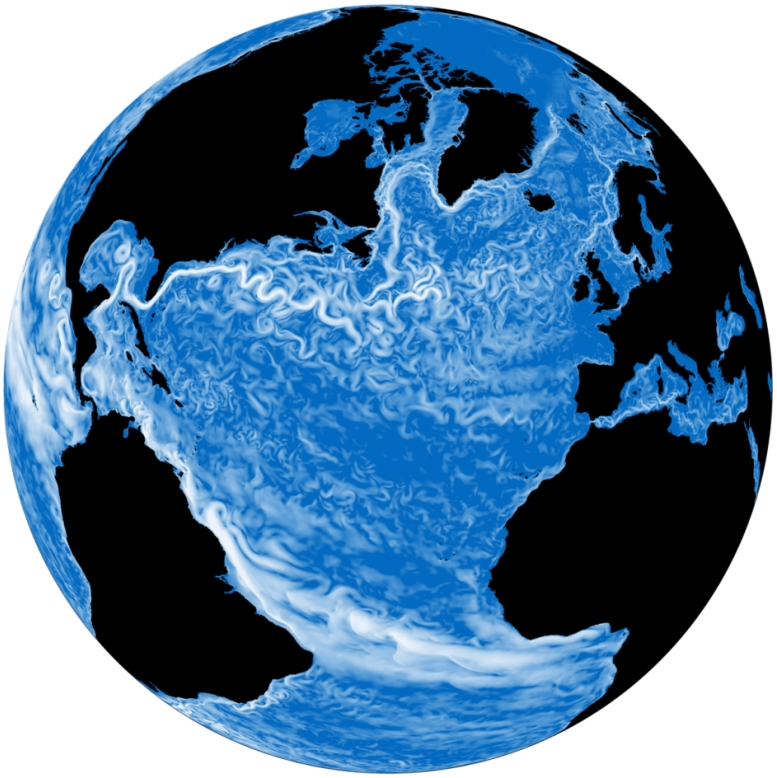
esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Applying Domain-Specific Language Tools to Production Weather and Climate Models: Computational Performance

Deliverable D2.4



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: March 17, 2023
Dissemination level: PU (for public use)

Lead authors:
Science and Technology Facilities Council (STFC): R. W. Ford and A. R. Porter
Federal Office of Meteorology and Climatology (MeteoSwiss): Matthias Roethlin and Carlos Osuna
Met Office (METO): Iva Kavcic

Other contributing authors:
Science and Technology Facilities Council (STFC): S. Siso, C. Dearden, N. Nobre

Project internal reviewer:
Barcelona Supercomputing Center (BSC): Mario Acosta

Contact details:
Project Office: esiwace@dkrz.de
Visit us on: https://www.esiwace.eu

Access our documents in Zenodo:
https://zenodo.org/communities/esiwace

Contents

1 Abstract /publishable summary

The models used in weather and climate simulation are large, complex and long-lived. It is then very difficult to adapt them to run well on supercomputers which are quickly evolving and tend to have a lifespan of only five years or so. Domain-Specific Languages (DSLs) are one solution to this problem: they enable the model developers to focus on the scientific challenges while the underlying tool chain takes responsibility for achieving good performance on a given supercomputer. This report describes the computational performance obtained through the application of the PSyclone and dusk/dawn DSL toolchains to full atmosphere and ocean models as used for weather and climate studies at various centres. These are: the UK Met Office's LFRic atmosphere model, the NEMO ocean model, the IFS-FVM atmosphere model and the ICON atmosphere model.

2 Conclusion & Results

The main outcomes of this work are:

- LFRic performance improvements demonstrated using new optimisations implemented in PSyclone.
- PSyclone used to add OpenMP parallelism to NEMO. Performance of this hybrid mode demonstrated on a production configuration on 1024 CPU cores.
- PSyclone used to add OpenACC parallelism to NEMO. Performance demonstrated on up to 192 GPUs.
- PSyclone used to add OpenMP offload parallelism to NEMO and demonstrated for the ORCA1 domain on a single GPU.
- dusk/dawn used to accelerate the ICON dycore and shown to match or outperform OpenACC performance for all dycore stencils
- Advanced optimisation techniques like fusion of and inlining of computations are introduced to the dusk/dawn tool-chain and their performance impact is discussed for a synthetic benchmarking suite.

3 Project objectives

This deliverable contributes directly and indirectly to the achievement of all the macro-objectives and 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 discusses the computational performance results obtained through the use of the DSLs in the four models addressed in this report: LFRic, NEMO, IFS-FVM and ICON. (These results were obtained following work on adapting the DSLs to the various models, as described in Deliverable D2.1 (Ford et al., 2022). The reader is therefore referred to D2.1 for more details of the functional changes.) LFRic is an example of a new model that has been developed with a DSL (PSyclone) from the outset. As a result, the main focus in this case has been the scientific development of the model and the development of PSyclone to support the necessary functionality. Work to date has therefore focused on performance on the existing, CPU-based machines used by the UK Met Office. NEMO is an example of an existing model where the developers are reticent to change their code unless it is to add new science functionality. As a result there are no directives in the code and it is currently only possible to run it on CPU-based machines. In this case the PSyclone DSL is used to add in directives and restructure the code where necessary, to allow NEMO to run on GPUs with no change to the original code. Since GPUs from different vendors require different programming models (e.g. NVIDIA hardware can be used from Fortran via either OpenACC or OpenMP directives while AMD hardware only supports OpenMP directives), PSyclone has been extended so as to support both OpenACC and OpenMP. Performance results are shown for both approaches. ICON and FVM (from the IFS suite) are examples of models where some of the existing code is re-written in a Domain-Specific Language. This then allows those sections of the model to be run on either GPU or CPU. The computational performance obtained for these four models are discussed in turn in the rest of this section.

4.1 PSyclone and LFRic

In this section we present how the developments to PSyclone (described in D2.1) have fed through to deliver performance improvements in the LFRic model. (Note that the associated LFRic infrastructure development work has been done by the Met Office with the support of the UK ExCALIBUR programme.)

4.1.1 Halo-exchange Reduction

PSyclone is responsible for adding appropriate distributed memory calls to LFRic code to ensure the code runs correctly in parallel. Given the structure and implementation of the LFRic code only two types of call are required 1) halo exchanges and 2) global sums. Halo exchange calls and global sum calls are added based on kernel implementations as described by kernel metadata and the order in which kernels are called as specified by 'invoke' calls in the algorithm layer. It is relatively simple to place global sums, however it is more difficult to place halo exchanges and run-time flags are used to indicate whether fields have dirty or clean halos in order to work out whether a halo exchange should be called or not in cases where this cannot be determined. It can be shown that, given a set of rules for the correct placement of halo exchanges PSyclone

will ensure that the minimum number of halo exchanges will be called. However, the rules themselves can be too restrictive. This section presents the results of extending kernel metadata in order to more precisely define the type of computation being applied in a kernel and as a result, reducing the number of halo exchanges that PSyclone needs to add to the code. The two changes are 1) adding a new function-space access type called `ANY_DISCONTINUOUS_SPACE` and 2) allowing certain kernels which modify fields on a continuous function space to be specified as `GH_WRITE`. For details on what these two changes mean please see Deliverable D2.1. This section simply provides the results of applying these two changes.

ANY_DISCONTINUOUS_SPACE

As mentioned in Deliverable D2.1, `ANY_DISCONTINUOUS_SPACE` metadata in LFRic kernels mark fields that are discontinuous without having to specify the particular type of field. These metadata are particularly useful in kernels that interface the existing Physics parameterisation schemes developed for the Unified Model (UM) that are implemented as single-column models.

`ANY_DISCONTINUOUS_SPACE` metadata were implemented in several tickets and commits to the LFRic trunk[1] from January to May 2020, leading to significant reduction of the number of halo exchanges in the LFRic Atmosphere (`lfric_atm`) model configuration. The additional improvement was implemented at the end of May 2022 by configuring PSyclone release 1.9.0 for LFRic to enable redundant computation to shared "dofs" for PSyclone built-ins (so called "annexed dofs"), arithmetic operations on whole fields directly implemented in the generated PSy-layer. Table 1 below presents the cumulative results of these changes in LFRic for the LFRic Atmosphere application from May 2020.

Application	AS	ADS	ADS + Annxd
lfric_atm	354,851	312,403 (88.03%)	164,024 (46.22%)
slow_physics	5,997	576 (9.6%)	0 (0%)
fast_physics	16,009	14,094 (88.03%)	10,454 (65.3%)

Table 1: Number of halo exchanges for the LFRic atmosphere application, Slow Physics and Fast Physics components for an Aquaplanet run at C96 cubed-sphere grid resolution (≈100 km horizontal) with 38 vertical levels and timestep $dt = 300$ s. Here "AS" and "ADS" denote kernel source code utilising `ANY_SPACE` metadata and `ANY_DISCONTINUOUS_SPACE` metadata, respectively (second and third column). "ADS + Annxd" in the fourth column denotes the implementation of both `ANY_DISCONTINUOUS_SPACE` metadata and redundant computation to shared ("annexed") dofs for built-ins. Figures in parentheses give the number of halo exchanges as a percentage of the corresponding figure before the introduction of the `ANY_DISCONTINUOUS_SPACE` and annexed-dofs functionality.

As can be seen from Table 2, implementation of `ANY_DISCONTINUOUS_SPACE` metadata and redundant computation resulted in significant runtime improvements in the Slow Physics component of the Aquaplanet configuration run and moderate runtime improvements of both Fast Physics components and the overall model runtime.

GH_WRITE with continuous function spaces

The potential of reduction of the number of halo exchanges in LFRic by using `GH_WRITE` access for fields on continuous function spaces was demonstrated prior to implementing PSyclone support. Table 3 shows runtime improvements for the manual adjustments to PSy-layer and LFRic kernels interfacing the UM explicit and implicit boundary layer (BL) schemes. These changes were fully implemented in LFRic after adopting PSyclone release 2.3.1 in June 2022.

[1] LFRic use the term "trunk" for the main branch as they use SVN for their version control.

MPI tasks (nodes)	Colums per task	Tot: ADS + Annxd	SP: ADS + Annxd	FP: ADS + Annxd
216 (6)	256	5.41%	16.81%	4.94%
144 (4)	384	4.70%	18.44%	3.73%
108 (3)	512	4.53%	18.59%	4.40%
72 (2)	768	3.40%	13.80%	4.06%

Table 2: Speed-up (%) compared to runtimes of `ANY_SPACE` Physics kernel metadata runs for the LFRic atmosphere application, Slow Physics (SP) and Fast Physics (SP). The Met Office Cray XC40/XCS architecture used for tests consists of Intel Xeon dual socket 18-core Broadwell nodes (i.e. 36 CPU cores per node) with Aries interconnect. The Intel 17.0.0.098 compiler was used to compile and run the test cases. The rest as in Table 1.

Scheme	Number of halo exchanges per timestep		C48 mean time		C192 mean time	
	Before	After	Before	After	Before	After
bl_exp_alg	74	7	55.08s	34.85s	14.76s	11.26s
bl_imp_alg	13	0	24.41s	23.62s	7.57s	7.44s

Table 3: The comparison of number of halo exchanges per timestep and mean runtimes for calls to the UM explicit and implicit boundary layer (BL) schemes before and after implementing `GH_WRITE` metadata in the relevant LFRic kernels. C48 and C192 configurations have ≈200 km and ≈50 km horizontal resolutions, respectively. The Cray XC40/XCS architecture used for the tests is outlined in caption of Table 2.

Using `GH_WRITE` metadata for continuous function spaces in LFRic where appropriate is also illustrated by ≈40% performance improvement of FFSL transport scheme illustrated in Table 4 below.

Component	Runtime at trunk revision [37670]	Runtime at trunk revision [37680]
gungho_transport_control	652.68s	388.90s

Table 4: Comparison between runtimes for a C192 (≈50 km horizontal resolution) "GAL9" configuration in the LFRic Atmosphere performance suite before and after implementing `GH_WRITE` metadata in relevant kernels in August 2022. All variables are transported using the Flux-Form Semi-Lagrangian (FFSL) advection scheme. The Cray XC40/XCS architecture used for the tests is outlined in caption of Table 2.

4.1.2 Mixed-precision Support

PSyclone support for mixed precision in LFRic, outlined in Deliverable 2.1, was the major highlight of PSyclone release 2.3.1 currently used in LFRic. The aim was to improve the computational performance of the model through the targeted use of lower-precision floating point numbers, primarily in the semi-implicit (SI) solver and transport schemes.

It is important to note here that the main goal of this work was to gain as much of performance improvement as possible whilst retaining scientific validity. The results presented here are for parts of the LFRic model where preliminary experiments indicated it would be possible to achieve this goal, and future work will explore whether reduced precision is feasible for other high-cost schemes.

In theory, the expected speed-up from going from 64-bit to 32-bit could be up to 50%. In practice, however, it is reasonable to expect less since some areas are not memory-bound and the global sums are still computed in double (64-bit) precision. This conclusion was supported by the Dynamics-only tests (Gungho performance suite) carried out in February and March 2022. The preliminary mixed-precision support for these

tests was accessed from then PSyclone master branch, which illustrates the continuous development feedback from LFRic to PSyclone and vice-versa. The results in Table 5 show the net speed-up on 64-bit and 32-bit LFRic branches for the Dynamics-only Held-Suarez test case. The solver speed-up on the 64-bit branch was mostly due to algorithmic optimisations applied to the Schur preconditioner, whereas the speed-up for the 32-bit branch also shows the impact of reduced precision.

Cray XC40 configuration	64-bit branch	32-bit branch
6omp 6nodes	1.06x	1.79x
6omp 12nodes	1.10x	1.55x
6omp 24nodes	1.09x	1.53x

Table 5: The net speed-up of the semi-implicit (SI) solver code compared to LFRic trunk for the Gungho (dynamical core) application that uses a C192 (≈50 km horizontal resolution) Held-Suarez test. The Cray XC40/XCS architecture used for the tests is outlined in caption of Table 2.

Following the adoption of PSyclone release 2.3.1 in LFRic in June 2022, the support for mixed precision has been implemented in LFRic in several parts of the model. Table 6 shows the cost of computing the semi-implicit (SI) operator and the overall SI solver for one member of the GC5-LFRic assessment suite before and after the introduction of 32-bit capability. The reduction in time for the SI operators is ≈55% (≈2.2x speed-up) and for the solver is ≈26% (≈1.35x speed-up), with the total model time reduced by ≈8%.

LFRic trunk revision number	SI operator computation cost	Semi-implicit solver cost	Total model cost
37279 (Jul 2022)	272.16s	188.72s	3377.71s
37648 (Aug 2022)	122.32s	139.62s	3098.62s

Table 6: Costs of the semi-implicit operators, solver and total model for a 6-day Gungho NWP configuration (one member of the GC5-LFRic assessment suite) before (revision [37279]) and after introducing 32-bit capability (revision [37648]). The cubed-sphere resolution is C192 (≈50 km horizontal resolution), and the configuration is run on 24 nodes of the Cray XC40/XCS architecture outlined in caption of Table 2.

Table 7 shows the speed-up associated with the implementation of mixed precision aimed at transport schemes in the LFRic Atmosphere performance suite for a C192 Aquaplanet run, also measured at the preliminary testing phase in February/March 2022. The overall transport speed-up (`gungho_transport_control`) is ≈30% when running on 6 nodes and ≈20% when running on 24 nodes for the lower 32-bit precision. The complete implementation of the mixed-precision support for transport schemes was committed to the LFRic trunk in January 2023.

	6 nodes		24 nodes	
Routine	ratio 32-bit/64-bit	difference (s)	ratio 32-bit/64-bit	difference (s)
gungho total	0.97	-41.42	0.98	-7.74
gungho_transport_control	0.70	-38.82	0.81	-5.67
mol_conservative_alg	0.63	-14.44	0.67	-2.83
wind transport	0.66	-7.91	0.72	-1.43
moist mixing ratio transport	0.70	-9.83	0.73	-1.93
tracer collection transport	0.64	-14.06	0.70	-2.81

Table 7: Comparison between runtimes for a C192 (≈50 km horizontal resolution) LFRic Atmosphere Aquaplanet configuration with transport routines implemented in 32-bit and 64-bit precisions. The Cray XC40/XCS architecture used for the tests is outlined in caption of Table 2.

Note, work on PSyclone mixed-precision support was partially funded by the ExCALIBUR (Exascale Computing: ALgorithms and Infrastructures Benefiting UK Research) Programme. LFRic implementation of the mixed-precision support, as well as the results presented above, were fully funded by the ExCALIBUR Programme.

4.1.3 I-first Kernels

As mentioned in Deliverable D2.1, another PSyclone contribution to better utilisation of the UM Physics parameterisation schemes in LFRic was support for i-first kernels. As shown in Figure 1, the implementation of i-first kernels enables calling UM schemes on whole domain.

Also, as in the case of implementation of mixed-precision support, implementation of i-first support in LFRic was partially funded by the ExCALIBUR Programme, with the results shown in this section produced by one of the Met Office-led work packages.

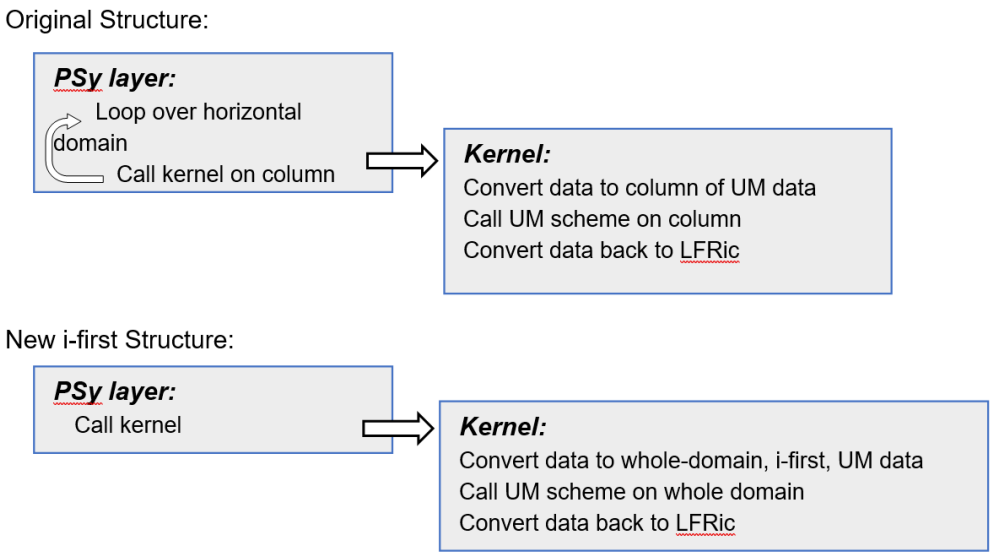


Figure 1: Original k-first and new i-first code structure of PSy and Kernel layers in LFRic for Physics kernels. Courtesy of Ben Shipway, Met Office.

Table 8 shows the cost of more expensive Physics schemes for a "basic-gal" LFRic Atmosphere configuration prior to implementation of i-first kernels in LFRic.

Table 9 compares runtimes of the Microphysics kernel re-written to call the UM Microphysics scheme with i-first looping and its original k-first implementation. The i-first data ordering significantly improved performance of the Microphysics scheme, with a speed up by a factor of 3.5 - 4 times (results from the i-first version of the code bit compared with the k-first version). Since December 2021 when this change was implemented for the Microphysics scheme, other Physics interface kernels were investigated and refactored as well.

4.1.4 LFRic Performance-Improvement Timeline

The previous subsections illustrate the difficulty with drawing a clear line between performance improvements gained by what is considered traditional optimisations and by the PSyclone DSL support. This is due to PSyclone being a core functionality in LFRic, where the generated PSy-layer not only applies optimisations but also employs the LFRic model infrastructure to extract and pass the correct information from the high-level field objects in the algorithm layer to the low-level kernels that implement mathematics on raw data.

Scheme	Time	Percentage of total runtime	Notes
whole model	2185.2s	100%	(of this ≈20.8% is Slow Physics and ≈4.5% is Fast Physics)
microphysics	247.4s	11.3%	
boundary layer	81.7s	3.7%	(≈69% UM explicit BL scheme, ≈% UM implicit BL scheme)
radaer	80.6s	3.7%	
radiation	67.1s	3.1%	(≈61% Long-Wave radiation scheme and ≈39% Short-Wave radiation scheme)
convection	18.0s	0.8%	
pc2	17.9s	0.8%	(split over various pc2 routines)
glomap aerosol	9.8s	0.4%	
cld	3.0s	0.1%	
jules	1.7s	$< 0.1\%$	

Table 8: Cost of more expensive Physics schemes in "basic-gal" LFRic Atmosphere configuration with C48 cubed-sphere grid resolution (≈200 km horizontal), running on 216 PEs for 720 timesteps.

Microphysics runtimes	k-first (original)	i-first
C48, 216 PEs, 720 timesteps	247.4s	63.9s
C192, 864 PEs, 60 timesteps	33.5s	9.6s

Table 9: k-first and i-first runtimes of the LFRic Microphysics kernel that interfaces the respective UM Physics scheme produced using the C48 (≈200 km horizontal) and C192 (≈50 km horizontal) configuration resolutions in the "basic-gal-demo" suite.

Hence, algorithmic optimisations in LFRic, such as more efficient transport schemes and multigrid solver, often depend on PSyclone support.

Figure 2 illustrates the predictions for performance improvements gained by this integrated approach. It was created in May 2022 by the Met Office's Dynamics Research team who are responsible for the algorithmic development of the LFRic model. The figure presents the predicted performance improvements of as the new scheduled algorithmic improvements are added. The scheduled algorithmic improvements since May 2022 to the time of this report have already been implemented.

Model Set-Up and Machine Configuration

The LFRic model is run at resolution C192 (≈50 km horizontal) on a cubed sphere, running with GA9 Physics with a 6-day NWP forecast taken from the GMED case study suite. The IO is representative of assessment runs rather than operation runs.

The test suite is run on the Met Office Cray XC40/XCS architecture which consists of Intel Xeon dual socket 18-core Broadwell nodes (i.e. 36 CPU cores per node) with Aries interconnect. The Intel 17.0.0.098 compiler is used to compile and run the model.

The same resources are used for running the Met Office's current model (the Unified Model (UM)) N320 or 48km configuration used for comparison. The UM N320 configuration uses the same GA9 Physics.

Discussion

The predictions are shown as light green, then yellow and finally salmon coloured bars. The heights of the bars indicates the time spent running the model so it can be seen that there is a gradual predicted performance improvement as new algorithmic improvements are added over time, with the actual algorithmic improvements

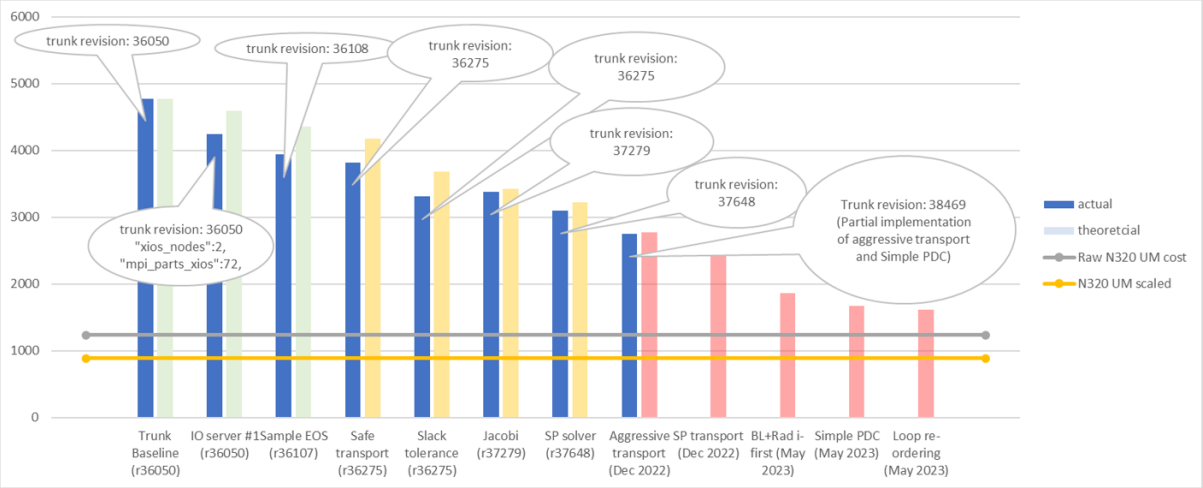


Figure 2: The predicted and actual performance improvements for LFRic as new algorithmic developments are added to the model.

being described underneath each of the bars. The different colours are meant to indicate confidence in the prediction, with green being the most confident, yellow the next and salmon the least. As one might expect, as the predictions go further into the future, the confidence in them drops accordingly. The grey horizontal bar is the time taken to run the same problem with the same effective resolution using the UM N320 configuration (48km). Whilst the two models have the same effective resolution it is a slightly unfair comparison as the UM actually contains around 40% more grid points than LFRic at the specified resolution due to its increased number of grid points towards to the poles. As a result, the orange line presents a scaled performance line that gives a fairer performance comparison. The Met Office are aiming to meet this lower performance target for the LFRic model, although it is not a hard constraint. However, as can be seen, this performance target is not predicted to be achieved through algorithmic improvements alone by May 2023. Whilst this is the case, there are also expected to be performance benefits through computational improvements which will hopefully enable LFRic to achieve parity at this resolution. It should also be noted that this example is at a relatively low resolution and that LFRic is designed to scale to higher resolutions where the UM will become limited by its regular grid. Therefore LFRic is expected to perform better than the UM at higher resolutions.

The actual performance improvements due the algorithmic improvements are shown by the blue bars with the speech bubbles specifying which particular version of LFRic trunk they were added. As can be seen, the actual and predicted performance improvements, as well as their cumulative benefit, are quite closely aligned, which gives confidence in the future predictions.

The two halo-exchange reduction optimisations that have been added to PSyclone (as reported in Deliverable D2.1 and discussed earlier) are not shown as part of the algorithmic improvements in Figure 2. One reason for this is that the first of these (adding support for `ANY_DISCONTINUOUS_SPACE`) was done before May 2022 and the second (adding support for `GH_WRITE` when modifying particular continuous function spaces) was devised and developed after May 2022. However, even if the latter were added to the figure it would not be expected to make that big a difference at the test resolution, as it is more about improving the scalability of the code. It can be seen that the multi-precision support that has been added to PSyclone (as reported in Deliverable D2.1 and discussed earlier in this section) is being used in two places by the Met Office. First, the "SP Solver" improvement (denoted as SI solver in the mixed-precision subsection earlier) was added to trunk in August 2022 giving the expected performance improvement, and second, the "SP transport" improvement was added in early January 2023. Further, it can also be seen that the i-first kernel support added to PSyclone (again as reported in Deliverable D2.1 and discussed earlier in this section) will be used in the "BL+Rad i-first" improvement for the boundary-layer and radiation Physics parametrisations which are scheduled to be

added to trunk in May 2023.

4.2 PSyclone and NEMO

In this section we present the details of the performance results obtained by STFC when using PSyclone to transform the NEMO ocean model for running in either hybrid mode (MPI+OpenMP) on CPU or for offloading to GPU. For the former, results are given for a $\frac{1}{4}$ degree global configuration while for the latter, results are given for two different global configurations of the model - a low-resolution (one degree) test case running on a single GPU, and a high-resolution ($\frac{1}{12}$ degree) test case that requires multiple GPUs running in parallel with MPI.

4.2.1 Machine Configuration

For the low-resolution test case on GPU, a workstation equipped with a 32GB NVIDIA Volta 100 GPU was used, hosted by an Intel Xeon W-2133 CPU. The scaling performance of NEMO across multiple GPUs in the high resolution test case was investigated using the JUWELS Booster supercomputer, equipped with 936 GPU nodes each comprising of two AMD EPYC 7402 CPUs (24 cores per socket) and four, 40GB NVIDIA Ampere 100 GPUs. The GPUs on JUWELS Booster are connected to each other via NVLink3, and the CPUs, GPUs, and network adapter are connected via two PCIe Gen 4 switches with 16 PCIe lanes going to each device. Both the GPU workstation and JUWELS Booster have the memory page size set to 4KB.

For benchmarking purposes, the standard compute nodes of the Hartree Centre's Scafell Pike supercomputer were used in the low resolution test case, comprising two Intel Skylake CPUs (Xeon Gold E5-6142 v5, 16 cores each at 2.6GHz) and 192 GB RAM per node. These tests used version 2018.0.128 of the Intel compiler. The MPI simulations on JUWELS Booster were benchmarked against the high memory nodes of the JUWELS Cluster supercomputer, each consisting of two Intel Xeon Platinum 8168 CPUs (24 cores each), and 192GB memory per node. These tests used version 2021.4.0 of the Intel compiler.

The OpenACC results presented here use version 22.7 of the NVIDIA HPC SDK and we set the environment variable `NV_ACC_POOL_THRESHOLD` to 75 in order to increase the size of the memory pool available to managed memory (the default is 50(%)).

Comparing performance between CPU and GPU architectures can be done in various ways (socket to socket or node to node) and including factors such as time to solution, hardware cost, and energy to solution. In the context of large models with flat performance profiles, effective use of a GPU requires that the "whole" model be run on it, avoiding the CPU (and associated data transfers) as much as possible. For simplicity therefore, we choose to compare the time to solution on a socket-to-socket basis.

4.2.2 NEMO-OCE on GPU

The ORCA1 global configuration of NEMO-OCE was chosen as the basis of the low resolution test case, since the horizontal resolution of 1 degree ensures that the model domain is small enough (362×332 grid points in the horizontal, with 75 vertical levels) to run comfortably on a single GPU without exceeding the available device memory. The CPU baseline for the ORCA1 configuration was achieved by running the model with MPI enabled on a fully populated 16-core Intel Skylake socket on Scafell Pike. For the high resolution test case, the ORCA12 configuration was used to provide a robust test of the scaling of NEMO-OCE across multiple GPUs with MPI. ORCA12 has a horizontal resolution of $\frac{1}{12}$th of a degree, with a global domain size of 4322×3606 grid points and 75 vertical levels. In both cases, simulations run for 100 timesteps unless stated otherwise, and the performance is quantified in terms of average time per timestep, taking care to exclude the time spent in the initialisation routines.

The output of diagnostics in NEMO-OCE is handled via XIOS, a library for managing I/O within climate codes. However, XIOS (and therefore diagnostic I/O) was switched off in our simulations to allow us to focus on the pure computational performance of the model. Acceleration of XIOS routines will be considered

separately as part of future work. It should be noted that a scheme for the treatment of icebergs, ICB, is also available within NEMO-OCE, but in its current form the ICB code is fundamentally serial in nature due to the data structures it uses. This proved to be a major barrier to obtaining acceptable GPU performance and thus for benchmarking purposes, we chose to disable the iceberg scheme for the present study. Similarly, the code section `dia_hsb`, which calculates diagnostics relating to conservation of ocean heat, salt and volume, was also turned off in both the GPU and CPU-only simulations. This is because such diagnostics involve reproducible global sum calculations which also proved difficult to offload efficiently to GPU.

4.2.3 Single GPU Performance (OpenACC and OpenMP)

We begin by considering the GPU performance of NEMO-OCE v4.0.2 in the absence of any MPI parallelism, where the code is processed with PSyclone and OpenACC transformations are applied. Figure 3 shows the evolution in the performance of the OpenACC version of NEMO ORCA1 running on a single NVIDIA V100 GPU. It reveals that performance has steadily improved over time in line with PSyclone development. Such developments include support for Fortran derived types, the ability to replace implicit loops with explicit loops, and the ability to hoist automatic arrays from the subroutine level to module level. When the timings are normalised with respect to the CPU-only baseline, the current OpenACC-accelerated version of NEMO-OCE is found to be 3.3× faster. (Note that this figure is for a version that has some additional, manual changes to the source generated by PSyclone. See below for a discussion of these.)

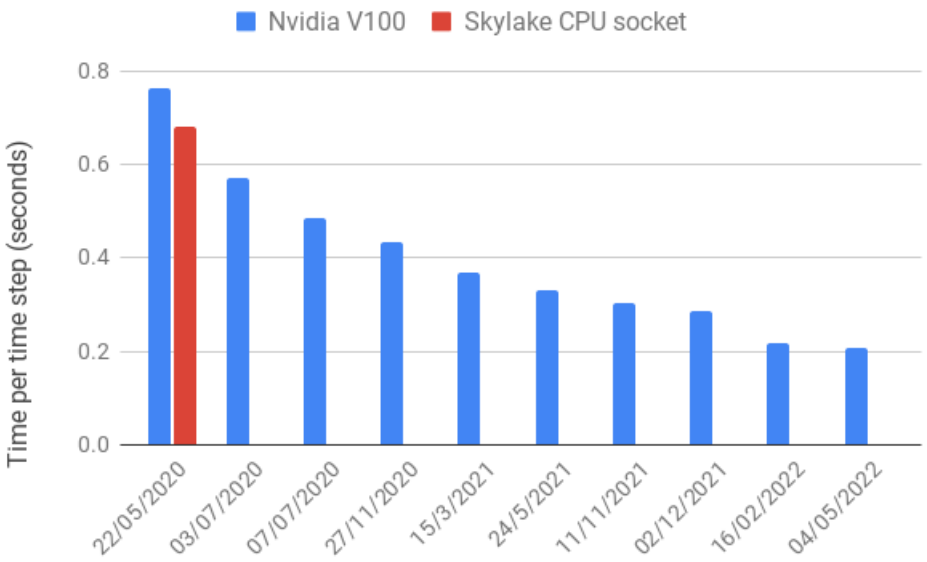


Figure 3: Evolution of NEMO-OCE GPU performance (ORCA1 configuration) during PSyclone development, shown in blue. Performance is shown in terms of average time per model timestep. For reference, the CPU baseline result is shown in red.

At the beginning of this project, PSyclone (with the NEMO API) was only capable of targeting a GPU via OpenACC and compiler support for OpenMP offload was itself not mature. PSyclone's existing support for OpenMP parallelisation on CPU has been extended to support the new directives offered by OpenMP 4.5 and 5.0. This has been combined with additional transformations that enable Fortran code that uses array syntax to be converted into explicit loops. As a result, PSyclone is now also capable of processing NEMO code so as to generate a version that targets GPUs via OpenMP offload instead of OpenACC. Figure 4 compares the current performance obtained when using either OpenACC or OpenMP to offload to a V100 GPU. We attribute the better performance of the OpenMP offload version to the fact that the NEMO source has to be transformed into a simpler form in order to permit the addition of the OpenMP directives. We have not

(yet) tried applying these additional transformations when adding OpenACC. In addition, there is still work to do to bridge the gap between the performance obtained purely using PSyclone and that obtained when some manual modifications are made to the code. The remaining manual modifications required for NEMO-OCE are listed below:

- Hoisting of automatic arrays at the subroutine level to module scope. This is necessary to prevent GPU page faulting associated with the creation/destruction of work arrays upon each entry/exit from a subroutine. Since Figure 4 was produced, this capability has now been incorporated into the PSyclone master branch, although a further, manual modification is necessary to allow PSyclone to deal with arithmetic expressions in the declaration of array dimensions. Our tests reveal that such expressions arise in the NEMO v4.2 source code due to the introduction of tiling in the lat-lon dimensions. See PSyclone issue #1969;
- Inlining of array-valued functions for OpenACC performance. See PSyclone issue #924;
- Add support for intrinsic functions `MAXVAL` and `MINVAL`, which accept optional mask and dim arguments. See PSyclone issue #840;
- Allow GPU compilation of scalar-valued functions when used inside an OpenACC `KERNELS` region. This situation was identified during the recent testing of NEMO v4.2 source code with PSyclone. See PSyclone issue #1985.

A further modification is required for NEMO+SI3, to introduce private/firstprivate clauses on the OpenACC Loop directive (see PSyclone issue #423). To achieve acceptable multi-GPU performance with MPI, some manual intervention is also necessary involving the use of GPUDirect; this is described in detail in Section 4.2.4.

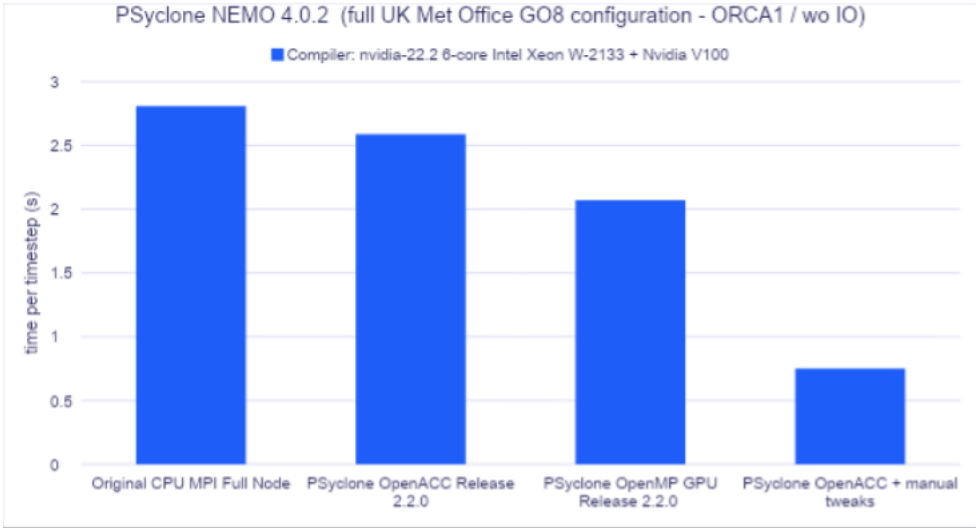


Figure 4: NEMO GPU performance (wall-clock time per simulation time step) with OpenACC and OpenMP. Note that the CPU used for comparison here is not an HPC-class chip and thus is shown for illustrative purposes only.

Figure 5 shows the breakdown of time spent in each code section for the ORCA1 configuration of NEMO-OCE running for 50 timesteps on a single NVIDIA V100 GPU, compared to an equivalent simulation running on a fully populated Skylake CPU socket of Scafell Pike. The figure shows that all but one of the code sections is accelerated in the GPU version, the exception being the dynamics surface pressure gradient code section `dyn_spg`, whose GPU performance is on a par with that of the CPU-only version. In the ORCA configurations, the surface pressure gradient term is solved using a time-splitting approach (`ln_dynspg_ts=.true.` in

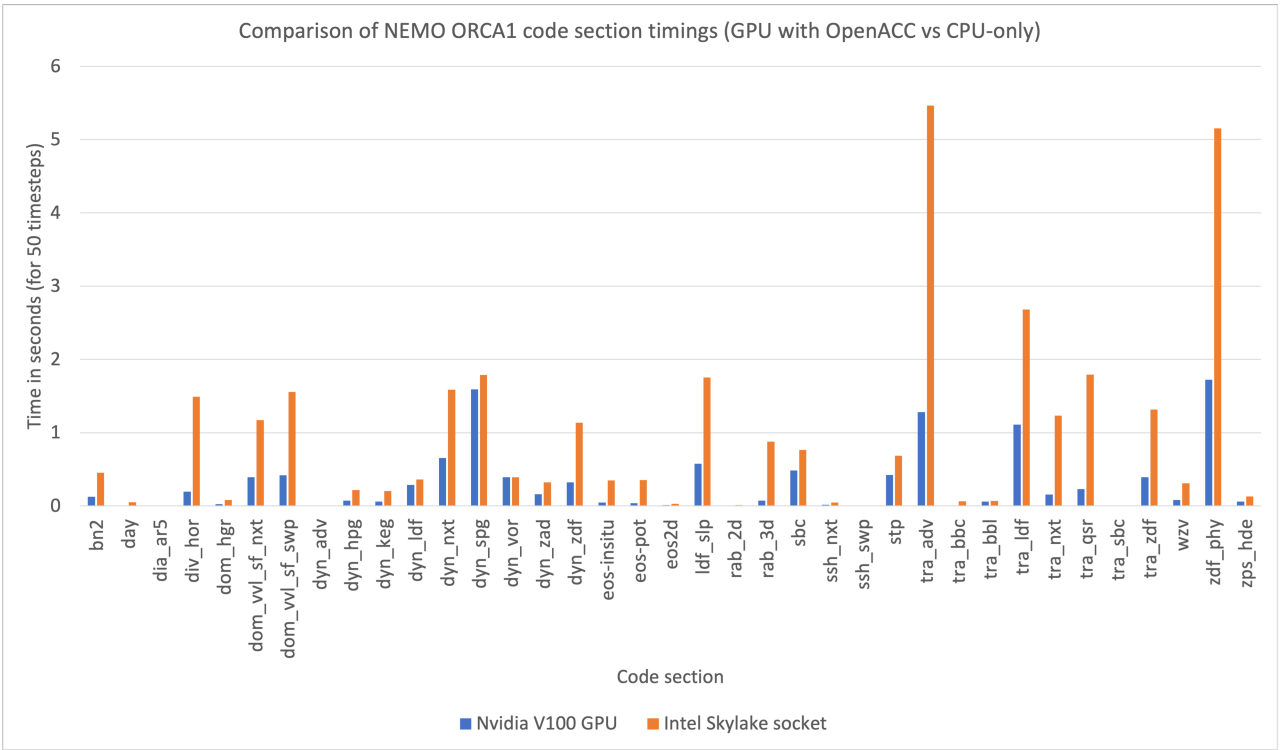


Figure 5: Plot showing the breakdown of time spent in each code section of NEMO-OCE for equivalent ORCA1 simulations running on a single NVIDIA Volta100 GPU (blue), and a Skylake 16-core CPU socket (orange).

`namelist_cfg`), whereby the pressure gradient term is computed using an iterative sub-stepping loop to ensure numerical stability. For ORCA1, there are 55 sub-steps in `dyn_spg` per model timestep, with 4 calls to the lateral boundary condition routines `lbc_lnk` per sub-step. Our investigations suggest that the relatively poor GPU performance of `dyn_spg` compared to the CPU version is a consequence of the high OpenACC kernel launch overhead arising from many small kernel launches within the sub-stepping loop.

In addition to the physical ocean model, PSyclone has also been used to process NEMO-OCE v4.0.2 with the interactive sea-ice module SI3 enabled (henceforth referred to as NEMO+SI3) and the OpenACC GPU performance has been analysed. Some sections of the SI3 code make use of a data packing approach, whereby gridpoints containing sea-ice are identified and their values stored contiguously in 1-D work arrays. Compute is then carried out on the work arrays before the results are unpacked, such that the sea-ice points in the original 2-D latitude-longitude arrays are updated with the new values. This method is used in the sea-ice ridging routines (`icedyn_rdgrft`), the thermodynamics routines (`icethd`) and `icecor`, which performs corrections on sea-ice variables at the end of each time step. Our investigations revealed a sequential dependency in the packing and unpacking routines which makes them inefficient on the GPU. Thus, prior to processing the SI3 code with PSyclone, the relevant parts of the SI3 code were first refactored such that the compute is instead performed on the original, multi-dimensional sea-ice arrays, with care being taken to mask out any points that do not contain sea-ice to avoid floating point exceptions. This approach removes the need for the costly packing/unpacking steps on the GPU, whilst also allowing PSyclone to exploit the inherent parallelism in the loops over latitude and longitude. The results are shown in Figure 6. Taken in isolation from the ocean component, the GPU performance of the refactored SI3 code on an NVIDIA V100 GPU is currently 1.57× faster than the original SI3 code running on CPU-only. This translates to a factor of 2.6 speed-up for NEMO+SI3 overall compared to a fully-populated Skylake socket. Figure 6 also suggests that further potential remains to improve the GPU performance of the SI3 code. For instance, some implicit data transfers

remain within `DO WHILE` loops that occur within `icethd` and `icedyn_rdgrft` which, when eliminated, will help to improve the performance of these routines.

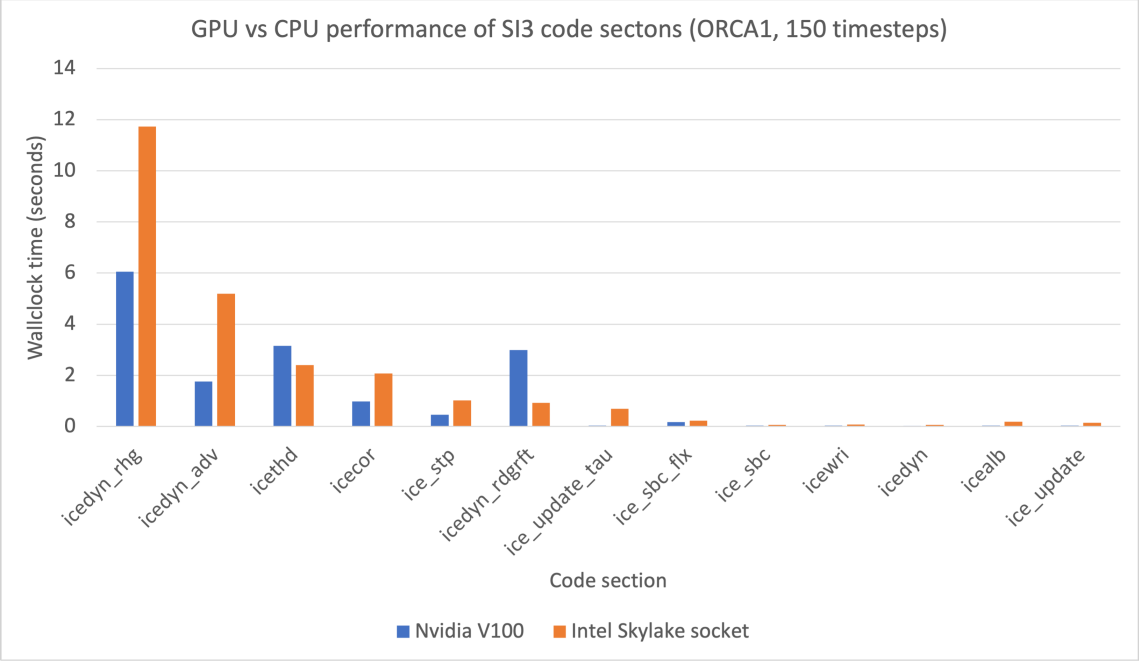


Figure 6: Plot showing the breakdown of time spent in each SI3 code section for equivalent ORCA1 simulations running on a single NVIDIA V100 GPU (blue), and a Skylake 16-core CPU socket (orange).

4.2.4 Multi-GPU Performance with MPI

To investigate the scaling performance of NEMO-OCE across multiple GPUs, ORCA12 simulations were carried out on the JUWELS Booster system using 4 MPI tasks per node, with each MPI task binding to a unique NVIDIA A100 device to ensure all the GPU resources per node were fully utilised. A major consideration for effective multi-GPU performance is the method by which the halo buffers resident in device memory are handled during MPI exchanges. The default CUDA-aware MPI approach is to stage device-resident halo buffers via the CPU. This is particularly important for node architectures such as those used on JUWELS Booster, where data transfers between device and host would use the slower PCI-e bus, making MPI communications costly. To prevent this, NVIDIA's GPUDirect RDMA technology (Remote Direct Memory Access) provides direct device-to-device communication via NVlink for inter-node GPU communications, and is available via the CUDA ToolKit. This eliminates the system CPUs and the required buffer copies of data via the system memory. However, a caveat to this is that GPUDirect is not yet compatible with managed memory. The current version of PSyclone relies on NVIDIA's managed memory technology to handle data transfers, and since the halo buffers are stored as allocatable arrays in NEMO, the device-resident halo buffers would still get staged via the CPU. This results in the system memory signalling a page fault, and without further intervention the multi-GPU performance of NEMO is compromised by the slow data transfer associated with each MPI communication. To work around this problem, our solution is to switch from using allocatable arrays to static arrays for the halo buffers in NEMO-OCE. In doing so, the halo buffers are no longer handled by managed memory, and GPUDirect can be used for direct device-to-device MPI communications. Although this refactoring is not yet handled by PSyclone, it is intended that this capability will be included as part of a future release.

Figure 7 shows results of the ORCA12 GPU+MPI performance for NEMO v4.0.2, showing the benefits of GPUDirect when switching to static arrays for the halo buffers. Our investigations reveal that one further code

change is also required to prevent page faulting within the north-fold boundary condition routines `lbc_nfd`. Such page faults occur as a result of work arrays being allocated at the subroutine level, which are then implicitly destroyed upon exit from the subroutine, meaning the arrays do not persist in device memory for the lifetime of the simulation. This would typically be addressed using PSyclone by applying a transformation to hoist the arrays in question so they have module scope; however this particular situation is complicated by the fact that the allocation of the work arrays depends on scalar variables which do not exist at the module level. Instead, our approach is to manually hoist the work arrays to the module level whilst also declaring them as static arrays, whose size is specified explicitly at compile time. Using this method, we are able to achieve performance of around 2.2 simulated years per day on 100 A100 GPUs of JUWELS Booster (blue line in Figure 7). This is approximately 2.2× faster than the CPU-only version running on 100 Skylake sockets of JUWELS Cluster. At higher GPU counts the results plateau, suggesting that the GPUs start to run out of work and MPI communications dominate.

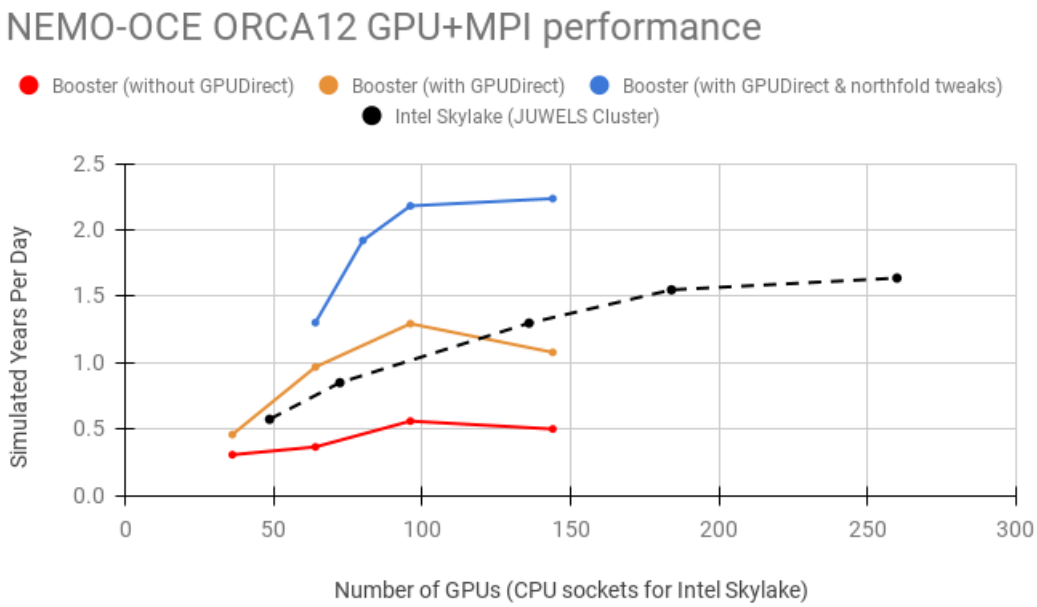


Figure 7: Scaling plot showing the multi-GPU OpenACC performance of NEMO v4.0.2 (ORCA12 configuration) on the JUWELS Booster machine, expressed in terms of Simulated Years Per Day. The CPU baseline result is generated using the CPU nodes of JUWELS Cluster.

To provide additional insight into the GPU+MPI performance, it is informative to consider the breakdown of wallclock time as a function of individual NEMO code sections. Figure 8 shows the time spent in each code section for the ORCA12 configuration of NEMO-OCE running for 100 timesteps on 80 GPUs of JUWELS Booster, compared to an equivalent simulation running on 80 fully populated Skylake CPU sockets of JUWELS Cluster. The results are qualitatively similar to the single GPU version, in that the vast majority of the code sections are accelerated except for `dyn_spg`. However with MPI enabled, the GPU performance of the `dyn_spg` code section is relatively worse, around three times slower than the CPU baseline. This can be attributed to the fact that within the sub-stepping loop of `dyn_spg`, the fragmentation of OpenACC `KERNELS` regions is even greater in the GPU+MPI version to allow for calls to the MPI routines. This results in a further increase in the kernel launch overhead.

The same issue also affects the GPU+MPI performance of the `icedyn_rhg` and `icedyn_adv` code sections in NEMO+SI3 (see Figure 9). Like `dyn_spg`, these routines also make use of iterative loops resulting in many calls to the `lbc_lnk` routines (485 calls per timestep from `icedyn_rhg` and 87 calls per timestep from `icedyn_adv` for ORCA12). We suggest that one way to improve the GPU performance of such code sections

is to replace calls to the north-fold boundary condition routines `lbc_nfd` and `lbc_nfd_nogather` with directly embedded code, so that as much code as possible can be included within a single OpenACC `KERNELS` block. Scheduling kernels to launch asynchronously, through use of the `async` clause with the `KERNELS` construct, can also help to hide the cost of the launch overhead within iterative loops and minimise the time when the GPU is idle. Third, making use of the new support in NEMO for halo depths greater than one would also improve performance by reducing the number of required calls to the `lbc_lnk` routines.

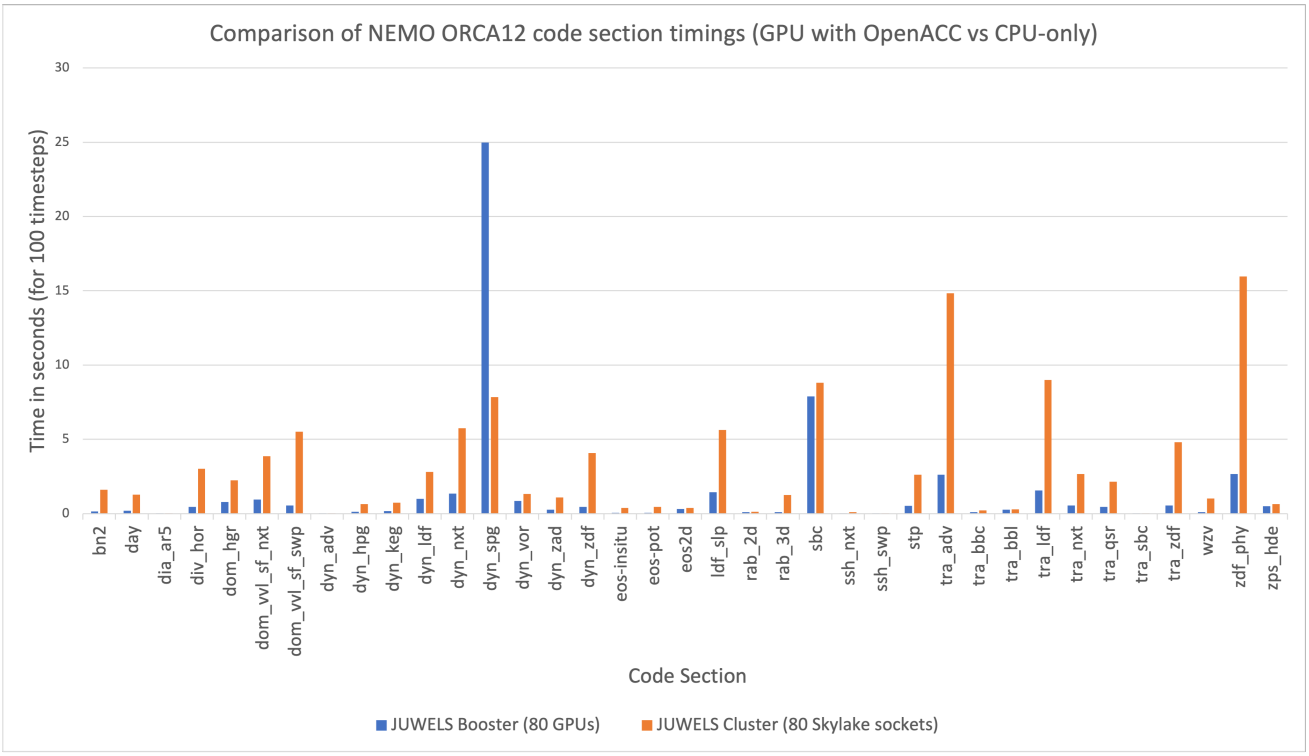


Figure 8: Plot showing the breakdown of time spent in each code section of NEMO-OCE for equivalent ORCA12 simulations running for 100 timesteps on 80 GPUs of JUWELS Booster, and 80 CPU sockets of JUWELS Cluster.

4.2.5 OpenMP-CPU Performance

In addition to adding OpenMP-offload directives so as to target GPU, PSyclone can also be used to add OpenMP directives to parallelise loops on CPU. As described in the previous section, this has required further development of PSyclone within this project because of the need to transform NEMO's extensive use of Fortran array notation into explicit loops.

For this work we have used an ORCA025 ($\frac{1}{4}$ degree) global configuration of NEMO incorporating both Ocean and Sea Ice components, supplied by ECMWF. This configuration has a global domain size of 1442×1207 grid points and 75 vertical levels. Since ECMWF have an operational requirement to be able to run NEMO in mixed (OpenMP+MPI) mode, they currently have a manually-implemented version which they maintain themselves. We are therefore able to compare the performance of the PSyclone-generated code with this version. For this comparison we run NEMO on 32 nodes of Scafell Pike with four MPI processes per node and eight OpenMP threads per process. (This was chosen as being representative of ECMWF's operational setup.) The results are shown in Figure 10.

The turquoise bars show the performance obtained with a version of the transformation script that simply attempts to parallelise the outermost loop of any loop nest it encounters. (If this loop proves not to be

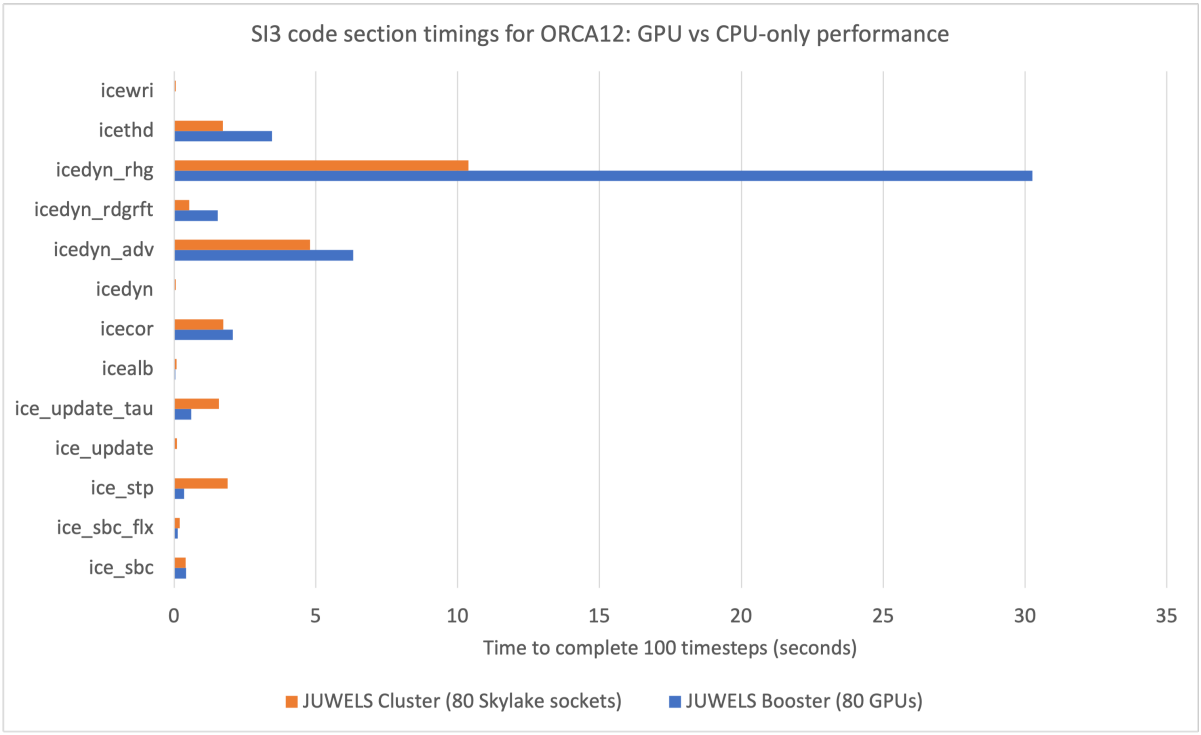


Figure 9: Plot showing the breakdown of time spent in each SI3 code section of NEMO+SI3 for equivalent ORCA12 simulations running for 100 timesteps on 80 GPUs of JUWELS Booster, and 80 CPU sockets of JUWELS Cluster.

parallelisable, then the next, inner loop is considered and so on.) This shows that about two thirds of the subroutines run at the same speed or faster (negative time difference) in the PSyclone-processed version, as compared to the ECMWF version. However, we can see that certain routines, particularly those dealing with the sea ice, are taking a lot longer in the PSyclone version.

An initial investigation of this issue has revealed that many of the ice routines contain nested loops where the outer loop is over ice categories (or similar) and has a very low trip count (~ 5). Since the transformation was parallelising this outermost loop, the resulting code was very inefficient. We have refined the optimisation script so that it now refuses to parallelise loops over ice categories or ice/snow layers which then automatically results in the inner loop over ice points being parallelised instead. The performance obtained with this approach is shown by the blue-grey bars in Figure 10. Although the ice routines are still slightly slower in the PSyclone version, this simple change to the transformation script has made a massive improvement. The wall-clock time for a complete model time-step now stands at 0.46s in the PSyclone version which is very close to the 0.41s obtained with the original ECMWF version.

We are therefore at the point where PSyclone has removed the need for ECMWF to maintain a separate, manual port of the NEMO source code containing the OpenMP directives. It is also likely that the PSyclone optimisation script can be further tuned to obtain "better" performance than the original ECMWF version in the near future.

4.3 dusk/dawn and ICON

The complete dry dycore, i.e. the model dynamics except for the tracer advection, of the ICON model was ported to dusk. This section reports on the performance of this ported dycore. For all benchmarks, an experiment of moderate resolution of roughly 40'000 columns at 65 vertical levels was used. Since the dusk/dawn tool-chain does not support MPI communications, no high resolution experiments that do not fit

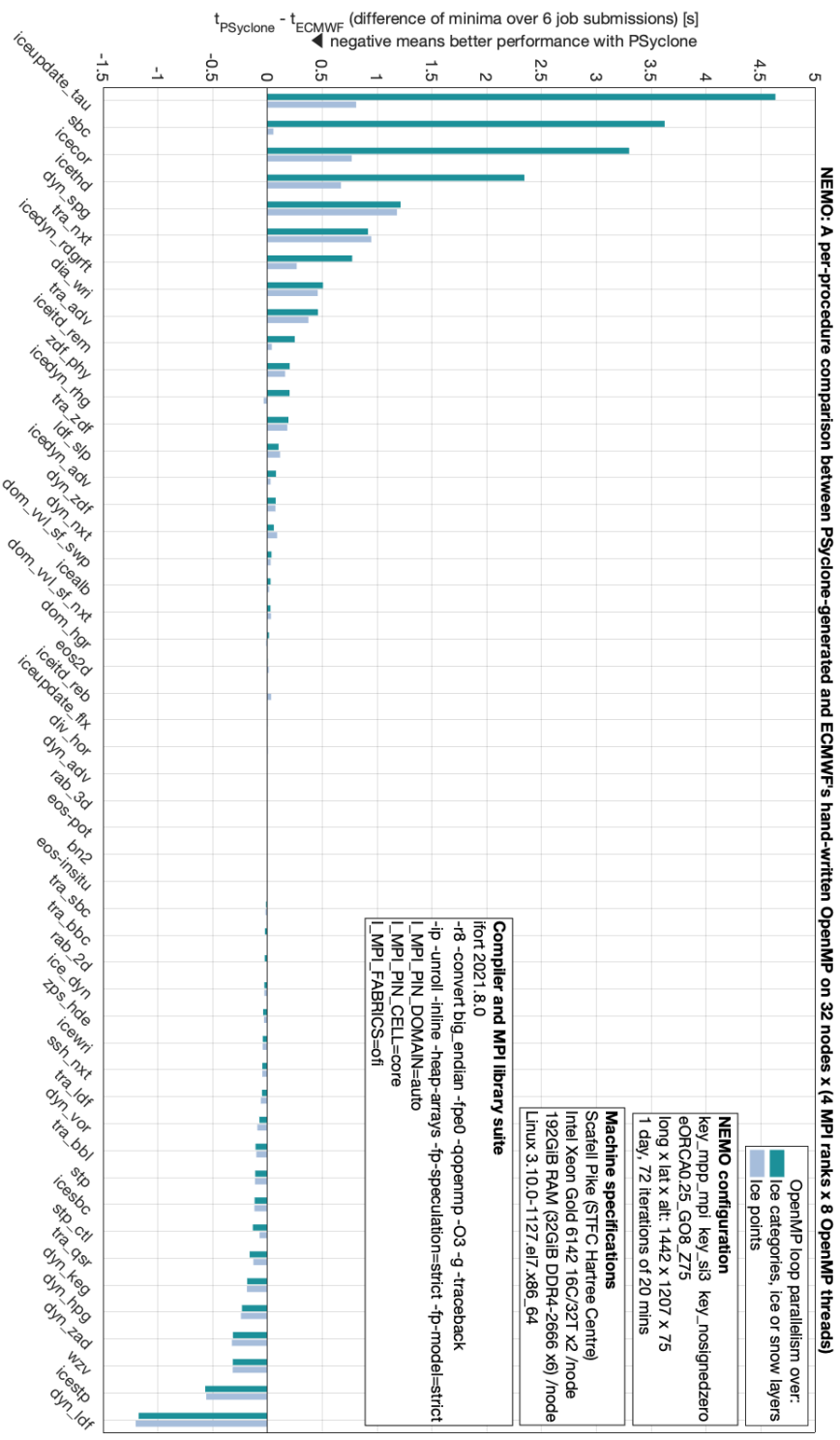


Figure 10: Plot comparing the performance of the ECMWF-manual and PSyclone-generated MPI+OpenMP versions of NEMO running on 32 nodes of Scafell Pike.

on a single GPU were run.

4.3.1 Machine Configuration

All tests were performed on the tsa super computer of MeteoSwiss, which features 18 GPU compute nodes equipped with 8 NVIDIA V100 GPUs each. However, as mentioned before, dusk/dawn does not support MPI

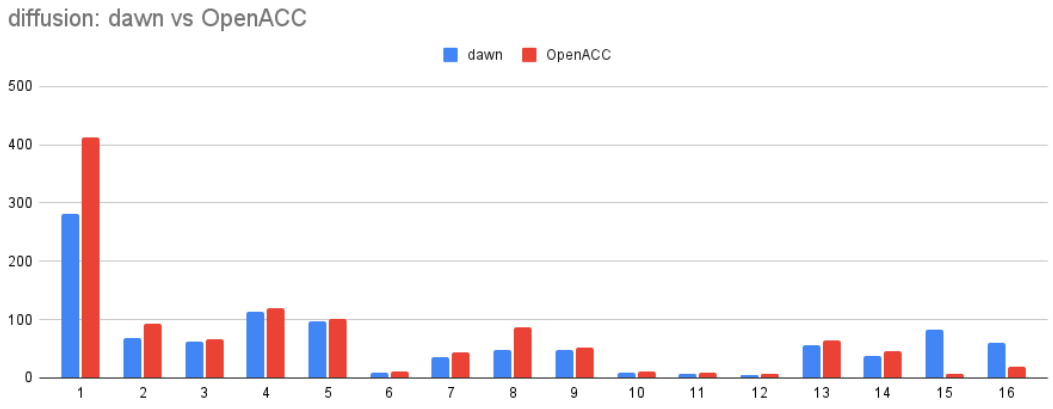


Figure 11: Performance of dawn vs OpenACC for the diffusion submodule.

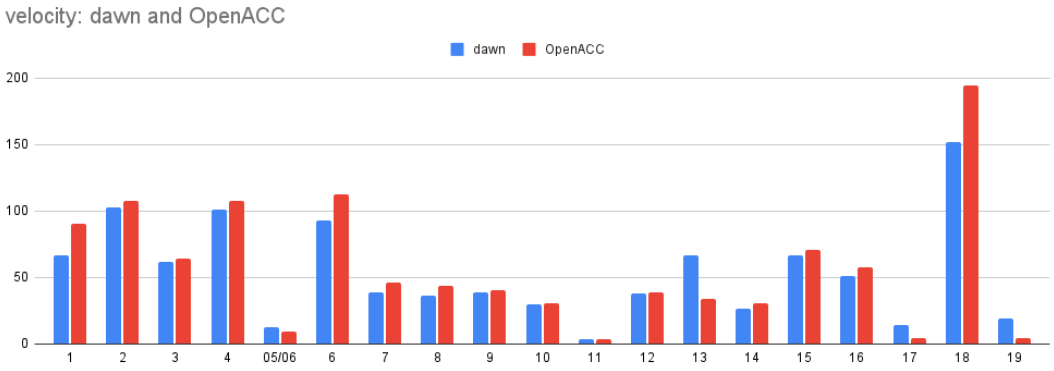


Figure 12: Performance of dawn vs OpenACC for the velocity advection submodule.

exchanges, hence every run was performed on a single node using a single GPU only. The parts of the ICON model which were not ported to dusk/dawn and remained in Fortran/OpenACC (e.g. Physics) as well as the OpenACC reference were compiled using the NVIDIA compiler version 21.2 and the CUDA code emitted by the dusk/dawn toolchain was compiled using CUDA 11.2.

4.3.2 Stencil by Stencil

From a computational perspective, the dycore of icon can be seen as collection of loop nests, iterating over the complete, or parts of, the computational domain. Such a loop nest is called a stencil in this text. We refrained from associating descrptive names with these loop nests, and simply numbered them consecutively for each module. Figures 11 to 14 show the performance of each dusk/dawn stencil side by side to its OpenACC reference. As can be seen, dawn outperforms the OpenACC references almost consistently. In fact, the average speedup over all stencils is roughly 13.5%.

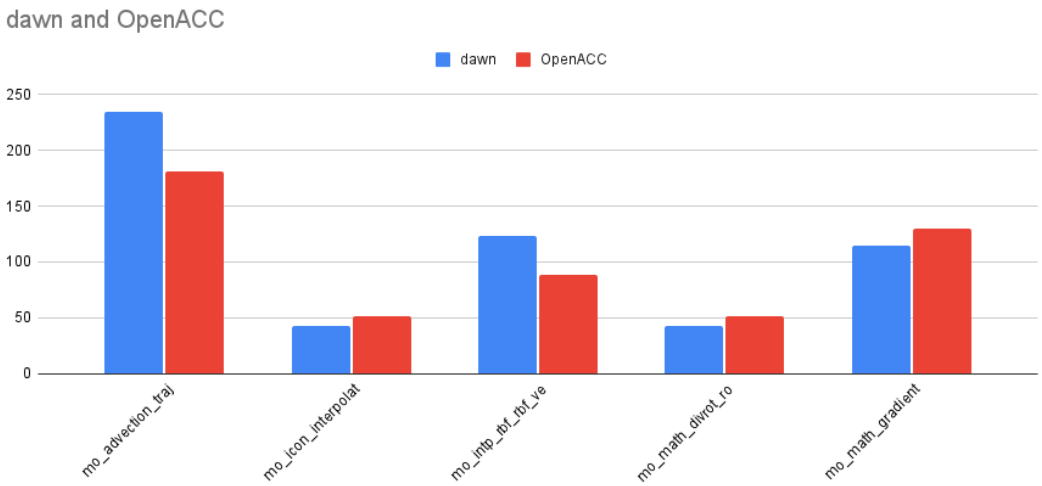


Figure 13: Performance of dawn vs OpenACC for the stencils being called in the solve nonhydro submodule.

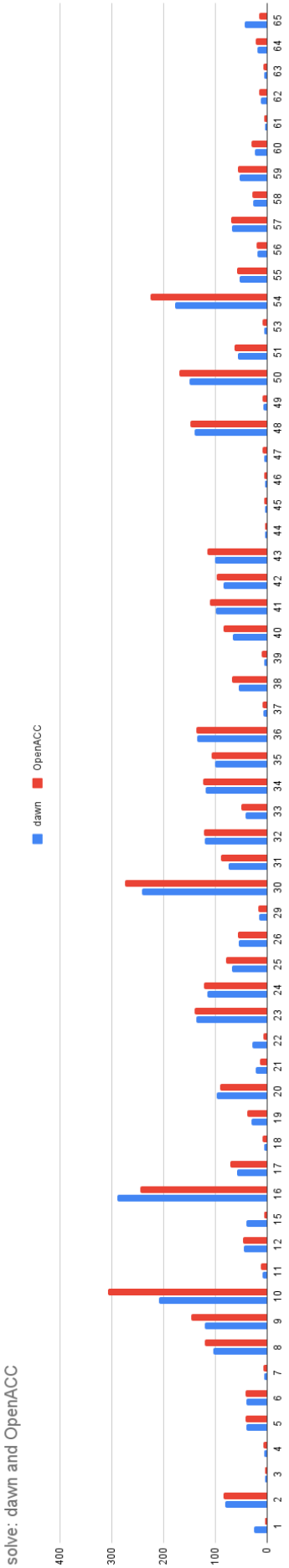


Figure 14: Performance of dawn vs OpenACC for solve nonhydro submodule.

4.3.3 Fusing

Translating the dycore stencil by stencil is very convenient for fine grained verification (see also Deliverable D2.2). However, this mode of operation does not yield the optimal performance. Stencils become quite small this way, and GPUs may not be properly saturated. Also, the compiler tool-chain has less code to reason about at any time, and optimisations may be missed by feeding it small isolated stencils. Thus, in a second step, the stencils were merged as much as possible, only breaking out back to the Fortran driver any time there is a halo exchange. In this configuration, the modules are translated into quite few stencil calls:

num sten	module name
5	diffusion
3	velocity advection
5	solve nonhydro

In this mode of operation, the velocity advection module is completely replaced by DSL function calls (in other words, the Fortran velocity advection module wouldn't even need to be compiled for correct operation). Thus, the timing of the velocity module is not indicated separately in the following table, but is part of the solve nonhydro module (typically taking about 7% of the total solve nonhydro runtime).

module name	OpenACC	DSL	speed-up
solve nonhydro	0.00724s	0.00488s	1.48x
diffusion	0.00148s	0.00100s	1.48x

This shows that the dusk/dawn DSL translation shows significant speedups over a very wide array of stencils, exhibiting a wide array of computational patterns (see also Deliverables D2.1, D2.2).

4.3.4 Performance of dusk/dawn compared to GT4Py

In Deliverables D2.1 and D2.2 it was explained that dusk and dawn is going to be replaced by a new tool chain called GT4Py. A natural question to ask is if the performance of dusk/dawn is matched by this new GT4Py tool chain. This can be answered in the affirmative. Comprehensive performance comparisons have been done on a stencil by stencil basis, showing that the dusk/dawn performance is closely matched by GT4Py. The timings of all 100 stencils will not be reproduced here, however, consider Figure 15 for a comparative timing of the diffusion submodule. The diffusion submodule was chosen because it exhibits almost any pattern present in the complete dycore, and hence represents a representative sample of stencils.

It is important to note that this does not indicate that the performance will also be equivalent for the fused situation. At the time of writing, the fusing of complete modules has not been performed. The GT4Py tool chain follows a functional programming model, potentially enabling different/more aggressive optimisations given larger stencils to process.

4.3.5 Inlining

This excursion shall serve to give the reader an understanding of what an optimisation pass in the DSL tool-chain might look like, and how it differs from optimisations conventional host compilers may perform. In conventional compilers "inlining" refers to the process of incorporating the function body of small functions into the call site. This has the benefit of foregoing the effort to set up a stack frame at the cost of (potentially) consuming more live registers. Modern compilers feature involved heuristics to decide which functions to inline. Some programming languages also allow the programmer to advise the compiler to inline a function using the `inline` keyword.

Exploiting domain knowledge, a DSL tool-chain for Finite Volume Computations on a triangular mesh, be it dusk/dawn or GT4Py, can perform another form of inlining than just discussed; stencil inlining. Since

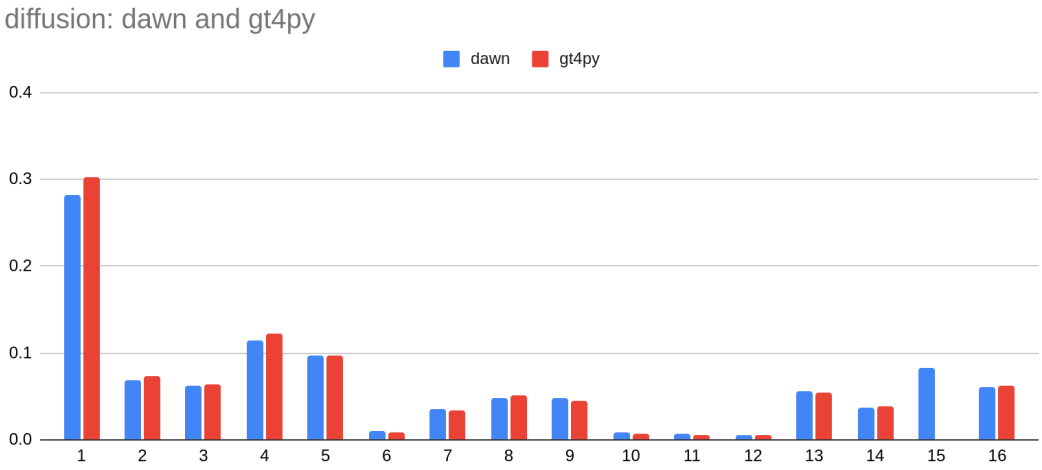


Figure 15: Performance of dawn vs. GT4Py for the diffusion submodule. As can be seen, the performance is very closely matched by the two tool-chains. The diffusion submodule was chosen because it featueres a very wide array of different computational patterns (close to every computational aspect of the complete dycore is reflected in diffusion).

the location (Cell, Edge, Vertex) where a computation is performed is a first class member of the DSLs type system, subsequent reductions can be combined into one nested reduction. See listing in Figure 16.

```
@stencil
def seq(f_e: Field[Edge],
        f_c: Field[Cell]):
  tmp_v: Field[Edge]
  with domain.upward:
    tmp_v = sum_over(Vertex>Edge, f_e)
    f_c = sum_over(Cell>Vertex, tmp_v)
```

```
@stencil
def inl(f_e: Field[Edge],
        f_c: Field[Cell]):
  with domain.upward:
    f_c = sum_over(Cell>Vertex,
                   sum_over(Vertex>Edge, f_e))
```

Figure 16: The stencil inlining pass demonstrated using dusk notation. The compiler tool-chain understands that the first reduction is onto the edges, and the second reduction reduces from edges onto cells. It can thus decide to inline the first reduction into the second.

Using the understanding about location types, the DSL tool-chain can turn two subsequent loops into a single, nested loop. This has two advantages: The memory to hold the intermediary result form the first reduction can be eliminated, and a "trip" to global memory (i.e. to store the intermediary result in said memory location, then read from it again in the second reduction) can be saved. This is an attractive optimisation on GPUs because of their steep memory hierarchy. Reading and storing values in global memory can be very expensive.

Unfortunately, this stencil inlining optimisation is not without its problems. Consider the listing in Figure 16 again. Count the number of times the body of the `sum_over` expressions has to be executed in both situations. In the sequential case that is `#vertex*|Vertex>Edge| + #cells*|Cell>Vertex| = #vertex*6 + #cells*3` and in the inlined case `#cells*|Cell > Vertex > Edge| = #cells*18`. Now, in a hexagonal triangle mesh (or nearly hexagonal triangle meshes like icosahedral triangle meshes), the ratio between vertices, cells and edges is $1 : 2 : 3$. Hence, the number of iterations for the situation in our example is $24 : 36 = 2 : 3$. This means that the speedup of the inlining optimisation needs be at least $1.\overline{33}\,x$ to "pay off". Figure 17 illustrates this for all chains of length two and a test mesh of roughly 20'000 columns. Figure 18 shows the actual run times for this mesh for inlined and sequential reductions.

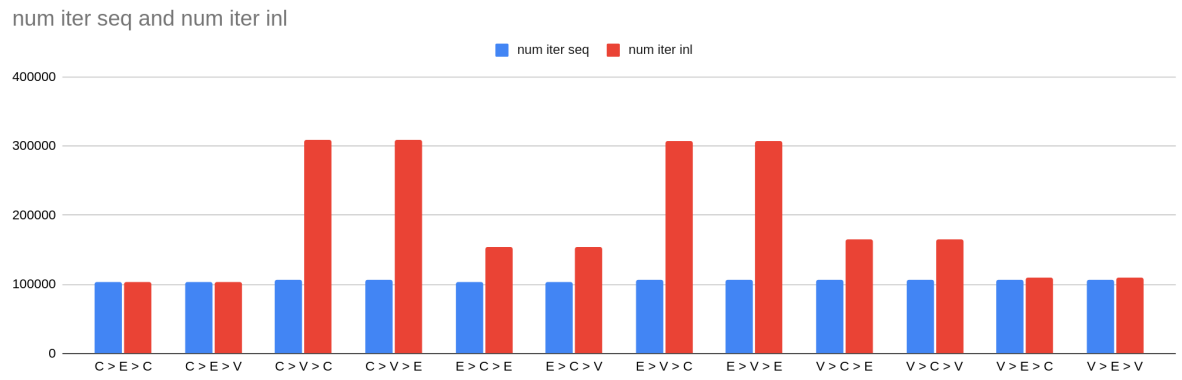


Figure 17: Number of iterations taken in the `sum_over` expression(s) for all chains of length 3, for either two reductions in succession (seq) or the equivalent nested (inl) version. See also listing in Figure 16.

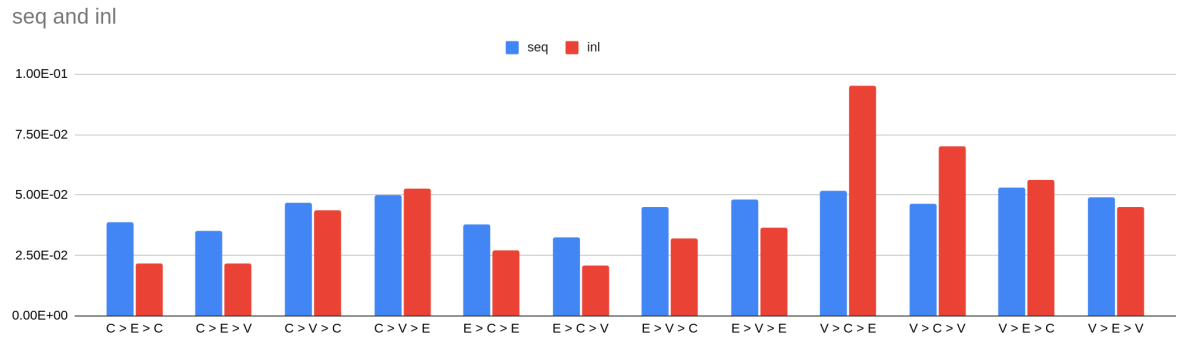


Figure 18: Runtime for reductions over all chains of length 3, for either two reductions in succession (seq) or the equivalent nested (inl) version. See also listing in Figure 16. As can be seen the transformation to a nested reduciton is only beneficial in some cases. In other cases, the performance is even defective after the transformation (e.g. `V>C>E`).

As can be seen from Figure 18 the impact of the inlining transformation is not always enough to compensate for the penalty of additional iterations. Potentially, Figure 17 could be used to inform the compiler when to execute the transformation. However, whether or not the transformation is beneficial is also affected by other factors, like caching behavior or the arithmetic intensity of the expression contained in the body of the `sum_over` concept. Hence, an optimisation that is strictly beneficial, or at least devoid from any performance defects, would be strongly preferable. In the following, two such strategies to improve upon the "simple" stencil inlining are presented.

Compressed Inlining

To understand how the simple inlining in previous elaborations can be improved further, the first thing to observe is that some locations in a chain may be visited more than once. This is illustrated for the familiar example `Cell > Vertex > Edge` in Figure 19. Now, if the outer reduction is a linear function of the inner reduction, we can just multiply its contribution by the number of times it is visited, instead of visiting more than once. To realise this multiplication, dusk offers the `weights` concept. This is illustrated in listing of Figure 20. The results for the same test mesh as used previously are reported in Figure 21.

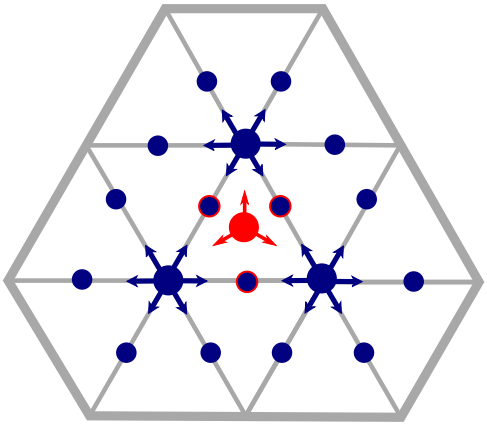


Figure 19: Multiplicity in the `Cell > Vertex > Edge` neighbor chain illustrated: the edges closest to the origin cell (origin indicated by red dot, edges closest to origin indicated by blue dots with red boundary) are visited twice.

```
@stencil
def seq(f_e: Field[Edge],
        f_c: Field[Cell]):
  tmp_v: Field[Edge]
  with domain.upward:
    tmp_v = sum_over(Vertex>Edge, f_e)
    f_c = sum_over(Cell>Vertex, tmp_v)
```

```
@stencil
def inl(f_e: Field[Edge],
        f_c: Field[Cell]):
  with domain.upward:
    f_c = sum_over(Cell>Vertex>Edge, f_e,
          weights=[2,2,2,1,1,1,1,1,1,1,1,1,1,1])
```

Figure 20: The compressed stencil inlining pass demonstrated using dusk notation. This time around, instead of nesting two reductions only one reduction is performed, using the concatenated neighbor chain `Cell > Vertex > Edge`. A purpose built weights vector accounts for the multiplicity not present in the concatenated chain.

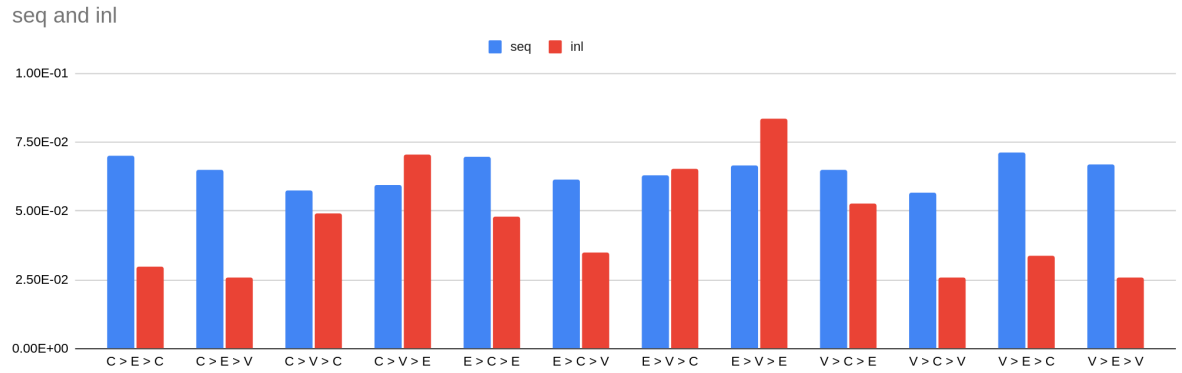


Figure 21: Runtime for reductions over all chains of length 3, for either two reductions in succession (seq) or the equivalent nested (inl) version, however, this time around the nested reduction uses the compressed version of the inlining optimisation as described in the paragraph "Compressed Inlining". As can be seen the optimisation is now almost strictly beneficial, with no defective cases.

While the results in Figure 21 are promising, the requirement with regards to linearity makes the optimisation not applicable in general. This would entail that either the user indicates whether linearity is given to the compiler, for example by a keyword, or the compiler performs an analysis pass the decide whether linearity

is given. The first option makes for poor user experience, and the other option introduces complexity into the compiler. Furthermore, linearity is not a natural requirement in this situation, making this pass potentially only available to a small subset of reductions in a given code base. In the next section yet another version of inlining is presented that again compresses the neighbor table but this time works in general.

Compressed and Unrolled Inlining

The basic idea of this optimisation of the simple inlining pass is, for a given nested reduction $\mathcal{A} > \mathcal{B} > \mathcal{C}$, where $\mathcal{A}, \mathcal{B}, \mathcal{C} \in \{\mathrm{Edge}, \mathrm{Cell}, \mathrm{Vertex}\}$: unroll the reduction completely, leave the neighbor list representing $\mathcal{A} > \mathcal{B}$ but take compress $\mathcal{A} > \mathcal{B} > \mathcal{C}$ to a "direct" neighbor list $\mathcal{A} > \mathcal{C}$ containing unique elements only. This avoids issuing multiple loads to read the same address as well as the pointer (index) chasing by indexing from the "outer" neighbor list into the "inner" one. It is hoped that this will become clearer using a worked example.

We will use a more involved stencil than previously, to highlight that this optimisation works for general stencils. In the listing below, the nesting of two subsequent reductions has already been performed. As described, in the following we will keep the $\mathrm{Cell} > \mathrm{Edge}$ list, but we will compress $\mathrm{Cell} > \mathrm{Edge} > \mathrm{Cell}$ into a special purpose $\mathrm{Cell} > \mathrm{Cell}$ list tailor made for this nested reduction.

```
@stencil
def cec_inl(kh_smag_e: Field[Edge, K], e_hat: Field[Edge],
            theta_v: Field[Cell, K], z_temp: Field[Cell, K]):
   with domain.upward.across[nudging:halo]:
       z_temp = sum_over(Cell > Edge, kh_smag_e*e_hat*sum_over(Edge>Cell, theta_v))
```

Compiling the above, the body of the CUDA kernel emitted could look like

```
for(int nbhIter0 = 0; nbhIter0 < C_E_SIZE; nbhIter0++) {
  int nbhIdx0 = ceTable[pidx * C_E_SIZE + nbhIter0];
  ::dawn::float_type lhs_7 = (::dawn::float_type)0;
  for(int nbhIter1 = 0; nbhIter1 < E_C_SIZE; nbhIter1++) {
    int nbhIdx1 = ecTable[nbhIdx0 * E_C_SIZE + nbhIter1];
    lhs_7 += theta_v[(kIter+0) * CellStride + nbhIdx1];
  }
  lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * e_hat[nbhIdx0]) * lhs_7);
}
z_temp[(kIter+0) * CellStride + pidx] = lhs_23;
```

Note the nested `for` loop structure reflecting the nested `sum_over` expressions from the dusk listing. Now, in a first step, the inner loop is unrolled.

```
for(int nbhIter0 = 0; nbhIter0 < C_E_SIZE; nbhIter0++) {
  int nbhIdx0 = ceTable[pidx * C_E_SIZE + nbhIter0];
  double inner = theta_v[(kIter+0) * CellStride + ecTable[nbhIdx0*E_C_SIZE+0]]
               + theta_v[(kIter+0) * CellStride + ecTable[nbhIdx0*E_C_SIZE+1]];
  lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * e_hat[nbhIdx0]) * inner);
}
z_temp[(kIter+0) * CellStride + pidx] = lhs_23;
```

Note the index/pointer chasing here. `nbhIdx0` is loaded from `ceTable` and consumed to look up an index from `ecTable`. The caching behavior of such an operation is generally bad. We now go one step further and also unroll the outer reduction

```
int nbhIdx0 = ceTable[pidx * C_E_SIZE + 0];
double inner0  = theta_v[(kIter+0) * CellStride + ecTable[nbhIdx0*E_C_SIZE+0]]
               + theta_v[(kIter+0) * CellStride + ecTable[nbhIdx0*E_C_SIZE+1]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner0);

int nbhIdx1 = ceTable[pidx * C_E_SIZE + 1];
double inner1  = theta_v[(kIter+0) * CellStride + ecTable[nbhIdx1*E_C_SIZE+0]]
               + theta_v[(kIter+0) * CellStride + ecTable[nbhIdx1*E_C_SIZE+1]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx1] * ehat[nbhIdx1]) * inner1);

int nbhIdx2 = ceTable[pidx * C_E_SIZE + 2];
 double inner2  = theta_v[(kIter+0) * CellStride + ecTable[nbhIdx2*E_C_SIZE+0]]
                + theta_v[(kIter+0) * CellStride + ecTable[nbhIdx2*E_C_SIZE+1]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner2);

z_temp[(kIter+0) * CellStride + pidx] = lhs_23;
```

So far, no optimisation was performed. The nested loop structure was just unrolled completely. Since the loop bounds are known at compile time, the CUDA host compiler would do this transformation when translating to GPU assembly instructions (SASS) either way. The first part of the actual optimisation is now to purpose build a special neighbor table `ccTable`.

```
int nbhIdx0 = ceTable[pidx * C_E_SIZE + 0];
double inner0 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+0]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+1]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner0);

int nbhIdx1 = ceTable[pidx * C_E_SIZE + 1];
double inner1 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+2]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+3]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx1] * ehat[nbhIdx1]) * inner1);

int nbhIdx2 = ceTable[pidx * C_E_SIZE + 2];
double inner2 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+4]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+5]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner2);

z_temp[(kIter+0) * CellStride + pidx] = lhs_23;
```

Note that the index/pointer chasing is now gone, the `ccTable` can be addressed directly using only the thread index `pidx`. Note also that the table reflect the first part of the chain ($\mathcal{C}\rceil\updownarrow\updownarrow > \mathcal{E}\lceil\}\rceil$) is still present because it is required to address the field located on edges. Now, the final part of the optimisation at hand is to exploit that `ccTable` only contains 4 different values. This is the compression step indicated in the title of this subsection.

```
int nbhIdx0 = ceTable[pidx * C_E_SIZE + 0];
double inner0 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+0]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+1]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner0);

int nbhIdx1 = ceTable[pidx * C_E_SIZE + 1];
```

```
double inner1 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+0]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+2]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx1] * ehat[nbhIdx1]) * inner1);

int nbhIdx2 = ceTable[pidx * C_E_SIZE + 2];
double inner2 = theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+0]]
              + theta_v[(kIter+0) * CellStride + ccTable[pidx*C_C_SIZE+3]];
lhs_23 += ((kh_smag_e[(kIter+0) * EdgeStride + nbhIdx0] * ehat[nbhIdx0]) * inner2);

z_temp[(kIter + 0) * CellStride + pidx] = lhs_23;
```

Figure 22 shows the runtimes of a select number of reductions of length three, i.e. reductions of the form $\mathcal{A} > \mathcal{B} > \mathcal{A}$ (starting location equals end location). Again, the optimisation is always strictly beneficial, with no defective cases. Compared to the optimisation in the paragraph "Compressed Inlining" however, this time around the optimisation is applicable in general.

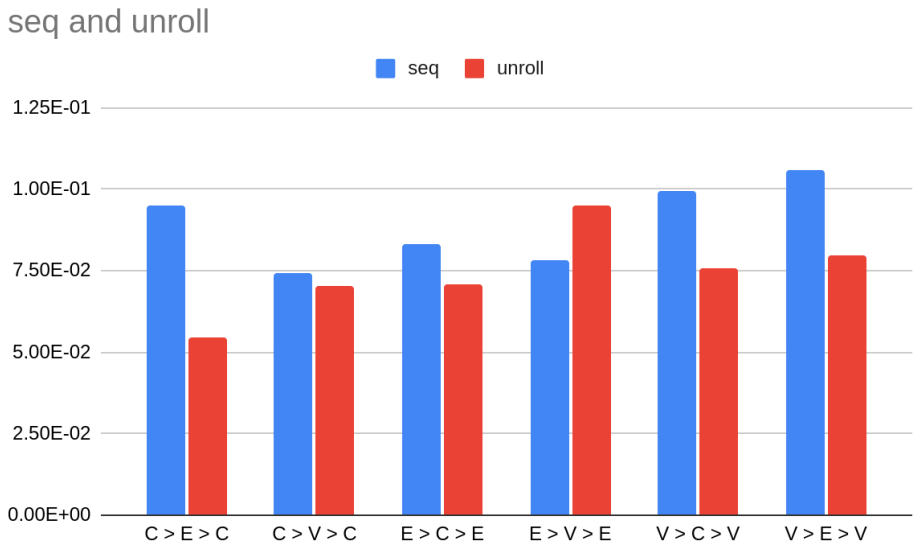


Figure 22: Runtime for reductions over all chains of length 3 where the starting location is equal to the end location, for either two reductions in succession (seq) or the equivalent nested (inl) version, however, this time around the nested reduction uses the compressed and unrolled version of the inlining optimisation as described in the paragraph "Compressed and Unrolled Inlining". As can be seen the optimisation is now almost strictly beneficial, with no defective cases.

Summary

Reduction inlining presents a challenging situation where avoiding memory traffic can be traded for additional hot loop iterations, where the size of the trade off is dependent on the specific sequence of reductions being merged. Three optimisations with regards to the inlining of reductions were presented. A simple version that can be readily implemented into any DSL tool chain, and two more involved versions. The compressed version is not applicable in general, and hence not recommended to be implemented as a pass in a DSL compiler. The compressed and unrolled version is, however setting up tailor made neighbor lists on a per stencil basis is extremely challenging. It is still the most promising candidate of the inlining optimisations presented, though.

4.4 IFS-FVM with GT4Py

Due to effort constraints there was little time to do any performance and/or performance portability analysis of the newly created GT4Py version of the dynamical core of FVM. However, once ported to GT4Py the DSL does support a number of user-selectable backends for CPU and GPU hardware (which is one of the big advantages of such a system).

Figure 23 presents first results using these different backends. The performance results are each for a single node and are snapshots of the current state of development of the DSL and dynamical core with further optimisation pending. Results are shown for two CPU backends and three different GPU backends (please see the figure caption for details). All runs were performed on Piz Daint which contains Intel Xeon E5-2690 v3 CPUs running at 2.60GHz with 12 cores and 64GB of RAM and NVIDIA Tesla P100 GPUs with 16GB of memory. All backends validate for the baroclinic wave instability simulation shown in Fig. 24 (for convenience of the reader this figure has been duplicated here from D2.2). Distributed multi-node execution is already possible and can be used to achieve high resolution with the model but was not investigated here.

It can be seen that at this stage of development the CPU backends give similar performance even with quite a dramatic change in data layout, with k-first slightly winning over i-first. The GPU backends all perform nearly identically and the GPU performance significantly outperform the CPU performance - presumably due to their greater memory bandwidth.

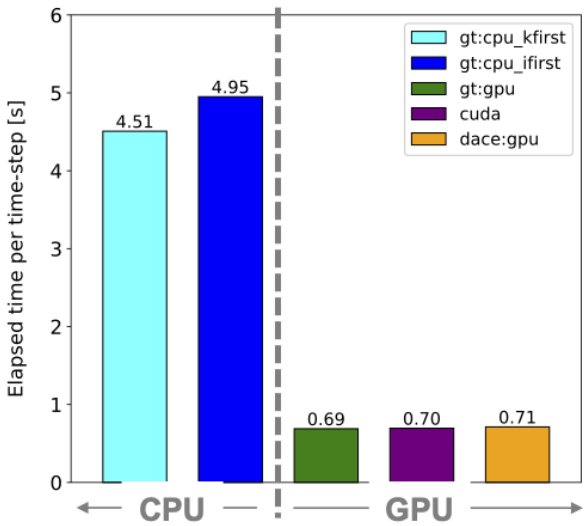


Figure 23: Single-node performance measures for the FVM dynamical core using various backends of GT4Py. Here, the model was run for the baroclinic wave instability test on the hybrid partition of the Piz Daint supercomputer at CSCS. Tested backends are from GridTools (Afanasyev et al., 2021) and DaCe (Ben-Nun et al., 2022).

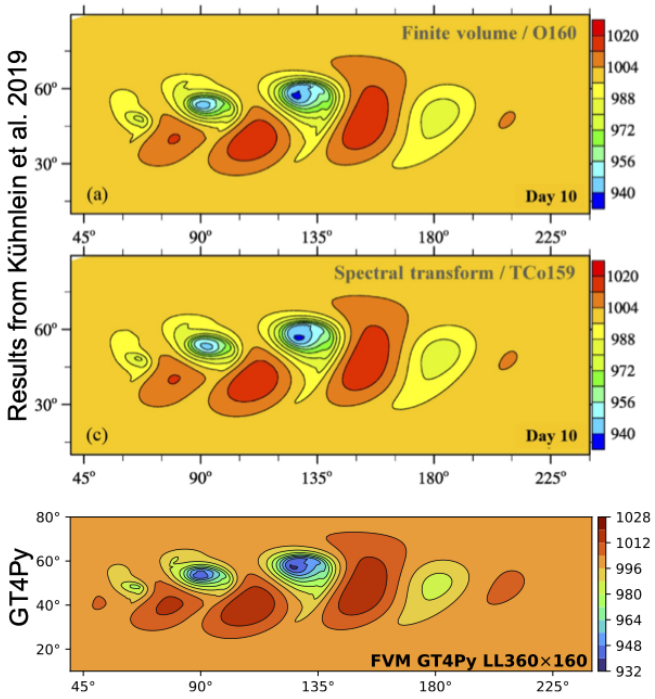


Figure 24: Pressure on lowest level (hPa) at day 10 of the baroclinic wave instability test. Shown are solutions for the FVM Fortran code (top panel), the spectral-transform IFS (middle panel) and the FVM GT4Py code (bottom panel). Note that a slightly different color scheme and labelling is used in the bottom panel, but the actual contour level values are identical in all three panels.

5 References

A. Afanasyev, M. Bianco, L. Mosimann, C. Osuna, F. Thaler, H. Vogt, O. Furher, J. VandeVondele, and Schulthess T. GridTools: A framework for portable weather and climate applications. *SoftwareX*, 15, 2021. doi: https://doi.org/10.1016/j.softx.2021.100707. URL `https://www.sciencedirect.com/science/article/pii/S2352711021000522`.

T. Ben-Nun, L. Groner, F. Deconinck, T. Wicky, E. Davis, J. Dahm, O. D. Elbert, R. George, J. McGibbon, L. Trümper, E. Wu, O. Fuhrer, T. Schulthess, and T. Hoefler. Productive Performance Engineering for Weather and Climate Modeling with Python. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'22)*, pages 1–14, Nov. 2022. doi: 10.1109/SC41404.2022.00078. URL `https://ieeexplore.ieee.org/document/10046105`.

Rupert Ford, Andrew Porter, Matthias Roethlin, Carlos Osuna, Christian Kühnlein, Iva Kavcic, and Nuno Nobre. Report summarising the adaptation of the proposed DSLs and the evaluation of the benchmarks and models proposed in this project - Deliverable D2.1, December 2022. URL `https://doi.org/10.5281/zenodo.7476411`.

Rupert Ford, Andrew Porter, Matthias Roethlin, Carlos Osuna, Christian Kühnlein, Iva Kavcic, Chris Deardon, Nuno Nobre, and Sergi Siso. Demonstration of the DSLs in the proposed models - Deliverable D2.2, January 2023. (confidential, provided via EC).

6 Changes made and/or difficulties encountered, if any

None.

7 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. Demonstrating the ability of DSLs to be used in the key European codes targeted in this deliverable - LFRic, NEMO, ICON and IFS - presents an opportunity to the code developers to adopt the DSL approach 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.

8 Sustainability

This deliverable is primarily concerned with the computational performance obtained by using the PSyclone and dawn DSLs[2] in the key European Weather and Climate models: NEMO, LFRic, ICON and IFS. Related deliverables discuss the extension of the DSLs to support these models (D2.1) and the practical issues around use of the DSLs in production models (D2.2).

The developments in this deliverable have benefited from close links with the now-completed ESCAPE-2 project, where dawn and GridTools were developed with IFS and ICON in mind. Further work on GT4Py continues in the EXCLAIM project which is strongly influenced by the dusk and dawn developments in ESiWACE2, as mentioned in Deliverable D2.1.

There are also strong links to the Met Office Next Generation Modelling Systems (NGMS) programme which is supporting PSyclone as both part of the developing LFRic atmosphere model and for application to a number of marine-systems models including NEMO, NEMOVAR, MEDUSA and Wavewatch III. PSyclone will also be evaluated for use in the Met Office's NAME dispersion model, which is used operationally for events such as nuclear accidents and volcanic eruptions as well as forecasts of air quality. Work will also continue in the recently funded ESiWACE3 project where CMCC and BSC will evaluate PSyclone for use in their NEMO configurations and ECMWF will continue to develop IFS-FVM in GT4Py, running on Exascale resources.

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 UK 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. Putting this integration onto NEMO trunk is now in progress (see `https://forge.nemo-ocean.eu/nemo/nemo/-/issues/140`). Work in this area is set to continue in ESiWACE3 with both CMCC and BSC committing to evaluate the PSyclone-processed NEMO code on GPU.

9 Dissemination, Engagement and Uptake of Results

9.1 Target audience

As indicated in the Description of the Action, the audience for this deliverable is:
In order to ensure the uptake of the deliverable by the targeted audience, the document will be distributed to all members of the consortium and Commission services.

9.2 Record of dissemination/engagement activities linked to this deliverable

Please see Table 10 and Table 11.

[2] As well as the GT4Py DSL that is closely related to dawn.

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)

9.3 Publications in preparation OR submitted

Please see Table 12.

9.4 Intellectual property rights resulting from this deliverable

dusk/dawn, PSyclone and fparser2 are open-source projects hosted on github with a BSD 3-clause license. The models evaluated in this project each have their own licences, which can be obtained from the model owners.

Table 10: 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	https://zenodo.org/record/3971994	30
Conference presentation	PSyclone and its use in LFRic	Platform for Advanced Scientific Computing (PASC) Conference, 12 Jun. 2019, online	Technical	https://zenodo.org/record/3252144	30
Participation to a workshop	PSyclone, LFRic, NEMO and SIR	1st ESCAPE-2 Dissemination Workshop, 21-22 October 2019, Reading, UK	Scientific Community	https://zenodo.org/record/3972115	60
Participation to a workshop	Recent Advances in PSyclone ... the PSyIR	6th ENES HPC Workshop, 26 May 2020, online	Scientific Community	https://zenodo.org/record/3972140	80
Presentation	Developing DSLs in ESiWACE2	ESiWACE2 Workshop on Emerging Technologies for Weather and Climate Modelling, 30 June 2020	Scientific Community	https://zenodo.org/record/3972151	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	https://zenodo.org/record/5795411, https://hps.vi4io.org/events/2020/esiwace-school	50
Presentations and tutorials	ESiWACE2 training course on Domain-specific Languages in Weather and Climate	23-27 Nov. 2020, online	Model developers	https://zenodo.org/record/5795411	30

Table 11: 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	https://zenodo.org/record/4922906	10
Presentation	LFRic and PSyclone: Meeting Challenges on the Road to Exascale	PASC 21, 5-9 July 2021, online	Scientific Community	https://zenodo.org/record/6078561	20
Presentation	A Whirlwind Tour of PSyclone	PASC 21, 5-9 July 2021, online	Scientific Community	https://zenodo.org/record/5961131	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	https://zenodo.org/record/5795411, https://hps.vi4io.org/events/2021/esiwace-school	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	https://zenodo.org/record/7378274	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	https://zenodo.org/record/7374307	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	https://indico.dkrz.de/event/45/	50

Table 12: 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 Kavcic, 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