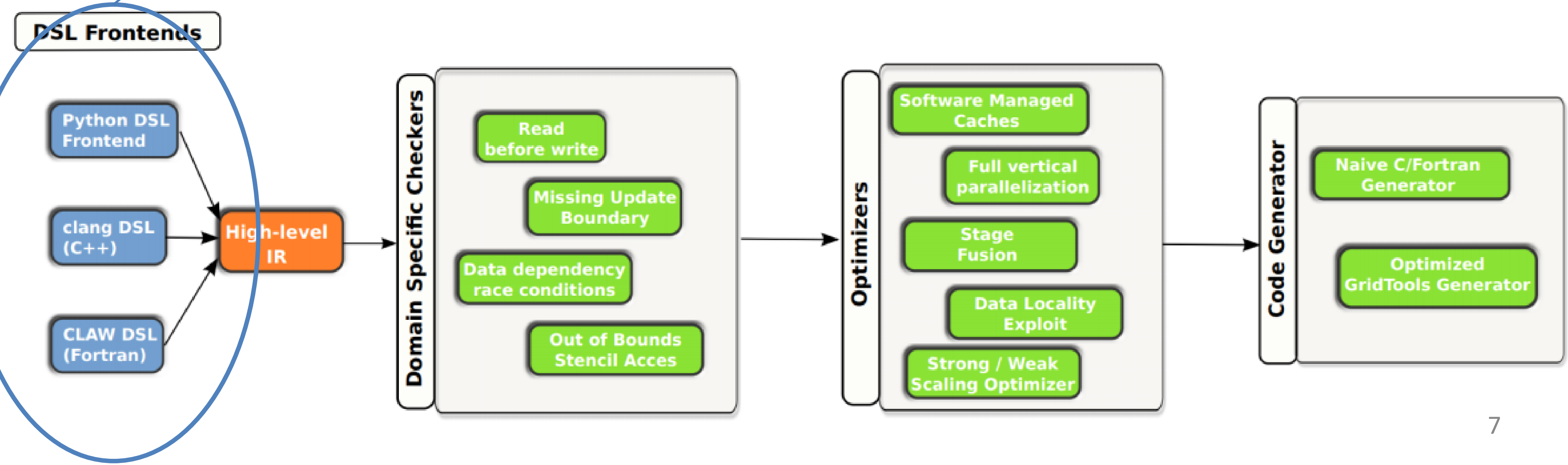
Programming models & libraries

Hardware specific instructions

```
!$ACC DATA PRESENT( u, div, fac_div, iidx, iblk), &
!$OMP PARALLEL
!$OMP DO PRIVATE(jb, i_sidx, i_eidx, jc,jk)
OMP_DEFAUL_SCHEDULE
DO jb = i_sblk, i_eblk
  CALL get_indices_c( ptr_patch, jb, i_sblk, i_eblk, &
           i_sidx, i_eidx, rl_start, rl_end)
  !$acc loop gang
  DO jk = slev, elev
  !$acc loop vector
    DO jc = i_sidx, i_eidx
      div(jc,jk,jb) = &
          u(iidx(jc,jb,1), jk, iblk(jc,jb,1)) * fac_div(jc,1,jb) + &
          u(iidx(jc,jb,2), jk, iblk(jc,jb,2)) * fac_div(jc,2,jb) + &
          u(iidx(jc,jb,3), jk, iblk(jc,jb,3)) * fac_div(jc,3,jb)
    ENDDO
  ENDDO
  !$acc end parallel
ENDDO
!$omp end do nowait
!$omp end parallel
```

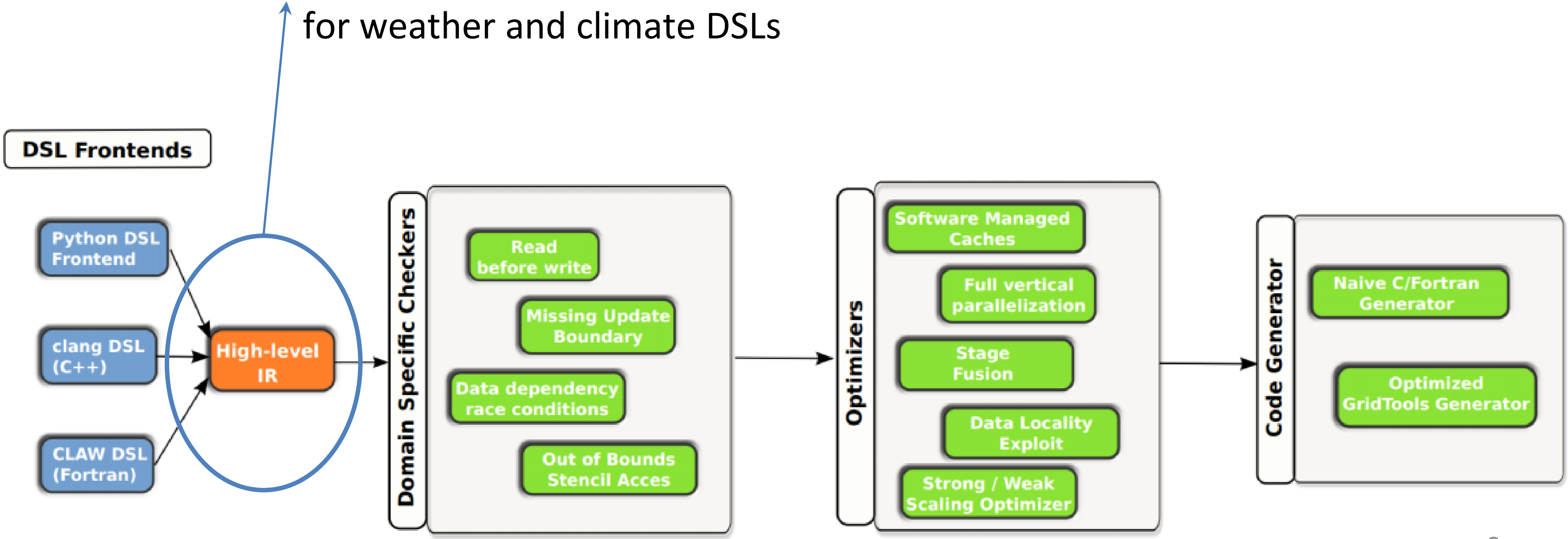
DSL Architectural Design

Task 2.2: develop DSL frontends for high scientific productivity



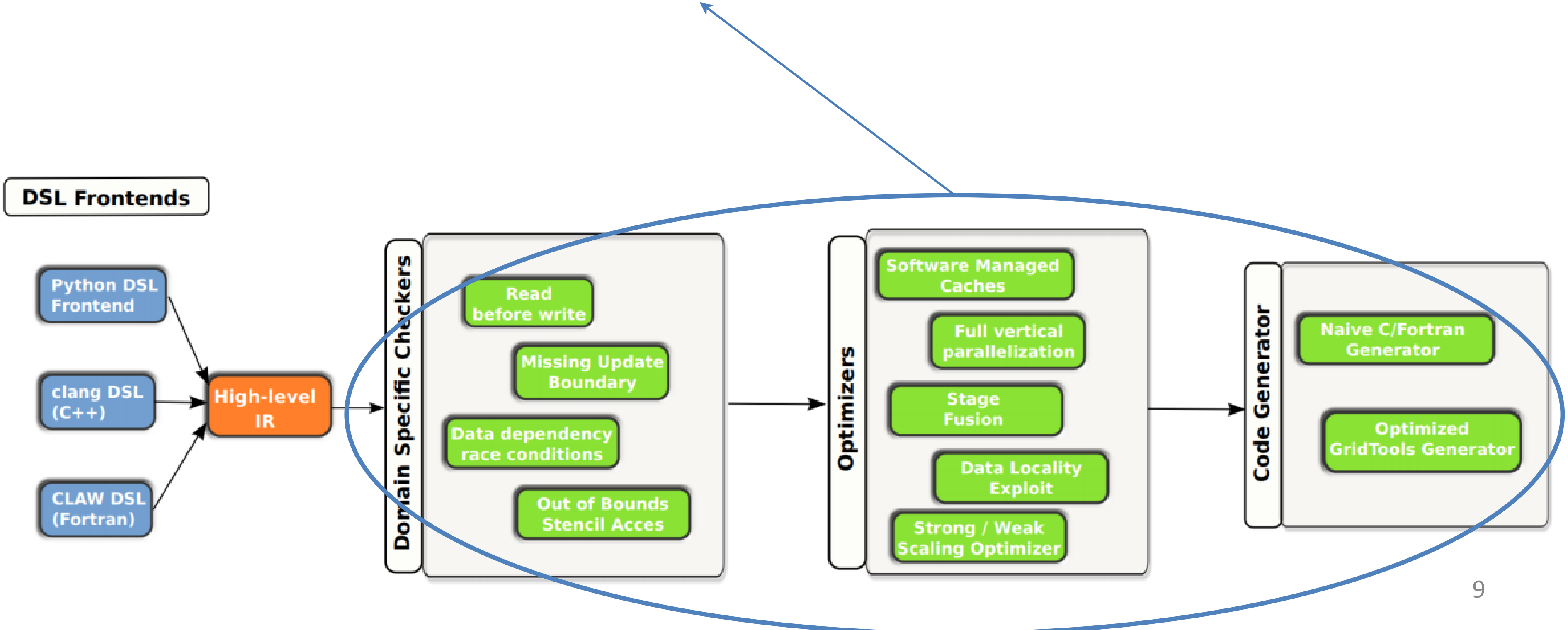
DSL Architectural Design

D2.3: Define and develop high-level intermediate representation (HIR) for weather and climate DSLs



DSL Architectural Design

Task 2.3: Implementation of the DSL toolchain



HIR

HIR

How to formalize a minimum set of orthogonal DSL concepts to capture computational patterns.

How to define a language that support our Models?

1. Start from computational patterns that appear repeatedly in our models
2. Simplify the example, removing all implementation details and optimizations

```
DO k = 1, ke
  DO i = 1, ie
    DO j = 1, 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
    DO j = 1, je
      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
```

How to define a language that support our Models?

3. Express it in pseudo-code that provides the semantics required to capture the algorithm

```
DO k = 1, ke
  DO i = 1, ie
    DO j = 1, 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
    DO j = 1, je
      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
```

→

```
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]
}
```

How to define a language that support our Models?

4. Extract sequence of language elements (D2.1)

```
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]
}
```

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 neighbour grid point (like i+1).

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

a set of integer (neighbor) offsets

LE5. sequence of AST statements: are the arithmetic computations to be performed at each grid point, and described by a full abstract syntax tree (AST)

LE6. dimension: a dimension identifier

How to define a language that support our Models?

5. Derive a full specification of a high-level intermediate specification (HIR) -> D2.3

4.11 `VerticalRegion` element

The `VerticalRegion` is the equivalent to Computation for the vertical dimension. The vertical dimension is treated specially since weather and climate codes can specialize computations for different regions of the vertical domain.

ContentsModel

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

((DimensionInterval)+,(Computation)+)

Child elements

name	description	R/O/A
`DimensionInterval`	Provides a specific range on the vertical dimension where the computations will be applied	O
`Computation`	Specifies the computation that contains the list of statements to be applied to this region	R

4.12 `Computation` element

The `Computation` defines an iteration loop over the specified GridDimension s of the domain (except for the vertical dimension, that is specified using the VerticalRegion element).

ContentsModel

((GridDimension)+,(BlockStmt))

Child elements

name	description	R/O/A
`GridDimension`	Specifies the dense dimensions where the computation is defined, convering the whole extent of the grid for that dimension	O
`BlockStmt`	Specifies the block with the list of statements that form the computation	R

For irregular grids, the GridDimension can only be specified for dense dimensions.

HIR is now complete to cover all patterns analyzed from dwarfs of

- Model category 1: Eulerian finite difference (FC) / finite volume (FV) physical parameterizations in Cartesian grids or cubed sphere
- Model category 2: FD / FV on irregular grids on the sphere

Model category 3: FE / Semilagrangian / DG might be incorporated in the future

DSL Front-end

DSL front-end: Task 2.2

- **2 Frontends developed: CDSL & dusk**

Following the spirit of modularity of the DSL toolchain that can accommodate multiple components: model specific syntax and elements, etc

CDSL

ESCAPE-2 DSL frontend (DKRZ)
Embedded in C++
AST parsed with gtclang (llvm)
Cartesian and unstructured grids
Fully supports HIR

dusk

ESiWACE-2 DSL frontend (MeteoSwiss)
Embedded in python
Specifically designed for ICON dynamical core
Supports a subset of the HIR

CDSL Example

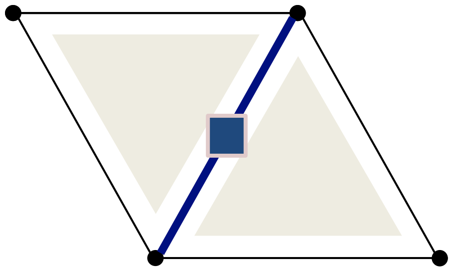
```
void e2e_via_c(EK_Field vn_e, EK_Field out_vn_e, ECEK_Field ece_op, ECK_Field ec_op, CK_Field scalar_field, EK_Field lsm_e, EK_Field
thick_edge) {
 vertical_region(start_level ,end_level) {
   compute_on(edges) {
     if (lsm_e == -2.0) {
       // reduction over neighbor cells:
       out_vn_e = vn_e * thick_edge * nreduce(cells, ec_op * scalar_field);
       // reduction over diamond edges:
       out_vn_e = out_vn_e + nreduce(cells.edges ,{0, 0, 1, 1}, vn_e * ece_op * thick_edge );
     }
   }
 }
}
```

dusk Example

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e: Field[Edge,K], ece_op: Field[Edge > Cell > Edge, K], ec_op: Field[Edge > Cell, K], lsm_e:
Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge * sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell > Edge, vn_e * ece_op * thick_edge, weights=[0, 0, 1, 1])
```

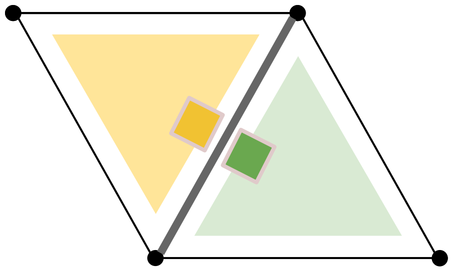
Field declarations on a staggered grid

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e: Field[Edge,K], ece_op: Field[Edge > Cell > Edge, K], ec_op: Field[Edge > Cell, K],
lsm_e: Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge * sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell > Edge, vn_e * ece_op * thick_edge, weights=[0, 0, 1, 1])
```



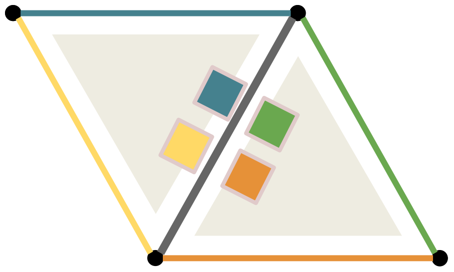
Field declaration: Sparse dimensions

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e: Field[Edge,K], ece_op: Field[Edge > Cell > Edge, K], ec_op: Field[Edge > Cell, K], lsm_e: Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge * sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell > Edge, vn_e * ece_op * thick_edge, weights=[0, 0, 1, 1])
```



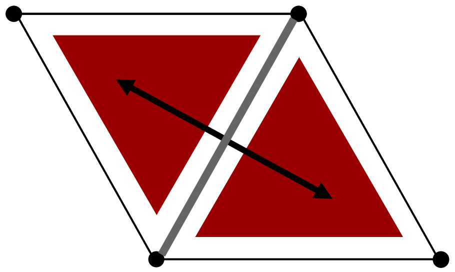
Field declaration: Sparse dimensions

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e: Field[Edge,K], ece_op: Field[Edge > Cell > Edge, K], ec_op: Field[Edge > Cell, K], lsm_e: Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge * sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell > Edge, vn_e * ece_op * thick_edge, weights=[0, 0, 1, 1])
```



neighbour reductions

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e: Field[Edge,K], ece_op: Field[Edge > Cell > Edge, K], ec_op: Field[Edge > Cell, K], lsm_e:
Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge * sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell > Edge, vn_e * ece_op * thick_edge, weights=[0, 0, 1, 1])
```



CDSL Example

```
void e2e_via_c(EK_Field vn_e, EK_Field out_vn_e,
ECEK_Field ece_op, ECK_Field ec_op, CK_Field
scalar_field, EK_Field lsm_e, EK_Field
thick_edge) {
 vertical_region(start_level ,end_level) {
   compute_on(edges) {
     if (lsm_e == -2.0) {
       // reduction over neighbor cells:
       out_vn_e = vn_e * thick_edge *
nreduce(cells, ec_op * scalar_field);
       // reduction over diamond edges:
       out_vn_e = out_vn_e + nreduce(cells.edges
,{0, 0, 1, 1}, vn_e * ece_op * thick_edge );
     }
   }
 }
}
```

dusk

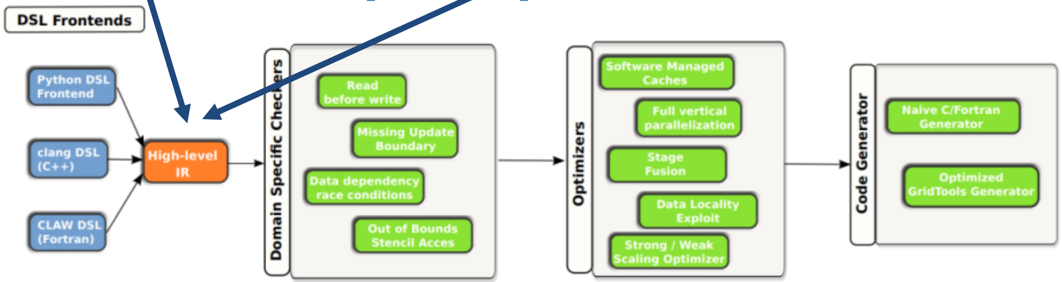
```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e:
Field[Edge,K], ece_op: Field[Edge > Cell > Edge,
K], ec_op: Field[Edge > Cell, K], lsm_e:
Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge *
sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell
> Edge, vn_e * ece_op * thick_edge, weights=[0,
0, 1, 1])
```

CDSL Example

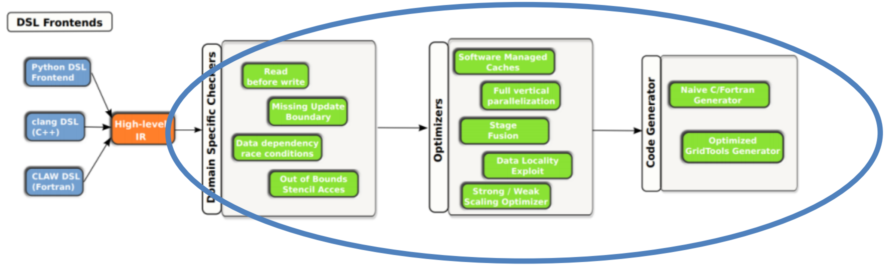
```
void e2e_via_c(EK_Field vn_e, EK_Field out_vn_e,
ECEK_Field ece_op, ECK_Field ec_op, CK_Field
scalar_field, EK_Field lsm_e, EK_Field
thick_edge) {
 vertical_region(start_level ,end_level) {
   compute_on(edges) {
     if (lsm_e == -2.0) {
       // reduction over neighbor cells:
       out_vn_e = vn_e * thick_edge *
nreduce(cells, ec_op * scalar_field);
       // reduction over diamond edges:
       out_vn_e = out_vn_e + nreduce(cells.edges
,{0, 0, 1, 1}, vn_e * ece_op * thick_edge );
     }
   }
 }
}
```

dusk

```
@stencil
def e2e_via_c(vn_e: Field[Edge, K], out_vn_e:
Field[Edge,K], ece_op: Field[Edge > Cell > Edge,
K], ec_op: Field[Edge > Cell, K], lsm_e:
Field[Edge, K], thick_edge: Field[Edge, K]):
 with domain.upward:
   if lsm_e == -2:
     # reduction over neighbor cells:
     out_vn_e = vn_e * thick_edge *
sum_over(Edge > Cell, ec_op * scalar_field)
     # reduction over diamond edges:
     out_vn_e = out_vn_e + sum_over(Edge > Cell
> Edge, vn_e * ece_op * thick_edge, weights=[0,
0, 1, 1])
```



DSL Toolchain Development

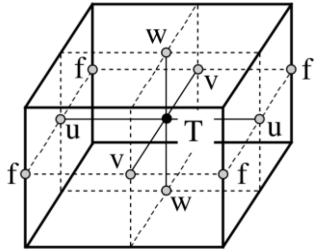


- Takes HIR as input
- Set of passes to organize computations for correct & efficient parallel implementations: fusion, inlining, split stages with synchronizations, software managed caching, etc
- Code generators:
  - C++ naïve (for debugging and reference code)
  - **Optimized CUDA for Cartesian and unstructured grids**

User Evaluation on Dwarfs

CDSL+dawn on Cartesian grids: the NEMO example

- Numerical schema: centered, second order, finite difference
- Spatial Domain discretization:
  - Homogeneous in all three space directions
  - Based on the Arakawa-C grid
  - Masks are used for land-points
  - Poles moved over land to avoid singularities
  - Tri-polar grid is used



- T: scalar points
- u, v, w: vector points
- f: vorticity points

Use of DSL code for advection

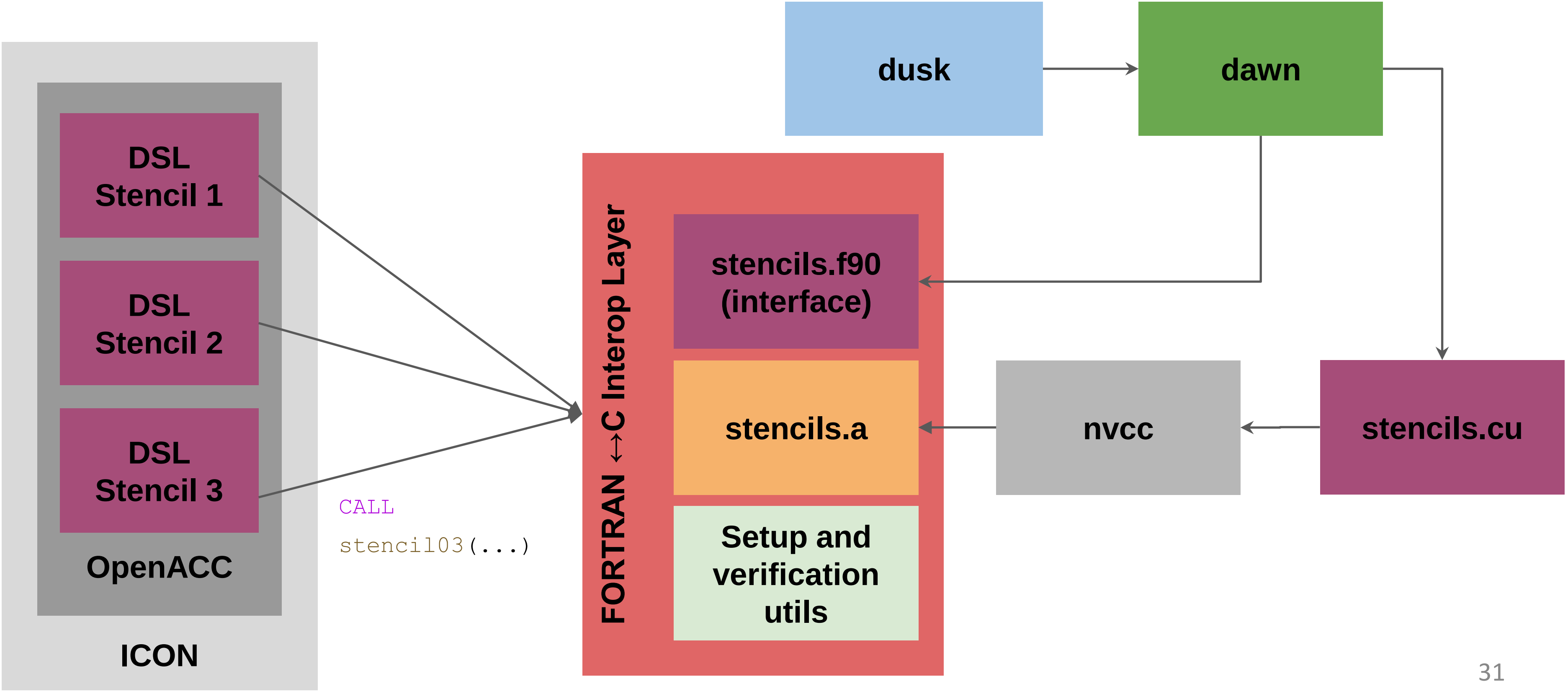
ESCAPE 2

```
DO jk = 1, jpkm1
 DO jj = 1, jpjm1
  DO ji = 1, fs_jpim1
   zwx(ji,jj,jk) = umask(ji,jj,jk) * ( ptb(ji+1,jj,jk,jn) - ptb(ji,jj,jk,jn) )
  END DO
 END DO
END DO

DO jk = 1, jpkm1              !-- Slopes
 DO jj = 2, jpj-1
    DO ji = 2, jpi-1
    zslpx(ji,jj,jk) = zwx(ji,jj,jk) + zwx(ji-1,jj,jk)
    END DO
 END DO
END DO

DO jk = 1, jpkm1              !-- Horizontal advective fluxes
 DO jj = 2, jpj-2
    DO ji = 2, jpi-2
    zu  = pun(ji,jj,jk) / ( e1u(ji,jj) * e2u(ji,jj) * fse3u(ji,jj,jk) )
    zflux(ji,jj,jk) = pun(ji,jj,jk) * ( ptb(ji+1,jj,jk,jn) + zu * zslpx(ji+1,jj,jk) )
    END DO
 END DO
END DO

DO jk = 1, jpkm1              !-- Tracer advective trend
 DO jj = 3, jpj-2
    DO ji = 3, jpi-2
    zu = 1. / ( e1t(ji,jj) * e2t(ji,jj) * fse3t(ji,jj,jk) )
    pta(ji,jj,jk,jn) = pta(ji,jj,jk,jn) - zu * ( zflux(ji,jj,jk) - zflux(ji-1,jj,jk) )
  END DO
 END DO
END DO
```

```
stencil advection_MUSCL {
  do {
    vertical_region (k_start, k_end - 1) {
      zwx = u_mask * (ptb(i+1) - ptb);
    }
      //-- Slopes of tracer
    vertical_region (k_start, k_end - 1)
      zslpx = zwx + zwx(i-1);
    }
      //-- Horizontal advective fluxes
    vertical_region (k_start, k_end - 1) {
      zu = pun / (e1u * e2u * fse3u);
      zflux = pun * (ptb(i+1) + zu * zslpx(i+1));
    }
      // Tracer advective trend
    vertical_region (k_start, k_end - 1) {
      zu = 1.0 / (e1t * e2t * fse3t)
      pta = pta - zu * (zflux - zflux(i-1));
    }
  }
}
```

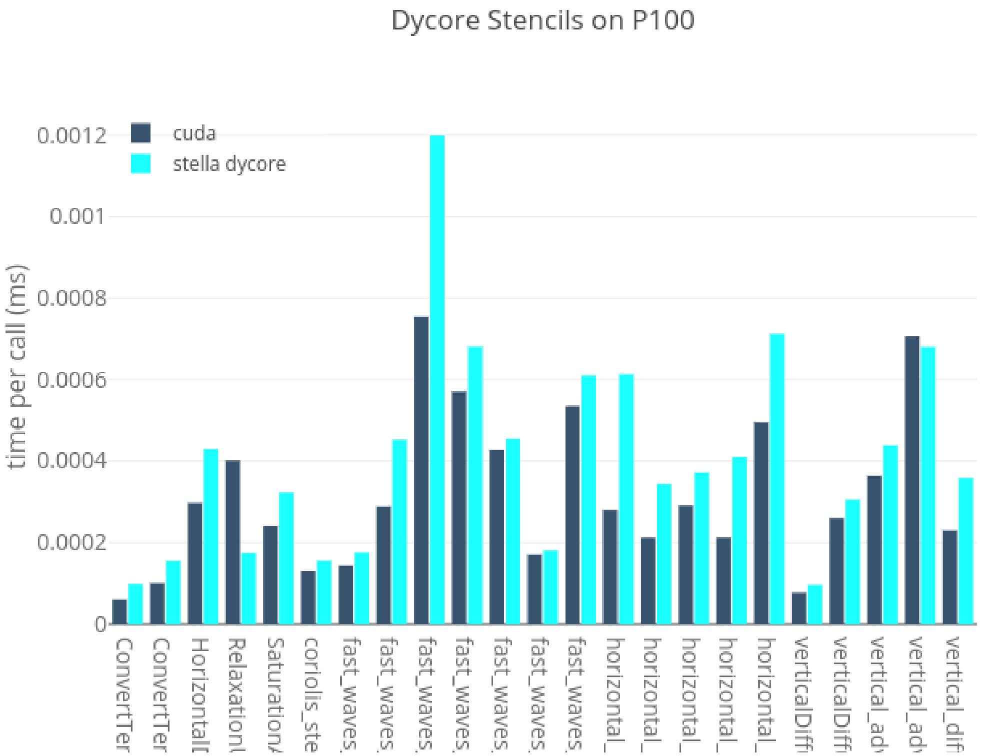
Interoperability and Integration into models

(see Florian Ziemen's slides)



Performance

cosmo dycore comparison of DSL toolchain vs GPU production



See Florian Ziemen's talk for results on ICON dycore

Final Considerations

- ESCAPE-2 was based on experiences and developments from various long term efforts in DSL developments: ESCAPE, GridTools, and PASC projects.
- Successfully established DSL concepts and language elements for high-level DSL to capture motifs of weather models.
- Provides for the first time a full high-level DSL toolchain, with various frontends and demonstrate on dwarfs.
- ESiWACE-2: Future establishing DSL on the community (see Florian Ziemen's talk)
- Gt4py and EXCLAIM developments are based on ESCAPE-2 DSL and will continue developing and evolving this work.

Many thanks to the entire WP2 team for all the great work and the fantastic journey!

Reinhard Budich (MPI)

Max-Planck-Institut für Meteorologie

Joerg Behrens (DKRZ)

DKRZ DEUTSCHES KLIMARECHENZENTRUM

Giacomo Serafini (MeteoSwiss)

ECMWF

Christoph Müller (MeteoSwiss)

Matthias Röthlin(MeteoSwiss)

Willem Deconinck (ECMWF)

MeteoSwiss

Tobias Wicky (MeteoSwiss)

Benjamin Weber (MeteoSwiss)

cmcc

Italo Epicoco (CMCC)

https://git.ecmwf.int/projects/ESCAPE/repos/cpp-dsl-front-end/browse
https://www.github.com/MeteoSwiss-APN/dawn.git
https://www.github.com/MeteoSwiss-APN/HIR.git