ESCAPE 2

D2.6: On the Usability and Performance of DSL toolchains

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

ESCAPE 2

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

Authors: **Jörg Behrens, Reinhard Budich, Mathias Röthlin**

Date **29. September 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 2021

Work Package/ Task: WP2

Document Owner: Reinhard Budich

Contributors: Jörg Behrens, Mathias Röthlin

Status: Final

Contents

1 Introduction **4**

2 Deployment **4**

3 Usability **5**

3.1 DSL Programmability 5

3.2 Completeness 7

3.3 Modularity 8

3.4 Model Integration 9

3.4.1 Toolchain Build Integration 9

3.4.2 DSL Kernel Access 10

4 Performance **11**

4.1 Discussion 11

5 Conclusion **12**

6 Outlook **12**

1 Introduction

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.
There is another theory which states that this has already happened.
Numerous times.... [2]

One of the key objectives of ESCAPE-2 is the development and application of a "domain-specific language (DSL) concept for the weather and climate community in order to maximize flexibility, programmability and performance portability to heterogeneous hardware solutions across different weather and climate models". The realization of this objective has been structured into several steps containing the definition of a DSL concept, the development of an open source toolchain, and the demonstration of its applicability. These steps have been described in deliverables D2.1-D2.5 [8, 4, 9, 5, 7]. Deliverable D2.6 at hand here reports on the usability and performance of applying the DSL toolchain and depends on the above list of previous deliverables.

The aim, as formulated in task T2.4, is "an in-depth evaluation of the DSL in terms of usability and performance [...], including a comparison between the original code and the DSL version and their behaviour on different systems."

Usability also has been discussed based on concrete algorithms of selected dwarfs in D2.4 [5] which are not repeated here but only summarized. Four main aspects of usability are considered: programmability, completeness, modularity and the ability to integrate the DSL solution into the model.

In addition to the evaluation of usability by reviewing previous results and experiences there are now first performance measurements on GPUs available. These are presented in comparison with OpenACC and discussed with respect to their sensitivity and to the applied optimizations.

2 Deployment

The SIR form generated by the front-end in JSON format is system independent [1]. It can be generated on one system and used on another. The system dependent toolchain behavior is associated with the availability of a CUDA GPU which allows additional internal tests and may require different performance tuning parameters depending on the device. Various run-time tests have been executed on the following systems:

- Mistral supercomputer at DKRZ
- Tsa supercomputer of MeteoSwiss
- Standard Ubuntu 20.04 desktop and server systems

[1] There is a *filename* element in SIR which caries the unessential but system dependent DSL source code path. Using the *-nopath* option of CDSL allows to generate bit-identical file content on different systems.

GPU performance tests were executed on Tsa (see section 4). Mistral is at it's end of lifetime and does not have any modern GPUs, so it was not considered a promising platform to do additional performance evaluations. CDSL internal correctness tests were executed on all systems:

For the evaluation of a proper toolchain build the fundamental DSL building blocks that correspond to the defined DSL concepts have been implemented in the form of unit tests into the front-end. Furthermore there are extensive built-in integration tests that capture the whole workflow from DSL source code up to the execution of the compiled binary with subsequent numerical comparison of Fortran and DSL solutions. Structured and unstructured grids are tested on CPU and - if available - on CUDA-GPUs. The unstructured tests use an original ICON grid provided by MPI-M and therefore allow reconstructing the stencil connectivity present in the full ICON model.

DSL versions of the NEMO and ICON dwarf have been evaluated on Mistral using CPU and CUDA back-ends. The DSL implementations pass the correctness criteria. This has been reported in detail in D2.4 [5].

3 Usability

In the following we describe different usability aspects of the DSL toolchain:

- *DSL Programmablity* describes how the DSL concepts are expressed in actual DSL source code and how this code is validated at compile time.
- *Completeness* describes the algorithmic space with respect to the analyzed dwarves and the initial ideas formulated in the D2.1 [8].
- *Modularity* describes the structure of the toolchain and its advantages.
- *Model Integration* describes integrated building and DSL kernel access from the perspective of a model developer.

3.1 DSL Programmability

The ESCAPE2 front-end CDSL follows the embedded DSL approach using C++ as host language. Therefore each DSL source code constitutes also valid C++ code. The opposite is not true: Not every C++ code is a valid DSL code. The ESCAPE2 DSL defines certain concepts that must be identifiable within the C++ source code, otherwise the code is rejected by the toolchain. The structure of a valid CDSL source code is sketched in listing 1:

```
// CDSL definitions
#include "dsl.hpp"
using namespace EDSL;
namespace edsl {
  // definition of iteration spaces:
```

```
  Gridspace cek_space(cells,edges,levels);
  Gridspace ck_space(cells,levels);
  Gridspace ek_space(edges,levels);

  // definition of field types:
  define_field_type(CEK_Field,cek_space);
  define_field_type(CK_Field,ck_space);
  define_field_type(EK_Field,ek_space);

  // definition of global scalars, e.g.:
  const int g = 42;

  // definition of kernels:
  void myfun1(CEK_Field stencil, CK_Field alpha, EK_Field beta) {
    vertical_region(start_level,end_level) {
      compute_on(cells) {
        alpha = nreduce(edges, stencil * beta) * g;
      }
    }
  }

  void myfun2(CEK_Field stencil, CK_Field alpha, EK_Field beta) {
    alpha = nreduce(edges, stencil * beta) * g;
  }

}
```

Listing 1: CDSL code example

The source code shows the definition of different entities:

- The DSL itself is described in a C++ header file that has to be included first. The following DSL source code has to be contained within the *edsl* namespace in order to be recognized by the toolchain.
- Predefined domain spaces, e.g. cells and levels, can be combined into so called *gridspaces*. A gridspace describes the degrees of freedom of field variables or iteration statements.
- Field type definitions bind a gridspace to a C++ type which then can be used to declare field variables. On unstructured grids there are dense fields which have only one unstructured degree of freedom and sparse fields which have two. Higher order sparse fields with more than two unstructured dimensions are currently not supported. Structured grids don't need sparse fields to describe stencils: they are described via scalar weights combined with directional field offsets in Cartesian coordinates.
- The toolchain also supports scalars of global scope but not global fields.
- Function definitions have to follow a certain structure that organizes domain iterations and local operations.
- Iterations covering the 3d physical space are expressed as two nested concepts: an outer *vertical_region* and an inner *compute_on* region. The

compute_ on concept accepts horizontal locations, e.g. *cells, edges or vertices* for unstructured grids and *longitudes, latitudes* for structured grids. The vertical extent can be narrowed with literal offsets to the symbolic start and end levels. This allows special implementations for surfaces.

There are two main classes of local operations available in functions: elemental algebraic operations and stencil reductions on field expressions over neighbors. These can be combined with iterations to formulate an algorithm.

The DSL abstraction is visible here by the lack of iteration indices. The abstraction can be increased further as shown in *myfun2* that is equivalent to *myfun1*. The additional compactness is a consequence of the available type information at compile time which allows the toolchain to infer iterations from field declarations. The same information is used to check the consistency of a field statement: the DSL does not allow a binary operation on incompatible degrees of freedom, e.g., $alpha(cells) = beta(edges)$ gives an error message.

At runtime debugging can be done with standard tools. However these tools will show the generated C++/CUDA code - not the DSL code. Therefore some understanding of the target language is required for programming in the current DSL.

The concepts of the ESCAPE-2 DSL have been described in D2.1 [8]. Details of the concrete implementation and the dialect of CDSL have been described in D2.2 [4]. The implicit semantics contained in the DSL concepts and field declarations make the DSL an expressive language beyond general purpose languages like Fortran or C++.

The parallel execution model [2] defines the order in which algorithmic elements within a kernel are executed: Vertical iterations over whole horizontal planes keep their sequential order in a parallel execution while iterations within each horizontal plane have no guaranteed execution order.

The independent horizontal evaluation semantic is very common in the targeted application domain. However, the fixed nesting of iteration spaces with an outer vertical iteration does not always match the implementation within ICON-O, i.e., a vertical loop-bound that is given by a horizontal depth field cannot be represented in the same way. Instead, an equivalent DSL formulation must use a 3d land-sea mask in a conditional statement applied to every iteration.

3.2 Completeness

The space of describable algorithms with the ESCAPE-2 DSL is less than that of a general purpose language. However, this corresponds to the main idea behind DSLs: to limit and capture the domain complexity using few selected concepts. The general scope of algorithms targeted by the DSL is the stencil-based weather and climate domain [8].

The applicability of the current implementation has been demonstrated for the structured NEMO dwarf and for selected kernels of the unstructured ICON

[2]https://github.com/MeteoSwiss-APN/dawn/wiki/SIR-Execution-Model

model used in an ICON-O advection test scenario [5]. The main result is that the DSL toolchain can be used for a wide range of relevant stencil problems.

Stencils with more than one sparse dimension are not supported yet. Such a stencil has been examined in D2.4 [5]. The current solution to such a problem is based on an experimental branch of the toolchain back-end Dawn that generalizes the *reduction* concept. On the model side this solution required a split of the original stencil tensor in order to fit into the current sparse field concept of the DSL. The original pseudo-code DSL formulation for such a higher order stencil envisaged a sparse field concept with multiple sparse dimension (see section 4.2.2.1 of D2.1 [8]). However, such a field concept has not been implemented yet. Instead, the toolchain supports a generalized single sparse dimension that is more efficient but less general than the original idea. In principle it should be possible to extend the current approach to recover generality in a future toolchain implementation.

On GPUs device data can be shared between toolchain generated CUDA code and OpenACC, and the execution order of DSL and OpenACC kernels can be controlled by using the same CUDA stream. This enables a mixed solution and therefore relaxes concerns about algorithmic completeness of the DSL.

For structured grids the toolchain seems complete with respect to the numerical aspect of the investigated NEMO dwarf. However, the lack of support for halo exchanges leads to additional data transfers between host and device in order to perform MPI communication on the host side. This problem applies to unstructured grids, too.

The original ambition formulated in D2.1, section 5 targeted *safety* of parallel implementations which is not realized in parallel models like MPI, OpenMP and OpenACC [8]. The DSL toolchain has the information needed to automatically decide about required halo updates and to execute these depending on the applied stencils. Such an automatism would eliminate the possibility of inconsistent states of decomposed fields. However, it is currently not possible to delegate this task to the toolchain.

The examined NEMO and ICON kernels have been translated using toolchain back-end variants that can generate naïve C++ or optimized CUDA code. In the future, additional variants have to be provided by the toolchain in order to realize performance portability.

3.3 Modularity

The architecture employed in the design of the toolchain is modular: The use of the SIR and IIR specification at the interfaces of the different components effectively decouples them. This allows for efficient extension of the toolchain by the user.

This is illustrated in figure 1. The (unstructued) toolchain currently supports two front-ends, namely CDSL and dusk, and has one code generator that emits code in two host languages (C++ and CUDA). Due to the well specified interfaces using SIR and IIR, respectively, this can be readily extended by the

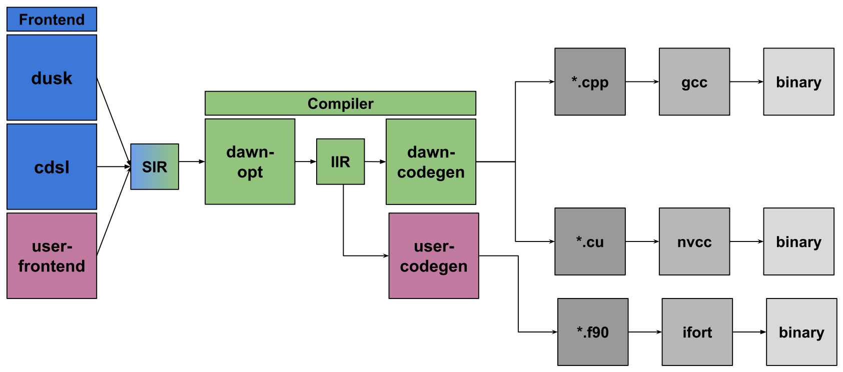

Figure 1: Design of the ESCAPE-2 DSL toolchain.

user. A new front end can be added by transposing user code into their respective SIR equivalents. Conversely, a new code generator can be added by inspecting the IIR emitted by `dawn-opt`, end emitting a language primitive in the desired host language for each IIR node. Since both SIR and IIR can be serialized to disk, both of these envisioned added components can potentially be implemented in any programming language of the users chosing.

3.4 Model Integration

3.4.1 Toolchain Build Integration

The build procedure of the full ICON model has been extended to use the ESCAPE-2 DSL toolchain as a compiler to generate hardware specific kernel codes from DSL source code. The generated toolchain output is compiled using model compatible compilers (in the case of mistral.dkrz.de: g++ and gfortran of gcc71 and cuda-10.0.130 for NVidia GPUs). The ICON model is then compiled in a second build phase using the generated libraries and Fortran modules of the first phase. The linking step requires the inclusion of the C++ standard library libstdc++ and, if GPUs are targeted, the CUDA libraries libcudart.so and libcudadevrt.a. In addition we use the toolchain library libcdslUnstruct.so that supports object oriented kernel access and data transfer for unstructured grids with Fortran.

The main feature of this procedure is that the DSL source code behaves like model source code insofar that it is an integral part of the established build scheme that triggers recompilation on modification.

The second aspect is that the hardware selection is completely absorbed by the build procedure. Neither the DSL code nor the model has any knowledge of the hardware target. This separation of concerns is possible because of the toolchain design and a kernel abstraction layer that provides identical Fortran interfaces to the model.

3.4.2 DSL Kernel Access

The choice of a C++ embedded approach for the front-end implementation was founded on technical aspects: The flexibility and compile-time type-safety of C++ and the available LLVM/clang tooling support that reduces the parsing task and also catches basic errors in DSL code. The host language however, does not extend the algorithmic space of the DSL. This is entirely defined by the HIR [9]. The actual main application target is a model implemented in modern Fortran. Therefore the toolchain can generate object oriented Fortran support that simplifies the DSL usage.

The command

```
$ cdsl -s cpp -b CUDAIco -o out.cpp DSL_example1.cpp
```

generates three files: `out.cpp`, `c_itf_out.cpp`, and `f_itf_out.f90`. For each DSL function these files provide Fortran types with type-bound procedures `init`, `run`, and `finalize` which connect to the corresponding kernel methods. The toolchain provides additional Fortran support that allows to specify, generate and transfer appropriate device data fields in a hardware independent way.

The data transform between model and back-end requires the specification of meta data that describes the size and kind of dimensions used in the allocation of a model variable, e.g., listing 2

```
    CALL e_vol_meta_data%init([nproma, nlev, nblks_edges], &
    & [cdsl_idx_dim_kind, cdsl_lev_dim_kind, cdsl_blk_dim_kind])
```

Listing 2: meta data specification example

describes the ICON data layout of a 3d velocity field on edges. The sequence of dimensional kind values describes the blocked data layout where horizontal planes are in general non-contiguous. In a setting optimized for GPUs-only `nproma` could be set to an extremely large value such that the layout consists of a single block only. In that case direct transfers without reshape would be possible.

In addition to the exchange of simple model data there is a more complex mesh object that is used in the kernel initialization for unstructured grids. This mesh has to be generated once within the model using Atlas [6] and existing direct connectivity maps between edges, vertices and cells. All required functionality is available in Fortran.

4 Performance

To get a first picture of the performance achieved by the code produced by the DSL toolchain, the complete diffusion submodule (`mo_nh_diffusion.f90`) was translated stencil by stencil, defined here as a `$ACC PARALLEL LOOP` region.

The procedure outlined in section 3.4.1 allows for “in situ” timings of the stencils directly in the ICON model. The grid data allocated by ICON is re-used by the DSL stencils executed, hence, there is no overhead associated with copying fields to the DSL context.

Timings recorded on the tsa supercomputer of MeteoSwiss using a single nvidia Tesla V100 GPU are summarized in figure 2. As can be seen, the DSL version either outperforms or matches the performance of the OpenACC version for each kernel, except in stencil 13. Save for this stencil, the performance increase using DSL ranges from 6 percent (stencil 6) to 87 percent (stencil 14), with an average performance increase of roughly 23 percent.

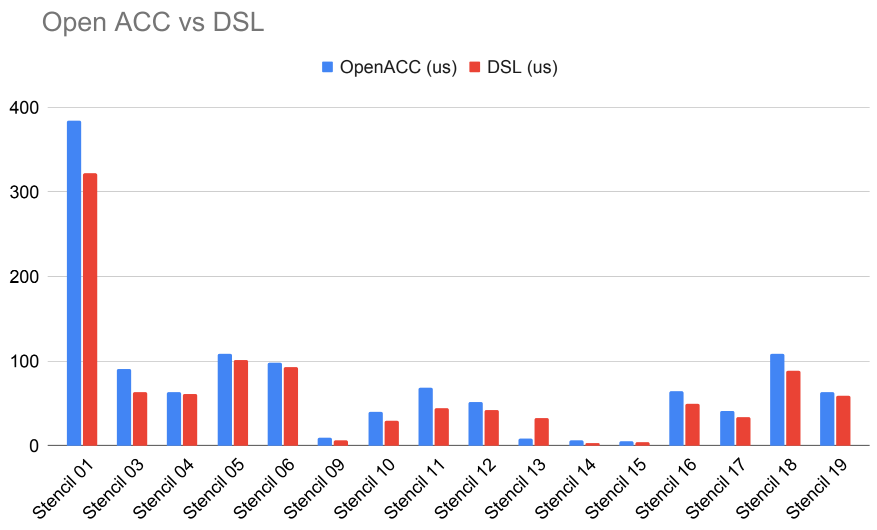

Figure 2: Timings of the diffusion submodule, OpenACC vs DSL

4.1 Discussion

While these timings look promising, it has to be noted that quite a few performance optimization on the DSL side were not employed. First of all, it would be readily possible to fuse some of these stencils using the DSL apporach into one. This was not done in order to enable a one-to-one comparison of singular DSL stencils to OpenACC parallel regions. Furthermore, only the default set of optimizations in the dawn compiler were activated. That is, aggressive or experimental compiler passes that are not proven on the whole dycore were deactivated. It also has to be noted that the performance of some DSL kernels is quite sensitive on the CUDA block size and thread granularity (number of vertical levels processed by a single CUDA thread), which was fine tuned manually on a per-stencil basis. Of course it would be desirable that these quantities are determined heuristically in the compiler, but current research suggests that this is a daunting task. Another ongoing process is to properly understand these performance differences using more involved profiling than simply comparing the run times to device further optimization strategies.

5 Conclusion

The ESCAPE-2 DSL toolchain constitutes a DSL concept suitable for the stencil based weather and climate domain. It separates syntax from functionality by specifying a High-level Intermediate Representation (HIR) and therefore allows syntax variations in order to express different preferences. The readable HIR format also allows writing additional tools, e.g., interface generators to support a certain programming style in the chosen application language.

The toolchain concept has been implemented in ESCAPE-2 by providing the C++ embedded front-end prototype CDSL and the optimizing back-end Dawn. Structured and unstructured grids are supported. The code generation currently provides debug-friendly C++ and optimized CUDA solutions. First performance measurements on Nvidia GPUs already show a clear advantage compared to OpenACC although not all available DSL performance optimization strategies have been applied yet.

The generated code can be used directly in C/C++ or Fortran codes, and the integration of DSL kernels into a full model source code and build system has been demonstrated for ICON. The DSL can capture a large number of relevant stencils, however, the coverage is not yet complete. For GPU targets this problem can be bypassed by using a mixed OpenACC/DSL approach. An equivalent strategy for CPU targets would combine host-language solutions with DSL solutions. However, the performance of the CPU related code generation for unstructured grids has not yet been analyzed.

It seems to be too early to expect full performance portability from the current toolchain implementation. However, this is an iterative process: Implementation and application are advancing together. For CUDA-GPUs the optimization aspect of the toolchain seems most advanced, approaching production quality.

6 Outlook

The ESCAPE-2 proposal states that the top-level project objectives 3 and 5 (see Table 1 in [3]) are achieved by, i.a.:

- sustaining community-wide code usability and maintainability beyond the lifetime of the project (objective 3)
- providing an open-source DSL toolchain software and support beyond the project lifetime to sustain & accelerate novel algorithm development and ensuring performance portability to emerging HPC hardware (objective 5)

Sustainability mentioned in these objectives can be linked to the open source nature and the modularity of the DSL toolchain (see section 3.3) which allow a continued development by the domain community. Modularity is also realized on a finer level within the implementation of the main toolchain components. This encapsulates complexity further and simplifies the continued development.

It may be expected that future projects that are interested in performance portability will contribute to the toolchain evaluation and development. Currently the “Centre of Excellence in Simulation of Weather and Climate in Europe” (ESiWACE) [1] applies and evaluates the DSL toolchain approach of ESCAPE-2.

References

[1] Excellence in simulation of weather and climate in europe, phase 2, January 2019. ESiWACE2, funded under H2020-EU.1.4.1.3.

[2] D. Adams. *The Hitchhiker’s Guide to the Galaxy.* San Val, 1995.

[3] Peter Bauer. Escape-2 - proposal number 800897, September 2017. H2020-CP-2016-v2.pdf Ver1.00 20170223.

[4] Jörg Behrens, 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), September 2019.

[5] Jörg Behrens, Carlos Osuna, Julia Duras, Italo Epicoco, Reinhard Budich, and Peter Korn. D2.4: Demonstration of domain specific language toolchain for selected weather and climate dwarfs, June 2021.

[6] Willem Deconinck, Peter Bauer, Michail Diamantakis, Mats Hamrud, Christian Kühnlein, Pedro Maciel, Gianmarco Mengaldo, Tiago Quintino, Baudouin Raoult, Piotr K. Smolarkiewicz, and Nils P. Wedi. Atlas : A library for numerical weather prediction and climate modelling. *Computer Physics Communications*, 220:188–204, 2017.

[7] C. Müller, M. Röthlin, and C. Osuna. D2.5: Implementation of comprehensive domain specific language toolchain, August 2021.

[8] C. Osuna, 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, March 2019.

[9] Carlos Osuna, Jörg Behrens, Reinhard Budich, and Tobias Wicky. D2.3: High-level intermediate (HIR) representation specification, September 2019.

Document History

Version	Author(s)	Date	Changes
1	Behrens, Budich, Röthlin	2021-09-08	Final touches
2	Behrens, Budich, Röthlin	2021-09-29	Reviewers comments dealt with

Internal Review History

Internal Reviewers	Date	Comments
Kim Serradell	2021-09-17	On debugging tools, performance, portability

Effort Contributions per Partner

Partner	Efforts
DKRZ	0,5
MeteoSuisse	5
MPI-M	3
Total	**8,5**

ESCAPE 2

ECMWF Shinfield Park Reading RG2 9AX UK

Contact: peter.bauer@ecmwf.int