D2.4: Demonstration of Domain Specific Language Toolchain for Selected Weather and Climate Dwarfs

Dissemination Level: Public

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800987.

www.hpc-escape2.eu

Funded by the European Union

Co-ordinated by ECMWF

Energy-efficient Scalable Algorithms for Weather and Climate Prediction at Exascale

Authors: Jörg Behrens, Carlos Osuna, Julia Duras, Italo Epicoco, Reinhard Budich, Peter Korn

Date **30. August 2021**

Research and Innovation Action

H2020-FETHPC-2017

Project Coordinator: Dr. Peter Bauer (ECMWF)

Project Start Date: 01/10/2018

Project Duration: 36 month

Published by the ESCAPE-2 Consortium

Version: 1.0

Contractual Delivery Date: Sep 2020

Work Package/ Task: WP2

Document Owner: Reinhard Budich

Contributors: Jörg Behrens, Carlos Osuna, Julia Duras, Italo Epicoco, Peter Korn

Status: Final

Contents

1 Executive Summary

This deliverable describes the programming and implementation of two parts of Earth System model codes, called dwarfs, in variants of a Domain Specific Language (DSL), which exercise structured and unstructured grids. These implementations can then be fed into the DSL-toolchain as it has been developed in ESCAPE-2, resulting in back-end-optimized code. The dwarfs generated fulfill the following points:

- The dwarfs in the complete toolchain can be executed on both CPUs and GPUs with CUDA support.
- They are available for download in the project repository [1] [2].
- They will be fed into work package 3 for further benchmarking and comparison with the originals.

The deliverable was delayed for several reasons, which are described as well as the countermeasures taken. Next steps will now be the interaction with work package 3 and to check and describe lessons learned for functionality (deliverable D2.5) and usability (deliverable D2.6) of the toolchain.

Concluding, it can be stated that the concepts developed in work package 2 have been applied successfully, and that DSL dwarf formulations are now available in a usable shape.

2 Introduction

2.1 Background

ESCAPE stands for *Energy-efficient SCalable Algorithms for weather Prediction at Exascale*. ESCAPE-2 continues what ESCAPE-1 started: develop world-class, extreme-scale computing capabilities for European operational numerical weather and climate prediction systems. A comprehensive overview of the project can be found at the project website `http://www.hpc-escape2.eu`.

One way to strive for enhanced computational performance is to develop generic programming approaches that ensure code portability and performance portability. In this context, Domain Specific Languages (DSLs) play an increasing role. Work package (WP) 2 of ESCAPE-2 *Programming Models and Domain-Specific Languages* i.a. has the objective to *„Demonstrate code adaptation and code generation via the DSL toolchain for a number of representative and fundamentally different mathematical algorithms and horizontal discretizations.“* (Bauer, 2018). The WP covered the requirements for a DSL, including a first definition of a DSL; it developed a front-end that parses a domain-scientist readable DSL. The front-end translates DSL code into a High-level Intermediate Representation (HIR). There now is a DSL toolchain developed in the WP.

[1] git clone `ssh://git@git.ecmwf.int/escape/dwarf-d-advection-muscl.git`
[2] git clone `ssh://git@git.ecmwf.int/escape/dwarf-d-icon-ocean.git`

With this report, we will cover the demonstration of the DSL with weather and climate prediction dwarfs, which were selected by the project for this WP.

2.2 Scope of this Deliverable

2.2.1 Objective of this Deliverable

Work package 2 *Programming Models and Domain Specific Languages* is aiming to define, develop, and apply a DSL toolchain to enhance code portability and performance portability.

Task 2.4 of this WP demonstrates the application of the DSL to weather and climate model dwarfs. It is lead by MPI-M, partners are MSWISS, ECMWF, CMCC, DKRZ, BSC, and RMI. *"This task will demonstrate and verify the usability of the newly developed DSL toolchain for code generation. The task will translate a specified set of representative dwarfs delivered by WP1 to the new high-level DSL and apply the DSL toolchain to generate code for multiple hardware targets (D2.4, MPI-M). These dwarfs are tested and verified against their original versions and subsequently supplied to WP3 for re-integration (Task 3.3) and benchmarking"* (Bauer, 2018).

This work is based on other deliverables of WP2:

- D2.1 (Osuna et al., 2019a) defined the functionality and structure of the DSL language elements and examined domain examples in the form of DSL pseudo-code. Important aspects are the unified approach for different grids, the index-free domain iteration, and the representation of fundamental grid operators, e.g., reductions. These abstractions are the fundamental building blocks of the ESCAPE-2 DSL that are used in D2.2 – D2.6.

- D2.2 (Behrens et al., 2019) provides the description and implementation of the ESCAPE-2 front-end (CDSL). The solution constitutes a concrete realization of the D2.1 pseudo-code DSL using a C++ embedded approach. CDSL has been continuously refined and is used to formulate the DSL representation of dwarf kernels shown in this deliverable.

- D2.3 (Osuna et al., 2019b) provides a specification document for the central API structure (HIR) that connects front- and back-end within the toolchain scheme. An extension of the HIR has been implemented as one of the transformation stages within CDSL and therefore D2.3 contributes to the toolchain application presented here.

- D2.5 will deliver the implementation of the ESCAPE-2 toolchain. Although D2.5 is scheduled for a later time, the implementation of the toolchain is already advanced enough to translate most DSL formulations considered here. Additionally, for one kernel of ICON-O, we used an experimental development branch of the back-end. This enables us to compile and execute the DSL dwarfs.

2.2.2 Work Performed in this Deliverable

To prove existing concepts of the DSL toolchain, two dwarfs extracted from highly representative weather and climate prediction models had been chosen for this deliverable: the so called *Dwarf-advection-MUS*, which is extracted from the NEMO ocean model (section 3.1), and the so called *ICON Ocean transport dwarf* (section 3.2) , which is part of the ICON ocean model. They have been chosen since they compute on regular - the NEMO dwarf - and on irregular grids - the ICON-O dwarf.

These dwarfs differ fundamentally in their stencil complexity and representation: On one hand, in the regular (NEMO) case, a local operator uses few translation-invariant stencil weights which are applied to grid fields via directional offsets. On the other hand, the irregular (ICON) case requires a full-domain sparse stencil field which is applied to grid fields via connectivity maps. These connectivity maps define the transition from the sparse dimension (which carries the stencil weights) to the domain elements of a certain location type (which carry physical state data). The challenge for the DSL is to provide abstract index-free DSL concepts to the user without losing the expressiveness required by the scientific domain. The selected unstructured dwarf assesses the applicability of these concepts.

Both dwarfs are considered to be integrated into the High Performance Climate and Weather (HPCW) benchmark suit, which is task 3.5 of the project.

For all dwarfs we have used the CDSL front-end and the Dawn back-end which constitute the current ESCAPE-2 toolchain, see section 4. For the NEMO dwarf we have two DSL implementations, using CDSL and GTClang.

2.2.3 Deviations and Counter Measures

The application of the DSL toolchain was delayed because of major obstacles:

- For unstructured grids front- and back-end development had to be partially redesigned and re-implemented at the point where the concept of stencil fields with multiple simple sparse dimensions defined in the DSL specification (Osuna et al., 2019a) was no longer considered to be efficient enough. A new maximal compact stencil representation that aggregates the target locations into a non-redundant set was chosen. The increased compactness reduces the numerical load at the cost of higher order connectivity maps. The new formulation is somewhat less general, and for at least one use case the DSL had to provide additional functionality in order to express the algorithm. The situation is still in an experimental state and one of the ICON-O kernels presented here gives a tentative evaluation of the approach. For the selected ICON-O dwarf the new DSL stencil concept required a reformulation of model internal operator weights.

- The ICON ocean dwarf does not fit into the dwarf concept that reduces complexity by extraction, due to the complexity of the ICON code structure. The work required to extract the selected functionality from ICON was considered to be beyond the means of this project. Instead, the selective aspect of the ICON-O dwarf was realized by the special run-path of the advection testbed in the ICON script environment. It was assumed that this limitation would provide enough focus to reduce the complexity on par to the original dwarf concept. However, the application of the DSL toolchain also involves the rather complex ICON grid infrastructure and the full model build procedure. This model-integration work was originally assigned to a different project task (task 3.3) but now had to be, and was, solved in this task.

- The unstructured part of the back-end was developed using atmospheric test-problems. It turned out that these tests did not cover the scope of ocean specific problems studied in the ICON-O dwarf. Therefore additional time has been spent on the analysis and correction of ocean-specific errors.
- Two of our co-workers were out of business due to health reasons for a few months in the time 2021/22.

Counter measures:

- In order to limit the delay the extent of the ICON-O dwarf functionality targeted for a re-implementation in CDSL has been reduced. This reduction has been partially compensated by adding ICON-grid related synthetic Fortran tests directly into the front-end, where they also serve as integration tests of front- and back-end. In summary, we still think that we can make a reasonable assessment of the applicability of the DSL toolchain.
- The additional work performed on integration aspects for the full ICON model greatly simplifies the generation of a HPCW benchmark version with DSL support. Therefore, we do not expect a limitation of project results by the rearrangement of work among tasks. Counter measures beyond the scope of this deliverable seem unnecessary.

3 Dwarfs Selected

For the project, some dwarfs have been identified to be subject to the application of DSLs. They seemed to be representative enough for climate and weather applications, and the data structures and algorithms used in typical cases.

3.1 The NEMO Dwarf *Dwarf-advection-MUS*

The *Dwarf-advection-MUS* implements and solves the tracer advection equation. It has been extracted from the NEMO4.0 (Madec and NEMO-ST, 2019) which is a European community model framework used for a wide range of applications, both regional or global, as a forced ocean model or as a model coupled with the sea-ice and/or the atmosphere.

The numerical techniques used to solve the Primitive Equations in the NEMO model are based on the traditional, centred second order finite difference approximation. Special attention has been given to the homogeneity of the solution in the three spatial directions. The arrangement of variables is the same in all directions. The spatial grid is the generalisation to three dimensions of the well-known "C" grid in Arakawa's classification. The discretised mesh is a uniform mesh with a grid size of unity. Discrete partial derivatives are formulated by the traditional, centred second order finite difference approximation while the scale factors are chosen equal to their local analytical value.

Several semi-discrete space forms of the tracer equations are available in NEMO depending on the vertical coordinates and on the physics used. This dwarf implements the Monotone Upstream Scheme for Conservative Laws (MUSCL), where the tracer at velocity points is evaluated assuming a linear tracer variation between two T-points. The MUSCL scheme for the tracer advection has been implemented by Lévy et al. (2001) . In this formulation, the tracer τ at velocity points is evaluated assuming a linear tracer variation between two T-points. For example, in the i-direction (which represents the longitude):

$$\tau_u^{mus} = \begin{cases} \tau_i = +\frac{1}{2}(1 - \frac{u_{i+1/2}\Delta t}{e_{1u}})\partial_i \tilde{\tau}, \quad if\ u_{i+1/2} \geq 0 \\ \tau_{i+1/2} = +\frac{1}{2}(1 + \frac{u_{i+1/2}\Delta t}{e_{1u}})\partial_{i+1/2} \tilde{\tau},\ otherwise \end{cases} \tag{1}$$

where $\partial_i \tilde{\tau}$ is the slope of the tracer on which a limitation is imposed to ensure the positive character of the scheme. The time stepping is performed using a forward scheme, that is the *before* tracer field is used to evaluate τ_u^{mus}.

For an ocean grid point adjacent to land and where the ocean velocity is directed toward land, two choices are available: an upstream flux or a second order flux. Note that the latter choice does not ensure the *positive* character of the scheme. Only the former, which is the actual flux selected in this dwarf, can be used on both active and passive tracers. The MUSCL scheme doesn't require an explicit diffusion operator; in fact it is diffusive enough so that it doesn't require additional diffusion, like other advection schemes.

This dwarf has been selected for DSL because it includes a traditional and uniform stencil on a regular mesh as highlighted in the following snippet of the Fortran code Listing 1. More details are available in the ESCAPE-2 repository[3].

[3]`https://git.ecmwf.int/projects/ESCAPE/repos/dwarf-d-advection-muscl/browse`

```
! computation of tracer slop
DO jk = 1, jpk-1; DO jj = 1, jpj-1; DO ji = 1, jpi-1
   zwx(ji,jj,jk) = umask(ji,jj,jk)*(ptb(ji+1,jj,jk)-ptb(ji,jj,jk))
END DO; END DO; END DO

CALL halo_exch(zwx)

DO jk = 1, jpk-1; DO jj = 1, jpj; DO ji = 2, jpi
   zslpx(ji,jj,jk) = (zwx(ji,jj,jk) + zwx(ji-1,jj,jk)) * 0.5
END DO; END DO; END DO

!advective fluxes
DO jk = 1, jpk-1;    DO jj = 1, jpj-1; DO ji = 1, jpi-1
   zzwx = ptb(ji+1,jj,jk,jn) + k * zslpx(ji+1,jj,jk)
   zzwy = ptb(ji  ,jj,jk,jn) + k * zslpx(ji  ,jj,jk)
   zflx(ji,jj,jk) = pun(ji,jj,jk) * ( s * zzwx + (1 - s) * zzwy )
END DO; END DO; END DO
```

Listing 1: stencil for advective fluxes in the MUSCL advection scheme

3.2 The *ICON-O Transport Dwarf*

The ICON Ocean Model (ICON-O) was developed at MPI-M over the last 10 years, for a reference see Korn (2017). The model is used in many different configurations, coupled and stand-alone, with a certain emphasis on high resolution and regionally focusing implementations. It contains a sea ice model and has advanced conservation properties.

The ICON-O model uses triangular cells where the variables are placed following the Arakawa C-type staggering (see Korn (2017) for a more complete description); variables such as temperature or pressure are located at the centre of a triangular cell, while the normal component of the velocity vector is positioned at the midpoint of a triangle edge. This choice of grid geometry and variable placement implies the existence of a computational mode. A computational cure to control this computational mode has been introduced. A result of major importance is that the new computational method for the solution of the dynamic equations of the ocean controlling this mode is compatible with the conservation laws (Korn, 2017; Korn and Danilov, 2017) and that it does not affect the wave propagation properties (Korn and Linardakis, 2018). The numerical method has also been extended to ocean parametrizations (Korn, 2018), and, more recently to generalized vertical coordinates (Singh and Korn, in preparation), where the z^* coordinates is of primary importance.

The ICON-O model has been chosen to demonstrate the DSL because its numerics and algorithms were sufficiently representative for the numerical solutions of similar equations in the field, and because it was the only unstructured ocean model owned by one of the project partners. The dwarf selected represents a transport equation in three dimensions, so we will call it the *ICON-O transport dwarf* below. It uses predefined diffusion coefficients as well as initial temperature and velocity fields of the sea water, and computes the time evolution of the sea temperature field on a so called *aqua-planet*, a test configuration of a planet without any land.

As opposed to the ideas at the beginning of the ESCAPE-2 project, it turned out to be easier in the case of ICON to use and configure the complete model framework for the building, compilation and execution for only this transport part, than ripping the framework apart and provide an environment and boundary data for this isolated case of a transport *dwarf*. Therefore, the ICON-O transport dwarf is incorporated within a testbed environment and can be called separately. This is why it might rather be called a test scenario than a dwarf. The dwarf itself calls general ICON and ICON Ocean routines. This allows, on the one hand, a more realistic computing example. But, on the other hand, the DSL implementation has to deal with the whole complexity of ICON and its infrastructure. This approach still seems to be preferable compared to the elaborate work required for ICON code extraction.

Its repository can be found at the ESCAPE-2 Software Collaboration Platform[4].

3.2.1 Kernels Chosen to Demonstrate the DSL Toolchain

In the following the term *kernel* is used to describe a functional unit that solves a given problem in the model - usually a Fortran subroutine. The focus here is on the algorithmic aspect. Currently two advection subproblems are targeted:

- **velocity prediction**
  The subroutine `compute_time_weighted_normal_velocity` calculates the predicted normal velocity vector v_n^{diag} from prognostic values v_n^{prog} at different time levels using the Adams-Bashforth semi-implicit time stepping parameter γ.

  $$v_n^{diag} = \gamma \cdot v_n^{prog[t_{new}]} + (1 - \gamma) \cdot v_n^{prog[t_{old}]} \tag{2}$$

  This mixing of 3D fields using scalar weight factors is done for each edge of the computational domain. The operation performs basic elemental operations without involving the grid specific connectivity. The simplicity should carry over to the DSL formulation and also allows to inspect the correctness of the basic toolchain functionality.

  The implementation within the ICON ocean code can be seen in the code snippet Listing 2. Since ICON uses a blocked data layout, e.g., (nproma, nlev, nblocks) for 3d state fields, the loop structure shows the typical blocked horizontal (loop over `blockNo` and `je`) and depth limited vertical iteration (loop over `jk`) of ICON-O.

[4]`https://git.ecmwf.int/projects/ESCAPE/repos/dwarf-d-icon-ocean/browse`

```fortran
! horizontal iteration
DO blockNo = subset_range%start_block, subset_range%end_block
  CALL get_index_range(subset_range,blockNo,start_edge_index,end_edge_index)
  DO je = start_edge_index, end_edge_index
  ! vertical iteration
    DO jk = 1, dolic_e(je,blockNo)
      vn_time_weighted(je,jk,blockNo) = ab_gam*vn_new(je,jk,blockNo) + (1.0_wp - ab_gam)*vn_old(je,jk,blockNo)
     END DO
  END DO
END DO
```

Listing 2: velocity prediction kernel

- **mapping edges to edges via cells**
  In a second step the subroutine `map_edges2edges_viacell`[5] (see Listing 3) with non-trivial horizontal dependencies has been evaluated. The kernel constitutes one of the central algorithmic elements of ICON-O. It calculates the volume fluxes and implements the filter of divergence noise (see operator $P^T P$ of Korn (2017) for details). An additional scalar tracer field `scalar` makes it applicable for tracer transport. Since the fluxes `vn_e` are located at midpoints of triangle edges, the operator region encloses two nearest neighbor shells around the stencil center: edge $\rightarrow$ cells $\rightarrow$ edges, resulting in 2 x 3 operator coefficients `edge2edge_viacell_coeff` on edges for triangular grids (see Figure 1). These coefficients are combined with normal velocities `vn_e` and geometric weights `thick_edge` on edges and the scalar tracer field `scalar` on cells in a twofold reduction operation that produces the output velocity field on the central edge. The reduction is hierarchical: a reduction on each of the two adjacent triangles (line 8-23 and line 24-39) is followed by a summation of the triangle results (line 41).

  This evaluation order defines the relation between scalar tracer field and operator coefficients and prevents the constant aggregation of the two operator coefficients that belong to the central edge. In other words, the usage of the scalar tracer field creates a data dependency between the edge locations and the iteration path that led to these locations. A simple aggregation scheme that tries to remove the redundancy of edge locations without regard for the path information would be generally invalid. In the model the path information is represented by two sparse operator dimensions stacked in a flat one-dimensional sparse data layout iterated by `ictr`.

[5]The full subroutine name is `map_edges2edges_viacell_3d_mlev_constZs`.

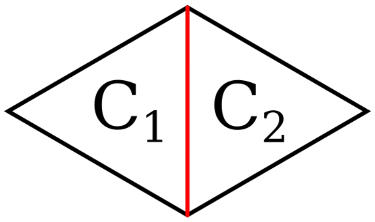

Figure 1: Two neighbouring triangular cells C_1 and C_2 within the ICON-O grid. In red, the central edge is indicated, where the volume flux `out_vn_e` to be computed is located.

```fortran
DO blockNo = start_block, end_block
  CALL get_index_range(edges_inDomain,blockNo,start_edge_index
    ,end_edge_index)
  ! vertical iteration
  level_loop: DO level = startLevel, endLevel
    ! loop over edges
    edge_idx_loop: DO je = start_edge_index, end_edge_index
      IF (patch_3d%lsm_e(je,level,blockNo) == sea) THEN
        ! neighbour cell C1 of the current edge
        ictr        = 0
        il_c        = patch_2d%edges%cell_idx(je,blockNo,1)
        ib_c        = patch_2d%edges%cell_blk(je,blockNo,1)
        scalar_cell = scalar(il_c,level,ib_c)
        t1 = 0
        DO ie = 1, no_primal_edges ! loop over edges of C1
          ictr       = ictr+1
          il_e       = patch_2d%cells%edge_idx(il_c,ib_c,ie)
          ib_e       = patch_2d%cells%edge_blk(il_c,ib_c,ie)
          thick_edge = prism_thick_e(il_e,level,ib_e)
          t1 = t1 &
        & + vn_e(il_e,level,ib_e) &
        & * edge2edge_viacell_coeff(je,level,blockNo,ictr) &
        & * thick_edge * scalar_cell
        END DO
        ! neighbour cell C2 of the current edge
        ictr        = no_primal_edges
        il_c        = patch_2d%edges%cell_idx(je,blockNo,2)
        ib_c        = patch_2d%edges%cell_blk(je,blockNo,2)
        scalar_cell = scalar(il_c,level,ib_c)
        t2 = 0
        DO ie = 1, no_primal_edges ! loop over edges of C2
          ictr       = ictr+1
          il_e       = patch_2d%cells%edge_idx(il_c,ib_c,ie)
          ib_e       = patch_2d%cells%edge_blk(il_c,ib_c,ie)
          thick_edge = prism_thick_e(il_e,level,ib_e)
          t2 = t2 &
        & + vn_e(il_e,level,ib_e) &
        &   * edge2edge_viacell_coeff(je,level,blockNo,ictr) &
        &   * thick_edge * scalar_cell
        END DO
        ! combining results from C1 and C2
        out_vn_e(je,level,blockNo) = t1 + t2
      ENDIF
    END DO edge_idx_loop
  END DO level_loop
END DO
```

Listing 3: Subroutine `map_edges2edges_viacell_3d_mlev_constZs`; Some identifiers have been shortened.

4 The Toolchain

4.1 The Concept

Since the end of the Dennard scaling (Frank et al. (2001)) era, technical progress has led to the adoption of new computer accelerators and computing architectures, hybridization and diversification of supercomputers.

The DSL toolchain aims at providing a solution to the portability, performance portability and maintainability problems of complex weather and climate models targeting multiple parallel and heterogeneous architectures. It achieves this by providing a high-level DSL language specific for weather and climate patterns. A comprehensive set of language elements that covers all the computational patterns of the weather and climate dwarfs explored in ESCAPE-2 was specified in the deliverable D2.1 (Osuna et al. (2019a)).

The design of the toolchain is depicted in Figure 2. A central component of the modular design is the high-level intermediate representation (HIR) which allows to communicate multiple DSL front-ends with the main toolchain compiler (Dawn). The HIR is a language agnostic representation of all the language concepts defined in ESCAPE-2 (Osuna et al. (2019a)). The large diversity of computational patterns, derived from the use of different grid types or discretization methods, identified within ESCAPE-2, advised against the limitations posed by a single DSL front-end. Like that, different DSL front-ends can provide model or grid specific syntax and, such, rely on the compiler toolchain by generating and providing HIR information to Dawn.

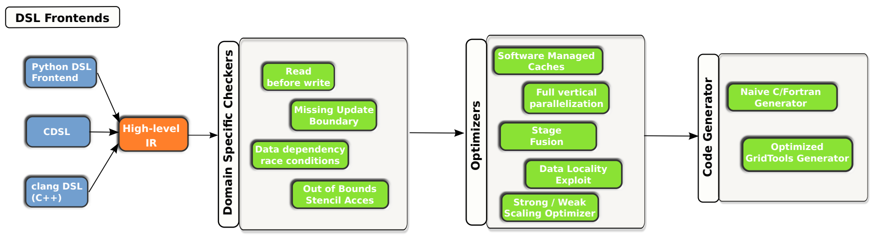

Figure 2: Design of the ESCAPE-2 DSL toolchain.

4.2 The Back-end Dawn

Dawn is the compiler toolchain that takes a HIR representation associated to the user equations and generates efficient code for any compiler architecture. Structured and unstructured grids are supported. Dawn adopts a modular design similar to modern compilers like LLVM. As shown in Figure 2, Dawn incorporates a set of compiler passes that gradually transforms the intermediate representation, applying step by step transformations, reorganizing statements or applying specific optimization techniques. Since the HIR is mainly a sequential specification for domain specific computations of weather and climate applications, a first set of passes (parallel IR builders) of Dawn transforms the HIR into a parallel IR that adheres to a parallel model of execution. In this transformation, it ensures the execution of the user equations will be safe and valid in parallel environment. Example of this type of passes are field versioning, required whenever a field is read and written to within the same parallel stage and stage splitting, that determines based on data dependencies whether statements need to be split into different stages that require synchronization of parallel computing elements. After the parallel IR builders, a set of optimization passes are ran, which aim at further applying optimization techniques that will improve the efficiency of the generated code. Examples of the optimization passes are: *stage reordering*, which reorders stages in order to group those connected with data dependencies in order increase data locality, *Stencil inlining* that perform a lazy evaluation of any temporary stencil values at the grid point whenever accessed by other stencil computations.

The final step of the toolchain is the code generators which generate efficient code that can be compiled by a general purpose compiler (C++ compilers or CUDA nvcc compiler). Further details can be found at Osuna et al. (2020).

4.3 The Front-end GTCLang

GTClang is an existing front-end for Dawn that provides a high-level descriptive language for expressing finite difference/volume methods on structured grids. It does not support unstructured grids. Its applicability for structured weather and climate codes was demonstrated for the dynamical core of COSMO (Osuna et al. (2020)). The implementation approach is a C++ embedded DSL that uses Clang to generate the AST (abstract syntax tree). It ties DSL semantic to predefined identifiers, e.g., i, j and k which carry a directional meaning in a Cartesian coordinate system. A more general approach is taken by the 2nd front-end of this deliverable, the CDSL.

4.4 The Front-end CDSL

The CDSL (Community Domain Specific Language) front-end constitutes the application entry point for the ESCAPE2 toolchain. It provides the user with the ability to express a broad range of algorithms of the climate and weather domain in a simple C++ embedded language.

The front-end parses, analyses and transforms the DSL source code into a form that allows further processing by the toolchain back-end. As a result of the full toolchain processing the user obtains a new source code in a standard language (C++ or CUDA), adapted to the chosen hardware. A semantic analysis allows the front-end to give sensible error messages in case of ill-formed DSL source code. In addition to DSL source code parsing and high level semantic analysis, the front-end also generates object oriented Fortran interface code that provides a convenient access to the kernel functionality. This simplifies integration of DSL formulated sub-problems into domain models which are usually written in Fortran.

The front-end implementation uses the Python bindings to libclang to parse the C++ host language. Subsequent processing of the AST is done in Python. Additional runtime support for Fortran is written in C++ and Fortran.

The programmability of CDSL benefits from the strong typing support of the C++ host language which allows to hide implicit semantics into types. Although the user is restricted by the DSL concepts for good reasons, there exists the possibility to "escape the DSL" by adding non-DSL C++ code in a special namespace. However, this only generalizes the elemental grid point computation - not the domain iteration.

CDSL implements the specification given in “D2.1: High-level Domain Specific Language (DSL) specification” (Osuna et al., 2019a). Parser and language dialect have been described in “D2.2 DSL front-end to parse DSL into high-level intermediate representation (HIR)” (Behrens et al., 2019). The interface between front-end and back-end has been defined in the document “D2.3: High-level Intermediate Representation (HIR) Specification” (Osuna et al., 2019b). In practice, however, there currently exist two variants of the high level intermediate representation concept. One variant closely follows the specification document D2.3 using a regular expression style to describe the content of language elements. Another one, also called SIR ("Stencil Intermediate Representation"), uses a more structured form. Both representations share the same abstraction level and can function as central toolchain representation. The CDSL front-end can produce both variants by deriving the SIR form from the HIR form, using suitable utilities provided by Dawn.

5 Results

5.1 GTClang Applied to the Dwarf-advection-MUS

The *Dwarf-advection-MUS* has been implemented with the GTClang front-end using the optimization toolchain based on Dawn. GTClang nativly supports finite difference stencils which perfectly matches with the requirements of the *Dwarf-advection-MUS*. After having mastered the use of the internal "concepts" of GTClang, the implementation of the dwarf was a trivial translation of the advection algorithm into GTClang language (as illustrated in the following snippet of code Listing 4). Nevertheless, the halo exchange using MPI is not supported by GTClang and this required a work-around to have a complete implementation of the dwarf. We have extended the size of the halo region to two lines and this allows to move the halo exchange after the advection scheme, hence we have implemented the advection kernel in GTClang and a Fortran interface of it has been included in the main program where the halo exchange is executed after the invocation of the advection kernel.

```
stencil advection_MUS_stencil {
   Do {
      //-- computation of tracer slop
      vertical_region(k_start, k_end - 1) {
         zwx = umask * (ptb(i + 1) - ptb);
         zslpx = (zwx + zwx(i - 1)) * 0.5;
      }

      //-- MUSCL horizontal advective fluxes
      vertical_region(k_start, k_end - 1) {
         zzwx = ptb(i+1) + k * zslpx(i+1)
         zzwy = ptb + k * zslpx
         zflx = pun * ( s * zzwx + (1 - s) * zzwy )
      }
   }
}
```

Listing 4: stencil for advective fluxes in the MUSCL advection scheme with GTClang

5.2 CDSL Applied to the Dwarf-advection-MUS

The ESCAPE2 DSL toolchain has the ambition to cover a major portion of the algorithmic space of the weather and climate domain. Therefore, structured and unstructured grids are supported by CDSL. To explore the applicability of the structured aspect the NEMO advection dwarf described above has also been formulated using CDSL.

5.2.1 CDSL Kernel Implementation

In contrast to the formulation using GTCLang the CDSL implementation followed closely the original Fortran implementation. This allowed an incremental development, where the full advection problem was split into several small code regions tested individually. The listings below show the translation of a typical loop nest to CDSL.

```
DO jk = 1, jpkm1                      !-- Tracer advective trend
  DO jj = 2, jpjm1
    DO ji = fs_2, fs_jpim1    ! vector opt.
      zbtr = 1. / ( e1t(ji,jj) * e2t(ji,jj) * fse3t(ji,jj,jk) )
      ! horizontal advective trends
      ztra = - zbtr * ( zwx(ji,jj,jk) - zwx(ji-1,jj,jk)    &
           &               + zwy(ji,jj,jk) - zwy(ji,jj-1,jk) )
      ! add it to the general tracer trends
      pta(ji,jj,jk,jn) = pta(ji,jj,jk,jn) + ztra
    END DO
  END DO
END DO
```

Listing 5: selected region of the Fortran NEMO-dwarf implementation

```
void calc_trend_backward(Field2d e1t, Field2d e2t, Field3d fse3t,
                         Field3d zwx, Field3d zwy, Field3d pta) {
  Field3d ztra, zbtr;
  Intrinsic_longitude ji;
  Intrinsic_latitude jj;

  vertical_region(start_level, end_level - 1) {
    zbtr = 1.0 / (e1t * e2t * fse3t);
    // horizontal advective trends
    ztra = -zbtr * (zwx - zwx(ji - 1) + zwy - zwy(jj - 1));
    // add it to the general tracer trends
    pta = pta + ztra;
  }
}
```

Listing 6: selected region of the CDSL NEMO-dwarf implementation

Not shown in the CDSL code are the user-defined field types *Field2d* and *Field3d* which are part of the DSL code. In contrast to the GTClang version there are no predefined index variable names (i, j, k) with implicit semantic. Instead, semantic must be bound to types or variables via explicit declaration, e.g., by using the type `Intrinsic_longitude`.

The driver code where the domain and meta data are specified and resource allocation and kernel execution are controlled was implemented in Fortran using the object oriented Fortran kernel interfaces generated by CDSL. E.g.:

```
CALL calc_trend_backward%init(core_domain)
CALL calc_trend_backward%run(e1t, e2t, fse3t, zwx, zwy, pta(jn))
```

Listing 7: excerpt of Fortran driver code

This is an example of the general focus of CDSL: Although the DSL code is embedded in C++, the target application is the Fortran model. The Fortran support is complete, i.e., no additional non-Fortran driver code had to be written in the implementation of the CDSL dwarf version.

5.2.2 Verification

The CDSL version of the NEMO dwarf extends the original CMake/ecBuild configuration to use the ESCAPE2 toolchain with the "naive C++" and CUDA backend. The code generation is an integral part of the build procedure and has been tested on the DKRZ HPC system Mistral. The dwarf can be executed in parallel using MPI but the communication is still done on the host side (CPU) - not on the device side (e.g., GPU). Therefore data has to be synced back and forth between device and host in order to do the halo exchange.

The original file-based data comparison scheme was changed in favor of a more debug-friendly solution that executes two different advection implementations in the same program: a reference version and a DSL version. In addition, a perturbation scheme has been implemented to increase the significance of the test. The verification scheme compares the resulting tracer fields using a certain tolerance. For the C++ backend the results were bit-identical while the CUDA backend required a relative tolerance of 10^{-12}.

5.3 CDSL Applied to the ICON-O Transport Dwarf

5.3.1 Challenges

The CDSL implementation of the ICON-O transport dwarf faced three major challenges:

- The benefit of reduced code complexity of the original dwarf concept did not apply because the extraction of a stand-alone transport kernel was not considered feasible given the rich dependencies within the ICON model. Instead, the dwarf was defined as limited code execution path at run-time. As a consequence the development, build, and execution of the dwarf was more extensive than, e.g., for the small NEMO dwarf. On the other hand, this complexity offered the opportunity to study more aspects of the full model DSL toolchain application at an early time and to iterate between comprehensive application and toolchain development. As a consequence we have CDSL support integrated into the ICON infrastructure that greatly simplifies the extension of the current dwarf to other performance relevant kernels. But we (MPI-M & DKRZ) do not yet cover as many kernels as we think to be desirable for a more general applicability assessment. Presented here are two ICON-O kernels containing the following operators:

  – An elemental single-point operator that combines velocity fields with scalar weights.

  – A horizontal operator with second-nearest neighbors applied to a 3d velocity field and a scalar tracer field.

  In addition to these model integrated use cases several simple unit tests have been implemented directly within the front-end repository. These tests read a global low-resolution ICON grid and exercise simple stencils on the grid. All unit tests are implemented twice and tested against each other: Fortran vs. CDSL. This somewhat recovers the low-complexity dwarf idea but uses synthetic kernels instead of extracted code.

- The initial DSL design was presuming the usage of multiple simple sparse dimensions that seemed to offer a straightforward transition from model to DSL implementation. However, concerns about run-time efficiency led to a new concept using only one sparse dimension with extended capabilities. It is possible that the idea of multiple sparse dimensions will be supported in the future to recover generality but at the time of writing of this deliverable this is not yet the case. One of the operators selected for the ICON-O dwarf was formulated in the model using two sparse dimensions and therefore had to be rewritten in a form that is representable with the toolchain.

- ICON uses a blocked data layout, e.g., (nproma, nlev, nblocks) for 3d state fields. The back-end, however, uses a non-blocked layout which generally makes direct data transfer between model and toolchain back-end impossible. A simple solution would be to choose a huge nproma value that contains all horizontal indices in a single block: (nproma, nlev). However, this would assume that all performance relevant kernels have been ported to the toolchain. Otherwise the remaining CPU executed kernels would run with an inefficient nproma parameter. On the other hand, a general nproma setting with an indirect data transfer would suffer from additional transfer costs caused by the data layout transformations.

  For the current development situation we chose to implement an indirect data transfer for all supported field variants that allows us to keep the standard nproma setting together with its vectorization behavior. For D2.5 we plan to evaluate the trade-offs stated here.

5.3.2 Verification

The verification of the toolchain generated kernels was done by direct element-wise comparison with the results of the Fortran reference implementation using a suitable relative tolerance criterion:

- For the C++-naive back-end generated code a fixed relative tolerance near the precision limit of the floating point model ($\epsilon = 10^{-15}$) was chosen.
- In the CUDA case we had to weaken the precision requirements in order to ignore the increased numerical noise that is not part of our current verification: We reduced the relative tolerance to $\epsilon = 10^{-12}$, and we introduced an absolute tolerance for small values: $\delta = \epsilon R_{max}$, where $R_{max} = max\{|R|\}$ is the maximum absolute value of the reference field R. The perceived reduced precision in the CUDA case may be caused by a combination of different compiler optimizations and a loss of significant bits for reduction operations with stencil weights of near-equal absolute value and different signs.

5.3.3 Elemental Expressions

This kernel shows a simple elemental-access-only pattern without spatial dependency. It tests the data transfer (not shown here) of 3d fields between host and device, the usage of scalars, and applies a typical ocean depth limiter. The CDSL implementation (see Listing 8) reflects this simplicity. The CDSL code is more compact than the Fortran version because the blocked iteration and the special data layout does not appear in this formulation.

```
void time_weigthed_vn_kernel(EK_Field vn_weighted, EK_Field vn_old,
                             EK_Field vn_n, E_Field dolic_e,
                             K_Field k_id) {
   vertical_region(start_level,end_level) {
      compute_on(edges) {
         if (k_id <= dolic_e) {
            vn_weighted = ab_gam*vn_new + (1.0 - ab_gam)*vn_old;
         }
      }
   }
}
```

Listing 8: CDSL stencil for ICON-O velocity prediction kernel

Although the index free (implicit) formulation is more abstract, it can also be cumbersome as the usage of the vertical field `k_id` shows. This field is a user defined identity map for levels and allows to restrict the calculation to the depth of `dolic_e` in an implicit formulation. It would be helpful if the DSL itself could provide this information. Also, the DSL requires the vertical iteration to be the outermost loop. This may be perceived as unnatural for this iteration space where the vertical iteration length depends on the horizontal iteration.

5.3.4 Second Neighbor Expressions

The original ICON-O operator `edge2edge_viacell_coeff` contains two sparse dimensions with a double visitation of the central edge location and is therefore not directly representable with the toolchain that only supports one sparse dimension. To work around this limitation we split the operator into two sparse parts attached to a central edge: a two-point stencil `ec_op` on direct neighbor cells and a four-point stencil `ece_op` on the outer edges of the diamond pattern (see black lines in Figure 1). The corresponding reduction operations `nreduce` are shown in Listing 9. The locations of each stencil can be represented using a single sparse dimension. However, the four sparse edges of the `ece_op` stencil need different tracer weights depending on the neighboring cell (line 19 and 20). For this purpose, we used an indirect assignment DSL feature that copies domain field values to the elements of a sparse dimension with corresponding locations. This is done in the loop statement (line 11). The resulting auxiliary tracer stencil `ec_aux` is attached to an edge and therefore can be used in the edge-domain iteration. However, it cannot be used directly in combination with the `ece_op` reduction (lines 18-21): `ec_aux` only has two weights, while the diamond pattern reduction needs four and each sparse `ec_aux` weight must be used twice. This calls for an extended DSL reduction concept that accepts weights (line 19) as sparse fields with sparse offsets (line 20). A solution has been implemented in the separate development branch *offsetReduction* of Dawn [6] that is used here. The final result is the sum of the two stencil reductions.

[6]developer repository: https://github.com/mroethlin/dawn.git

```
void e2e_via_c_3d_mlev_constZs_kernel(
  EK_Field vn_e, EK_Field out_vn_e, ECEK_Field ece_op,
  ECK_Field ec_op, ECK_Field ec_aux, 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) {

       // aux field:
       loop_on(cells) { ec_aux = scalar_field; }

       // 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,
                                     {ec_aux, ec_aux, ec_aux, ec_aux},
                                     {0, 0, 1, 1},
                                      vn_e * ece_op * thick_edge );
      }
    }
  }
}
```

Listing 9: CDSL formulation of the ICON-O *mapping edges to edges via cells* kernel. Field type definitions are not shown.

The CDSL code is much more compact than the original Fortran code (Listing 3). However, the complexity of the auxiliary field construction requires some familiarity with the ESCAPE2 DSL concepts and the reduction functionality probably contains too many parameter for an intuitive understanding. It is not clear yet if this *offsetReduction* concept will be merged into the main Dawn or if an alternative and more general approach can be implemented. The solution closest to the original model formulation would require multiple sparse dimensions.

6 Discussion

The application of the ESCAPE2 DSL toolchain to the selected dwarfs as it is presented here demonstrates that the toolchain concept and its implementation generally work for structured and unstructured climate and weather codes. The toolchain is able to produce correct code for CPU and/or GPU. Both devices have been successfully used to test the portability of the DSL dwarfs. The backend choice has no influence on the DSL formulation or the usage in the model, only the build process had to be adapted. This demonstrates that *separation of concerns* works well in the approach chosen.

But it also became clear that the functionality for unstructured grids currently supported does not yet cover the target domain sufficiently since, e.g.:

- Operators with two sparse dimensions on unstructured grids cannot be expressed. This is not a fundamental restriction of the concept but rather of the current state of the implementation.

- For the structured NEMO dwarf there was no workaround required to find a proper DSL formulation. Here, the toolchain appeared to be mature and nearly complete. The exception is MPI communication which currently has to be handled by the model.

For the concrete problem of the ICON-O dwarf we have shown a prototype solution as a workaround for said limitation. But it is not clear yet if this approach will enter the toolchain definition finally supported. A proper solution would also take the usability aspect into account which we have not addressed in great detail here: The effort required to rewrite the original operator into two DSL-compliant operators was considerable, i.e., a suitable transform of constant stencil weights had to be implemented in the model. Another workaround that should be resolved in the future is the cumbersome implementation of a typical depth limiter for the ocean level iteration.

Having a high level intermediate representation (HIR) has proven to be a functional design that allows multiple front-ends to offer different forms of language elements. However, the high level of abstraction in the center of the toolchain also limits the variability of different front-ends. Therefore, the structural aspect of CDSL is very similar to GTClang. Escaping the DSL is currently only supported for elemental operations that do not break the abstraction. What is missing is an abstraction stack with at least two levels instead of a single HIR in order to express lower level DSL concepts without losing portability. This should replace the need to escape into a non-portable general purpose language formulation.

Looking beyond the ICON-O dwarf example we see the greater challenge of the full model production situation. This not only requires a solution to the current limitations but also a sustainability concept to protect the investment into a new programming language. This must also include a development process to extend the DSL capabilities according to the requirements of the weather and climate model community, e.g., support for general unstructured grids beyond the special triangular case. This aspect touches the scope of D2.6 which will report on the usability of the toolchain.

References

Bauer, Peter. Grant Agreement - Number 800897 - ESCAPE-2, **2018**. `https://confluence.ecmwf.int/display/ESCAPEII/Project+Documents`.

Behrens, Jörg, Reinhard Budich, Leonidas Linardakis Peter Korn, Ralf Mueller, Carlos Osuna, Giacomo Serafini and Tobias Wicky. D2.2 DSL front-end to parse DSL into high-level intermediate representation (HIR), **2019**. `https://confluence.ecmwf.int/display/ESCAPEII/Deliverables`.

Frank, David J, Robert H Dennard, Edward Nowak, Paul M Solomon, Yuan Taur and Hon-Sum Philip Wong, **2001**. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288.

Korn, P. and S. **Danilov**, **2017**. Elementary dispersion analysis of some mimetic discretizations on triangular c-grids. *Journal of Computational Physics*, 330:156–172. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2016.10.059.

Korn, Peter, **2017**. Formulation of an unstructured grid model for global ocean dynamics. *Journal of Computational Physics*, 339:525–552. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2017.03.009.

Korn, Peter, **2018**. A structure-preserving discretization of ocean parametrizations on unstructured grids. *Ocean Modelling*, 132:73–90. ISSN 1463-5003. doi:https://doi.org/10.1016/j.ocemod.2018.10.002.

Korn, Peter and Leonidas **Linardakis**, **2018**. A conservative discretization of the shallow-water equations on triangular grids. *Journal of Computational Physics*, 375:871–900. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2018.09.002.

Lévy, Marina, Audrey Estublier and Gurvan Madec, **2001**. Choice of an advection scheme for biogeochemical models. *Geophysical Research Letters*, 28 (19):3725–3728. doi:10.1029/2001GL012947.

Madec, Gurvan and NEMO System Team **NEMO-ST**. *NEMO ocean engine*, **2019**.

Osuna, C., J. Behrens, R. Budich, W. Deconinck, J. Duras, I. Epicoco, O. Fuhrer, C. Kühnlein, L. Linardakis, T. Wicky and N. Wedi. D2.1: High-level domain specific language (dsl) specification, **2019a**. `https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5c2e37f98&appId=PPGMS`.

Osuna, Carlos, Jörg Behrens, Reinhard Budich and Tobias Wicky. D2.3 High-level intermediate (HIR) representation specification, **2019b**. `https://confluence.ecmwf.int/display/ESCAPEII/Deliverables`.

Osuna, Carlos, Tobias Wicky, Fabian Thuering, Torsten Hoefler and Oliver Fuhrer, **2020**. Dawn: a high-level domain-specific language compiler toolchain for weather and climate applications. *Supercomputing Frontiers and Innovations*, 7(2). ISSN 2313-8734. https://www.superfri.org/superfri/article/view/314.

Singh, Vikram and Peter **Korn**. A structure-preserving ocean model in generalized vertical coordinates, **in preparation**.

D2.4: Demonstration of Domain Specific Language Toolchain for Selected Weather and Climate Dwarfs

Dissemination Level: Public

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800987.

www.hpc-escape2.eu

Funded by the European Union

Co-ordinated by ECMWF

Energy-efficient Scalable Algorithms for Weather and Climate Prediction at Exascale

Authors: **Jörg Behrens, Carlos Osuna, Italo Epicoco, Reinhard Budich, Julia Duras**

Date **31. May 2021**

Research and Innovation Action

H2020-FETHPC-2017

Project Coordinator: Dr. Peter Bauer (ECMWF)

Project Start Date: 01/10/2018

Project Duration: 36 month

Published by the ESCAPE-2 Consortium

Version:

Contractual Delivery Date:

Work Package/ Task:

Document Owner:

Contributors:

Status:

Table of Contents

Figures

Tables

Body Text (Normal)

Caption 1

Note:

Figure 1: Example Figure

1. References

8 Heading 1

8.1 Table example

Title	Date	Time	Copy
Copy	Copy	Copy	Copy
Copy	Copy	Copy	Copy

Table 1: Table Caption

9 Conclusion

10 References

Document History

Version	Author(s)	Date	Changes
1	**Jörg Behrens, Carlos Osuna, Julia Duras, Italo Epicoco, Reinhard Budich, Peter Korn**	**2021-06-28**	

Internal Review History

Internal Reviewers	Date	Comments
Sergey Kosukhin, MPI-M	**2021-07-05**	

Effort Contributions per Partner

Partner	Efforts
MSuisse	**2**
DKRZ	1
CMCC	0.8
MPI-M	6
Total	**9.8**

ESCAPE 2

ECMWF Shinfield Park Reading RG2 9AX UK

Contact: peter.bauer@ecmwf.int