

Accelerating the Dutch Atmospheric Large-Eddy Simulation (DALES) Model with OpenACC

Esclapez, Lucas; Soucasse, Laurent; Jungbacker, Caspar; Jansson, Fredrik; de Roode, Stephan R.; Costa, Pedro; van den Oord, Gijs; Sclocco, Alessio

DOI

[10.1109/IPDPS64566.2025.00066](https://doi.org/10.1109/IPDPS64566.2025.00066)

Publication date

2025

Document Version

Final published version

Published in

International Symposium on Parallel and Distributed Processing (IPDPS)

Citation (APA)

Esclapez, L., Soucasse, L., Jungbacker, C., Jansson, F., de Roode, S. R., Costa, P., van den Oord, G., & Sclocco, A. (2025). Accelerating the Dutch Atmospheric Large-Eddy Simulation (DALES) Model with OpenACC. In *International Symposium on Parallel and Distributed Processing (IPDPS)* (pp. 678-688). IEEE. <https://doi.org/10.1109/IPDPS64566.2025.00066>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

**Green Open Access added to [TU Delft Institutional Repository](#)
as part of the Taverne amendment.**

More information about this copyright law amendment
can be found at <https://www.openaccess.nl>.

Otherwise as indicated in the copyright section:
the publisher is the copyright holder of this work and the
author uses the Dutch legislation to make this work public.

Accelerating the Dutch Atmospheric Large-Eddy Simulation (DALES) model with OpenACC

Lucas Esclapez
Netherlands eScience Center
 Amsterdam, The Netherlands
l.esclapez@esciencecenter.nl
 0000-0002-2438-7292

Laurent Soucasse
Netherlands eScience Center
 Amsterdam, The Netherlands
 0000-0002-5422-8794

Caspar Jungbacher
Dept of Geoscience & Remote Sensing,
Delft University of Technology,
 Delft, The Netherlands
 0009-0001-1031-9010

Fredrik Jansson
Dept of Geoscience & Remote Sensing,
Delft University of Technology,
 Delft, The Netherlands
 0000-0003-4712-5938

Stephan R. de Roode
Dept of Geoscience & Remote Sensing,
Delft University of Technology,
 Delft, The Netherlands
 0000-0003-3217-8009

Pedro Costa
Process & Energy Department,
Delft University of Technology,
 Delft, The Netherlands
 0000-0001-7010-1040

Gijs van den Oord
Netherlands eScience Center
 Amsterdam, The Netherlands
 0000-0001-8367-1333

Alessio Sclocco
Netherlands eScience Center
 Amsterdam, The Netherlands
 0000-0003-3278-0518

Abstract—This paper presents the GPU porting through OpenACC directives of the Dutch Atmospheric Large-Eddy Simulation (DALES) application, a high-resolution atmospheric model. The code is written in Fortran 90 and features parallel (distributed) execution through spatial domain decomposition. We assess the performance of the GPU offloading, comparing the time-to-solution on regular and accelerated HPC nodes. A weak scaling analysis is conducted and portability across NVIDIA A100 and H100 hardware is discussed. Finally, we show how targeted kernels can benefit from further optimization with Kernel Tuner, a GPU kernels auto-tuning package.

I. INTRODUCTION

Atmospheric models are among the most challenging physical models to solve numerically. Model complexity and associated computational costs come from the chaotic nature of the airflow dynamics and its coupling with various transport processes such as precipitation, chemical reactions, solar and thermal infrared radiation and ocean/land interactions. The accuracy and reliability of the simulations strongly depend on the spatial and temporal resolutions with which the model is discretized and, in return, on the efficient use of significant computational resources.

Atmospheric numerical models naturally rely on parallel implementation on powerful supercomputers. The spatial discretization employed in atmospheric models is well suited for domain decomposition, with independent mesh partitions distributed on compute clusters using MPI. The last decade has seen the massive use of GPU accelerators in high performance computing (HPC), and GPUs represent 3/4 of the computational power of the machines in the Top500 list today [1].

The climate community has thus explored different ways to efficiently use GPUs.

The community early work focused on converting selected computationally expensive kernels to CUDA [14], leading to up to 20x speedup for given compute kernels and 1.3x speedup at the solver level (on a single CPU versus single GPU basis). Subsequent efforts aimed at accelerating entire solvers and led to the development of solvers such as NIM-CUDA [9], GALES [18] or MicroHH [20], the later two introducing a C++/CUDA re-write of a previous Fortran code base. However, such drastic approaches require significant code development investments and disruptive changes for the users. In recent years, directive-based acceleration using OpenACC has been successfully applied to weather and climate Fortran code bases [13], [15], [8], striking a compromise between implementation efforts and performance on GPU.

This paper addresses the GPU porting of a high-resolution atmospheric model called the Dutch Atmospheric Large-Eddy Simulation (DALES) model through OpenACC directives. We first present the DALES implementation and our acceleration strategy in Sec. II. The simulation experiments we perform to assess the offloading performance are described in Sec. III. We then discuss the offloading results in Sec. IV, obtained on a single-node basis and different hardware, namely NVIDIA A100 and NVIDIA H100 (Snellius supercomputer). This section also covers the multi-node performance and a weak scaling analysis. Finally, we address how Kernel Tuner [21], an auto-tuning GPU application, can further improve the performance in Sec. V, and we conclude in Sec. VI.

II. DALES FRAMEWORK

A. Modelling

DALES is an atmospheric large-eddy simulation code, used to model atmospheric processes such as convection, clouds, and precipitation. It is generally run with a horizontal resolution of 10 to 200 m and a vertical resolution of 10 to 50 m, meaning that it can explicitly simulate clouds and convective updrafts. The model combines a large-eddy simulation of fluid dynamics with models of physical processes in the atmosphere. The dynamics is solved for incompressible air, in the anelastic approximation [2].

The fluid dynamics part contains advection of momentum and scalars in finite difference formulation. Several advection schemes are available, see Ref. [10]. Time integration is performed by an explicit third order Runge-Kutta scheme. Since the flow is assumed incompressible, a Poisson equation for the pressure field is solved at each time step to ensure mass conservation. As the spatial discretization along horizontal directions uses a regular grid spacing, a FFT-based approach for the finite-difference Poisson equation is used in the lateral directions [3]. Subgrid turbulence, i.e. diffusion caused by turbulent eddies smaller than the grid size, is modelled with the Deardorff scheme [6]. A prognostic model variable for the subgrid turbulent kinetic energy is used to estimate diffusion coefficients for momentum and other transported quantities [5]. Such a *subgrid-scale model* is almost what defines a large-eddy simulation: the model resolves the “large” turbulent eddies and parameterizes effects of the smaller ones.

The physical processes include:

- buoyancy of air dependent on temperature and moisture content;
- condensation of water vapor, forming cloud droplets and releasing latent heat, hereafter referred to as “thermodynamics”;
- “cloud microphysics”, i.e. the conversion of cloud droplets to rain droplets, and other processes related to rain or snow;
- radiative transfer, solved using the independent column approximation within both the infrared range (longwave) and visible solar light range (shortwave). There are several schemes for simulating the radiative transfer. One of them is RTE-RRTMGP [16], which we focus on here since it supports offloading to GPUs;
- atmospheric chemistry, modelling chemical reaction between chemical species in the atmosphere (optional and not considered in this article).

B. Implementation

DALES is written in Fortran 90 and employs an MPI-based domain decomposition approach for parallelism. The code base consists of about 75k lines of codes, organised in physics, numerics and utility modules. The basic data structures, stored in data modules, are Fortran multi-dimensional arrays which are allocated only once upon initialization since the computational grid is fixed. 3D-arrays are indexed ijk , meaning

the k indices associated with the vertical direction are not addressed contiguously. The domain decomposition partitions the computational grid into z -pencils (in the direction normal to the ground), generating Cartesian sub-domains whose size in the horizontal directions is inversely proportional to the number of MPI ranks. On each partition, halo cells are added in the horizontal directions to ensure data consistency across MPI ranks using non-blocking send/receive exchanges. The number of halo cells varies from 1 to 3 depending on the advection scheme stencil size. DALES does not feature shared-memory parallelization (e.g. using OpenMP).

DALES includes a bundled netlib fast Fourier Transform (FFT) module to ensure portability of the Poisson solver, but also relies on external libraries such as FFTW for increased efficiency and flexibility. Finally, DALES employs NetCDF for most of its input/output operations except checkpointing which uses plain binary format.

C. Acceleration

Owing to the size of the code base and the long duration of the tuning and validation of the model parameters, a complete overhaul of the solver in a more modern and GPU-friendly programming language (e.g. C++) is not an option. Additionally, DALES is used by researchers and students alike (see Ref. [7] for instance), with the later group often having a limited computer science background thus making Fortran’s imperative programming style a good choice. OpenACC enables to leverage the high throughput of GPUs with limited disruption to the code base and has been successfully applied to several fluid and weather solvers over the past few years [22], [15], [8], [4]. The following provides an overview of the main aspects of the DALES acceleration.

1) *Data management*: Data containers in DALES are distributed across the physics, numerics and utility Fortran modules, each module managing the dynamic memory allocation and deallocation of the multi-dimensional arrays necessary for its computations. In early GPUs (e.g. NVIDIA Kepler and older), the limited available device memory required careful management of the data loaded in the device memory for each kernel, and data transfers to and from the device was found to be a significant bottleneck software developers had to overcome in order to obtain good performance on GPUs. Leveraging the larger memory available on modern GPUs, the entire memory of the CPU is mirrored on the device in DALES, with each module using the `enter data copyin` clause once after initialization of the data on the CPU, and clearing the device memory with `exit data delete` clause before deallocating the arrays on the CPU. Thus, data is always assumed to be already present on the device when launching compute kernels and data copy from the device to the host memory only happens when performing diagnostics or checkpointing.

2) *Overall porting strategy*: DALES typical compute subroutines fall into one of the following categories:

- nested independent ijk or $ijkn$ loops, traversing the three spatial dimensions and possibly a fourth dimension in the scalar index space.
- similarly nested loops but carrying a dependency over the k index corresponding to the vertical direction (normal to the ground).

The sizes of the ijk loops are case dependent and can be found in a wide range of values when considering the typical use case of DALES. As such, a premature optimization of the OpenACC pragmas for a given case size is not considered. Rather, our default approach consists in using the `collapse` clause on as many tightly nested loops as possible in order to expose a maximum of parallelism to the GPU while remaining portable, with the *stride-1* inner loop index, as follows:

Listing 1. Typical DALES triply-nested Fortran loop

```
!$acc parallel loop collapse(3)
do k = 1, kmax
  do j = 1, jmax
    do i = 1, imax
      phi(i,j,k) = ...
    enddo
  enddo
enddo
```

Alternate OpenACC clauses, including changing the `parallel loop` default parameters and caching will be discussed in Sec. V. Most of DALES compute kernels are concise, such that register pressure is not a limiting factor except in a few specific cases that will be discussed Sec. V. DALES ijk loops often include an `if` statement for the surface layer ($k = 1$). To avoid branching within the device code and the resulting thread divergence, the $k = 1$ case is launched in a separate kernel, using the `async` clause to hide the launch latency. For ijk loops carrying a k dependency, atomic operations are used when the dependency is strictly additive (e.g. in the cloud sedimentation kernels, where the cloud droplets are transferred between adjacent k levels) or a temporary array is introduced.

When possible, consecutive independent kernels are fused but DALES often intertwine compute kernels and statistics sampling in order to extract diagnostics able to provide insights into the complex physics at play. In this later case, `async` clauses in separate queues allows to hide launch latency while keeping the kernels in separate subroutines for readability and convenience.

3) *Refactoring*: In order to enable collapsing as many loops as available, local refactoring of several compute kernels was necessary to displace k -only dependent computations into the innermost loop, at the small price of repeated calculations. More extensive refactoring was needed in microphysics and thermodynamics, where differences in the local state can introduce significant thread divergence. Two examples of such refactoring are:

- All the rain microphysics computations in DALES are masked to take advantage of the observation that rain

droplets are generally present in a limited range of altitude (k -levels) and clustered in "showers" in the spanwise directions. The actual range of k in microphysics ijk loops is based on a scan of the mask, initially relying on k -only loop combined with Fortran-array syntax and an `exit` statement on CPU. This approach performs poorly on GPUs, where a reduction across a fully collapsed ijk loops was used. In this case, separate versions of the code are needed since the GPUs approach proved more expensive on CPU.

- DALES uses a saturation adjustment scheme in the thermodynamics routines to diagnose cloud liquid water content. To adjust the liquid water potential temperature accordingly, a Newton-Raphson algorithm is needed. The local conditions in each computational cell can lead to a varying number of Newton iterations needed to reach the desired accuracy, introducing thread divergence. Replacing the iterative algorithm by explicitly unrolling a fixed number of iterations deemed sufficient for accuracy alleviated the divergence issue and resulted in a more efficient kernel on both CPU and GPU.

4) *External libraries*: Alongside this acceleration effort, DALES relies on external libraries that already features GPU offloading.

Radiative transfer calculations are performed with the RTE-RRTMGP libraries [16]. The library is made of two parts, both implemented in Fortran and featuring GPU porting through OpenACC directives. The first part (RRTMGP) computes the radiative properties of molecules and clouds from the input state of the atmosphere by interpolating tabulated data. The second part (RTE) solves the radiation problem and output radiative fluxes across the spatial domain. Each part has to be performed independently for each vertical column and each spectral wavelength, which makes GPU acceleration efficient. However, it is in practice not possible to run all column calculations simultaneously because of memory constraints. The total number of columns is thus divided into several batches. It is important to maximize the number of columns per batch (or minimize the number of batch) as multiple calls to RRTMGP and RTE libraries induce extra computational costs associated with kernel launch overhead.

In order to perform the 3D FFT required by the Poisson solver on the GPU, we rely on the cuFFT library. Fourier transforms are performed one direction of space after another. Because Fourier transforms are non local, it needs to access all the field data along the direction performed, across the domain partition. This requires reorganizing the 3D field memory layout by performing a transposition, which is communication intensive with all-to-all communications. The same applies for the inverse Fourier transform, once the Poisson equation for the Fourier modes is solved. Finally, we should note that cuFFT does not support FFTW's "half-complex" numbering format used in DALES for in-place real-to-complex transforms of size N . To circumvent this, we adopted the approach used in the CaNS [4] and employed a minimal pre-/post-processing step of cuFFT's input/output, along with a re-ordering of the



Fig. 1. 3D visualization of the clouds in one Cloud Botany simulation. Clouds are shown in white and rain is shown in gray. The air temperature near the surface is shown in blue, dark blue areas being cold pools associated with rain.

modified wavenumbers.

III. SIMULATION EXPERIMENTS

A. Reference test case

As a test case for the GPU porting project we chose one simulation from the *Cloud Botany* ensemble of simulations. *Cloud Botany* [12] is a set of 103 DALES simulations of shallow cumulus clouds over the subtropical ocean, exploring how the clouds and their mesoscale organization behave under different physical conditions, for example changing sea surface temperature or wind speed. The simulated domains were 150 km wide in both horizontal directions, large enough to permit organization on a larger scale (called mesoscale in this context, for scales between single clouds and the scale of whole weather systems). One objective of the ensemble was to evaluate how mesoscale cloud organization affects the cloud-climate feedback [11], i.e. to assess whether these clouds in a warmer climate will reflect more or less sunlight back into space. Figure 1 shows a typical 3D visualization of clouds and rain that can be extracted from the *Cloud Botany* simulations.

This reference test case is useful to accelerate with GPUs, because: (i) it utilizes the core of the model and the optional physical models for solar radiative transfer and rain; (ii) it is a case where it is scientifically interesting to make the simulation domain as large as computationally feasible: a larger domain gives more space for mesoscale organization.

In the following simulation experiments, the grid size in the horizontal directions is fixed to 100 m but the number of points is changed between cases in order to address different problem sizes (in the range 50-100 km). However, the vertical domain size and spatial discretisation remain the same for all tests, namely a height of 7 km discretised using 175 points. As a first step, the model is integrated in time for about 24 to 36 h (model time) in order to reach a statistically steady state. All simulations will be run from this state for a model integration time of 10 min to make representative performance assessments. We made sure this statistically steady state falls at day time so that it features shortwave radiation.

B. Verification

To verify the implementation of the OpenACC directives we checked correctness of the model output between CPU

and GPU implementations. Due to the chaotic nature of the system, round-off errors quickly lead to different simulation trajectories. We will thus base the comparison on space and time-averaged statistics, which should remain close. Starting from the same statistically steady state, the simulation is pursued using either a single CPU (AMD Rome 7H12) or a single GPU (NVIDIA A100) for 30 mn model time and a spatial resolution of about 500×500 points in the horizontal directions, accumulating statistics every minutes and performing planar-averages in the horizontal. Figure 2 compares CPU and GPU output for various quantities of the model, namely the temperature, the resolved turbulent kinetic energy representative of the turbulence intensity, the liquid water specific humidity showing where the clouds are, the cloud fraction which is fraction of the area with clouds and the net flux for both longwave and shortwave radiation. Comparison shows excellent agreement for both first order and second order statistics (turbulent kinetic energy), with a maximum relative difference of $\simeq 3\%$ observed locally on the precipitation mean droplet diameter.

IV. PERFORMANCE

A. Single node

1) *Hardware and set up:* In order to assess the performance of the GPU acceleration for different GPU hardware, we compare model run times obtained on a node-basis on the Dutch national supercomputer Snellius [19]. We namely consider the following computing nodes:

- a CPU node consisting of 192 AMD Genoa 9654
- a GPU node consisting of 4 NVIDIA A100
- a GPU node consisting of 4 NVIDIA H100

The number of spatial points in the horizontal directions (number of grid columns) is adapted as indicated in Table I in order to accommodate with different MPI partition of the spatial domain and to ensure that the GPU memory is filled.

The performance of DALES on the Botany case are measured during a short simulation lasting 10 min of model time and restarting after the day-long initialization run. Only the time spent in the time stepping loop is included in the measurements and the experiments are repeated 5 times. Because the number of time steps and the number of computational cells differ between the runs (due to the chaotic nature of the flow and the scaling of the computational domain), we present timing data in units of μ s per time step per million of cells.

2) *Timings:* Table II shows the averaged performance data for the CPU-only, the A100 and the H100 runs for each component of the DALES solver. For each hardware, the execution time and the fraction of the total time is presented, and the speed up compared to the CPU-only run is also added for the GPU-accelerated cases. The overall speed-up of DALES on a node basis is around 4 on A100 and 11.5 on H100. The fractions of the time spent in each component of the code remaining close to their CPU value with one exception: on A100, the performance of the Poisson solver are below average with a speedup close to 2. Further testing

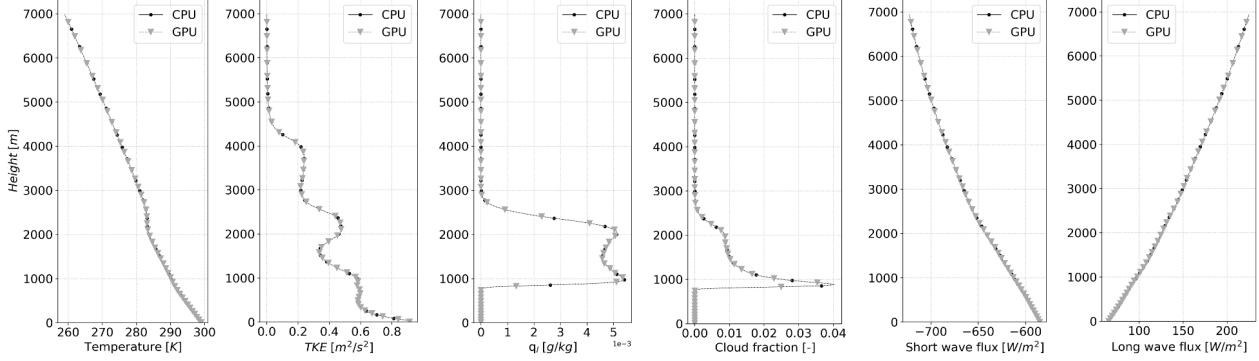


Fig. 2. Vertical profiles of time and planar-averaged fields: temperature, resolved turbulent kinetic energy (TKE), liquid water specific humidity q_l , cloud fraction, shortwave and longwave net radiative fluxes (counted positively from Earth to space). Plot for both GPU and CPU implementation.

Hardware	MPI tasks	nb. of grid columns per task	total nb. of grid columns
CPU AMD Genoa (Snellius)	192	63×84	1.016×10^6
GPU NVIDIA A100 (Snellius)	4	512×512	1.049×10^6
GPU NVIDIA H100 (Snellius)	4	512×512	1.049×10^6

TABLE I

SINGLE-NODE COMPARISONS. CONSIDERED HARDWARE WITH ASSOCIATED NUMBER OF MPI TASKS, NUMBER OF GRID COLUMNS PER TASK AND TOTAL NUMBER OF GRID COLUMNS.

showed that the cuFFT solver available with NVHPC 24.5 (CUDA 12.1.1) on Snellius is less efficient than that available with NVHPC 22.7 (CUDA 11.7.0). The later software stack brings the Poisson solve speedup on par with the remainder of the code, but performance data obtained with the former stack were kept in Table II to maintain a consistent software stack across both NVIDIA GPUs. Both stencil-based functions (advection, subgrid) show similar trends with above averaged speedups while thermodynamics exhibits below averaged speedup, driven by an increased need for synchronisation and horizontal reductions. The significantly higher speed-up obtained on GPUs for the surface component mainly results from the lower number of MPI ranks compared to the CPU case, since the surface computation include reductions in order to obtain integrated ground friction and other quantities of interests.

Timings of the components of the Poisson solver are presented in Table III. It can be seen that most of the time is spent in the forward and backward FFT steps which, as mentioned before, require MPI all-to-all communication operations. The forward and backward FFT steps only experience a small speedup when comparing timings on the A100 GPU node to the CPU node. Speedups on the FFT steps comparable to those observed on H100 GPU node were attained when using the NVHPC 22.7 (CUDA 11.7.0) distribution.

Radiation calculation is another computationally expensive step in the time loop (about 12 % of CPU timings). As explained in Sec. II-C4, these calculations are divided into two steps for both longwave and shortwave spectral regions: the computation of radiative properties (RRTMGP) and the radiative transfer equation (RTE). Table IV gives additional timings for each of these substeps. Total speed-up is about

9 and 14 for A100 GPUs and H100 GPUs, respectively. The speed-up is roughly the same for each of the individual sub-step, although RRTMGP LW and RTE SW are proportionately a little faster. This homogeneous speed-up is expected since radiation calculations are independent for each vertical column and each spectral band.

Finally, note that while the size of the DALES test case has been kept constant on all hardware, more GPU memory is available on H100 and larger test cases could be run on these hardware, further increasing the speedup using the performance metric listed in Table II.

B. Weak scaling

To evaluate the ability of DALES to scale with the needs of researchers, the A100 case is scaled from 1 to 64 GPUs (0.25 to 16 compute nodes), leveraging the periodicity of the test case to duplicate the initial data in the horizontal directions in order to keep the workload per GPU constant. Figure 3 shows the results of the overall DALES weak scaling efficiency (t_{1GPU}/t_{NGPU}) as well as the efficiency of individual components. DALES parallel efficiency is found to drop quickly, mostly driven by the poor scaling performance of the Poisson solve, whereas the other components maintain an efficiency close to one as very little to no communications are needed. Improvement of DALES scaling performance, in particular exploring alternate projection approaches or improving the communication pattern of FFT-based solvers [17], is left for future work.

It is worth mentioning here that strong scaling is disregarded in this study, since the objective of acceleration is to address larger spatial domains.

Timer	CPU		A100GPU			H100GPU		
	Timing	Fraction	Timing	Fraction	speed-up	Timing	Fraction	speed-up
Thermodynamics	1178.9	12.1	221.8	9.1	5.32	120.7	14.3	9.77
Poisson	2729.2	28.1	1442.0	59.0	1.89	220.5	26.1	12.38
Subgrid	1002.3	10.3	131.6	5.4	7.62	89.9	10.6	11.15
Radiation	1172.3	12.1	132.8	5.4	8.83	86.1	10.2	13.61
Microphysics	607.2	6.3	97.0	4.0	6.26	73.5	8.7	8.26
Advection	922.1	9.5	106.7	4.4	8.64	74.3	8.8	12.41
Time stepping	733.2	7.6	56.2	2.3	13.05	37.6	4.4	19.5
ExtForces	497.9	5.1	61.9	2.5	8.04	45.7	5.4	10.89
Halo-Exchange	501.0	5.2	112.0	4.6	4.47	42.7	5.0	11.73
Surface	246.1	2.5	3.2	0.1	76.91	4.3	0.5	57.23
Checks	44.5	0.5	5.9	0.2	7.54	4.6	0.5	9.67
Timestep loop	9711.7	100.0	2444.2	100.0	3.97	846.2	100.0	11.48

TABLE II
SINGLE NODE CPU AND GPU TIMINGS IN μ S/STEP/MCELLS.

Timer	CPU		A100GPU			H100GPU		
	Timing	Fraction	Timing	Fraction	Speedup	Timing	Fraction	Speedup
Fill RHS	516.6	18.9	42.3	2.9	12.21	23.3	10.5	22.17
Forward FFT	946.7	34.7	603.3	41.8	1.57	86.9	39.3	10.9
Tridiagonal solve	110.5	4.0	10.0	0.7	11.05	6.2	2.8	17.82
Backward FFT	1025.5	37.6	659.4	45.7	1.55	89.5	40.5	11.46
Apply correction	129.9	4.8	126.6	8.8	1.03	15.1	6.8	8.60
Total	2729.2	100	1441.6	100	1.20	220.9	100	7.8

TABLE III
SINGLE NODE CPU AND GPU POISSON SOLVER TIMINGS IN μ S/STEP/MCELLS.

Timer	CPU		A100GPU			H100GPU		
	Timing	Fraction	Timing	Fraction	speed-up	Timing	Fraction	speed-up
RRTMGP LW	332.2	28.3	35.3	26.6	9.41	22.2	25.8	14.96
RRTMGP SW	277.0	23.6	34.0	25.6	8.15	21.4	24.9	12.94
RTE LW	166.7	14.2	19.6	14.8	8.51	13.4	15.6	12.44
RTE SW	345.6	29.5	37.7	28.4	9.17	25.0	29.0	13.82
Radiation total	1172.3	100.0	132.8	100.0	8.83	86.1	100.0	13.62

TABLE IV
SINGLE NODE CPU AND GPU RADIATION TIMINGS IN μ S/STEP/MCELLS.

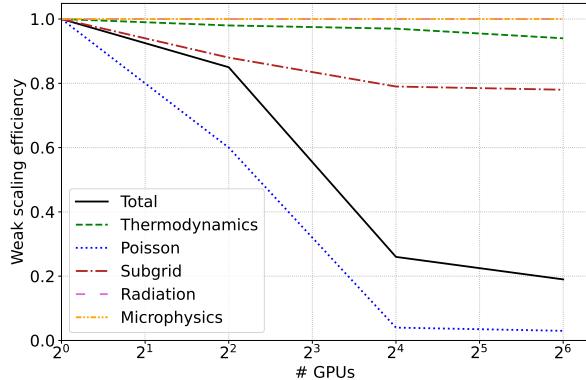


Fig. 3. Weak scaling of DALES on the botany case using Snellius A100 GPUs

V. OPTIMIZATION

The acceleration strategy presented in Sec. II-C enables to quickly accelerate the entire code base, giving entire freedom to the compiler in terms of kernel launch parameters and ensuring relatively good portability. However, a handful of kernels exhibit low occupancy (<50 %) and require more

careful consideration. Two strategies are employed in order to increase the performance of those kernels: 1) manually modifying the kernels to improve memory access, cache usage and register pressure; 2) using Kernel Tuner [21] (KT) to auto-tune the kernel launch parameters.

A. Presentation of Kernel Tuner

Kernel Tuner (KT) [21] is a generic auto-tuning framework for GPU applications. It is written in Python and supports the tuning of computational kernels written in CUDA, HIP, and OpenCL, but was also recently extended to handle directive-based languages like OpenACC and OpenMP.

To do so, KT provides a set of helper functions that allows users to extract the tunable kernels from the rest of the code, together with all the necessary C preprocessor macros, and takes care of the allocation and initialization of the used data structures.

The selection of tunable kernels and identification of the data structures is guided by the user, by adding some ad-hoc KT pragmas to the source code; these pragmas are the only changes to the original code needed for tuning.

Kernel Tuner was used to find the optimal configuration of launch parameters for the GPUs used in the experiments. All the KT data reported hereafter are averaged over 20 calls to

Listing 2. Momentum diffusion operator kernel where km and e are respectively the eddy viscosity and the subgrid TKE

```

!$acc parallel loop num_gangs(ngang) vector_length(vlength) collapse(cfactor)
do k = 1, kmax
  do j = 1, jmax
    do i = 1, imax
      e_t(i,j,k) = (((km(i+1,j,k)+km(i,j,k))*(e(i+1,j,k)-e(i,j,k)) &
                     -(km(i,j,k)+km(i-1,j,k))*(e(i,j,k)-e(i-1,j,k))) &
                     * nuf(k)/dx2 &
                     + ((km(i,j+1,k)+km(i,j,k))*(e(i,j+1,k)-e(i,j,k)) &
                     -(km(i,j,k)+km(i,j-1,k))*(e(i,j,k)-e(i,j-1,k))) &
                     * nuf(k)/dy2 + &
                     + ( rhoh(k+1)/rhof(k) * (dzf(k+1)*km(i,j,k)+dzf(k)*km(i,j,k+1) &
                     * (e(i,j,k+1)-e(i,j,k))) / dzh(k+1)**2 &
                     -rhoh(k)/rhof(k) * (dzf(k-1)*km(i,j,k)+dzf(k)*km(i,j,k-1) &
                     * (e(i,j,k)-e(i,j,k-1))) / dzh(k)**2 &
                     ) &
                     )
    enddo
  enddo
enddo

```

the targeted kernel with a given set of parameters in order to mitigate runtime variability.

B. Targeted kernels

The compute kernels targeted for optimization are the stencil-based kernels in the advection and subgrid scale diffusion operator. Combined, these kernels take up to 20% of the total compute time (see Table II). Note that tuning of launch parameters in typical thermodynamics and microphysics kernels was also attempted without obtaining any gain compared to the compiler-selected values.

The subgrid diffusion operator is close to a weighted 3D 7-points stencil, which is a benchmark for a number of stencil kernel domain specific language [23], [24]. These studies showed that the 3D 7-points stencil is difficult to optimize because only close neighbors (+/-1 in each direction) are involved, such that data locality is not as critical as higher order kernels. Additionally, these kernels are memory bound and thus less sensitive to kernel launch parameters. For instance the subgrid turbulent kinetic energy (TKE) production through strain (second term on the right-hand-side of Eq. 12 in [10]) kernel is given in Listing 2.

In this kernel, the stencil in the z direction differs from the horizontal ones due to the staggered nature of the grid and the non-uniform grid spacing. A variant of the kernel in Listing 2 for computing subgrid scalar diffusion adds a fourth loop on the number of scalars. This fourth loop can be placed around the ijk do loops, further extending the index space when using collapse, or nested as an inner loop and treated sequentially using the loop seq clause such that each GPU thread performs the computation for all the scalars. The advection kernels (not listed here for the sake of concision) features similar access pattern as those presented in

Listing 3. Example of manual loop tiling

```

!$acc parallel loop num_gangs(ngang) &
!$acc& vector_length(vlength) &
!$acc& collapse(cfactor)
do k = 1, kmax
  do bj = 0, nbj
    do bi = 0, nbi
      !$acc loop seq collapse(2)
      do j = 1, j_b
        do i = 1, i_b
          ...
        enddo
      enddo
    enddo
  enddo
enddo

```

Listing 2, but the higher order flux reconstruction (6^{th} -order) needs a wider stencil, reaching $+/-3$ in each direction.

An alternative approach to dealing with stencil operation consists in replacing the collapse clause with a tile(tx,ty,tz), directing the compiler to split each of the nested loop into a outer tile loop and an inner element loop, in hope that it might result in better cache reuse when already present neighboring cell data is accessed again. The same objective can also be targeted by manually tiling the inner ij loops with $nbi-nbj$ blocks of size i_b-j_b as depicted in Listing 3.

Here, a loop seq clause is used on the inner ij element loops such that each GPU thread performs the computation for all the elements in the tile.

In summary, the kernel launch parameters targeted by KT are:

- `num_gang` maps to the length of the thread block grid in the x direction (`gridDim.x` with a CUDA backend)
- `vector_length` maps to the block size in the x direction (`blockDim.x` with a CUDA backend)
- `collapse` indicates how many of the tightly nested loop index spaces are flatten to a single one
- `tx,ty,tz` the tile size in the three spatial dimension employed by OpenACC

C. Experiments on selected kernels

In a first step, the scalar subgrid diffusion kernel `diffcsv` is manually tweaked in order to improve performance. The following changes are implemented: 1) using the `cache` directive to load anisotropic grid spacing data in software-managed caches instead of multiple read into 1D array, 2) kernel fission of the subgrid kinetic energy sources to reduce register pressure (from 108 down to 62 + 56), 3) replace `ijkn` collapsed nested loop with a `ijk` collapsed one and a `loop seq` on the n component. Figure 4 shows the results. Using a combination of 1) and 3), the overall performance of the subgrid module can be increased by close to 5%. Reducing register pressure by splitting the long kernel into two leads to lower performance as the resulting kernels are effectively run sequentially since each can still occupy the entire GPU.

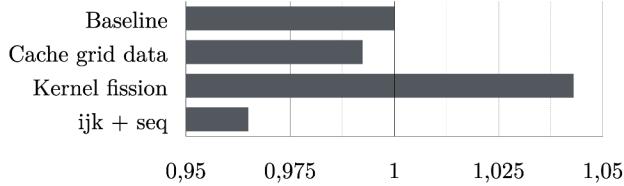


Fig. 4. Run times of the subgrid module with three optimisation strategies (using the `cache` directive, using kernel fission and using `ijk` collapsed loop and n sequential loop), relative to the baseline acceleration strategy.

We then use KT on two specific subgrid kernels: `diffcsv` (which is similar to Listing 2 but including an outer loop on scalar and 4-dimensional Fortran arrays) and `sources`, their contribution accounting for 30% of the subgrid module. The first kernel is of interest as it evaluates the diffusion term for transported scalars in DALES, the number of which can vary from 2 (as in the Botany case) to a few dozens when chemical species are considered; the second kernel exhibit significant reuse of the neighboring data to evaluate cross-derivatives and is a good candidate for improved performance using tiling. All the tests are conducted on a grid size similar to that of the botany case on a single GPU, and using both A100 and H100 GPUs in order to evaluate the portability of the tuned kernels.

Figure 5 shows the distribution of the `diffcsv` kernel timings for an increasing number of transported scalars and kernel launch parameters in the following range: `vector_length` $\in [16 : 512]$, `num_gangs` $\in [256 : 65535]$, `cfactor` $\in [2 : 4]$ and with or without the `loop seq` optimization

shown in Fig. 4 (a total of 256 parameter sets per subplot). Each subplot is a classical histogram plot, oriented vertically such that the bar length (abscissa) is the number of parameter sets that leads to a timing within a given time bin (ordinate). The left panel corresponds to A100 timings, while the mirrored right one is H100 timings. Additionally the timing with the default (compiler-selected) parameters are indicated with a dashed red line. Optimizing the kernel launch parameters only resulted in significant performance improvements on A100, with a maximum gain close to 35% in the $n = 10$ case. On H100, only small improvements were obtained, up to 10%. Using `loop seq` always led to better performance, especially for a small number of scalars. Interestingly, the timing distribution on H100 are not simply a translation of the A100 one, and the optimum parameter set differs slightly between the two GPUs: larger `vector_length` and `num_gangs` led to optimum on H100 compared to A100. However, the parameters leading to optimum performance on H100 gave close to optimum performance on A100 (within 5%), and vice-versa.

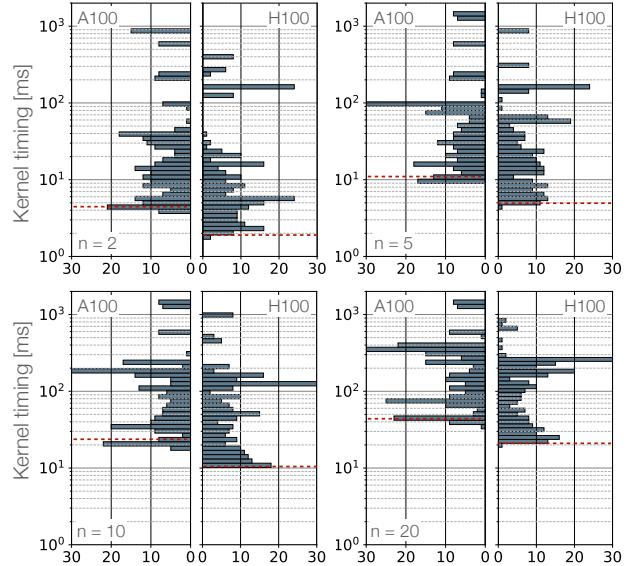


Fig. 5. Measured kernel timing distribution for the `diffcsv` kernel, varying the number of scalar n . On each subplot, the left and right panel corresponds to A100 and H100 GPUs respectively. Dashed red lines indicate default parameters timing.

On the second kernel, we could not find launch parameters leading to better performance than the defaults with KT when using only the `collapse` clause. Instead, we employed 1) the `tile(tx,ty,tz)` clause discussed in Section V-B and 2) the manual tiling of Listing 3. Figure 6 shows the kernel timing for optimization 1) and 2) on the left and right, respectively. KT allows to automatically test hundreds of combinations of `(tx,ty,tz)` and i_b, j_b values, for values of `vector_length` and `num_gangs` in the same range as for Fig. 5. Using OpenACC `tile` led to up to 15% performance gain on A100 and H100 compared to the

baseline `collapse(3)` clause with default vector length and number of gangs. On both GPUs, the best performance are obtained using $tx = 64$, $ty = 4$ and $tz = 2$, and only differs on the value of `vector_length`. Once again, the optimum values of `vector_length` obtained on A100 led to performance on H100 close to the optimum, and the reciprocal. In contrast, manually introducing tiles led to a small decrease in performance at best and much worse performance in most cases.

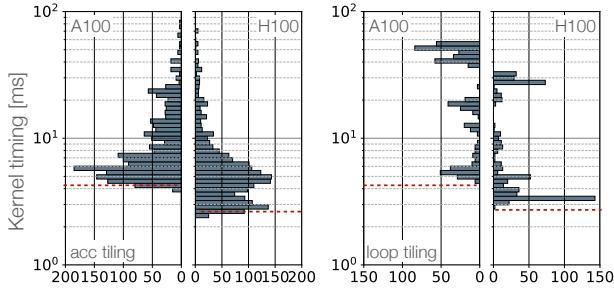


Fig. 6. Measured kernel timing distribution for the sources kernel, using OpenACC tiling clause (left) and introducing *ij* tiles (right). On each subplot, the left and right panel corresponds to A100 and H100 GPUs respectively. Dashed red lines indicate default parameters timing.

The `tile(tx,ty,tz)` clause is then employed on another stencil-based kernels, the 6th-order, wider stencil, advection kernel. Additionally, we evaluate if the problem size can affect the potential performance gain by dividing the initial *ij* loop span (512 x 512) by 2 and 4 in each direction. The results obtained on the advection kernel are qualitatively similar with those obtained on the subgrid `source` kernel: up to 15% performance gain on A100 and only marginal gain on H100 GPUs. To illustrate the effect of the tile size on performance, Fig. 7 shows the kernel timing for the (512 x 512) loop span as a function of the tile size tx which was found to correlate best with performance. For each value of tx , a small random perturbation was added to visualize the timing dispersion due to the other tuning parameters. In particular, data are colored by the number of gangs. Large values of tx significantly improve the kernel performance, while a large number of gangs is beneficial at small values of tx . However, a smaller number of gangs leads to better performance when large tx are used. The KT timing results for smaller loops span are qualitatively similar to Fig. 7, but no performance gain could be obtained compared to the default kernel launch parameters.

Finally, we implemented the set of parameters that led to performance improvements throughout the code base in order to evaluate an overall gain from the optimization with KT. More specifically, we introduced a `collapse(3)` with a `loop seq` in the scalar space in place of all the `collapse(4)` for the *ijkn* loops, and we replaced `collapse(3)` with a `tile(tx,ty,tz)` clause for all the stencil-based kernels, keeping the A100 best launch parameters. The resulting overall speed-up of the Botany case listed in Table II was 1.7% on A100 and 2.6% on H100. A closer

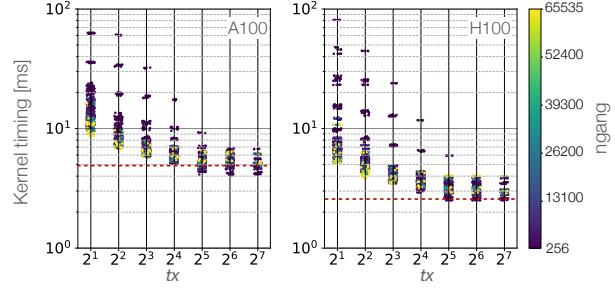


Fig. 7. Advection kernel timing for the (512 x 512) *ij* loops span, as function of the tile size in *x* direction, colored by the number of gangs used in the launch parameters, left: A100, right: H100. Dashed red lines indicate default parameters timing.

look at specific kernel timings shows that propagating the kernel launch parameters optimized with KT on the handful of kernels reported here to all of the stencil-based kernels has a mixed outcome. Several kernels are effectively faster, close to the performance gains observed with KT, but the performance of some other kernels (also stencil-based kernels, but with different access pattern and intermediate computations) is degraded. This last point further highlights the difficulty to perform optimization at the application level and point towards an avenue of improvement for auto-tuning tools such as KT.

VI. CONCLUSION

This paper presented the GPU porting of a large-eddy-simulation atmospheric model (DALES), which is a multi-physics Fortran code designed to run on a large spatial domain with spatial resolution typically of the order of 5-100 m and for a long integration time of the order of several days. The use of OpenACC directives facilitated the porting, mainly consisting of OpenACC collapsed loops over the three dimensions of space and fusing of as many successive, independent kernels as possible. The porting also benefited from GPU-aware libraries for radiation calculations (RTE-RRTMGP) and Fourier transforms (cuFFT). Still, refactoring some parts of the code was necessary, especially in the microphysics and thermodynamics sections, where differences in the local state introduced significant thread divergence.

Acceleration performance was assessed from a test case featuring cloud transport and radiative transfer. Run times between CPU and GPU implementations were compared on a single-node basis. A speed-up of 4 and 12 has been obtained for nodes of 4 NVIDIA A100 and 4 NVIDIA H100 respectively, compared to a node of 192 AMD CPUs. The most computationally intensive sections of the code were found to be the thermodynamics, the subgrid diffusion, the radiative transfer and the Poisson solver. Overall, the fractions of the time spent in each component of the code remained close to their CPU value. However, the Poisson solver was found to perform poorly on the A100 because of a loss of efficiency of the cuFFT library in the associated software stack. Thermodynamics routines also exhibited below-average

speed-up because of required synchronisations and horizontal reductions.

The weak scaling analysis showed that the acceleration efficiency drops quickly beyond a single node. The Poisson solver, based on Fourier transforms, is responsible for this loss of efficiency, as performing FFT across the whole spatial domain requires all-to-all communications and memory rearrangements, which do not scale. We might consider implementing alternative Poisson solvers that could better scale with many GPUs. Having said that, we stress that the single-node acceleration remains highly valuable for research, as the simulations must be performed for a large integration time (up to several hundred hours) to accumulate statistics.

A noteworthy aspect of this work is the use of Kernel Tuner, an auto tuning application, to optimize the GPU execution of stencil-based kernels associated with subgrid diffusion and advection processes. For the subgrid diffusion kernel, significant performance improvements were obtained on A100 GPUs, with a maximum gain close to 35 %, while limited performance improvements were obtained on H100 GPUs, with a maximum gain close to 10 %. We noticed that the optimal parameters differ slightly between the two GPUs, even though using the optimal parameters of one GPU on the other one resulted in near optimal performance. For subgrid diffusion source and 6th order advection kernels, we attempted to replace collapsing clauses with tile clauses and it only resulted in better performance on the A100 GPUs, around 15 %.

Finally, we examined how the tuned parameters can be transferred to similar kernels (i.e. stencil based kernels) across the entire DALES code base. The overall DALES performances gain only showed small improvements compared to the initial port, and fine grain profiling revealed the mitigated effects of the tuned parameters. These observations deserve further investigation and underline the challenge associated with application level performance tuning.

Ongoing efforts in DALES focus on evaluating and improving the performances of the present OpenACC port on AMD GPU hardware, as well as investigating alternate strategies for solving the Poisson linear system in order to improve DALES scaling. In future steps, we would also like to investigate the effect of mixed precision on acceleration performance and code accuracy. DALES already allows switching some quantities to single-precision, which will result in the ability to address larger resolutions on the GPU.

VII. ACKNOWLEDGEMENTS

The authors acknowledge funding by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement No 101093054.

The authors acknowledge the support of the Ruisdael Observatory, scientific research infrastructure which is (partly) financed by the Dutch Research Council (NWO, grant no. 184.034.015)

This work used the Dutch national e-infrastructure with the support of the SURF Cooperative using grants no. vus21036 and EINF-10608.

Caspar Jungbacher acknowledges support by the Refreeze the Arctic Foundation (RAF) <https://refreezethearticfoundation.com/about/>.

REFERENCES

- [1] TOP500 List, November 2024. <https://top500.org/lists/top500/list/2024/11/>.
- [2] Steven J. Böing, Harm J. J. Jonker, A. Pier Siebesma, and Wojciech W. Grabowski. Influence of the subcloud layer on the development of a deep convective ensemble. *Journal of the Atmospheric Sciences*, 69(9):2682 – 2698, 2012.
- [3] Pedro Costa. A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. *Computers Mathematics with Applications*, 76(8):1853–1862, 2018.
- [4] Pedro Costa, Everett Phillips, Luca Brandt, and Massimiliano Fatica. GPU acceleration of CaNS for massively-parallel direct numerical simulations of canonical fluid flows. *Computers & Mathematics with Applications*, 81:502–511, January 2021.
- [5] Stephan R de Roode, Harm JJ Jonker, Bas JH van de Wiel, Victor Vertregt, and Vincent Perrin. A diagnosis of excessive mixing in smagorinsky subfilter-scale turbulent kinetic energy models. *Journal of the atmospheric sciences*, 74(5):1495–1511, 2017.
- [6] J. W. Deardorff. Stratocumulus-capped mixed layers derived from a three dimensional model. *Boundary-Layer Meteorology*, 18:495–527, 1980.
- [7] Ariana Frassoni, Dayana Castilho, Michel Rixen, Enver Ramirez, João Gerd Z De Mattos, Paulo Kubota, Alan James Peixoto Calheiros, Kevin A Reed, Maria Assunção F da Silva Dias, Pedro L da Silva Dias, et al. Building the next generation of climate modelers: Scale-aware physics parameterization and the “grey zone” challenge. *Bulletin of the American Meteorological Society*, 99(11):ES185–ES189, 2018.
- [8] Marco A Giorgetta, William Sawyer, Xavier Lapillonne, Panagiotis Adamidis, Dmitry Alexeev, Valentin Clément, Remo Dietlicher, Jan Frederik Engels, Monika Esch, Henning Franke, et al. The ICON-A model for direct QBO simulations on GPUs (version iconcscs: baf28a514). *Geoscientific Model Development*, 15(18):6985–7016, 2022.
- [9] Mark W Govett, Jacques Middlecoff, and Tom Henderson. Running the nim next-generation weather model on GPUs. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 792–796. IEEE, 2010.
- [10] T. Heus, C. C. van Heerwaarden, H. J. J. Jonker, A. Pier Siebesma, S. Axelsen, K. van den Dries, O. Geoffroy, A. F. Moene, D. Pino, S. R. de Roode, and J. Vilà-Guerau de Arellano. Formulation of the dutch atmospheric large-eddy simulation (dales) and overview of its applications. *Geoscientific Model Development*, 3(2):415–444, 2010.
- [11] Martin Janssens, Fredrik Jansson, Pouriya Alinaghi, Franziska Glassmeier, and A. Pier Siebesma. Symmetry in mesoscale circulations explains weak impact of trade cumulus self-organisation on the radiation budget in large-eddy simulations. *preprint*, 2024.
- [12] Fredrik Jansson, Martin Janssens, Johanna H. Grönqvist, A. Pier Siebesma, Franziska Glassmeier, Jisk Attema, Victor Azizi, Masaki Satoh, Yousuke Sato, Hauke Schulz, and Tobias Kölling. Cloud botany: Shallow cumulus clouds in an ensemble of idealized large-domain large-eddy simulations of the trades. *Journal of Advances in Modeling Earth Systems*, 15(11):e2023MS003796, 2023. e2023MS003796 2023MS003796.
- [13] Xavier Lapillonne and Oliver Fuhrer. Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science. *Parallel Processing Letters*, 24(01):1450003, 2014.
- [14] John Michalakes and Manish Vachharajani. GPU acceleration of numerical weather prediction. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–7, 2008.
- [15] Matthew Norman, Jeffrey Larkin, Aaron Vose, and Katherine Evans. A case study of cuda fortran and openacc for an atmospheric climate kernel. *Journal of computational science*, 9:1–6, 2015.
- [16] Robert Pincus, Eli J. Mlawer, and Jennifer S. Delamere. Balancing accuracy, efficiency, and flexibility in radiation calculations for dynamical models. *Journal of Advances in Modeling Earth Systems*, 11(10):3074–3089, 2019.

- [17] Joshua Romero, Pedro Costa, and Massimiliano Fatica. Distributed-memory simulations of turbulent flows on modern GPU systems using an adaptive pencil decomposition library. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, pages 1–11, 2022.
- [18] Jérôme Schalkwijk, Eric J Griffith, Frits H Post, and Harm JJ Jonker. High-performance simulations of turbulent clouds on a desktop PC: Exploiting the GPU. *Bulletin of the American Meteorological Society*, 93(3):307–314, 2012.
- [19] SURF. Snellius hardware description, <https://servicedesk.surf.nl/wiki/display/WIKI/Snellius+hardware>, 2024.
- [20] Chiel C Van Heerwaarden, Bart JH Van Stratum, Thijs Heus, Jeremy A Gibbs, Evgeni Fedorovich, and Juan Pedro Mellado. Microrh 1.0: A computational fluid dynamics code for direct numerical simulation and large-eddy simulation of atmospheric boundary layer flows. *Geoscientific Model Development*, 10(8):3145–3165, 2017.
- [21] Ben van Werkhoven. Kernel tuner: A search-optimizing gpu code auto-tuner. *Future Generation Computer Systems*, 90:347–358, 2019.
- [22] Yidong Xia, Jialin Lou, Hong Luo, Jack Edwards, and Frank Mueller. OpenACC acceleration of an unstructured CFD solver based on a reconstructed discontinuous Galerkin method for compressible flows. *International Journal for Numerical Methods in Fluids*, 78(3):123–139, 2015.
- [23] Charles Yount, Josh Tobin, Alexander Breuer, and Alejandro Duran. Yask—yet another stencil kernel: A framework for hpc stencil code-generation and tuning. In *2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*, pages 30–39, 2016.
- [24] Tuowen Zhao, Samuel Williams, Mary Hall, and Hans Johansen. Delivering performance-portable stencil computations on CPUs and GPUs using bricks. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 59–70. IEEE, 2018.