



Report summarising the adaptation of the proposed DSLs and the evaluation of the benchmarks and models proposed in this project

Deliverable D2.1



The project Centre of Excellence in Simulation of Weather and Climate in Europe Phase 2 (ESiWACE2) has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 823988.

About this document:

Work package in charge: WP2 Establish, evaluate and watch new technologies for the community

Actual delivery date for this deliverable: December 23, 2022

Dissemination level: PU (for public use)

Lead authors:

Science and Technology Facilities Council (STFC): Rupert Ford and Andrew Porter

Federal Office of Meteorology and Climatology (MeteoSwiss): Matthias Roethlin and Carlos Osuna

Other contributing authors:

European Centre for Medium-Range Weather Forecasts (ECMWF): Christian Kühnlein

Met Office (METO): Iva Kavcic

STFC: Nuno Nobre

Project internal reviewers:

Stichting Netherlands EScience Center (NLeSC): Ben van Werkhoven

Deutsches Klimarechenzentrum GmbH (DKRZ): Julia Duras

Contact details:

Project Office: [esiwace@dkrz.de](mailto:esiwace@dkrz.de)

Visit us on: <https://www.esiwace.eu>



Access our documents in Zenodo:

<https://zenodo.org/communities/esiwace>

Disclaimer: This material reflects only the authors' view and the Commission is not responsible for any use that may be made of the information it contains.

## Contents

<b>1</b>	<b>Abstract/publishable summary</b>	<b>4</b>
<b>2</b>	<b>Conclusion &amp; Results</b>	<b>4</b>
<b>3</b>	<b>Project objectives</b>	<b>4</b>
<b>4</b>	<b>Detailed report on the deliverable</b>	<b>5</b>
4.1	DSL Improvements . . . . .	5
4.1.1	PSyclone Improvements . . . . .	5
4.1.2	dawn Improvements . . . . .	11
4.2	Comparison of DSLs . . . . .	16
4.2.1	Benchmark and Machine Configuration . . . . .	17
4.2.2	Performance Results . . . . .	18
<b>5</b>	<b>Changes made and/or difficulties encountered, if any</b>	<b>22</b>
<b>6</b>	<b>How this deliverable contributes to the European strategies for HPC</b>	<b>22</b>
<b>7</b>	<b>Sustainability</b>	<b>22</b>
<b>8</b>	<b>Dissemination, Engagement and Uptake of Results</b>	<b>23</b>
8.1	Target audience . . . . .	23
8.2	Record of dissemination/engagement activities linked to this deliverable . . . . .	24
8.3	Publications in preparation OR submitted . . . . .	24
8.4	Intellectual property rights resulting from this deliverable . . . . .	24

## 1 Abstract/publishable summary

This deliverable discusses the extensions and developments performed to two DSLs, PSyclone and dusk/dawn, required to support the models of ESiWACE2. PSyclone was further developed to improve the performance and scalability of the Met Office's LFRic atmosphere model and to improve the performance of the generated GPU version of the full NEMO ocean model. PSyclone is already used in LFRic and as a result of these improvements the NEMO community are also considering adopting PSyclone for use in their model. The dusk/dawn DSL was also extended to support specific patterns of ICON and to provide integration of the DSL code into the Fortran codebase of ICON. Dusk/dawn provides a pythonic high-level language to express the ICON algorithms. The optimised backend provides significant performance improvement on NVIDIA GPUs over the existing operational OpenACC code. Lastly, the potential for interoperability between the two DSLs is demonstrated in a NEMO benchmark code and it is shown that by combining the two tools, greater performance and improved performance portability can be obtained.

## 2 Conclusion & Results

The PSyclone developments for LFRic have helped improve the performance and scalability of the Met Office's next generation atmosphere model LFRic and the developments for NEMO have significantly improved the performance of the generated GPU version, as well as making it more portable by adding OpenMP GPU support. The dusk and dawn developments enabled MeteoSwiss to translate the complete dycore of ICON, enabling significant speed ups. Lastly, the new SIR backend for PSyclone has shown that whilst the PSyclone and dawn DSLs are quite different, it is possible for them to interoperate by mapping between their internal representations in cases where they both support the same domain. Results for a NEMO benchmark show that this interoperability improves overall performance and performance portability on a NEMO benchmark.

## 3 Project objectives

This deliverable contributes directly and indirectly to the achievement of both of the macro-objectives and all but one of the specific goals indicated in section 1.1 of the Description of the Action:

Macro-objectives	Contribution of this deliverable
(1) Enable leading European weather and climate models to leverage the available performance of pre-exascale systems with regard to both compute and data capacity in 2021.	X
(2) Prepare the weather and climate community to be able to make use of exascale systems when they become available.	X

Specific goals in the workplan	Contribution of this deliverable
Boost European climate and weather models to operate in world-leading quality on existing supercomputing and future pre-exascale platforms	X
Establish new technologies for weather and climate modelling	X
Enhance HPC capacity of the weather and climate community	X
Improve the toolchain to manage data from climate and weather simulations at scale	
Strengthen the interaction with the European HPC ecosystem	X
Foster co-design between model developers, HPC manufacturers and HPC centres	X

## 4 Detailed report on the deliverable

This section is separated into two main parts. First, the improvements that were made to the two Domain-Specific Language (DSL) systems (PSyclone and dusk/dawn) are detailed. These improvements have been made in order to support three of the four Weather and Climate models targeted in this project; NEMO (Madec and Team), LFRic (Adams et al. (2019)), and ICON. In the fourth case, the structured version of FVM (which is part of the IFS suite) was evaluated using the GT4Py DSL<sup>1</sup> that was already available at the start of the project and therefore no DSL development was required. In the second part of this section the PSyclone and dusk/dawn DSLs are compared with each other. The evaluation part of this report focuses on a NEMO benchmark as this code can be targeted by both the PSyclone and dusk/dawn DSLs, thereby allowing their comparison. Please see Deliverable D2.3 for more details on benchmark selection.

### 4.1 DSL Improvements

This section details the DSL improvements and additional DSL concepts that were added to the PSyclone and dusk/dawn DSLs in order to successfully run the models targeted by the DSLs in this project as well as efficiently run them on different architectures. PSyclone was being extended to support the Met Office's LFRic Atmosphere model and the NEMO ocean model. dusk/dawn was being extended to support the ICON atmosphere model.

In the next section the PSyclone improvements are described. The following section describes the improvements made to dusk/dawn.

#### 4.1.1 PSyclone Improvements

**Releases and Documentation** At the beginning of the ESiWACE2 project, the PSyclone release was 1.7.0. Since then, there have been 6 major releases and the current version is 2.3.1. The code is freely available under a BSD 3-clause licence and is openly developed on GitHub (<https://github.com/stfc/psyclone>), where all improvements can be viewed if required. PSyclone also has a user guide (<https://psyclone.readthedocs.io/>), a developer guide (<https://psyclone-dev.readthedocs.io/>) and a reference guide (<https://psyclone-ref.readthedocs.io/>).

During the project, various tutorials have been developed for users to learn about PSyclone. Some of these use Jupyter notebooks (see <https://github.com/stfc/PSyclone#tutorial>) and some are intended for a user to work through on their local machine (<https://github.com/stfc/PSyclone/tree/master/tutorial/practicals>).

<sup>1</sup>This is the structured version of GT4Py. More recent implementations use the same name but are implemented differently, as mentioned later in this document.

**fparser2** fparser2 is a Fortran parser used by PSyclone to parse Fortran code into PSyclone's internal representation (PSyIR). The parser is implemented purely in Python and thus has minimal dependencies. It is open source under a BSD-3 licence and is openly developed on GitHub, see [www.github.com/stfc/fparser](http://www.github.com/stfc/fparser).

During the ESIWACE2 project, fparser2 has had to be extended to support a number of new Fortran constructs that it was not previously able to cope with, resulting from the parsing of NEMO and LFRic code, as well as a number of bugs. Some of this work was required because fparser2 originally only supported versions of Fortran up to and including 2003 whereas LFRic makes use of some 2008 features.

Due to the improvements in `fparser2`, it is able to parse the Met Office's UM and LFRic codes, the NEMO code, and the majority of ECMWF's IFS code. As well as being used in PSyclone, `fparser2` is also used by the Met Office in their 'Stylist' formatting checker (<https://github.com/MetOffice/stylist>) and by ECMWF in their 'Loki' DSL, which optimises IFS Physics parameterisations.

**PSyIR** PSyclone was designed to make use of the PSyclone Intermediate Representation (PSyIR) when generating LFRic PSy-layer code. However, the PSyIR was limited to a few specialised nodes to support LFRic code generation. During the ESiWACE2 project the PSyIR has been significantly extended to support more general code constructs in a language-agnostic way and is now also used to represent NEMO code and LFRic kernel code. In order to achieve this, the PSyIR has had to be extended to support many code concepts including `if` statements, `case` statements, `while` loops, `where` statements, array syntax, array slices and structures.

The PSyIR itself does not contain any domain-specific concepts; it is specialised in order to support LFRic concepts and separately specialised to support NEMO concepts. Code can be 'raised' from generic-PSyIR to LFRic-specific or NEMO-specific PSyIR and 'lowered' again from domain-specific PSyIR to generic-PSyIR, which can then be output by PSyclone's backends.

Figure 1 outlines how the raising and lowering works in the NEMO API. The lowest of the three levels is language-specific, the middle layer is language independent and the top layer is domain-specific. The original Fortran code is shown in the bottom left box. This is then parsed by `fparser2` into its Fortran-specific internal representation. This representation is then raised into language-independent PSyIR. The language independent PSyIR is then raised into domain-specific PSyIR capturing any relevant domain-specific concepts for NEMO. Transformations are then applied at this level (indicated by the arrow leaving and returning to the box labelled NEMO IR). Once all transformations have been applied, the NEMO IR is then lowered to language-independent PSyIR and the Fortran back-end is used to lower this to parallel Fortran code. The alternate path via the box labelled SIR is discussed later.

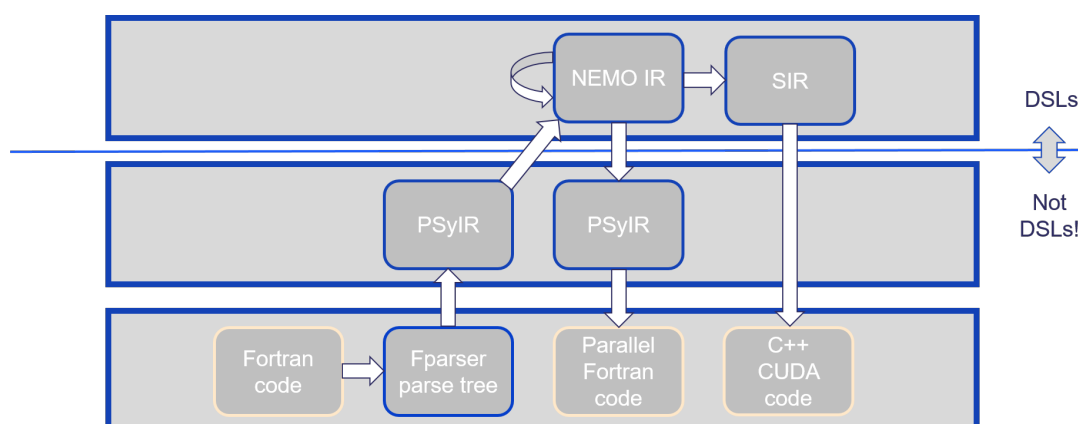


Figure 1: PSyclone NEMO API

**LFRic halo exchanges** In the LFRic API, PSyclone is responsible for the generation of the code that implements both the shared and distributed memory parallelism (see Adams et al. (2019)). In the distributed

memory case this involves the addition of halo exchange calls and global sums at the required locations in the generated code. To determine where halo exchanges and global sums are required, PSyclone follows rules that are based on the metadata describing a kernel and the ordering of kernels specified in invoke calls in the algorithm layer. An advantage of using rules is that the code is guaranteed to be correct. Further, in accordance with the specified rules, the minimum number of required halo exchanges are placed in the code. This is known because in cases where PSyclone is not able to definitively determine that a halo exchange is required, run-time flags are used.

However, whilst the minimum number of halo exchanges are placed according to the rules, this is not necessarily the minimum number required by an application. The reason for this is that the metadata describing kernels is not specific enough in certain cases and halo exchanges may therefore be placed where they are not required.

Before going further it is useful to understand that, in broad terms, when data on a continuous function space is accessed this can potentially result in a halo exchange whereas when data on a discontinuous function space is accessed this typically does not result in a halo exchange.

During the ESiWACE2 project, two cases have been determined where more specific metadata can reduce the number of halo exchanges placed by PSyclone and these are now discussed in turn.

The first case is the addition of `ANY_DISCONTINUOUS_SPACE` metadata. LFRic Kernel metadata specifies which function space a field operates on. In some cases a kernel may be able to be applied to multiple function spaces. In this case the metadata specifies this using the `ANY_SPACE` keyword. If PSyclone sees this keyword then, without any additional information, it must assume that the associated field might be continuous and follow more conservative rules about halo exchange placement. However, scientists sometimes know that a field is discontinuous even though they do not know the particular type of field. By adding the `ANY_DISCONTINUOUS_SPACE` keyword to the LFRic kernel metadata and using this instead of `ANY_SPACE` where possible, PSyclone is able to assume that the field is discontinuous which results in fewer halo exchanges.

The second case is the addition of `GH_WRITE` for an access to a continuous field. When a continuous field is modified in a kernel the access is specified as being `GH_INC`. This indicates that field values (degrees of freedom, hereafter 'dofs') are updated by more than one cell (as they are shared). The use of `GH_WRITE` indicates that the values being written are the same, irrespective of the cell, and therefore it does not matter that the same value might be written multiple times. In this case PSyclone does not need to worry about the outermost halo being dirty on entry to the kernel and this can result in a reduction in the number of halo exchanges required.

**LFRic mixed precision** The LFRic scientists want to make use of mixed precision in their model in order to improve its performance. By using lower precision in appropriate parts of the model, the memory and computational requirements of the code can be reduced, thereby speeding up the model. In LFRic, the solver was tackled in the first instance as it forms a large component of the overall model time and it uses an iterative scheme in which early iterations only require lower precision.

In order to support mixed precision, PSyclone needed to be extended so that it parses the algorithm layer to extract the particular precision specified for LFRic fields and then use this precision when generating the PSy-layer. The actual precision is required in the PSy-layer so that the Fortran compiler can call the kernel with the appropriate precision via a generic interface.

Given that mixed precision kernels now have multiple implementations, PSyclone also needed to be extended to support the idea of a kernel having more than one subroutine implementation.

**LFRic 'i-first' kernels** The data for fields (such as temperature and moisture) in weather and climate models is often stored as 3-dimensional arrays, one dimension each for latitude, longitude and levels. As codes typically use `i`, `j` and `k` as loop iterators, `i` and `j` have become synonymous with latitude and longitude and `k` with levels. Models which store their field data with levels being contiguous in memory are often termed

'k-first' and models which store their field data with latitude or longitude contiguous in memory are often collectively termed 'i-first'.

LFRic uses a k-first data layout for its field data. This was chosen as the horizontal mesh is treated as being fully unstructured, and therefore requires indirection to access neighbour data, whilst the vertical is structured so can be directly addressed. Using a k-first layout hides the cost of indirection as a lookup is only required once for all elements in a column.

However, the Physics parametrisations used by LFRic are independent of each other in the horizontal so do not require a lookup. Further they were originally written as i-first code (and are used this way in the Met Office's current model). In some of the Physics parametrisations, running them as k-first, rather than i-first results in a significant performance degradation. It was therefore decided to support an i-first data layout in LFRic as well as a k-first data layout. Data would need to be re-ordered between kernels with different orderings but tests indicated that there would still be a performance benefit.

PSyclone was extended so that it would support an initial implementation of such i-first kernels. In this initial implementation it was decided to support these kernels by allowing the kernels to manage their iteration space internally rather than a kernel iterating over a single column. The reason for this was due to the complexity of the existing Physics code making any manual restructuring difficult and the fact that this code is also used in the current Met Office model making it simpler to keep the code as single source. Further, it allows PSyclone to provide the data in k-first order and for the kernel to be responsible for the re-ordering of the data from k-first to i-first (and back again). Future implementations will restructure the kernel code and extend PSyclone so that kernels can be i-first without needing to manage the iteration space and data re-ordering.

To support this approach, new metadata was added to support these new LFRic kernels to indicate that PSyclone should not control the iterations and should not create any looping structures in the PSy-layer.

**NEMO full code processing** At the beginning of the ESiWACE2 project, PSyclone was able to generate PSyIR for a large proportion of the NEMO code base. However, the final code-generation still relied upon the underlying fparser2 parse tree. This made implementing transformations difficult and also meant that certain code structures could not be transformed (e.g. one-line `if`- and `where`-statements). Since the focus of the application of PSyclone to NEMO was to achieve performance on GPUs, this was a critical limitation since any un-transformed code would be left to execute on the CPU, incurring the cost of transferring data both from and back to the GPU. A key part of the work done on PSyclone in ESiWACE2 has therefore been to improve the PSyIR and its Fortran frontend and backends such that virtually the entire NEMO code base can be re-created using purely the PSyIR backend.

Once the dependence on fparser2 for generating code had been removed, the next key step was to improve PSyclone's 'coverage' of the NEMO code. Those parts of a code that cannot be captured in PSyIR are represented by `CodeBlock` nodes. In general, PSyclone cannot transform regions containing such nodes and therefore it is important to minimise the number of them, especially in computationally-important parts of the code. Therefore, the work done on the PSyIR (described in Section 4.1.1) to improve the variety of code that can be captured was fundamental to improving coverage and thus the quantity of code that could be ported over to the GPU.

The NEMO build system is based upon the Met Office's FCM which incorporates a pre-processing step. Once a build has completed, the complete set of pre-processed NEMO source files are available and it is these that are the inputs to PSyclone. Since there are approximately 260 source files and 100,000 lines of code in a build of the OCE-SI3 configuration, running PSyclone across all of the code can become complex. This is for two main reasons:

1. since both PSyclone and fparser2 are under development, some source files have features that one or both of them cannot handle;



2. some source files, once processed with PSyclone, reveal bugs in the compiler. (Either the code won't compile or it will crash at runtime).

In order to manage this complexity we have therefore developed a master script ([https://github.com/stfc/PSyclone/blob/master/examples/nemo/scripts/process\\_nemo.py](https://github.com/stfc/PSyclone/blob/master/examples/nemo/scripts/process_nemo.py)) which allows the user to specify those files which must be completely ignored, those to which only profiling calipers will be added (see below) and those which will be fully transformed. For the latter, the master script is responsible for launching PSyclone with the appropriate optimisation script. For adding OpenACC kernels regions to NEMO, this is `kernels_trans.py` ([https://github.com/stfc/PSyclone/blob/master/examples/nemo/scripts/kernels\\_trans.py](https://github.com/stfc/PSyclone/blob/master/examples/nemo/scripts/kernels_trans.py)). Essentially, this optimisation script works its way through the supplied source code, adding the largest possible OpenACC kernels regions. In doing so, it also excludes any code features that are known to cause problems due to compiler bugs. (Over time, the compiler used in this project has improved significantly and therefore the amount of code that must now be excluded is not significant.)

When automatically transforming a large code base, it is impractical for the HPC expert to manually check all of the generated code (and the associated compiler output). Therefore, we have found that the best approach is to use profiling to identify poorly-performing regions of the code. These can then be investigated and the optimisation script updated appropriately. If new PSyclone functionality is required then a manually-modified version of the offending source file can be used in place of the PSyclone-generated one until that functionality is available.

**NEMO transformations** At the outset of the ESiWACE2 project, the primary PSyclone transformation used with the NEMO code base was `ACCKernelsTrans`. This introduces `!$acc kernels` regions into the code which instruct the compiler to automatically decide how to offload to the GPU (including performing dependence analysis to determine whether a loop may be safely parallelised). Movement of data between CPU and GPU was handled by NVIDIA's 'managed memory' technology.

However, OpenACC directives can only be used to target NVIDIA hardware: if one wishes to make use of AMD or Intel hardware (from Fortran) then OpenMP offload directives (introduced with version 4.5 of the OpenMP specification) must be used instead. Further, 'managed memory' is an NVIDIA feature and, although other manufacturers are also developing 'Unified Memory' technologies, they are significantly less mature than NVIDIA's offering.

In addition to these GPU considerations, ECMWF have a requirement that the version of NEMO used in their operational suite be parallelised using OpenMP (this is due to the way it is coupled to their IFS atmosphere model). Currently they maintain a separate, manual port of NEMO in order to provide this functionality but it would be preferable if an OpenMP version could be generated automatically by PSyclone.

Consequently, in ESiWACE2 we have developed new transformations in PSyclone that are capable of adding both OpenMP thread-parallel and OpenMP offload directives to NEMO source. This has proved significantly more challenging than OpenACC for two reasons:

1. there is no equivalent of OpenACC's `kernels` directive, and
2. OpenMP directives can only be applied to explicit loops.

The first of these challenges means that PSyclone must take on the responsibility of determining which loops can be safely parallelised. The second challenge is a problem because NEMO makes very heavy use of Fortran array syntax in order to keep the code as concise as possible. We have therefore implemented two transformations (`Reference2ArrayTrans` and `ArrayRange2LoopTrans`) which may be used to replace all uses of array syntax with explicit loops. Alongside this (and outside of ESiWACE2), dependence-analysis functionality has been developed for the PSyIR such that it is now possible to check whether a loop may be safely parallelised. This functionality is now used by the various OpenMP and OpenACC transformations that add explicit parallelism to a code.

It is common practice in NEMO for a subroutine to have its own, local work arrays, often allocated on the stack ('automatic arrays' in Fortran terminology). These create a performance problem when running with managed memory on GPU: each time the subroutine is called, a new array is allocated on the CPU and then copied over to the GPU when it is accessed there. In order to avoid this problem, we have implemented the `HoistLocalArraysTrans` transformation which automatically identifies such arrays and promotes them to module scope, adding a suitable `allocate` statement to the subroutine body. In this way, such work arrays are allocated just once on the GPU.

In addition to these performance-oriented transformations, we have also been working on functionality to provide an alternative to the use of NVIDIA's managed-memory technology for handling data movement between GPU and CPU. In the LFRic API, computation is guaranteed only to occur within kernel subroutines and therefore it is relatively straightforward to ensure that data remains resident on the GPU by adding appropriate `!$acc enter data` directives to generated PSy-layer code. `!$acc enter data` is used since it copies data to the GPU the first time it is encountered, leaving it there for the remaining duration of the program. Since the NEMO API does not have the physical PSyKAI separation of concerns, it is much harder to ensure that data remains resident on the GPU. After processing with PSyclone in the usual way, a given routine will, in general, have regions of computation contained within `!$acc kernels` directives and other regions that are not and therefore will execute on the CPU. (Ideally, no computation would happen on the CPU but I/O and code regions that PSyclone cannot understand and/or that the compiler cannot handle are currently left on the CPU.) Data movement for such a routine is then handled in a two-step process:

1. An `!$acc enter data` is added for all variables that are accessed within the code region(s) that are within `!$acc kernels`;
2. For each region of code that will execute on the CPU, an `!$acc update` directive is inserted to bring any necessary data back from the GPU first. After each such region, another `!$acc update` directive is inserted to send any modified data back to the GPU.

Experiments with this implementation showed that this strategy could result in inefficient code and therefore step 2 was modified such that `!$acc update` directives are only inserted a) before an `!$acc kernels` region that accesses the modified data, in which case only the modified data that is accessed within the `!$acc kernels` is updated, or b) at a procedure boundary, including at calls to other procedures, in which case we conservatively update all the modified data. Note that, as the same procedure might have multiple call sites, we cannot statically delay the update to happen inside the procedure since we do not know what data needs updating until runtime. This functionality is now working for the tracer-advection benchmark (originally extracted from NEMO by Silvia Mocavero of CMCC, see Porter et al. (2017)) and we are experimenting with applying it to the full NEMO code.

As discussed in the previous section, applying transformations automatically across a large code base can be difficult. It is therefore very useful to be able to profile the performance of the resulting code so that any problematic regions can be identified. Since much of our work has focused on making use of NVIDIA GPUs, the profiler we have used the most has been NVIDIA's Nsight. As with NVIDIA's compiler, the functionality of this tool has been improving over time but at the outset of this project, although it was good at showing what was happening on the GPU, crucially it provided no information if a section of code was not running on the GPU. We have therefore extended PSyclone's PSyData API to support NVIDIA's nvtx (NVIDIA Tools eXtension) library. The PSyclone optimisation script was then also extended such that profiling calipers were automatically added around those regions of the code where no OpenACC directives had been placed. The value of this approach is illustrated by the example screenshot taken from Nsight in Figure 2.

**SIR backend** In order to support the interoperability between PSyclone and dusk/dawn a new PSyclone back-end was developed to translate from PSyclone's PSyIR internal representation into dusk/dawn's Stencil Intermediate Representation (SIR). A benefit of this interoperability is that Fortran code input into PSyclone

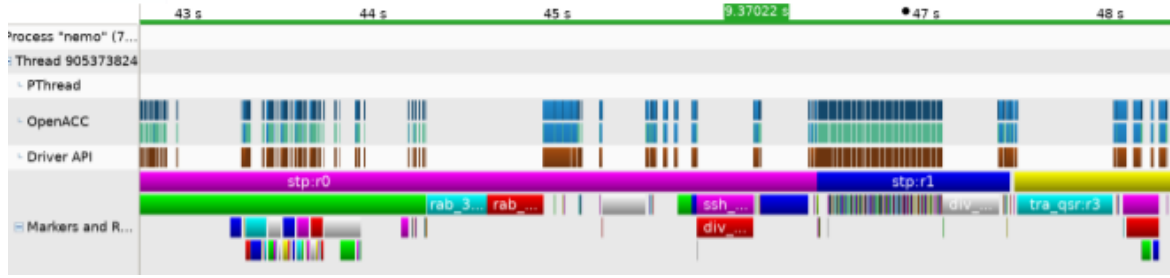


Figure 2: A timeline produced by NVIDIA's Nsight profiler for a run of NEMO on a single GPU at a very early stage of this work. Note that the origins of the blank spaces in the rows labelled 'OpenACC' can be identified thanks to the profiling-marker information in the rows below.

using the NEMO API can potentially be output using dawn's backend, which can produce optimised CUDA code. This is discussed further in deliverable D2.2 (due in project month 49). A high level view of this can be seen in Figure 1.

PSyclone supports backends which are implemented as Visitors. These visitors translate PSyclone's PSyIR into the target representation. The implementation of these visitors in PSyclone makes it intuitive and relatively simple to create new backends and there is documentation on how to do this in the PSyclone developers guide (see [https://psyclone-dev.readthedocs.io/en/stable/psyr\\_backends.html](https://psyclone-dev.readthedocs.io/en/stable/psyr_backends.html)).

As there was mostly a natural mapping between PSyIR nodes using the NEMO API and the Python SIR binding (see Deliverable D2.2 for more details) it was not too difficult to implement the mapping as a visitor. The example listing below shows that a PSyclone backend subclasses PSyIRVisitor. It also shows a slightly simplified version of the mapping from a PSyIR IfBlock node to an SIR Python `make_if_stmt` function. As can be seen, a PSyIR IfBlock contains a `textttcondition`, an `if_block` and an optional `else_body`, which, once mapped to SIR, then make up the arguments to the SIR's `make_if_stmt`.

```
class SIRWriter(PSyIRVisitor):
    # ...
    def ifblock_node(self, node):
        cond_expr = self._visit(node.condition)
        cond_part = f"make_expr_stmt({cond_expr})"
        then_statements = self._visit(node.if_body)
        then_part = f"make_block_stmt(['{then_statements}'])"
        if node.else_body:
            else_statements = self._visit(node.else_body)
            else_part = f"make_block_stmt(['{else_statements}'])"
        else:
            else_part = "None"
        return f"make_if_stmt({cond_part}, {then_part}, {else_part}),\n"
```

The performance results of using PSyclone and dawn together via PSyclone's SIR backend compared with using PSyclone's standard Fortran back-end for a NEMO benchmark are presented in Section 4.2.

#### 4.1.2 dawn Improvements

The DSL toolchain dusk/dawn is the first version of high-level python DSL used for ICON. It is mainly developed by MeteoSwiss and freely available on GitHub (<https://github.com/dawn-ico/dusk> and <https://github.com/MeteoSwiss-APN/dawn>). At the beginning of the ESiWACE2 project the DSL backend dawn was only able to accept stencil computations on regular lattices. However, ICON is a model that discretises the continuum using an icosahedral triangular mesh. This new requirement needed a rich set of additional

features to be implemented in dawn. To expose these features to the user, a new front end that is purpose built for more general meshes, called dusk, was also designed and implemented. As the IFS-FVM model uses a regular lattice, in principle, dawn could have been used as its front-end language. However, due to effort constraints in ESiWACE2 and developments in related projects the GT4Py front end was used instead, see the paragraph "Transition to GT4Py" for more information. The most important features of the dusk and dawn toolchain are highlighted in the remainder of this section.

**Language Features** The key conceptual difference between computations on a regular lattice and computations on a more general mesh are

- Variables are associated with a location type, i.e. Edge, Cell or Vertex on the general mesh<sup>2</sup>
- Neighbours cannot be accessed by integer offsets along a coordinate axis on the general mesh; a more expressive concept is needed to access the neighbours

The first requirement was addressed by elevating the location of a quantity to a fundamental type of the languages' type system, while the second was realised by introducing a generalised reduction operator. These two features are highlighted in Figure 3 and Figure 4.

```

stencil lap {
  storage in, out;
  Do {
    vertical_region(k_start, k_end) {
      out = - 4*in[i,j]
            + in[j - 1] + in[j + 1]
            + in[i - 1] + in[i + 1];
    }
  }
};

```

```

@stencil
def lapl(inF: Field[Vertex],
        outF: Field[Vertex]):
    with domain.upward:
        outF = -4*inF +
               sum_over(Vertex>Edge>Vertex, inF)

```

Figure 3: The same discrete finite difference Laplacian implemented in gtclang (Osuna et al., 2020) and dusk. Dusk highlights that the input and output fields are residing on vertices, and that the east, west, north and south neighbours of a point can be reached by traversing the mesh to the neighbouring vertices via the edges.

Even though the stencil presented in Figure 3 is simple, it already highlights the necessity of one of the salient features of the reduction concept; the neighbour chains. While the most basic Finite Volume stencils may indeed only look at the closest neighbours to any given location type, e.g. the edges to a cell when computing the divergence, a lot of computations in ICON do in fact require neighbourhoods that are quite a bit more complex. Examples of this include Finite Difference stencils on the triangular mesh or interpolation methods. To support such computations, the user can define a "neighbour chain", i.e. instruct the toolchain to visit an ever increasing neighbourhood by stating which elements (Cell, Edge, Vertex) to visit next. Two such examples are presented in Figure 5.

However, if one studies even the most basic discretised equations in (Zängl et al., 2015) one will notice that there is yet another ingredient missing. Consider:

$$\nabla \phi_c = \frac{1}{A_c} \sum_{e \in \mathcal{N}_e(c)} \phi_e l_e f_e \quad (1)$$

where  $\phi$  denotes some flux,  $A_c$  the cell area,  $\mathcal{N}_e(c)$  the set of edge neighbours  $e$  to cell  $c$ ,  $l_e$  the length of edge  $e$ , and  $f_i$  a factor that orients the flux across the edge  $i$  such that it points outside the cell  $c$ . Trying to

<sup>2</sup>Technically, this is also true for most codes operating on Cartesian meshes. However, this fact was not represented in dawn, and the responsibility to distinguish between quantities located on different location types is placed on the user.

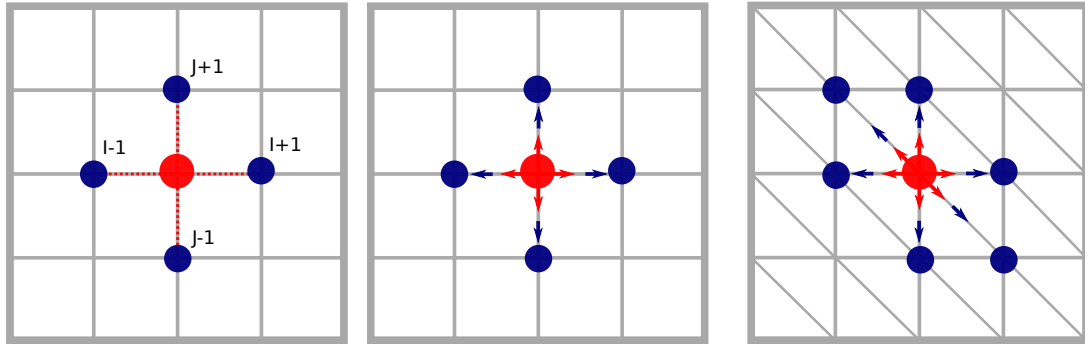


Figure 4: left: accessing the vertex neighbours of a vertex on a Cartesian mesh using the unique property of regular lattices that neighbours can be uniquely identified using offsets along the coordinate axis. Middle: accessing the same neighbourhood by identifying the 4 edge neighbours to a vertex, and subsequently identifying the vertex neighbours to said edges. Right: Same as middle, but now on a hexagonal (possibly icosahedral) triangle mesh. Note that on this mesh, the approach on the left would not be applicable, since neighbours can not be identified using offsets.

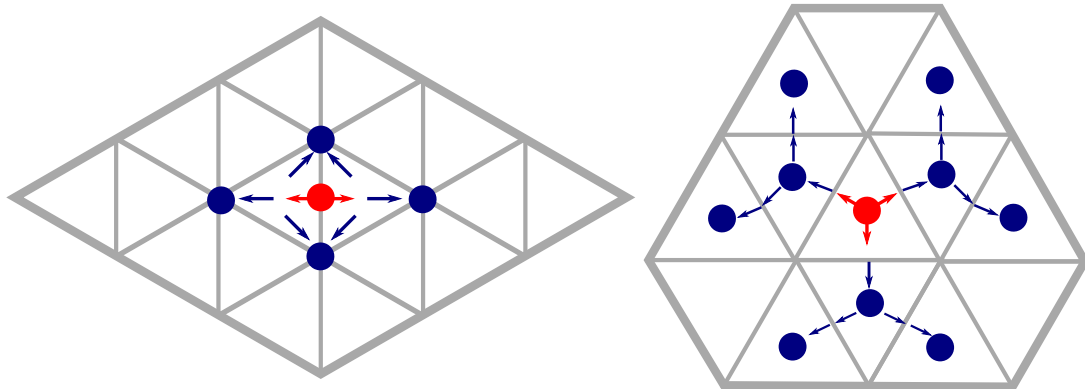


Figure 5: Two neighbourhoods larger than the immediate neighbours to an element ("neighbour chains"). Left: The Laplacian Finite Difference stencil can be recovered on a triangular mesh by considering the centre of an edge, and the 4 vertex neighbours adjacent to the two cells neighbouring the original edge (Edge > Cell > Vertex). Right: neighbourhood as used in the Radial Basis Function interpolation (Franke, 1982) in ICON. 9 neighbouring cells are considered to interpolate a quantity from cells to cells (Cell > Edge > Cell > Edge > Cell).

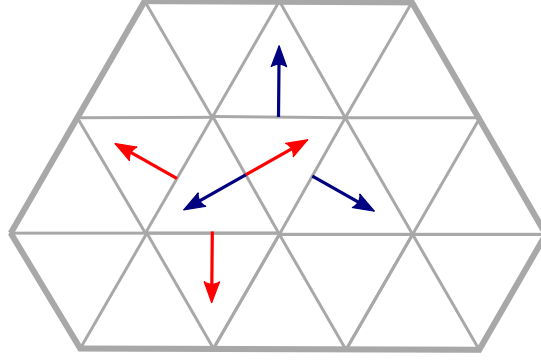


Figure 6: The need for sparse fields illustrated by an orientation factor that orients cell normals such that they point outwards of the cell,  $f_e$  in Equation 1. As can be seen, two neighbouring cells sharing an edge can not share the same orientation. Thus, two signs need to be stored on this edge. Or alternatively three signs need to be stored per cell.

implement this equation into a computer program it will quickly be noticed that the quantity  $f_e$  cannot be stored on the edges. In fact, if this quantity is to be precomputed, each cell needs to store 3  $f_e$  for each of its neighbouring edges. See also Figure 6 which illustrates this problem. Such quantities that are defined on the neighbours of a location are called "sparse dimensions" in dusk, and their usage is illustrated in Figure 7.

```
@stencil
def div(phi_c: Field[Cell], A_c: Field[Cell],
        phi_e: Field[Edge], l_e: Field[Edge], f_e: Field[Cell>Edge]):
    with domain.upward:
        phi_c = 1/A_c*sum_over(Cell > Edge, phi_e*l_e*f_e)
```

Figure 7: Equation 1 implemented in dusk. Note the sparse field  $f_e$  that is located on  $\text{Cell} > \text{Edge}$ , i.e. a sparse dimension.

Finally, a concept is offered to simplify trivial sparse dimensions. Consider:

$$\nabla_{\underline{n}}\psi_e = \frac{1}{\hat{l}_e} (\psi_{c_1} - \psi_{c_2}) \mid c_{1,2} \in \mathcal{N}_e(e) \quad (2)$$

where  $\nabla_{\underline{n}}\psi_e$  is the directional derivative of  $\psi$  along edge normal  $\underline{n}$  of edge  $e$ ,  $\hat{l}_e$  is the dual edge length and  $c_{1,2}$  are the cell neighbours of  $e$ . Of course, this equation could be implemented in dusk using a sparse dimension to reflect which of the  $\psi_{c_i}$  to multiply with  $+1$ ,  $-1$ , respectively. However, if the mesh data structure is designed in such a way that the first cell neighbour to an edge that is returned is always the one which is pointed at by the edge normal, the sparse dimension would become trivial, storing the same two values for each edge. This optimisation is used heavily throughout the ICON model. To reflect this, the "weights" concept was introduced. Its usage is illustrated in Figure 8.

```
@stencil
def grad_n(psi_e: Field[Edge], l_hat: Field[Edge],
           psi_c: Field[Cell]):
    with domain.upward:
        psi_e = 1/l_hat*sum_over(Edge > Cell, psi_c, weights = [1, -1])
```

Figure 8: Equation 2 implemented in dusk. Note the usage of the optional argument `weights` which can be used to express a sparse field that is constant throughout the domain.

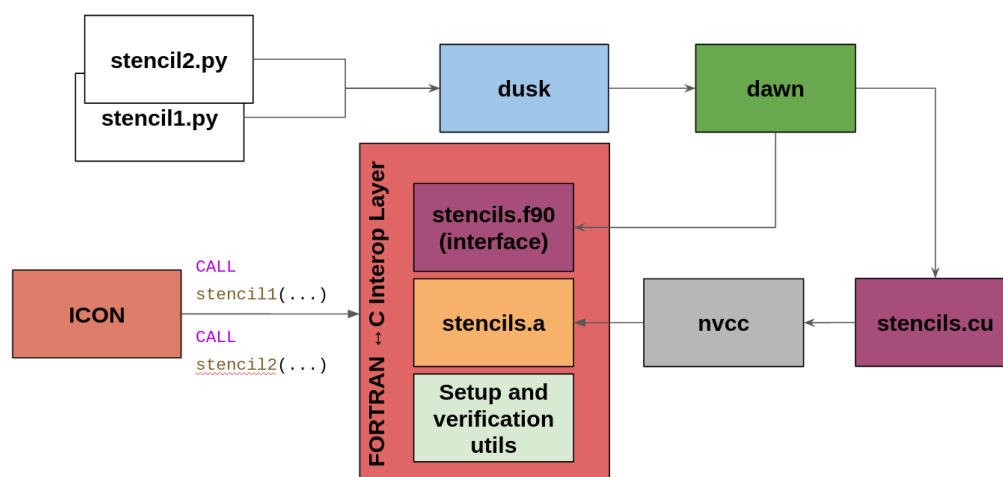


Figure 9: Schematic view of the integration facilities provided by the dusk / dawn (and GT4Py) toolchain. The components shown in purple are emitted by the toolchain. The set-up and verification utilities are provided as library functions shipped with the toolchain and called in the wrapper functions to the stencil. See also deliverable 2.2 for more details.

**Integration** Originally, dawn was only emitting a single file containing the stencils implementation in the desired backend language. For the dawn features described above, the only backend languages available in dawn are C++ and CUDA. The C++ backend is intended for debugging, and hence not required to interface with Fortran. The CUDA code, however, is the performance code the dycore stencils are translated into. Thus, facilities to call these CUDA stencils from the Fortran driver ("bindings") are required.

A substantial amount of engineering was put into the automatic generation of these bindings. Basically, for each stencil a C header file is generated, to which a Fortran interface which is also generated may bind to by means of the Fortran iso C bindings. The functions contained in this Fortran interface can then be called by the driver code. Additionally, logic is injected into these bindings that enable automatic checking of the stencil code against reference code computed in Fortran as well as serialisation in case said verification fails. Please see Figure 9 for a schematic overview of the bindings and ESiWACE2 Deliverable D2.2 *Demonstration of the DSLs in the proposed models*, due in project month 49, for a detailed explanation.

**Transition to GT4Py** Whilst the dusk/dawn toolchain was under active development for the most part of the ESiWACE2 project, it was superseded by the declarative dialect of GT4Py in 2022. The development of dusk/dawn was frozen and the dynamical core (dycore) was ported to this new DSL. The ported code is available at <https://github.com/C2SM/icon4py> (access on demand). The GT4Py declarative dialect follows a different programming paradigm than dusk; functional instead of imperative. Its frontend language is also different in the sense that it adapts a different perspective on the computations, where a complete field is considered at once ("FieldView"), whereas dusk highlighted processing each element (Vertex, Edge or Cell) individually.

There is also an earlier structured dialect of GT4Py. This dialect is general enough to support IFS-FVM. In fact, no extensions to GT4Py were required to support this model, be it for global or limited area mode. Please see deliverable D2.2 for a detailed description of the IFS-FVM model in GT4Py.

GT4Py, be it in structured or declarative dialect, is also embedded into python. From now on in this text, "GT4Py" shall indicate the declarative dialect of GT4Py. Consider Figure 10 for a side-by-side comparison of example Figure 7 between dusk and GT4Py.

The complete dry dycore of ICON was ported to GT4Py. On the driver side, only minimal changes were

```

@stencil
def div(phi_c: Field[Cell],
        A_c: Field[Cell],
        phi_e: Field[Edge],
        l_e: Field[Edge],
        f_e: Field[Cell>Edge]):
    with domain.upward:
        phi_c =
            1/A_c*sum_over(Cell > Edge,
                           phi_e*l_e*f_e)

@field_operator
def div(
    A_c: Field[[CellDim], float],
    phi_e: Field[[EdgeDim], float],
    l_e: Field[[EdgeDim], float],
    f_e: Field[[C2EDim], float]
) -> Field[[CellDim], float]:
    phi_c =
        1/A_c*neighbor_sum(
            phi_e(C2E) * l_e(C2E) * f_e,
            axis=C2EDim)
    return phi_c

```

Figure 10: Side-by-side comparison of the Equation 1 implemented in dusk (left) and GT4Py (right). dusk highlights the local perspective by stating the iteration domain explicitly and hiding all offsets in the neighbour loop implied by `sum_over`. GT4Py highlights the view on full fields, e.g. by requiring the dense fields inside the `neighbor_sum` statement to be "shifted" by C2E explicitly and stating that the `neighbor_sum` happens along the axis labelled C2EDim.

required since the integration features in Section 4.1.2 were ported to the GT4Py toolchain. Since both GT4Py and dusk are concise DSLs embedded into ICON, the translation process from dusk to GT4Py was quite straight forward, especially since dusk stencils could be replaced individually by GT4Py stencils and tested within the ICON model.

## 4.2 Comparison of DSLs

This section provides the evaluation part of this report and focuses on a NEMO benchmark. The reason for choosing this NEMO benchmark is that the complete code can be targeted by both the PSyclone and dusk/dawn DSLs, thereby allowing their comparison.

The NEMO benchmark is for tracer advection (the propagation of quantities such as salinity) which is one of the most computationally costly routines in the model. The original version of the tracer-advection benchmark was constructed by Silvia Mocavero of the Euro-Mediterranean Center on Climate Change (CMCC) as part of the IS-ENES Project (Porter et al. (2017)). It is a standalone program based upon the MUSCL tracer-advection scheme implemented in NEMO. Note that this benchmark code also forms the core of one of the ESCAPE-2<sup>3</sup> Weather and Climate Dwarfs<sup>4</sup> where it is embedded within a more generic routine in order to emulate time-stepping behaviour. The benchmark is contained within the 'PSycloneBench' repository hosted on GitHub: <https://github.com/stfc/PSycloneBench>.

The original tracer-advection benchmark is written as a single Fortran program which does not separate the benchmark setup (array allocation, initial values etc.) and completion (array de-allocation, checking of results etc.) from the computation part. This lack of separation is problematic when transforming to SIR, as the SIR is designed to capture the computational part of a program. In order to concentrate on the translation from PSyIR to SIR it was decided to manually create a modified version of the original tracer-advection benchmark where the computation is provided in a separate computational-kernel subroutine. In a production implementation we would expect PSyclone to separate the computational part of the code itself, however this has not yet been implemented. For more details on the benchmark and any required PSyclone transformations to allow the code to be transformed to SIR, please see Deliverable D2.3 *Demonstration of PSyclone-CLAW using the GridTools toolchain with the proposed benchmarks, making use of the HIR* (confidential).

<sup>3</sup><https://hpc-escape2.eu>

<sup>4</sup><https://hpc-escape2.eu/sites/default/files/2021-11/ESCAPE-2-D3-5-V1-0.pdf>



When writing Deliverable D2.3 a number of issues were found with the generated CUDA code. To work round these problems 1) the computational-kernel subroutine was further split into 10 parts, 2) any scalar constants written outside of a loop were removed and where they were read the constant was replaced by its value, and 3) local variables first written to within an if statement were given an initial value before the if statement.

Since Deliverable D2.3 was written, improvements have been made to both PSyclone and dawn so that the computational-kernel subroutine no longer needs to be split into separate parts. Further, as reported in Deliverable D2.3, the Fortran intrinsics were originally replaced by equivalent code before translating to SIR. However, dawn does support intrinsics. Therefore, PSyclone's SIR backend was extended to output the appropriate SIR code for intrinsics and the associated PSyclone transformations were removed from the script. The only special case was the Fortran SIGN function, where there is no direct C equivalent. This function was implemented in the backend in the following way: [Fortran] SIGN(A,B) => [C] FABS(A)\*SIGN(B). The use of a single computational kernel and native intrinsics allows for a fair comparison between PSyclone-generated Fortran code with directives and dawn-generated CUDA code.

#### 4.2.1 Benchmark and Machine Configuration

The analysis of the PSyclone- and dawn-generated versions of the benchmark have been performed on an NVIDIA V100 GPU (with 32GB of memory) hosted by an Intel CPU.

For the Fortran and OpenACC forms of the code, version 22.11 of the NVIDIA HPC-SDK was used for compilation with the following flags: `-O3 -acc=gpu -gpu=managed`. Since Deliverable D2.3 was written, PSyclone has been extended such that it is capable of adding data-movement directives to the generated code. This removes the need for NVIDIA's 'managed memory' and thus this additional version of the code is compiled without the `-gpu=managed` flag. For consistency with other GPU vendors, we use the term 'unified memory' (UM) rather than 'managed memory' and therefore results obtained using this technology have 'UM' in their description.

Each of the versions of the benchmark has been run for a range of (square) horizontal domain sizes varying from  $8 \times 8$  up to  $1024 \times 1024$ . The very small domain sizes are not actually large enough to exercise the GPU but are included so as to permit comparison with CPU performance. The number of vertical levels has been held constant at 75 since this is the value used in the eORCA025 and ORCA12 configurations of NEMO.

In order to normalise the results, the quantity plotted is 'performance' in millions of grid points updated per second, calculated as  $N_{points}/(\delta t \times 10^6)$  where  $N_{points} = N_{side} \times N_{side} \times 75$  and  $\delta t$  is the measured elapsed time for a single iteration. This quantity was calculated by running the benchmark for a large number of iterations (typically 2000 for domains up to  $256^2$  reducing to 500 for the larger domains) and computing the average time per iteration, excluding the very first timestep. (Thus excluding any one-time setup costs which will be negligible in a production run consisting of many thousands of time steps.)

We have measured the performance of various forms of the benchmark, all of which begin with the same Fortran source described earlier (all of the compute kernels contained within a single subroutine):

1. Fortran with compute kernels contained in a single subroutine, CPU (serial);
2. Fortran transformed by PSyclone to add OpenACC KERNELS regions;
3. Fortran transformed by PSyclone into SIR-compliant form, CPU (serial);
4. SIR-compliant form further transformed to add OpenACC KERNELS regions;
5. As above but with explicit data-movement directives added by PSyclone;
6. CUDA version generated by dawn from the SIR produced by PSyclone.

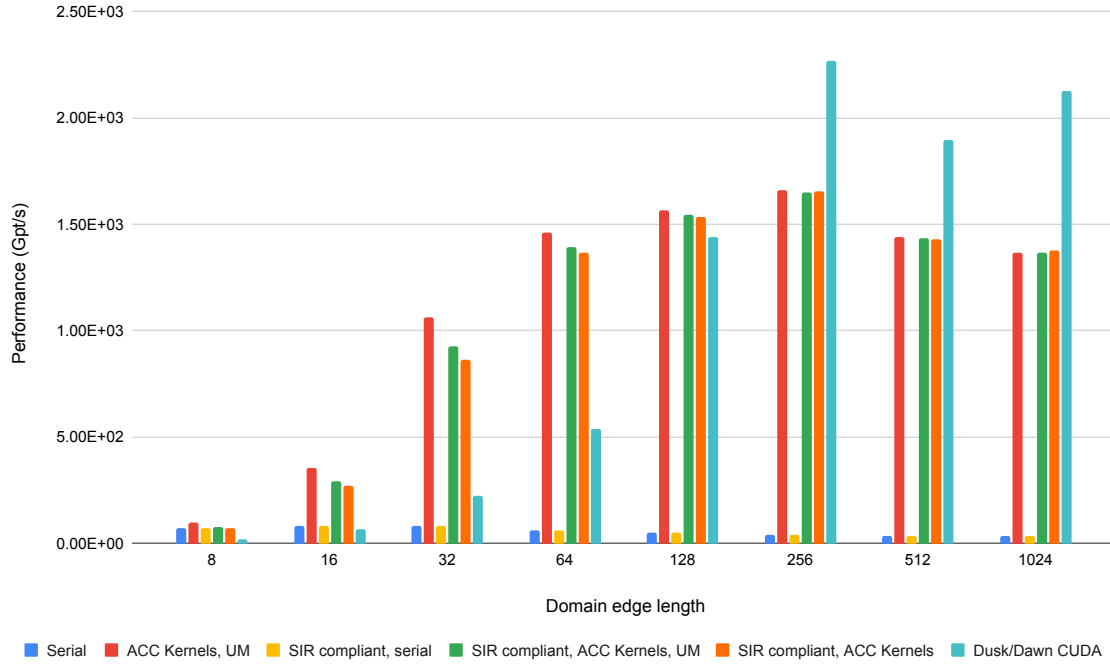


Figure 11: Performance comparison of various forms of the tracer-advection benchmark.

The Fortran driver for the CUDA version is based on that for the original but has been extended so as to first copy data over to the device (using OpenACC) and then to ensure that the call to the compute routine is passed the addresses of the various arrays on the device rather than the host. The call to the compute routine directly from Fortran is possible thanks to the Fortran interface code that dawn generates.

#### 4.2.2 Performance Results

Figure 11 shows the results obtained for the six versions of the benchmark. Considering the results for the two serial forms of the benchmark, it can be seen that transforming the source into a form suitable for conversion into SIR has a negligible effect on performance. Of the OpenACC results, the version that has *not* been transformed into a SIR-compliant form gives the best performance up to a domain size of  $128^2$ . For domains larger than this, the performance of the three different OpenACC versions is very similar.

Moving now to compare the performance obtained with OpenACC with that obtained from the dawn-generated CUDA version, we can immediately see that at large problem sizes ( $256^2$  and greater) the latter gives a significant up-lift in performance. For both tool chains, peak performance is obtained when the domain is  $256^2$ ; beyond that performance degrades. Figure 12 shows the duration of each kernel launched during a single time-step of the OpenACC version of the benchmark for both the  $256^2$  and  $512^2$  domains. If we assume that a kernel is achieving good streaming performance, then its duration should be proportional to the domain size. Figure 12 highlights those kernels where this is not the case. We have then investigated the performance of one of those kernels (the leftmost one in Figure 12) using NVIDIA's 'Nsight Compute' tool. This tool reports (unsurprisingly) that the kernel is extremely memory-bandwidth bound and the performance drop-off can be attributed to a fall in the L2 Cache hit rate as the problem size is increased (Figure 13). In turn, this is due to stencils resulting in accesses with large strides.

Dawn optimises for this situation by performing blocking and redundant computation. This reduces the stride-length of memory accesses, improving cache performance. For the  $1024^2$  domain, the memory throughput achieved for the most costly kernel is 737GB/s for the dawn-generated code but 616GB/s for the corresponding OpenACC solution. The main reason for this seems to be that dawn's heavily optimised CUDA

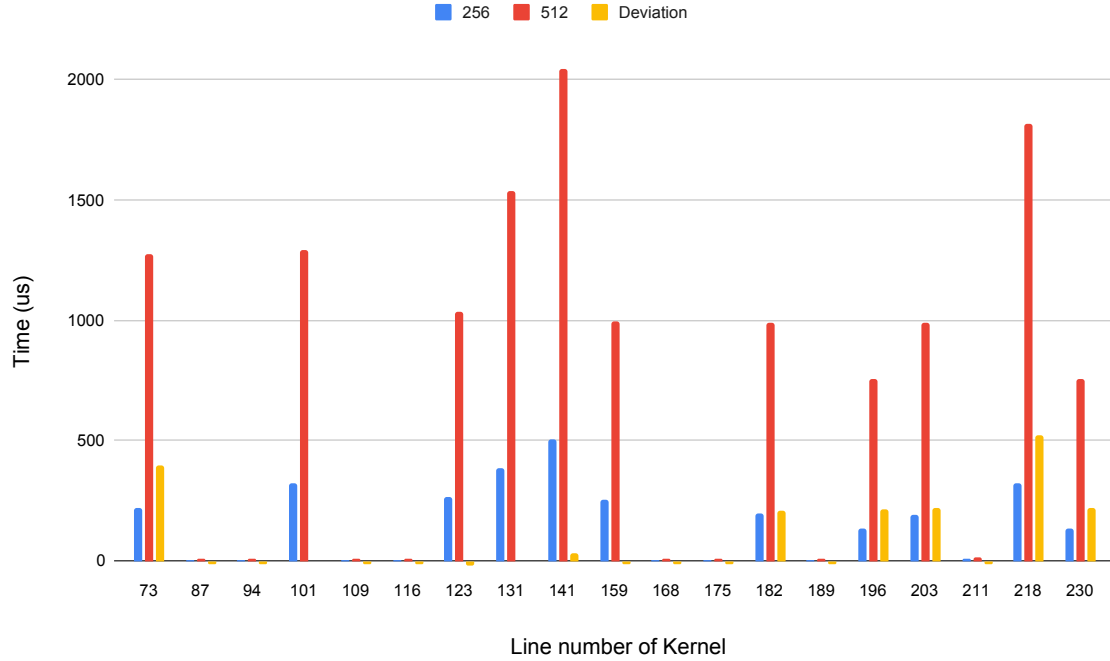


Figure 12: Duration of each kernel launched during a single time-step of the OpenACC version of the benchmark for the  $256^2$  and  $512^2$  domains ( $t_{512}$  and  $t_{256}$ , respectively). The orange bar shows the result of  $t_{512} - 4 \times t_{256}$  and thus highlights those kernels that perform less well for the  $512^2$  domain.

code succeeds in using fewer GPU registers (30 versus 52) which in turn allows a theoretical occupancy of 100% instead of 56% for the OpenACC version. The ability to run more threads on the device improves latency-hiding and thus the dawn-generated code is able to achieve significantly better memory throughput.

At small problem sizes Figure 11 shows that the OpenACC versions tend to strongly outperform the CUDA version. Our investigations with ‘Nsight Compute’ have shown that this is because the optimisations that dawn performs and that are so beneficial at larger problem sizes are actually detrimental to performance in this regime because they result in fewer threads being spawned on the device. More work is required to allow these optimisations to be disabled in dawn.

The comparison of the OpenACC versions with and without unified memory is interesting: for domains of  $128^2$  and larger the version without UM performs very slightly better. However, for the smaller domains the situation is reversed with the UM version performing better. We have investigated this using NVIDIA’s Nsight profiler and Figure 14 compares the time-lines for runs with and without UM for the  $64^2$  domain. There appear to be two competing effects: without UM, the kernels run slightly faster (compare the width of the top, blue horizontal lines showing GPU activity) but the gaps between the timesteps (large white space at either side of the blue bars) are shorter in the UM version. The reason for the latter difference is that PSyclone’s solution for explicitly managing data movement makes use of the `enter` data directive. While this is successful in ensuring that data is only moved over to the GPU the first time the compute subroutine is called, it turns out that there is a cost associated with it, even when no data is moved. By inserting timing calipers around the `enter` data directive we measured this cost to be 0.013 ms or roughly 25% of the measured whitespace (excluding the very first time the region is visited as that is when the data is transferred to the GPU). For domains larger than  $64^2$  it therefore seems likely that the beneficial effect of the slightly faster kernels outweighs the additional cost incurred by the `enter` data directive.

Overall, the benchmark displays two regions with each DSL performing best in one of them. PSyclone

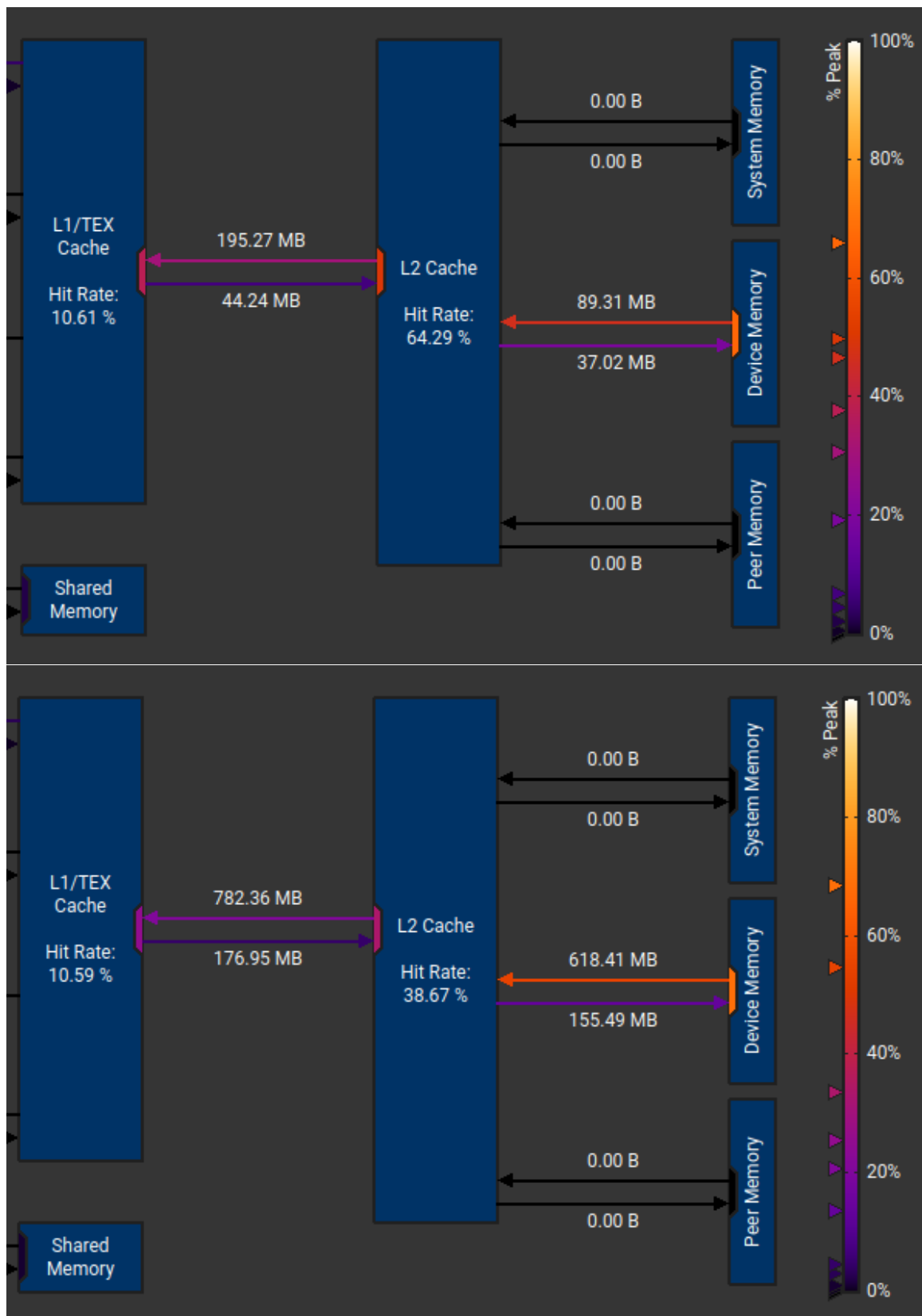


Figure 13: Memory-bandwidth reports for the first kernel running with domains of  $256^2$  (top) and  $512^2$  (bottom).

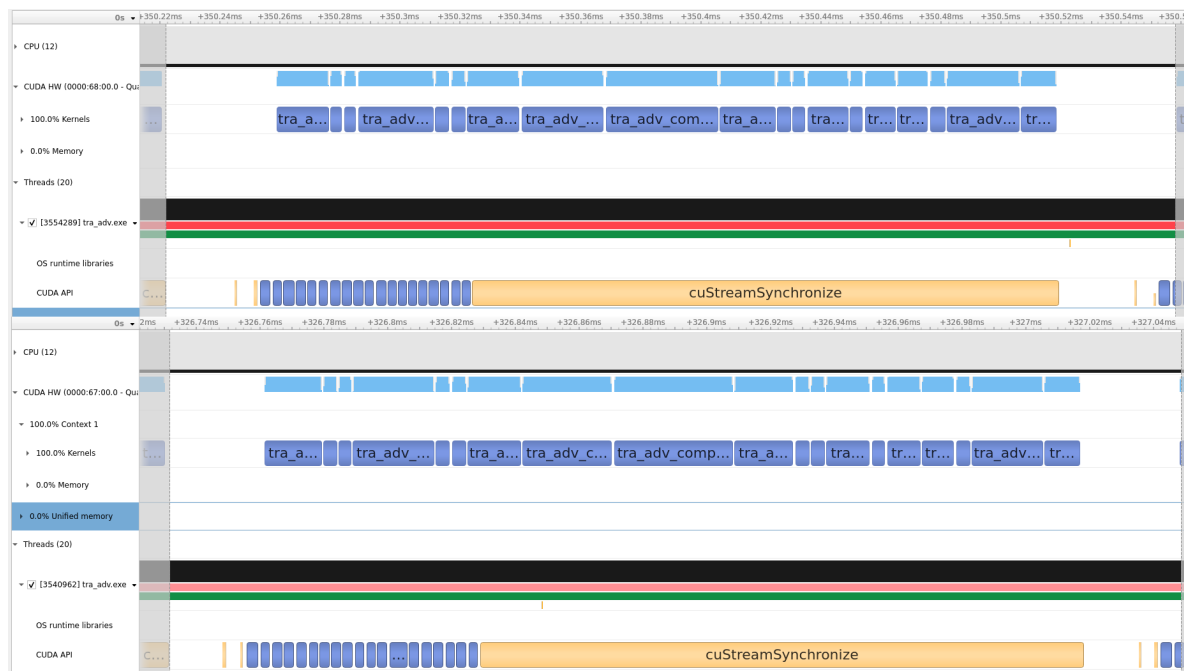


Figure 14: Comparison of the Nsight time-lines for a single time-step in the versions with (bottom) and without (top) UM for the  $64^2$  domain.

gives the best performance for smaller problem sizes and dawn gives the best performance for larger problem sizes. By combining the two tools, it is possible for either solution to be produced from the same source code ensuring that the best solution can be used depending on the required problem size. This improves overall performance and performance portability. Of course each tool could potentially add or remove optimisations that improve their performance to be closer or even match the other but in practice it is often the case that different solutions perform better in different regimes and it is difficult to provide the best solution in all cases. Combining solutions is one way to get round this problem.

## References

- S.V. Adams, R.W. Ford, M. Hambley, J.M. Hobson, I. Kavčič, C.M. Maynard, T. Melvin, E.H. Müller, S. Mullerworth, A.R. Porter, M. Rezny, B.J. Shipway, and R. Wong. Lfric: Meeting the challenges of scalability and performance portability in weather and climate models. *Journal of Parallel and Distributed Computing*, 132:383–396, 2019. ISSN 0743-7315. doi: <https://doi.org/10.1016/j.jpdc.2019.02.007>. URL <https://www.sciencedirect.com/science/article/pii/S0743731518305306>.
- Richard Franke. Scattered data interpolation: Tests of some methods. *Mathematics of computation*, 38(157): 181–200, 1982.
- Gurvan Madec and NEMO System Team. *NEMO ocean engine*. URL <https://www.nemo-ocean.eu/>.
- Carlos Osuna, Tobias Wicky, Fabian Thuring, Torsten Hoeffler, and Oliver Fuhrer. Dawn: a high-level domain-specific language compiler toolchain for weather and climate applications. *Supercomputing Frontiers and Innovations*, 7(2):79–97, 2020.
- A. R. Porter, R. W. Ford, S. Mocavero, S. Fiore, and G. Aloisio. Application of the psykal approach to the nemo ocean model. Technical report, 2017. URL [https://is.enes.org/archive-1/phase-2/documents/na2-working-documents/application-of-the-psykal-approach-to-the-nemo-ocean-model/at\\_download/file](https://is.enes.org/archive-1/phase-2/documents/na2-working-documents/application-of-the-psykal-approach-to-the-nemo-ocean-model/at_download/file).

Günther Zängl, Daniel Reinert, Pilar Rípodas, and Michael Baldauf. The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, 141(687):563–579, 2015.

## 5 Changes made and/or difficulties encountered, if any

The title of this deliverable talks about the evaluation of benchmarks and the original expectation was for a number of benchmarks to be transformed. However, as already explained in Deliverable D2.3, it is only feasible to compare the two DSLs when they support a common domain. In our case the overlapping domain (structured finite difference codes) happened to fit well with the NEMO code, therefore work concentrated on a single benchmark associated with that.

This deliverable's title also talks about the evaluation of the models. Given that the two DSLs have been developed for different models in this project, their evaluation has concentrated on scalability and performance, which will be presented in Deliverable D2.4 *Report on the performance of DSL compilers with the proposed models* (due in project month 50).

## 6 How this deliverable contributes to the European strategies for HPC

Domain-specific languages (DSLs) are a promising approach for the development of maintainable performance-portable codes for exascale machines. Extending the abilities of these DSLs to demonstrate their potential for key European codes presents an opportunity to the code developers to adopt the DSL approach as part of their development process in order to be able to run their models efficiently on exascale machines. For example, NEMO is not currently able to run on GPUs. However, GPUs make up the majority of the compute performance in most existing pre-exascale machines and planned exascale machines. Fortunately, the DSLs developed in ESiWACE2 will allow NEMO to run on these machines.

## 7 Sustainability

This deliverable is concerned with the extension of PSyclone and dawn to models that have not yet made use of DSLs and to improve the performance and scalability of models that already make use of these DSLs, as well as the comparative performance results of the two DSLs on a NEMO-based benchmark. Related deliverables will discuss the running of the full models using the DSLs (D2.2, due in project month 49) and will study the performance of codes generated by PSyclone and dawn (D2.4, due in project month 50).

The developments in this deliverable have benefited from close links with the now-completed ESCAPE-2 project, where dawn and GridTools were developed. There were also more general links to the IMMERSE project where CMCC had been looking at using PSyclone to apply loop-transformation optimisations to NEMO.

There are also strong links to the Met Office Next Generation Modelling Systems (NGMS) programme which is supporting PSyclone development in a number of areas. New PSyclone functionality is being developed for the LFRic atmosphere model. PSyclone's GPU support for NEMO is being extended to OpenMP offload (as an alternative to OpenACC) as well as being extended to support SI3, NEMOVAR, MEDUSA and Wavewatch III. PSyclone is being developed to support the automatic generation of adjoint Kernels from tangent-linear kernels as part of their Data Assimilation system. PSyclone will also be evaluated for use in the Met Office's NAME dispersion model next year, which is used operationally for events such as nuclear accidents and volcanic eruptions as well as air quality forecasts. As a result of this collaboration, the Met

Office have committed to support the maintenance and development of PSyclone. The Australian Bureau of Meteorology (BOM) are also already providing effort to help with the development of PSyclone.

Further, as part of the UK's ExCALIBUR programme, PSyclone is being extended to support OpenMP tasking directives for evaluation in the LFRic model and in a separate project PSyclone's intermediate representation is being translated to MLIR, which would allow PSyclone to make use of optimisations available in MLIR and LLVM.

Related work also continues in the DestinE project where fparser2 is being used as the front-end to the Loki code transformation tool. Work will also continue in the recently funded ESiWACE3 project where CMCC and BSC will evaluate and extend PSyclone for use in their NEMO configurations running on Exascale resources.

The dawn toolchain developments were integral in the design and implementation of GT4Py, its successor DSL. The GT4Py developments will transition and be integrated into the EXCLAIM project. The goals of EXCLAIM will continue the efforts started in ESCAPE-2 and ESiWACE2. The port of the ICON dycore to the DSL, developed in ESiWACE2 will be gradually extended until it covers entire model configurations, with all main components (physical parametrisations, dynamical core, advection schemes, MPI communication, etc) implemented in the python GT4Py language. The EXCLAIM project will develop the toolchain entirely in python and use it for production use cases of the ICON model at C2SM and MeteoSwiss. The EXCLAIM project is a long term project that ensures the sustainability of the developments initiated by ESCAPE-2 and ESiWACE2 with dawn.

PSyclone GPU support for NEMO is also maturing and it is hoped that this will become the accepted - and supported - way to run NEMO on GPUs. To help with wider adoption, progress is regularly communicated to the NEMO HPC and Systems teams. Colleagues at the National Oceanography Centre (NOC) - who are members of the NEMO Systems team - have demonstrated a simple way to integrate PSyclone into the current NEMO build system, as part of the ExCALIBUR Marine Systems project mentioned earlier. Putting this integration onto NEMO trunk is now a part of the workplan that NOC have agreed with the Systems Team. If PSyclone were accepted as part of the NEMO development and release process then this should also help with its sustainability and with the relevance of the interoperability work in this deliverable.

## 8 Dissemination, Engagement and Uptake of Results

### 8.1 Target audience

As indicated earlier in the document, the audience for this deliverable is 'the general public' (PU). Please see the table below for the different options.

X	The general public (PU)
	The project partners, including the Commission services (PP)
	A group specified by the consortium, including the Commission services (RE)
	This reports is confidential, only for members of the consortium, including the Commission services (CO)

The results of this deliverable have been presented at a number of conferences and workshops and will be reflected in a number of papers, please see Sections 8.2 and 8.3. We will also continue to present this work at upcoming conferences and workshops, such as the SIAM Geosciences Conference in Bergen, June, 19-22, 2023 where we are scheduled to give a presentation on DSLs for Weather and Climate modelling in a mini-symposium.

The results of the deliverable are also being taken up in other projects, such as DestinE and the Met Office NGMS programme, and will be taken up in future projects such as ESiWACE3. This document will act as a useful reference guide for these projects.

## **8.2 Record of dissemination/engagement activities linked to this deliverable**

Please see Table 1 and Table 2.

## **8.3 Publications in preparation OR submitted**

Please see Table 3.

## **8.4 Intellectual property rights resulting from this deliverable**

Dawn, PSyclone and fparser2 are open source projects hosted on github each with a BSD 3-clause license.



Table 1: Record of dissemination / engagement activities linked to this deliverable

Type of dissemination and communication activities	Details	Date and location of the event	Type of audience	Zenodo Link / Other Link	Estimated number of persons reached
Conference presentation	PSyIR: the PSy Intermediate Representation	Platform for Advanced Scientific Computing (PASC) Conference, 12 Jun. 2019, online	Technical	<a href="https://zenodo.org/record/3971994">https://zenodo.org/record/3971994</a>	30
Conference presentation	PSyclone and its use in LFRic	Platform for Advanced Scientific Computing (PASC) Conference, 12 Jun. 2019, online	Technical	<a href="https://zenodo.org/record/3252144">https://zenodo.org/record/3252144</a>	30
Participation to a workshop	PSyclone, LFRic, NEMO and SIR	1st ESCAPE-2 Dissemination Workshop, 21-22 October 2019, Reading, UK	Scientific Community	<a href="https://zenodo.org/record/3972115">https://zenodo.org/record/3972115</a>	60
Participation to a workshop	Recent Advances in PSyclone ... the PSyIR	6th ENES HPC Workshop, 26 May 2020, online	Scientific Community	<a href="https://zenodo.org/record/3972140">https://zenodo.org/record/3972140</a>	80
Presentation	Developing DSLs in ESiWACE2	ESiWACE2 Workshop on Emerging Technologies for Weather and Climate Modelling, 30 June 2020	Scientific Community	<a href="https://zenodo.org/record/3972151">https://zenodo.org/record/3972151</a>	50
Presentations and tutorials	PSyclone and dawn at the ESiWACE2 Summer School on Effective HPC for Climate and Weather	24-28 Aug. 2020, online	Students	<a href="https://zenodo.org/record/5795411">https://zenodo.org/record/5795411</a> , <a href="https://hps.vi4io.org/events/2020/esiwace-school">https://hps.vi4io.org/events/2020/esiwace-school</a>	50
Presentations and tutorials	ESiWACE2 training course on Domain-specific Languages in Weather and Climate	23-27 Nov. 2020, online	Model developers	<a href="https://zenodo.org/record/5795411">https://zenodo.org/record/5795411</a>	30

Table 2: Record of dissemination / engagement activities linked to this deliverable

Type of dissemination and communication activities	Details	Date and location of the event	Type of audience	Zenodo Link / Other Link	Estimated number of persons reached
Presentation	A Whirlwind Tour of PSyclone	Argonne National Labs Invited Presentation, 11 May 2021	Scientific Community, Industry	<a href="https://zenodo.org/record/4922906">https://zenodo.org/record/4922906</a>	10
Presentation	LFRic and PSyclone: Meeting Challenges on the Road to Exascale	PASC 21, 5-9 July 2021, online	Scientific Community	<a href="https://zenodo.org/record/6078561">https://zenodo.org/record/6078561</a>	20
Presentation	A Whirlwind Tour of PSyclone	PASC 21, 5-9 July 2021, online	Scientific Community	<a href="https://zenodo.org/record/5961131">https://zenodo.org/record/5961131</a>	20
Presentation and tutorial	PSyclone and dawn at the 2nd ESIWACE2 Summer School on Effective HPC for Climate and Weather	23-26 Aug. 2021, online	Students	<a href="https://zenodo.org/record/5795411">https://zenodo.org/record/5795411</a> , <a href="https://hps.vi4io.org/events/2021/esiwace-school">https://hps.vi4io.org/events/2021/esiwace-school</a>	50
Conference presentation	Performance Portability for Existing Weather & Climate Models using PSyclone: Application to the NEMO Ocean Model	19th ECMWF workshop on HPC in Meteorology, 20-24 Sep. 2021, online	Technical	<a href="https://zenodo.org/record/7378274">https://zenodo.org/record/7378274</a>	100
Conference presentation	PSyclone: A code generation and transformation system for weather and climate DSLs	Ateliers de Modélisation de l'Atmosphère, 8-10 Jun. 2022, Toulouse	Technical	<a href="https://zenodo.org/record/7374307">https://zenodo.org/record/7374307</a>	50
Presentations	PSyclone for LFRic; ESIWACE2 DSLs for ICON and NEMO	ESIWACE2 2nd Virtual Workshop on Emerging Technologies for Weather and Climate Modelling, 7 Oct 2022, online	Scientific Community	<a href="https://indico.dkrz.de/event/45/">https://indico.dkrz.de/event/45/</a>	50

Table 3: Publications related to this deliverable

In preparation OR submitted?	Title	All authors	Title of the periodical or the series	Is/Will open access be provided to this publication?
In preparation	PSyclone: A Code Generation and Transformation System for Weather and Climate Models	R. W. Ford, J. Henrichs, I Kavacic, C. M. Maynard, A. R. Porter, S. Siso	Geoscientific Model Development	Yes
In preparation	Using PSyclone 2.3.1 to Achieve Performance Portability for the NEMO (v4.0) and NEMO-MEDUSA ocean models.	Dearden C. and Ford, R. W. and Henrichs, J. and Porter, A. R. and Siso, S. and Müller, S.	Geoscientific Model Development	Yes