The ICON-A model for direct QBO simulations on GPUs (version icon-cscs:baf28a514)

Marco A. Giorgetta[1], William Sawyer[2], Xavier Lapillonne[3], Panagiotis Adamidis[4], Dmitry Alexeev[5], Valentin Clément[6], Remo Dietlicher[3], Jan Frederik Engels[4], Monika Esch[1], Henning Franke[1, 12], Claudia Frauen[4], Walter M. Hannah[7], Benjamin R. Hillman[8], Luis Kornblueh[1], Philippe Marti[6], Matthew R. Norman[9], Robert Pincus[10], Sebastian Rast[1], Daniel Reinert[11], Reiner Schnur[1], Uwe Schulzweida[1], and Bjorn Stevens[1]

[1]Max-Planck-Institut für Meteorologie, Hamburg, Germany
[2]Centro Svizzero di Calcolo Scientifico, Lugano, Switzerland
[3]Bundesamt für Meteorologie und Klimatologie MeteoSchweiz, Zürich-Flughafen, Switzerland
[4]Deutsches Klimarechenzentrum, Hamburg, Germany
[5]NVIDIA, Zürich, Switzerland
[6]Center for Climate Systems Modeling, ETH, Zürich, Switzerland
[7]Lawrence Livermore National Laboratory, Livermore, USA
[8]Sandia National Laboratories, Albuquerque, USA
[9]Oak Ridge National Laboratory, Oak Ridge, USA
[10]Lamont-Doherty Earth Observatory, Columbia University, Palisades, USA
[11]Deutscher Wetterdienst, Offenbach, Germany
[12]International Max Planck Research School on Earth System Modelling, Hamburg, Germany

Correspondence: Marco Giorgetta (marco.giorgetta@mpimet.mpg.de)

Abstract. Classical numerical models for the global atmosphere, as used for numerical weather forecasting or climate research, have been developed for conventional central processing unit (CPU) architectures. This ~~now~~ hinders the employment of such models on current top performing supercomputers, which achieve their computing power with hybrid architectures, mostly using graphics processing units (GPUs). Thus also scientific applications of such models are restricted to the lesser computer power of CPUs. Here we present the development of a GPU enabled version of the ICON atmosphere model (ICON-A), motivated by a research project on the quasi-biennial oscillation (QBO), a global scale wind oscillation in the equatorial stratosphere that depends on a broad spectrum of atmospheric waves, which origins from tropical deep convection. Resolving the relevant scales, from a few kilometers to the size of the globe, is a formidable computational problem, which can only be realized now on top performing supercomputers. This motivated porting ICON-A, in the specific configuration needed for the research project, in a first step to the GPU architecture of the *Piz Daint* computer at the Swiss National Supercomputing Centre, and in a second step to the *Juwels-Booster* computer at the Forschungszentrum Jülich. On *Piz Daint* the ported code achieves a single node GPU vs. CPU speed-up factor of ~~6.3, and now~~ 6.4, and allows global experiments at a horizontal resolution of 5 km on 1024 computing nodes with 1 GPU per node with a turnover of 48 simulated days per day. On *Juwels-Booster* the more modern hardware in combination with an upgraded code base allows for simulations at the same resolution on 128 computing nodes with 4 GPUs per node and a turnover of 133 simulated days per day. Additionally, the code still remains functional on CPUs as it is demonstrated by additional experiments on the *Levante* compute system at the German Climate

Computing Center. While the application shows good weak scaling over the tested 16-fold increase in grid size and node count, making also higher resolved global simulations possible, the strong scaling on GPUs is relatively ~~weak~~poor, which limits the options to increase turnover with more nodes. Initial experiments demonstrate that the ICON-A model can simulate downward propagating QBO jets, which are driven by wave meanflow interaction.

1 Introduction

Numerical weather prediction (NWP) and climate research make use of numerical models which solve discretized equations for fluid dynamics on the globe. For NWP and many research applications the resolution is chosen as high as possible for the available computing resources. Higher resolution allows to explicitly compute atmospheric processes over a larger range of scales, and thus to compute more faithfully the dynamics of the global atmosphere. Examples of small scale features, which are relevant for NWP or climate research, are cumulus clouds, gravity waves generated by orographic obstacles and convective clouds, or turbulent motions in the boundary layer. The advantages of higher resolution, however, comes at higher costs and especially longer time-to-solution, which practically limits the maximum resolution that can be afforded in specific applications. In climate research most global simulations of the atmospheric circulations still use resolutions of a few tens of kilometers to about 200 km for simulations over decades to centuries. But the most ambitious global simulations now reach already resolutions of ~~ca. 5~~ just a few kilometer (?), which means that basic structures of tropical deep convection can be computed explicitly. Such simulations are still the exception and limited to short time periods, owing to the slow turnover in terms of simulated time per wall clock time unit. A specific reason for the limitation of such simulations in resolution or simulated time is that the computing codes of these numerical models have been developed and optimized for conventional central processing unit (CPU) architectures, while the most advanced and powerful computer systems employ now hybrid architectures with graphics processing units (GPUs). Thus, the most powerful computing systems are effectively out of reach for most existing computing codes for numerical weather prediction and climate research.

This holds also for the ICON model system, which has been developed since the early 2000s for use on either cache based or vectorizing CPUs, with components for atmosphere, land and ocean. The build up of the hybrid *Piz Daint* compute system at the Swiss National Supercomputing Centre, however, created a strong motivation to port the ICON model to GPUs in order to benefit from the immense compute power of *Piz Daint* resulting from up to 5704 GPUs. With this motivation it was decided to port the atmospheric component of ICON (ICON-A) in two specific configurations to GPUs so that the development effort can be limited initially to a subset of the ICON codes. The model configuration in the focus of the presented work was designed for the *Quasi-biennial oscillation in a changing climate (QUBICC)* project, for which global simulations at horizontal resolutions of 5 km or better and vertical resolutions of a few hundred meters up to the middle stratosphere are planned to investigate the dynamics of the quasi-biennial oscillation (QBO), a global scale zonal wind oscillation in the equatorial stratosphere, for which the main characteristics are reviewed in **?**, and an overview of the QBO impacts is given in **?**. Using this very high resolution is essential for the QUBICC project so that the dynamical links from small scale and quickly evolving tropical deep convection to the global scale and slowly varying wind system of the QBO can be directly computed. This makes a substantial difference to existing simulations of the QBO in coarser models, where deep convection and the related gravity wave effects must be parameterized. The uncertainty in the parameterization of convection and gravity waves, as necessary in coarser models, is the main reason for problems in simulating the QBO (**??**).

Until now only a few attempts have been made to port general circulation models for the atmosphere or the ocean to GPUs, using different methods. **?** presented an early attempt, in which only the most costly part of the NICAM model, the ~~two-dimensional~~ horizontal dynamics, was ported to GPUs, for which these parts were re-programmed in CUDA Fortran. **?** ported the COSMO5 limited area model to GPUs by using directives and by re-writing the dynamical core from Fortran to C++ and employing the STELLA domain specific language. Similarly **?** ported their LICOM3 model by rewriting the code in the time loop from Fortran to C and further to HIP. In the case of the NIM weather model, **?**, however, decided to work with directives only so that the same code can be used on CPU, GPU and Many Integrated Core (MIC) processors. Other models have partial GPU implementations, such as WRF, **?**, in CUDA-C and MPAS, **?**, with OpenACC. In our attempt, after initial steps described later, it was decided to stay with the standard ICON Fortran code wherever possible and thus to work with directives, so that ICON-A works on CPUs and GPUs. In the CPU case applications shall continue to use the proven parallelization by MPI domain decomposition mixed with OpenMP multi-threading, while in the GPU case parallelization should now combine the MPI domain decomposition with OpenACC directives for the parallelization on the GPU. OpenACC was chosen because this was the only practical option on the GPU compute systems used in the presented work and described below. Specifically OpenMP version 5 was not available on these systems. Consequently the resulting ICON code presented here includes now OpenMP and OpenACC directives.

In the following we present the model configuration for QUBICC experiments, for which the ICON-A model has been ported to GPUs (Sect. **??**), the relevant characteristics of the compute systems *Piz Daint*, *Juwels-Booster* and *Levante* used in this study (Sect. **??**), the methods used for porting ICON codes to GPUs (Sect. **??**), the validation methods used to detect porting errors (Sect. **??**), the results from benchmarking on the three compute systems (Sect. **??**), selected results from first QUBICC experiments (Sect. **??**), and the conclusions.

2 Model configuration for QUBICC experiments

The QUBICC experiments make use of very high resolution grids, on which dynamics and transport are explicitly solved. This means that only a small number of processes needs to be parameterized in comparison to the low resolution simulations presented by **?**. This reduced physics package, which we call the Sapphire physics, comprises parameterizations for radiation, vertical turbulent diffusion, and cloud microphysics in the atmosphere, and land surface physics, as detailed in section **??**. Thus the model components to be ported to GPU include dynamics, transport, the aforementioned physics parameterizations, and additionally the essential infrastructure components for memory and communication. The following subsections provide more details on the model grids defining the resolution and the components computed on these grids.

2.1 Horizontal grid

The horizontal resolution needs to be high enough to allow the explicit simulation of tropical deep convection, and at the same time simulation costs must be limited to realistic amounts, as every doubling of the horizontal resolution multiplies the computing costs by a factor of 8, resulting from a factor 2 for each horizontal dimension and a factor 2 for the necessary shortening of the time step. From earlier work made with ICON-A it is understood that $\Delta x = 5\mathrm{km}$ is the smallest resolution, for which deep convection is simulated in an acceptable manner (**?**), and for which realistic gravity wave spectra related to the resolved convection can be diagnosed (**??**). As any substantial increase in horizontal resolution is considered to exceed the expected compute time budget, a horizontal mean resolution of $\Delta x = 5\mathrm{km}$ is used, as available on the R2B9 grid of the ICON model, see Table 1 in **?**. The specific ICON grid-id is 0015, referring to a north-south symmetric grid, which results from a 36 degree longitudinal rotation of the southern hemispheric part of the ICON grid after the initial R2 root bisection of the spherical icosahedron. (Older setups as in **?** did not yet use the rotation step for a north-south symmetric grid.)

2.2 Vertical grid

The vertical grid of the ICON-A model is defined by a generalized smooth-level vertical coordinate (**?**) formulated in geometric height above the reference ellipsoid, which is assumed to be a sphere. At the height of 22.5 km the model levels transition to levels of constant height, which are as well levels of constant geopotential. For the QUBICC experiments a vertical grid is chosen that has 191 levels between the surface and the model top at a height of 83km. This vertical extension and resolution is chosen as a compromise between a number of factors:

- A high vertical resolution is needed to represent the dynamics of vertically propagating waves, considering waves which can be resolved horizontally. The resolution should also be sufficient in the shear layers of the QBO, where the Doppler shifting shortens the vertical wave lengths of upward propagating waves with phase speeds similar to the velocity of the meanflow in the shear layer. Typically a vertical resolution of a few hundred meters is wanted.
- The vertical extent of the model should be high enough to allow the simulation of the QBO in the tropical stratosphere without direct numerical impacts from the layers near the model top, where numerical damping is necessary to avoid numerical artifacts. In practice the ICON model uses the upper boundary condition of zero vertical wind, $w = 0$, and applies a Rayleigh damping on the vertical wind (**?**). This damping starts above a given minimum height from where it is applied up to the top of the model, using a tanh vertical scaling function changing from 0 at the minimum height to 1 at the top of the model (**?**). Based on experience the depth of the damped layer should be ca. 30 km. Combining this with the stratopause height of ca. 50 km, a top height of ca. 80 km is needed.
- A further constraint is the physical validity of the model formulation. The key limitation consists in the radiation scheme RTE+RRTMGP, which is developed and validated for conditions of local thermal equilibrium (LTE) between the vibrational levels of the molecules involved in radiative transitions and the surrounding medium. This limits the application of this radiation scheme to levels below atmospheric pressures of 0.01 ~~hPa~~hPa. The atmospheric pressure of 0.01 ~~hPa~~

hPa corresponds to a height of ca. 80 km with a few km variation depending on season, latitude and weather. Other complications existing at higher altitudes, ~~beside~~ besides non-LTE, are strong tides, and processes which are not represented in the model, as for instance atmospheric chemistry and effects from the ionized atmosphere. Such ~~complication~~ complications shall be avoided in the targeted model setup.

- Computational cost increases approximately linearly with the number of layers. Thus, less layers would allow more or longer simulations at the same costs.

As a compromise a vertical grid is chosen that has 191 levels between the surface and the model top at a height of 83 km. The first layer above the surface has a constant thickness of 40 m. Between this layer and a height of 22.5 km the height of the model levels, and thus the layer thicknesses, vary following the implemented smooth-level algorithm. Above 22.5 km all remaining model levels are levels of constant height. The resulting profile of layer thickness versus layer height is shown in Fig. **??** for a surface point at sea level (~~black~~blue profile) and the highest surface point on the R2B9 grid, which is in the Himalayas and has a height of 7368 m ~~. Thus the~~ (red profile). The vertical resolution ranges from 300 m at 12 km, near the tropopause, to 600 m at 50 km height near the stratopause. Over high terrain, however, a relatively strong change in vertical resolution appears near 13 km height, which unfortunately cannot be avoided with the existing implementation of the smooth level algorithm.

2.3 Dynamics

For the QUBICC experiments the time step is adjusted to the higher resolution in the horizontal and vertical grids: $\mathrm{d}t = 40\mathrm{s} = 5 \cdot \mathrm{d}t_{\mathrm{dyn}}$, where $\mathrm{d}t_{\mathrm{dyn}}$ is the time step of the substeps used in the dynamical core. The model time step $\mathrm{d}t$ is thus slightly shorter than the 45 s time step used for the same horizontal resolution in **?**. The reason is the increased vertical resolution which imposes narrower limits for stability in the vertical tracer transport.

2.4 Tracer transport

The QUBICC experiments include a total of 6 tracers for water vapor, cloud water, cloud ice, rain, snow and graupel. This enlarged set of tracers compared to **?** is related to the more detailed cloud microphysics scheme that predicts also rain, snow and graupel, see below. For efficiency reasons the transport of hydrometeors, i.e. cloud water, cloud ice, rain, snow and graupel, is limited to heights below 22.5 km, assuming that none of these hydrometeors exist in the vicinity of this stratospheric height level and above. Concerning the configuration of the transport scheme (**?**), the horizontal advection for water vapor has been changed from a combination of a semi-Lagrangian flux form and a third order Miura scheme with sub-cycling, to a second order Miura scheme with sub-cycling. The choice of the simpler scheme is related to the difficulty of a GPU implementation for a semi-Lagrangian flux form scheme~~, as discussed later~~. (The GPU port of this scheme is currently ongoing.) Sub-cycling means that the integration from time step n to $n+1$ is split into 3 sub-steps to meet the stability requirements. This sub-cycling is applied only above 22.5 km height, i.e. in the stratosphere and mesosphere, where strong winds exist. The other tracers,

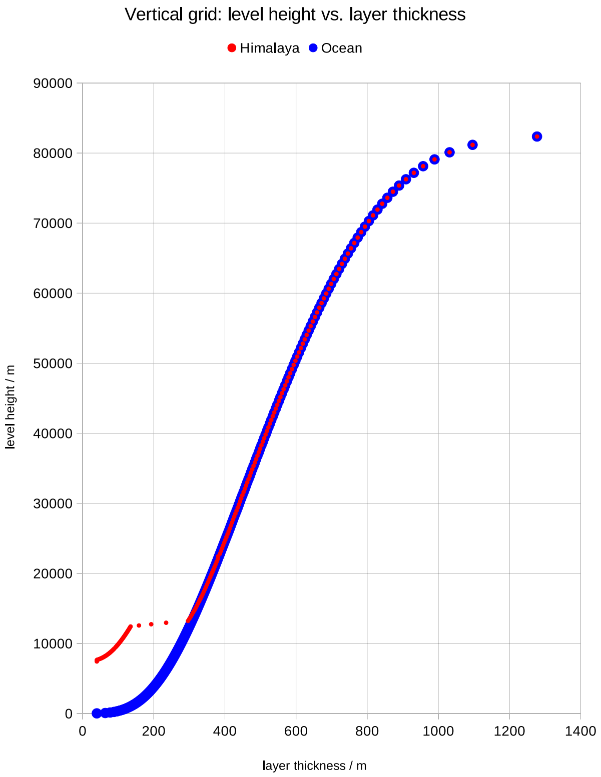

Figure 1. Level height vs. layer thickness from the surface to the model top for a surface height of 0 m over ocean (blue), and for the highest surface point of the R2B9 grid at 7368 m at 87°6’45” E / 27°55’52” N in the Himalayas (red). The lowermost layer has a constant thickness of 40 m, thus is centered at 20 m height above the surface. The uppermost layer is centered at 82361 m and has an upper bound at 83000 m. At heights above 22500 m, the vertical resolution profile is independent of the surface height.

the hydrometeors, are also transported with the second order Miura scheme, though without sub-cycling because they are not transported above 22.5 km.

2.5 Physics

The QUBICC experiments make use of the Sapphire physics package for storm resolving simulations. This package deviates from the ECHAM based physics package described in **?** in a number of points. First of all the physics package excludes parameterizations for convection, atmospheric and orographic gravity waves, and other sub-grid scale orographic effects. These processes are mostly resolved, though not completely, at the grid resolutions used in QUBICC experiments. Further the scientific goal of the QUBICC experiments ~~consist in~~ includes the investigation of the QBO forcing based on the resolved dynamics of deep convective clouds and related waves, which can be granted by excluding parameterized representations of these processes. As a result, the Sapphire physics package is considerably smaller and the model structurally simplified.

The atmospheric processes which still require parameterizations are radiation, the vertical diffusion related to unresolved eddies, and the cloud microphysics. Additionally land processes must be parameterized for the interactive representation of the lower atmospheric boundary conditions over land.

2.5.1 Radiation

From the beginning of the GPU port it was clear that the radiation code was a special challenge due to its additional dimension in spectral space that resolves the shortwave and longwave spectra. Further, initial work on the original PSRAD radiation scheme (**?**) showed that a substantial refactoring would have been necessary for a well performing GPU version of PSRAD, with uncertain outcome. Therefore the decision was taken to replace PSRAD by the new RTE+RRTMGP radiation code (**?**), which was designed from the beginning to work efficiently on CPUs and GPUs, with separate code kernels for each architecture. Thus the ICON code for QUBICC employs now the RTE+RRTMGP code. From a modeling point of view RTE+RRTMGP employs the same spectral discretization methods as PSRAD, namely the k-distribution method and the correlated-k approximation. Differences exist however in using absorption coefficients from more recent spectroscopic data in RTE+RRTMGP, and in the number and distribution of discretization points, so-called g-points, in the SW and LW spectra. While PSRAD used 252 g-points (140 in the longwave spectral region and 112 in the shortwave), RTE+RRTMGP versions on *Piz Daint* and *Juwels-Booster* use 480 (256 LW + 224 SW) and 240 (128 LW + 112 SW) g-points, respectively. Scattering of longwave radiation by cloud particles is not activated in RTE+RRTMGP, so that also in this aspect it is equivalent to the older PSRAD scheme.

As the calculations for the radiative transfer remain the most expensive portion of the model system, a reduced calling frequency, as common in climate and numerical weather prediction models, remains necessary. For QUBICC experiments the radiation time step is set to $\Delta t_{\mathrm{rad}} = 12\mathrm{min} = 18 \cdot \mathrm{d}t$, thus a bit shorter and more frequent than in the simulations of **?**, where $\Delta t_{\mathrm{rad}} = 15\mathrm{min} = 20 \cdot \mathrm{d}t$ was used.

Concerning the atmospheric composition the radiative transfer depends on prognostic fields for water vapor, cloud water, and cloud ice, and on externally specified time dependent greenhouse gas concentrations for CO_2, CH_4, N_2O, CFC11, and CFC12, and O_3, as prepared for the historical simulations of CMIP6 (**?**).

In the spirit of allowing only explicitly modeled scales, all tracers used in the radiation are assumed to be homogeneous within each cell. Thus no parameterized effect of cloud inhomogeneities on the optical path of cloud water and cloud ice is applied in the QUBICC simulations.

~~Further, rain,~~ Rain, snow and graupel concentrations are neglected in the radiative transfer, and for practical reasons no aerosol forcing has been used in the initial experiments.

2.5.2 Vertical diffusion

For the representation of the vertical turbulent diffusion of heat, momentum and tracers the ~~same~~ total turbulent energy parameterization of **?** is used, ~~again~~ which is implicitly coupled to the land surface scheme, see below.

2.5.3 Land surface physics

Land processes in ICON-A are ~~described by~~ parameterized in the JSBACH land surface model which provides the lower boundary conditions for the atmosphere and is implicitly coupled to the atmospheric vertical diffusion parameterization. The infrastructure, ICON-Land, for this ICON-A land component has been newly designed in a Fortran2008 object-oriented, modular and flexible way. The specific implementations of physical, biogeophysical and biogeochemical processes constituting the JSBACH model have been ported from the JSBACH version used with the MPIESM/ECHAM modeling framework (**??**).

For the experiments described in this study, JSBACH has been used in a simplified configuration that uses only the physical processes and in which the sub-grid scale heterogeneity of the land surface properties in each grid box is described by lakes, glaciers and only one single vegetated tile, as in ICON-A (**?**).

2.5.4 Cloud microphysics

Cloud microphysics is parameterized by the "graupel" microphysics scheme (**?**, Sect. 5.2 and 5.3), which is a single moment microphysics scheme for water vapor, cloud water, cloud ice, rain, snow and graupel. All hydrometeors are also transported. For efficiency reasons the computation of cloud microphysics and the transport of cloud tracers are limited to heights below 22.5 km height.

2.5.5 Cloud cover

In the spirit of allowing only explicitly modeled scales, it is assumed that all fields controlling cloud condensation and thus cloud cover are homogeneous in each cell. Thus the instantaneous cloud cover in a cell is diagnosed as either 0 or 1 hundred percent, depending on the cloud condensate mass fraction exceeding a threshold value of $10^{-6}\mathrm{kg/kg}$. Total cloud cover in a column thus is either 0 or 1 hundred percent.

2.6 Coupling of processes

The coupling of the processes described above, the transformations between dynamics variables and physics variables, as well as the time integration follow closely the setup described in Sect. 3 of **?**, also using the simplified case of their Equation 8, for which the scheme is displayed in Fig. **??**. However, a difference with respect to **?** consists in the coupling between the physical parameterizations and is shown in Figure **??**. The coupling scheme applied in our study couples radiation, vertical diffusion with surface land physics, and cloud microphysics sequentially instead of using a mixed coupling scheme (cf. Fig. 6 of **?**).

3 The compute systems

In this section the compute systems used for the presented work are described briefly, with *Piz Daint* at the Swiss National Supercomputing Centre (CSCS) being the main system where the GPU port of ICON was developed and experiments have been carried out during the first year of a PRACE allocation. The second year PRACE allocation was shifted to *Juwels-Booster*

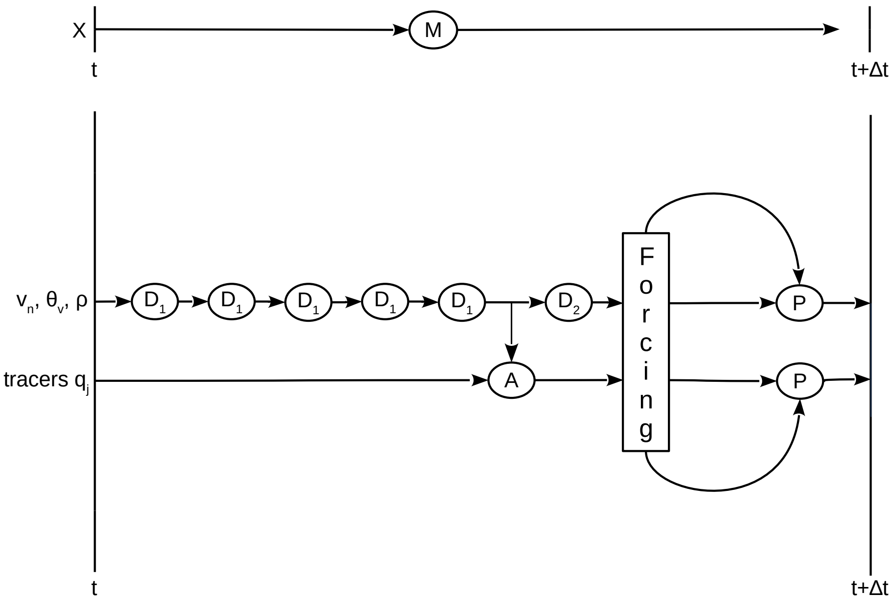

Figure 2. The model operator M propagates the model state X from time t to $t + \mathrm{d}t$, with X consisting of the variables v_n, θ_v and ρ, which are processed by the dynamics, and tracer mass fractions q_i, which are processed by the advection scheme A. The dynamics consists of a sequence of 5 sub-steps (D_1), each propagating the dynamics variables by $\mathrm{d}t/5$, followed by horizontal diffusion (D_2). The advection makes use of the air mass flux computed in the dynamics to achieve consistency with continuity. The intermediate state resulting from dynamics and advection ~~(A)~~ is used for the computation of the forcing, which is applied in the physics update (P) that produces the new state $X(t + \mathrm{d}t)$.

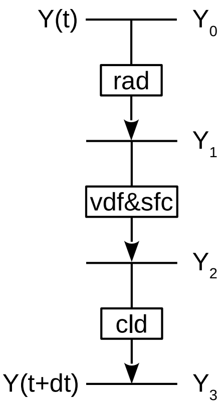

Figure 3. The forcing consists of three sequentially coupled components for radiative heating (rad), vertical diffusion (vdf) coupled implicitly to land surface processes (sfc), and cloud microphysics (cld). Each component computes its contribution to the forcing from a provisional state Y expressed in the physics variables T, m, q_i, u and v.

at the Forschungszentrum Jülich (FZJ), where the GPU port of ICON was further optimized followed by new scaling tests and experiments. Last but not least the same code was used also for additional scaling tests on the new *Levante* computer at the German Climate Computing Center (DKRZ), which is a CPU architecture, thus demonstrating the portability across a number of platforms. The maximum sustained throughput $R_{\max}$ from the HPL (high-performance linpack) benchmarks are used to normalize the performance across the machines. Because ICON is often memory bandwidth limited the HPCG (high performance conjugate gradient) benchmarks would be a more informative norm, however these are not available for *Levante*.

3.1 The compute system *Piz Daint* at CSCS

The main work of porting ICON to GPUs including extensive testing, benchmarking, and performing the first set of experiments for the QUBICC project was carried out on the *Piz Daint* computer at CSCS. *Piz Daint* is a hybrid Cray XC40/XC50 system with 5704 XC50 nodes and 1813 XC40 dual-socket CPU nodes (**?**) with a linpack performance of $R_{\max} = 21.2\,\mathrm{PFlop/s}$ (**?**). The work presented here is targeting the XC50 nodes of the machine, which contain an Intel Xeon E5-2690 v3 CPU with 12 cores and 64 GB memory and a NVIDIA Tesla P100 GPU with 16 GB memory.

The main software used for compiling the ICON code is the PGI/NVIDIA compiler, which on *Piz Daint* is currently the only option for using OpenACC directives in a large Fortran code like ICON that makes use of Fortran 2003 constructs. Software versions of essential packages used from *Piz Daint* for building the ICON executable are listed in Table **??**.

Table 1. System software used for compiling ICON on *Piz Daint*, *Juwels-Booster*, and *Levante*.

Software	*Piz Daint*	*Juwels-Booster*	*Levante*
Compiler	pgi/20.1.1	pgi/21.5	intel/2022.0.1
MPI communication	cray-mpich/7.7.16	OpenMPI/4.1.1	OpenMPI/4.1.2
CUDA Toolkit	cudatoolkit/11.0.2	CUDA/11.3	-
NetCDF	cray-netcdf/4.7.4.0	netCDF/4.7.4	netcdf-c/4.8.1, netcdf-fortran/4.5.3
HDF5	cray-hdf5/1.12.0.0	HDF5/1.10.6	HDF5/1.12.1

3.2 The compute system *Juwels-Booster* at FZJ

After the first version of ICON-A for GPUs was working on *Piz Daint*, the newer *Juwels-Booster* system at FZJ became available. This led to a second version of the ICON GPU code, with model improvements and further optimizations of the GPU parallelization, both benefiting the computational performance of the model.

The *Juwels-Booster* system at FZJ comprises 936 nodes, each with 2 AMD EPYC Rome CPUs and 256 GB memory per CPU, and 4 NVIDIA A100 GPUs with 40 GB memory per GPU (**?**). The maximum linpack sustained performance of this system is 44.1 $\mathrm{PFlop/s}$ (?). The main software used for compiling the ICON code on *Juwels-Booster* is also shown in Table **??**. Also here the PGI~~compiler~~ /NVIDIA compiler with OpenACC is the only option to use the model on GPUs.

3.3 The compute system *Levante* at DKRZ

The third compute system used for scaling tests is the new CPU system *Levante* at DKRZ, which is the main provider of computing resources for MPI-M and other climate research institutes in Germany. *Levante* is used here to demonstrate the portability of the code developed for GPU machines back to CPUs, and also to measure the performance on a CPU machine for comparison to the GPU machines.

The *Levante* system entered service in March 2022 consisting of a CPU partition with 2832 nodes, each with 2 AMD EPYC Milan x86 processors. A GPU partition with 60 GPU nodes, each with 4 NVIDIA A100 GPUs is presently being installed. The 2520 standard CPU nodes have 128 GB memory per CPU, while others have more memory (?). When fully operational *Levante* is expected to have a LINPACK $R_{\max} = 9.7\ \mathrm{PFlop/s}$. Benchmarks during the installation phase of ~~Levante~~ *Levante* arrived at a LINPACK $R_{\max}$ of 7 PFlop on 2048 CPU nodes (?). The software used for compiling is listed in Table **??**.

4 Porting ICON to GPUs

4.1 General porting strategy

On current supercomputer architectures, GPU and CPU have separate memories, and the transfer of data between the two goes via a slow connection compared to the direct access of the local memory of each device. When considering the port of an application to GPU, the key decision is which part can be run on CPU or GPU, so that data transfer between them can be minimized. Since the compute intensity, i.e., ratio of floating point operations to memory load, of typical computation patterns in weather and climate models is low, it becomes clear that all computations occurring during the time loop need to be ported to the GPU to avoid data transfers within it.

ICON-A inherently operates on three-dimensional domains: the horizontal is covered with a space-filling curve, which is decomposed first between the MPI processes and then, within each domain, split up into `nblocks` blocks of arbitrary size `nproma` in order to offer flexibility for a variety of processors. The vertical levels form the other dimension of size `nlev`. Most, but not all, of the underlying arrays have the index order (`nproma`, `nlev`, `nblocks`), possibly with additional dimensions of limited size.

The basic idea of the GPU port is to introduce the OpenACC `PARALLEL LOOP` statements around all the loops that operate on the grid data. We identify the following main approaches to improve the performance of such approach:

- employing structured data regions spanning multiple kernels to avoid any unnecessary CPU-GPU data transfers for the automatic arrays;
- collapsing horizontal and vertical loops where possible to increase the available parallelism;

- fusing adjacent similar loops when possible by writing an embracing `PARALLEL` region with multiple loops using `LOOP GANG(static:1) VECTOR`;
- using `ASYNC` clause to minimize the launch latency;
- "scalarization", i.e., using scalar temporary variables instead of `nproma`-sized arrays where possible;
- restructuring and rewriting a few loops that are not directly amenable to efficient porting, for example, using CLAW, see Section **??**.

In the GPU port of ICON we assume that `nproma` is chosen as large as possible, ideally such that all cell grid points of a computational domain including first and second level halo points fit into a single block thus yielding `nblocks` $= 1$. Therefore the `nproma` dimension is in general the main direction of parallelism. Considering the data layout with `nproma`, with unit stride in memory, this needs to be associated with the "vector" OpenACC keyword to ensure coalesced memory access.

4.2 GPU memory

Due to the 16 GB memory limitation on the P100 GPUs of *Piz Daint*, it was crucial to limit the allocation of ICON data structures on the GPU. To this end, OpenACC's *selective deep copy* was used, in which all relevant arrays are allocated only if needed and then copied individually to the GPU just before the main time loop. At its end, the data ~~types~~ structures are deleted on GPU, because all subsequently required data have been updated on the host within the loop. The selective deep copy required a new Fortran module `mo_nonhydro_gpu_types`, which is inactive for CPU compilation.

Within the time loop all calculation (Dynamics, Physics) is performed on the device, except for minor computation whose results (at most one-dimensional arrays) can be copied to the device with minimal overhead with `UPDATE(DEVICE)` clauses.

ICON uses an unstructured grid formulation, meaning that accesses to cell, edge and vertex neighbors go through indexing arrays, i.e., indirect addressing. Therefore, within the time loop all graph information also has to reside on the device memory.

4.3 Porting the dynamical core

The ICON non-hydrostatic dynamical core algorithms have been extensively documented in **?**. In this section, the dynamical core, or "dycore", is defined as (1) the non-hydrostatic solver, (2) tracer advection, (3) ~~diffusion, and finally~~ horizontal diffusion, (4) operators such as interpolations, divergence, gradient and other stencil computations, and finally (5) all infrastructure called by these — not necessarily exclusively — such as communication~~, as well as interpolations, divergence, gradient and other stencil computations.~~

. Only the accelerator implementation of the dynamical core is discussed in this section. The validation of the accelerator execution ~~, which actually took much longer than the implementation,~~ is discussed in Sect. **??**.

4.3.1 Non-hydrostatic solver

In a preliminary phase, OpenCL and CUDAFortran versions of a prototype non-hydrostatic dycore were created as a proof of concept. ICON developers were not willing to include these paradigms into their code base and insisted on an implementation with only compiler directives.

This methodology was explored first in the ICON dycore and the underlying infrastructure was ported to GPUs using OpenACC directives. These improvements were also incorporated into the ICON development code base, and this work was documented in **?**. In this dycore version, kernels operated on the full three-dimensional (`nproma`, `nlev`, `nblocks`) domain, in other words over three nested `DO` loops. Due to this approach, the optimal block size `nproma` was in the range 500-2000.

However, this approach turned out to be a considerable limitation: in the physical parameterizations the loop over all blocks is many subroutine levels above the loops over the block and the levels. Although it is in theory possible to construct OpenACC parallel region with a complex and deep subroutine call tree, ~~it proves in practice~~ in practice it proves not to be a viable approach with the available OpenACC compilers (PGI and Cray). In order to avoid a complex programming technique, it was decided to refactor the dynamical core to parallelize only over the two inner dimensions, `nproma` and `nlev`, when possible, see Listing **??**. With this approach the optimal `nproma` is chosen as large as possible, i.e., having effectively one block per MPI subdomain and thus a single iteration in the `jb` loop in Listing **??**.

Listing 1. Most common loop structure in dynamical core with asynchronous execution.

```
    DO jb = i_startblk , i_endblk

        CALL get_indices_c(p_patch , jb , i_startblk , i_endblk , &
          i_startidx , i_endidx , rl_start , rl_end)

!$ACC PARALLEL IF( i_am_accel_node .AND. acc_on ) DEFAULT(NONE) ASYNC(1)
          !$ACC LOOP GANG VECTOR COLLAPSE(2)
          DO jk = 1, nlev
            DO jc = i_startidx , i_endidx
              :
            ENDDO
          ENDDO
!$ACC END PARALLEL

         :
     ENDDO
```

Generally kernels are denoted with the `ACC PARALLEL` directive, which allows the user to be more prescriptive than the higher level descriptive `ACC KERNELS` directive, which is used in ICON only for operations using Fortran array syntax. Usage of `KERNELS` for more complicated tasks tended to reduce performance.

There are code ~~divergences~~ differences between CPU and GPU in the non-hydrostatic solver. On the accelerator it is advantageous to use scalars within loops, while for the CPU frequently two-dimensional arrays perform better, see Listing **??**.

Listing 2. Register variables outperform arrays on GPU. One of roughly 10 code divergences in the dynamical core

```
!$ACC PARALLEL IF( i_am_accel_node .AND. acc_on )  DEFAULT(NONE) ASYNC(1)
          !$ACC LOOP GANG VECTOR COLLAPSE(2)
          DO jk = nflatlev(jg)+1, nlev
            DO jc = i_startidx, i_endidx
!Original code had advantages with older vector compilers,
              z_w_concorr_mc_m1 =  &   ! original: z_w_concorr(jc,jk-1)
                p_int%e_bln_c_s(jc,1,jb)*z_w_concorr_me(ieidx(jc,jb,1),jk-1,ieblk(jc,jb,1)) + &
                p_int%e_bln_c_s(jc,2,jb)*z_w_concorr_me(ieidx(jc,jb,2),jk-1,ieblk(jc,jb,2)) + &
                p_int%e_bln_c_s(jc,3,jb)*z_w_concorr_me(ieidx(jc,jb,3),jk-1,ieblk(jc,jb,3))
                :
              z_w_concorr_mc_m0 =  &  ! original:  z_w_concorr(jc,jk)
                p_int%e_bln_c_s(jc,1,jb)*z_w_concorr_me(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) + &
                p_int%e_bln_c_s(jc,2,jb)*z_w_concorr_me(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) + &
                p_int%e_bln_c_s(jc,3,jb)*z_w_concorr_me(ieidx(jc,jb,3),jk,ieblk(jc,jb,3))
              p_nh%diag%w_concorr_c(jc,jk,jb) =                              &
                p_nh%metrics%wgtfac_c(jc,jk,jb)*z_w_concorr_mc_m0 +        &
                (1._vp - p_nh%metrics%wgtfac_c(jc,jk,jb))*z_w_concorr_mc_m1
!Original: ... *z_w_concorr(jc,jk-1) + (1._vp - p_nh%metrics%wgtfac_c(jc,jk,jb))*z_w_concorr(jc,jk)
            ENDDO
          ENDDO
!$ACC END PARALLEL
```

After extensive refactoring and optimizations, such as asynchronous kernel execution and strategically placed `ACC WAIT` directives, the resulting dycore version performed at least as well on GPUs as the original GPU version with triple-nested parallelism, with the former operating with `nblocks` = 1 or a very small integer, and thus the largest possible `nproma`. See Sect. **??** for complete performance comparisons, in particular between CPU and GPU.

4.3.2 ~~Transport schemes~~Tracer transport

~~Transport schemes predict the large-scale redistribution of atmospheric tracers such as water substances, as in the model setup used here, and chemical constituents or aerosols in the atmosphere due to air motion. Mathematically, advection solves one of the fundamental laws of physics, namely the equation of tracer mass continuity.~~

~~In ICON transport, the numerical solution to the tracer mass continuity equation is based on so called space-time finite volume methods (?). Finite volume methods are derived on the cell-integrated form of the underlying partial differential equation.~~

There are several different variants of horizontal and vertical advection, depending on whether the scheme is Eulerian or semi-Lagrangian, what sort of reconstructions (second or third order) and which type of time-stepping is employed. All of these variants ultimately can be considered stencil operations on a limited number of neighboring cells, i.e., physical quantities defined in cell centers, vertices or edges. As such, the structure of the corresponding kernels is usually similar to Listing **??**.

In several parts of the code specific ~~optimization, using~~ optimizationa, so called *index lists* as shown in Listing **??** are ~~used~~ employed for better performance on CPUs, in particular for vector machines. The advantage of an index list is that the subsequent calculation can be limited only to the points which fulfill a certain criterion, which is generally quite rare, meaning the list is sparse and thus quite small. In addition such an implementation avoids the use of if statements which makes it easier for compiler to auto-vectorize this code section. For the GPU parallelization such index list implementation has unfortunately a negative impact on performance as the list creation is a sequential operation.

Listing 3. Index lists used in vertical flux calculation with reconstruction by the piece-wise parabolic method.

```
DO jc = i_startidx , i_endidx

  ! jk_shifted must fall within the range [top_bound, bot_bound] in order
  ! to pass the following if condition. Unfortunately, the range depends on
  ! the sign of w.
     :
  IF ( z_aux(jc) > p_cellmass_now(jc,jk_shifted(jc,jk,jb),jb)   &
    &  .AND. jk_shifted(jc,jk,jb) <= bot_bound                  &
    &  .AND. jk_shifted(jc,jk,jb) >= slevp1_ti                  ) THEN
     :
    ! Fill index lists with those points that need index shifts
    ! Note that we have to use a scalar counter instead of a vector, like
    ! i_listdim(nlist,jb). Otherwise this loop will not vectorize.
    counter_ji = counter_ji + 1
    i_indlist(counter_ji,nlist,jb) = jc
    i_levlist(counter_ji,nlist,jb) = jk
      :
  ENDIF

END DO ! end loop over cells
```

On an accelerator, numerous execution threads will be competing to increment `counter_ji` and insert indices into `i_indlist, i_levlist`. We overcame this by using OpenACC atomics or parallel algorithms based on exclusive scan techniques. However, in some cases the proper GPU algorithm is to operate over the full loop. The GPU executes both code paths of the `IF` statement, only to throw the results of one path away. The algorithm for `vflux_ppm4gpu` is functionally equivalent (Listing **??**).

Listing 4. Index list-free implementation adapted for accelerator execution.

```
!$ACC PARALLEL DEFAULT(NONE) PRESENT(z_cfl) ASYNC(1) IF( i_am_accel_node .AND. acc_on )
!$ACC LOOP GANG VECTOR PRIVATE( z_mass, jks ) COLLAPSE(2)
        DO jk = slevp1_ti , elev
          DO jc = i_startidx , i_endidx
            z_mass = p_dtime*p_mflx_contra_v(jc,jk,jb) ! total mass crossing jk'th edge
            IF (z_mass > 0._wp) THEN
              jks = jk   ! initialize shifted index
```

```
                    DO WHILE( (z_mass > p_cellmass_now(jc,jks,jb)) .AND. (jks <= nlev-1) )
                       ! update Courant number
                    ENDDO
                    ! now we add the fractional Courant number
                    z_cfl(jc,jk,jb) = z_cfl(jc,jk,jb) + MIN(1._wp,z_mass/p_cellmass_now(jc,jks,jb))
                 ELSE
                    :
                       ! update Courant number
                    :
                    ! now we add the fractional Courant number
                    z_cfl(jc,jk,jb) = z_cfl(jc,jk,jb) + MAX(-1._wp,z_mass/p_cellmass_now(jc,jks,jb))
                 ENDIF
              ENDDO ! jc
           ENDDO ! jk
```

Some horizontal advection schemes and their flux limiters require halo exchanges in order to make all points in the stencil available on a given process. The communication routines are described in Sect. **??**.

4.3.3 ~~Non-hydrostatic~~ Horizontal diffusion

The dynamical core contains several variants of horizontal diffusion. The default approach is a more physically motivated second-order Smagorinsky diffusion of velocity and potential temperature combined with a fourth-order background diffusion of velocity, using a different discretization for velocity that is formally second-order accurate on equilateral triangles.

Most of the horizontal diffusion contains kernels in the style of Listing **??**, but again there are index lists for the normal CPU calculation. Listing **??** illustrates how the index lists are avoided at the cost of a temporary 3-D array.

Listing 5. The OpenACC version uses a temporary 3D array `enh_diffu_3d` defined in cell centers and a revised MAX statement on the edge grid to avoid the construction of `iclist/iklist` from previous loop.

```
#ifndef _OPENACC
        DO jb = i_startblk,i_endblk
           IF (icount(jb) > 0) THEN
              DO ic = 1, icount(jb)
                 jc = iclist(ic,jb)
                 jk = iklist(ic,jb)
                 enh_diffu = tdlist(ic,jb)*5.e-4_vp
                 kh_smag_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)) = &
                      MAX(enh_diffu,kh_smag_e(ieidx(jc,jb,1),jk,ieblk(jc,jb,1)))
                 kh_smag_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)) = &
                      MAX(enh_diffu,kh_smag_e(ieidx(jc,jb,2),jk,ieblk(jc,jb,2)))
                 kh_smag_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)) = &
                      MAX(enh_diffu,kh_smag_e(ieidx(jc,jb,3),jk,ieblk(jc,jb,3)))
              ENDDO
           ENDIF
```

```
    ENDDO
#else
    :
    DO jb = i_startblk,i_endblk
      CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, i_startidx, i_endidx, rl_start, rl_end)
!$ACC PARALLEL LOOP DEFAULT(NONE) GANG VECTOR COLLAPSE(2) ASYNC(1) IF( i_am_accel_node .AND. acc_on )
      DO jk = nlev-1, nlev
        DO je = i_startidx, i_endidx
           kh_smag_e(je,jk,jb) = MAX(kh_smag_e(je,jk,jb),                         &
                                     enh_diffu_3d(iecidx(je,jb,1),jk,iecblk(je,jb,1)), &
                                     enh_diffu_3d(iecidx(je,jb,2),jk,iecblk(je,jb,2)) )
        ENDDO
      ENDDO
!$ACC END PARALLEL LOOP
    ENDDO
#endif
```

4.3.4 Dynamical core ~~infrastructure~~operators

The dynamical core also calls horizontal operators such as averaged divergence or cell-to-vertex interpolation. These operators, along with numerous related stencil operations required in other parts of the model, were also ported with OpenACC. These almost always adhere to the style of Listing **??**, and are thus straightforward to port to OpenACC.

4.3.5 Dynamical core infrastructure

Essentially all of the halo exchanges occur in the dynamical core, the horizontal flux calculation of advection, or in the dynamics-physics interface. During the exchange, the ~~surface~~ lateral boundary cells of a vertical prism residing on a given process ~~is~~ are written into the ~~halo surface~~ lateral halo cells of vertical prisms residing on its neighboring processes. Since these halo exchanges are performed within the time loop, the halo regions are in device memory. Two mechanisms are ~~provided~~ available to perform the exchange:

- Update the prism surface on the CPU, post the corresponding MPI_Isend, and Irecv with a temporary (host) buffer, and after the subsequent MPI_WAIT operation, update receive buffer on the device, and copy the buffer to the halo region solely on the device.

- Pass GPU pointers to the same Isend and Irecv routines in a GPU-aware MPI implementation. The final copy to the halo region is again performed on the device.

These two mechanisms illustrated in Listings **??** and **??** are easily woven together with logicals in the corresponding OpenACC `IF` clauses.

Listing 6. High level halo receive operation with optional GPU-to-GPU communication

```
!$ACC DATA CREATE( send_buf, recv_buf ) PRESENT( recv, p_pat ) IF (use_gpu)

    IF (iorder_sendrecv == 1 .OR. iorder_sendrecv == 3) THEN
      ! Set up irecv's for receive buffers
      DO np = 1, p_pat%np_recv ! loop over PEs from where to receive the data

        pid    = p_pat%pelist_recv(np) ! ID of receiver PE
        irs    = p_pat%recv_startidx(np)
        icount = p_pat%recv_count(np)*ndim2
        CALL p_irecv(recv_buf(1,irs), pid, 1, p_count=icount, comm=p_pat%comm, use_g2g=use_g2g)
      ENDDO
    ENDIF
      :
    CALL p_wait
!$ACC UPDATE DEVICE( recv_buf ) IF (.NOT. use_g2g)
```

Listing 7. Implemention of p_irecv, which supports GPU-to-GPU communication

```
    !$ACC HOST_DATA USE_DEVICE( t_buffer ) IF ( use_g2g )
    CALL p_inc_request
    CALL mpi_irecv(t_buffer, icount, p_real_dp, p_source, p_tag, &
         p_comm, p_request(p_irequest), p_error)
    !$ACC END HOST_DATA
```

4.4 Physical Parameterizations

The provision of the physical forcing for the time integration is organized in four levels outlined in Listing **??**. The first level, which is the dynamics physics interface, transforms the provisional variable state $X(t)$ that results from dynamics and transport (Fig. **??**) to the physics variable state Y that is the input for the physical parameterizations (Fig. **??**). And on return from the physics the collected total tendencies from physics in Y variables are converted to tendencies in ~~X variablesfor dynamics and computes the new dynamics~~ the X variables, followed by the computation of the new state $X(t+\mathrm{d}t)$. These tasks involve loops over blocks, levels and grid points as in dynamics. Their parallelization on the GPU therefore follows the pattern used in the dynamics codes, see Listing **??**.

At the second level the physics main routine calls the physical parameterizations of the Sapphire configuration in the sequence shown in Fig. **??** by use of a generic subroutine. This routine contains the block loop from which ~~a parameterization interface , as specified by argument,~~ the specified parameterization interface routine is called for each single data block. Thus the computation below this level concerns only the `nproma` dimension over cells and the `nlev` dimension over levels, and in some cases extra dimensions for instance for tracers or surface tiles. ~~This second level contains only OpenMP directives for the parallelization of the block loop, but not OpenACC directives because the parallelization on GPUs is employed only within data blocks~~Note that the second level interface and the generic subroutine do not compute any fields, and therefore do not use the GPU and are free of OpenACC directives.

Listing 8. Lines from the 1st to 3rd level interfaces and the generic routine with the block loop used in the 2nd level interface for calling the 3rd level interface routines for specific parameterizations, here for the example of cloud microphyiscs "graupel" (mig).

```
SUBROUTINE interface_iconam_echam(..., patch, ...)     ! <-- 1st level interface
  < uses GPU, ACC directives for data and loop parallelization >
  CALL echam_phy_main(patch, ...)
-------------------------------------------------------------------------------
SUBROUTINE echam_phy_main(patch, ...)                  ! <-- 2nd level interface
  < does not use GPU, no ACC directives >
  CALL omp_loop_cell_prog(patch, interface_echam_mig, ...)
-------------------------------------------------------------------------------
SUBROUTINE omp_loop_cell_prog(patch, routine, ...)
  < does not use GPU, no ACC directives >
  < get grid index jg, start and end indices jbs and jbe for block loop >
!$OMP PARALLEL DO PRIVATE(jb, jcs, jce)
  DO jb = jbs, jbe
    < get start and end indices jcs and jce for cell loops in "routine" >
    CALL routine(jg, jb, jcs, jce, ...)
  END DO
!$OMP END PARALLEL DO
-------------------------------------------------------------------------------
SUBROUTINE interface_echam_mig(jg, jb, jcs, jce, ...)  ! <-- 3rd level interface
  < uses GPU, ACC directives for data and loop parallelization >
```

The third level consists of the interfaces to the ~~specfic~~ specific parameterizations. These interfaces provide the access to the global memory for the parameterizations by `USE` access to memory modules. The equivalent variables in GPU memory, which have been created before and updated where necessary, are now declared ~~as present either for individual variables~~individually as present, as for instance the 3-dimensional atmospheric temperature `ta` and the 4-dimensional array `qtrc` for tracer mass fractions in `$ACC DATA PRESENT(field%ta, field%qtrc)`. This practice was followed in the code used on *Piz Daint*. The newer code on *Juwels-Booster* instead declares the entire variable construct as present instead of its components, like `$ACC DATA PRESENT(field)`. Beside the memory access these interfaces use the output of the parameterization for computing the provisional physics state for the next parameterization in the sequentially coupled physics, and for accumulating the contribution of the ~~parameterizaion~~ parameterization tendencies in the total physics tendencies. These tasks typically require loops over the `nproma` and `nlev`~~dimension~~dimensions, but sometimes also over additional dimensions like tracers. The typical loop structure follows Listing **??**.

Listing 9. Most common loop structure over levels `jk` and cells `jc` in parameterizations and their interfaces.

```
 !$ACC PARALLEL DEFAULT(NONE) ASYNC(1)
 !$ACC LOOP GANG VECTOR COLLAPSE(2)
 DO jk = 1,nlev
   DO jc = jcs,jce
      :
   END DO
 END DO
!$ACC END PARALLEL
```

Particular attention is paid to the bit-wise reproducibility of sums, as for instance for vertical integrals of tracer masses computed in some of these interfaces in loops over the vertical dimension. Here the `!$ACC LOOP SEQ` directive is employed to fix the order of the summands. This bit-wise reproducibility is important in the model development process because it facilitates the detection of unexpected changes of model results, as further discussed in Sect. **??**.

Finally, the forth level exists in the parameterizations. The parameterizations used here are inherently one dimensional, as they couple levels by vertical fluxes. This would allow to encode them for single columns, but for traditional optimization reasons, these codes include a horizontal loop dimension, which in ICON is the `nproma` dimension. Therefore these parameterizations include this additional loop beside those required for the parameterization itself~~, and hence~~. The presence of these horizontal loops favored the GPU parallelization~~in general follows~~, which therefore generally follow the pattern in Listing **??**. ~~Subsequently some~~ Some parameterization specific modifications ~~for~~ of the GPU parallelization are pointed out in the following sections.

4.4.1 Radiation

As pointed out earlier, the GPU implementation aims at using maximum block sizes, so that all grid points and levels within a computational domain fit into a single block, and hence a single iteration of the block loop suffices. Using large block sizes, however, means also that more memory is required to store the local arrays, which is a challenge especially on *Piz Daint* with the small GPU memory capacity. This problem turned out to be most pronounced for the radiation code, owing to the extra spectral dimension. On *Piz Daint* this meant that a single block per domain would not have been feasible. This issue was resolved by allowing for a sub-blocking parameter `rrtmgp_column_chunk` (`rcc`) in the radiation code, so that the original blocks of size `nproma` are broken up into smaller data blocks for input and output of the radiation scheme. This radiation block size can be specified as necessary and is typically ca. 5% to 10% of `nproma` when using the smallest possible number of nodes. But, at the largest node counts during strong scaling as shown in the experiments below, `nproma` can become small enough so that no sub-blocking in the radiation is needed and `rcc` is set to the number of grid points in the computational domain.

As explained in **?**, RTE+RRTMGP comprises a set of user-facing code, written in object-oriented Fortran 2008, which is responsible for flow control, input validation, etc. Computational tasks are performed by computational kernels using assumed-size arrays with C bindings to facilitate language interoperability. For use on GPUs a separate set of kernels was implemented in Fortran using OpenACC directives, with refactoring to increase parallelism at the cost of increased memory use relative to the original CPU kernels. The Fortran classes also required the addition of OpenACC data directives to avoid unnecessary data flows between CPU and GPU.

RTE+RRTMGP, like ICON, operates on sets of columns whose fastest-varying dimension is set by the user and whose second-fastest varying dimension is the vertical coordinate. Low-level CPU kernels are written as loops over these two dimensions, with higher-level kernels passing results between low-level kernels while looping over the slowest-varying spectral dimension. This approach, illustrated in Listing **??**, keeps memory use modest and facilitates the reuse of cached data. Low-level GPU kernels, in contrast, operate on all three dimensions at once. When the calculation is parallelizable in all dimensions~~(~~, i.e where values at every spatial and spectral location are independent~~)~~, the parallelization is over all $\texttt{rcc} \times \texttt{nlev}(=191) \times \texttt{ngpt}(=\mathcal{O}(125))$ elements at once. Some loops have dependencies in the vertical; for these the GPU kernels are parallelized over the column and spectral point, with the vertical loop performed sequentially within each horizontal and spectral loop (see Listing **??**).

Listing 10. Example loop structure for CPU kernels in RTE+RRTMGP. High level kernels operate on all levels and columns in the block but only one spectral point (g-point) at a time. In this example the loop over g-points is performed at one higher calling level.

```
!
! Element-wise loop for fully independent calculations
!
do ilay = 1, nlay
  do icol = 1, ncol
    tau_s = tau(icol, ilay)
    ! Intermediate computation
    ...
    source_up(icol,ilay) =    Rdir * dir_flux_inc(icol)
  end do
end do

!
! Vertically-dependent loop (e.g. for radiation transport)
!
do ilev = nlay, 1, -1
  denom           = 1._wp/(1._wp - rdif(:,ilev)*albedo(:,ilev+1))         ! Eq 10
  albedo(:,ilev) = rdif(:,ilev) + &
                   tdif(:,ilev)*tdif(:,ilev) * albedo(:,ilev+1) * denom ! Equation 9
  ...
end do
```

Listing 11. Example loop structure for GPU kernels in RTE+RRTMGP. For the GPUs kernels operate on all levels, columns, and spectral points at once, expect where dependencies in the vertical require the vertical loop to be done sequentially.

```
    !$acc  parallel loop collapse(3)
    do igpt = 1, ngpt
      do ilay = 1, nlay
        do icol = 1, ncol
          tau_s = tau(icol,ilay,igpt)
          ! Intermediate computation
          ...
          source_up(icol,ilay,igpt) =    Rdir * dir_flux_inc(icol,igpt)
        end do
      end do
    end do
```

```
!
! Vertically-dependent loop (e.g. for radiation transport)
!
!$acc parallel loop gang vector collapse(2)
!$omp target teams distribute parallel do simd collapse(2)
! Note loop over levels is performed sequentially
do igpt = 1, ngpt
  do icol = 1, ncol
    do ilev = nlay, 1, -1
      denom (icol,ilev,igpt) = 1._wp/(1._wp - rdif(icol,ilev,igpt)*albedo(icol,ilev+1,igpt)) ! Eq 10
      albedo(icol,ilev,igpt) = rdif(icol,ilev,igpt) + &
                               tdif(icol,ilev,igpt)*tdif(icol,ilev,igpt) * &
                               albedo(icol,ilev+1,igpt) * denom(icol,ilev,igpt)             ! Eq 9
      ...
    end do
  end do
end do
```

Most sets of kernels in RTE+RRTMGP now contain two separate implementations organized in distinct directories with identical interfaces and file naming. A few sets of kernels (e.g. those related to summing over spectral points to produce broadband fluxes) were simple enough to support the addition of OpenACC directives into the CPU code.

Though the original plan was to restrict OpenACC directives to the kernels themselves it became clear that the Fortran class ~~front ends~~ front-ends contain enough data management and small pieces of computation that they, too, required OpenACC directives, both to keep all computations on the GPU and to allow the sharing of data from high level kernels (for example, to reuse interpolation coefficients for the computation of absorption and scattering coefficients by gases). The classes therefore have been revised such that communication with the CPU is not required if all the data used by the radiation parameterization (temperatures, gas concentrations, hydrometeor sizes and concentrations, etc.) already exists on the GPU.

4.4.2 Land surface physics

One of the design goals of the new ICON-Land infrastructure has been to make it easy for domain scientists to implement the scientific routines for a specific land model configuration, such as JSBACH. Except for the (lateral) river routing of runoff, all processes in JSBACH operate on each horizontal grid cell independently, either 2-D or 3-D with an additional third dimension such as soil layers, and therefore don't require detailed knowledge of the memory layout or horizontal grid information. To further simplify the implementation, the 2-D routines are formulated as Fortran elemental subroutines or functions thus abstracting away the field dimensions of variables and loops iterating over the horizontal dimension. As intermediate layer between the infrastructure and these scientific routines JSBACH uses interface routines which are responsible for accessing variable pointers from the memory and calling the core (elemental) calculation routines. These interfaces make extensive use of Fortran array notation. Consequently, the implementations of the land processes models have no explicit loops over the npromadimension, in contrast to the atmospheric process models, where the presence of these loops favored the GPU parallelization.

Instead of re-factoring large parts of JSBACH to use explicit ACC directives and loops and thus hampering the usability for domain scientists, the CLAW (**?**) source-to-source translator has been used for the GPU port. CLAW consists of a domain-specific language (DSL) and a compiler allowing it to automate the port to OpenACC with much fewer directives and changes to the model code than are necessary with pure ACC. For example, blocks of statements in the interface routines using array notation can simply be enclosed by `!$claw expand parallel` and `!$claw end expand` directives and are then automatically expanded into ACC directives and loops. An example of this mechanism, as used in the JSBACH interface to radiation, is shown in Figures 8 and 9 of **?**.

The elemental routines discussed above are transformed into ACC code by using an additional CLAW feature: the CLAW Single Column Abstraction (SCA) (**?**). The CLAW SCA has been specifically introduced in CLAW to address performance portability for physical parameterizations in weather and climate models which operate on horizontally independent columns. Using the CLAW SCA translator, the only changes necessary in the original Fortran code of JSBACH were to

- prepend the call to an elemental routine by the CLAW directive `!$claw sca forward`,
- insert `!$claw model-data` and `!$claw end model-data` around the declaration of scalar parameters in the elemental routine that need to be expanded and looped over,
- insert a `!$claw sca` directive in the beginning of the statement body of the elemental routine.

The CLAW SCA transformation then automatically discards the `ELEMENTAL` and `PURE` specifiers from the routine, expands the flagged parameters to the memory layout specified in a configuration file and inserts ACC directives and loops over the respective dimensions. Examples for the original and transformed code of the JSBACH routine calculating the net surface radiation is shown in Figure 5 and 6, respectively, of **?**.

More details on the CLAW port of JSBACH including ~~code examples and~~ performance measurements for the radiation component of JSBACH can be found in **?**.

4.4.3 Cloud microphysics

Cloud microphysical processes are computed in three sequential steps: (1) saturation adjustment for local condensation or evaporation, (2) the microphysics between the different hydrometeors and the vertical fluxes of rain, snow and graupel, and (3) again saturation adjustment for local condensation or evaporation. Here the code for the saturation adjustment exist in a CPU and GPU variant, selected by a compiler directive. The CPU code sets up one dimensional lists of ~~grid~~ cell and level indices, where the adjustment requires Newton iterations, while the GPU code uses a logical 2 dimensional mask with `nproma` and `nlev` dimensions for the same purpose. The CPU code then loops over the cells stored in the index lists while the GPU code employs a two-dimensional loop structure in which computations happen only for the cells selected by the mask. Beside the different means to restrict the computations to the necessary cells, the CPU code is also optimized for vectorizing CPUs, which is the reason that the loop over the list occurs within the condition for ending the Newton iteration cycles, while the GPU code checks this within the parallelized loops. The related code fragments are shown in Listing **??** and Listing **??** .

Listing 12. Code structure of the Newton iteration for the saturation adjustment on CPU, making use of 1d-lists `iwrk` for the cell index and `kwrk` for the level index of cells where the adjustment needs to be computed iteratively.

```
count = 0
DO WHILE (ANY(ABS(twork(1:nsat)-tworkold(1:nsat)) > tol) .AND. count < maxiter)
  DO indx = 1, nsat
    :
    IF (ABS(twork(indx)-tworkold(indx)) > tol) THEN
      ! Here we still have to iterate ...
      i = iwrk(indx)
      k = kwrk(indx)
      tworkold(indx) = twork(indx)
      :
      twork(indx) = twork(indx) - ...
    END IF
  END DO
  count = count + 1
END DO
```

Listing 13. Code structure for saturation adjustment on GPU, making use of a 2d-mask `iwrk` for the cell index and `kwrk` for the level index of cells where the adjustment needs to be computed iteratively.

```
!$acc parallel default(present)
!$acc loop gang vector collapse(2) private(... , count)
DO k = klo,kup
  DO i = ilo,iup
    count = 0
    DO WHILE (ABS(twork(i,k)-tworkold(i,k)) > tol .AND. count < maxiter .AND. iter_mask(i,k))
      ! Here we still have to iterate ...
      tworkold(i,k) = twork(i,k)
      :
      twork(i,k) = twork(i,k) - ...
      count = count + 1
    END DO !while
  END DO !i
END DO !k
!$acc end parallel
```

5 Validation

The ICON development on CPU makes use of test suites comprising simplified test experiments for a variety of model configurations running on a number of compute systems using different compilers and parallelization setups. This includes the AMIP experiment discussed in **?**, but shortened to 4 time steps. The test suite for this experiment checks for reproducible results with respect to changes of the blocking length, amount and kind of parallelization (MPI, OpenMP or both) as well as checks for differences to stored reference solutions. This test suite was also implemented on *Piz Daint*, where the experiments have been run by pure CPU binaries as well as GPU-accelerated binaries. Output produced on these different architectures — even if performed with IEEE arithmetic — will always produce slightly different results due to rounding. Therefore, above mentioned tests for bit-identity cannot be used across different architectures. The problem is made worse by to the chaotic nature of the underlying problem. These initially very small changes, which are on the order of floating point precision, quickly grow across model components and timesteps which makes distinguishing implementation bugs from chaotically grown round-off differences a non-trivial task.

This central question of CPU vs. GPU code consistency was addressed in three ways, (1) the “ptest” mode, (2) by serializing the model state before and after central components~~and~~, and (3) tolerance tests of the model output. Details for these methods are given in the following subsections. The first two methods are able to test a small fraction of the code in isolation where chaotic growth of round-off differences is limited to the tested component and thus small. We found that tolerating relative errors up to differences of 10^{-12} with double precision floating point numbers (precision roughly at 10^{-15}) did not result in many false-positives (a requirement for continuous integration) while still detecting most bugs. Even though most of the code is covered by such component tests, there is no guarantee that passing all these tests leads to correct model output. To ensure this, a third method had to be implemented. This method came to be known as the "tolerance test" because tolerance ranges could not be assumed constant but had to be estimated individually for each variable across all model components and over multiple time-steps. It should be emphasized that, while the introduction of directives took only weeks of work, the validation of proper execution with the inevitable round-off differences between CPU and GPU execution took many months.

5.1 Testing with ptest mode

The pre-existing internal “ptest” mechanism in ICON allows the model to run sequentially on one process and in parallel on the "compute processes" with comparisons of results at synchronization points, such as halo exchanges. This mechanism was extended for GPU execution with the addition of IF statements in kernel directives, so that the GPUs are active only on the compute processes. Listing **??** illustrates all of the above-mentioned ideas. In particular, the global variable `i_am_accel_node` is .TRUE. on all processes which are to execute on accelerators but .FALSE. on the worker node delegated for sequential execution.

If the ptest mode is activated when a synchronization point is encountered, arrays calculated in a distributed fashion on the MPI compute processes are gathered and compared to the array calculated on the single sequential process. Synchronization points can either be halo exchanges, or manually inserted `check_patch_array` invocations which can compare any arrays in the standard 3-D ICON data layout.

While this method was very handy at the beginning of the effort to port the model to GPUs, especially for the dynamical core of the model, extending it beyond the pre-existing mechanism turned out to be cumbersome. At the same time, the Serialbox library offered a very flexible way to achieve the same goal without running the same code on different hardware at the same time.

5.2 Serialization

Besides the “ptest” technique mentioned two other approaches were used for the validation of GPU results. First the full model code was used with test experiments, which typically use low resolution, just a few time steps, and only the component of interest. Examples are AMIP experiments on the R2B4 grid, as used in **?**, but for 4 time steps only, with or without physics, or with only a single parameterization. This approach has the advantage that the experiments can be compared to other related experiments in the common way, based on output fields as well as the log files.

The second approach, the "serialization" method uses such experiments only to store all input and output variables of a model component. Once this reference data is stored, the test binary (usually utilizing GPUs) reads the input data from file and calls the model component in exactly the same way as the reference binary (usually running exclusively on CPUs). The new output is stored and compared to the previously generated reference data, for instance to check for identical results or for results within defined tolerance limits due to round-off differences between pure CPU and GPU-accelerated binaries. This serialization mechanism was implemented in ICON for all parameterizations (but not dynamics or transport, which were tested with the technique mentioned in **??**), which were ported to GPU, and was primarily used during the process of porting individual components to GPU. The advantage of this method is the fine test granularity that can be achieved by surrounding arbitrary model components with the corresponding calls to the serialization library.

5.3 Tolerance testing

The methods discussed in the last two sections are valuable tools to locate sources of extraordinary model state divergence (usually due to implementation bugs) as well as frequent testing during optimization and GPU code development. However, they do not guarantee the correctness of the model output. This problem is fundamentally different from component testing because chaotic growth of initial round-off differences is not limited to a single component but quickly accumulates across all model components and simulation timesteps. This section introduces a method to estimate this perturbation growth and how it is used to accept or reject model state divergence between pure CPU and GPU-accelerated binaries.

The idea is to generate for each relevant test experiment on CPU an ensemble of solutions, which diverge due to tiny perturbations in the initial state. In practice the ensemble is created by perturbing the state variables ~~conisting~~ consisting of the vertical velocity$w$, normal velocities on cell edgesv_n, the virtual potential temperature θ_v, the Exner function Π, and the densityρ by uniformly distributed random numbers in the interval [-1e-14, 1e-14]. The resulting ensemble, which consist of the unperturbed and 9 perturbed simulations, is used to define for each time step of the test the tolerance limits for all output variables. In practice, we do not compute the tolerance limit for each gridpoint, but define a single value for each variable and timestep by applying different statistics across the horizontal dimension and selecting the largest value for each statistic across the vertical dimension. The test is currently implemented using minimum, maximum and mean statistics in the horizontal. This approach has proved to be effective to discard outliers. Applying the same procedure to the model output from the GPU-accelerated binary allows to compare the test results with the pure CPU reference under the limits set by the perturbed CPU ensemble. This method proved effective in highlighting divergences in the development of the GPU version of the ICON code over a small number of time steps.

6 Benchmarking Results

Once the GPU port of all components needed for the planned QUBICC experiments was completed, practical testing was started with the first full experiment shortened to two simulation hours – a computational interval that proved to be sufficient to provide robust performance results. For the technical setup it was found that a minimum of ca. 960 GPU nodes of *Piz Daint* was necessary for the memory of a QUBICC experiment. Because performance was affected when getting close to ~~the limit,~~ 960 nodes, a setup of 1024 nodes was chosen for the model integration. This number $1024 = 2^{10}$ also has the advantage that it more easily facilitates estimates of scalings for factor of two changes in node counts. Similarly a minimal numbers of 128 compute nodes was determined for a QUBICC experiment on *Juwels-Booster*and *Levante*.

An additional small number of nodes was allocated for the asynchronous parallel output scheme. The number is chosen such that the output written hourly on *Piz Daint* and two-hourly on *Juwels-Booster* and *Levante* is faster than the integration ~~of 90 or 180 time steps, respectively~~over these output intervals. As a result the execution time of the time loop of the simulation is not affected by writing output. Only writing output at the end of the time loop adds additional time.

These setups were used for the science experiments including the experiment discussed in section **??** as well as starting points for the benchmarking experiments.

6.1 Benchmarking experiments

The test experiment for benchmarking consists of precisely the configuration of dynamics, transport and physics as for the intended QUBICC experiments. Only the horizontal grid size, the number of nodes, and parameters to optimize parallelization were adjusted as needed for the benchmark tests. The length of the benchmark test is 180 time steps corresponding to 2 hours simulated time, using the same 40 s time step in all tests. ~~Hence to the extent the configuration should change for the scientific use of ICON on coarser grids ,~~

Three different grid sizes are used for benchmarking. First, for single node testing on *Piz Daint*(section **??**) the R2B4 grid is used, because this grid is 1024 times smaller than the R2B9 grid used for experiments running on 1024 *Piz Daint*nodes. Second, for the strong scaling analysis the R2B7 grid is used, which is 16 times smaller than the R2B9 grid. Accordingly the minimal number of nodes used for the strong scaling tests is 16 times smaller than for the R2B9 setup used in experiments so that this smallest setup is again comparable in memory consumption to the R2B9 setup for the experiments, and further so that at least four node doubling steps are possible within the limits of the computer allocations. The actual ranges of compute nodes n_{cn} used for the strong scaling tests for the R2B7 grid on the three computers can be seen in Table **??**. Third, for the weak scaling analysis the R2B9 grid is used so that it can be compared with the R2B7 tests with 16 times smaller number of grid points and nodes.

The allocations on *Juwels-Booster*and *Levante*allowed to run benchmark tests for the R2B9 grid also on larger node numbers so that the strong and weak scaling could also be analyzed also for higher node numbers, as tabulated in Table **??**.

Among the three grids used here, only the benchmark test on the original QUBICC grid (R2B9) has a physically meaningful configuration, while benchmark tests with smaller grid sizes are not configured for meaningful experiments. Timings reported below for benchmark tests on smaller grid sizes therefore should not be interpreted as timings for ICON experiments configured for such reduced resolutions, e.g., through the introduction of additional processes or changes in time steps~~, the timings would also change~~.

Two measures of scaling are introduced. Strong scaling S_{s} measures how much the ~~time to solution~~time-to-solution, T_{f}, is increased for a fixed configuration with $2n_{\mathrm{cn}}$ computing nodes compared to one half of the ~~time to solution~~ time-to-solution with n_{cn} nodes. Weak scaling S_{w} measures how much T_{f} increases for a two fold increase in the horizontal grid-size (with n_{gp} grid points) balanced by a two-fold increase in the node count, n_{cn}. These are calculated as

$$S_{\mathrm{s}} \equiv \frac{T_{\mathrm{f}}(n_{\mathrm{gp}}, n_{\mathrm{cn}})/2}{T_{\mathrm{f}}(n_{\mathrm{gp}}, 2n_{\mathrm{cn}})} \quad \text{and} \quad S_{\mathrm{w}} \equiv \frac{T_{\mathrm{f}}(n_{\mathrm{gp}}, n_{\mathrm{cn}})}{T_{\mathrm{f}}(2n_{\mathrm{gp}}, 2n_{\mathrm{cn}})} \approx \left(\frac{T_{\mathrm{f}}(n_{\mathrm{gp}}, n_{\mathrm{cn}})}{T_{\mathrm{f}}(16n_{\mathrm{gp}}, 16n_{\mathrm{cn}})} \right)^{1/4} \equiv (S_{\mathrm{w},16})^{1/4} \tag{1}$$

respectively. Because global grids can more readily be configured with grid resolution changing in factors of 2 and consequently n_{gp} changing in multiples of four, and to minimize the noise for the ~~very good~~ weak scaling, S_{w} is estimated through experiments with a 4-fold increase in resolution and 16-fold increases in the computational mesh and node count. ~~In the ideal case S_{s} and S_{s} would both be unity. Values less than 0.5~~ An ideal scaling would result in $S_{\mathrm{s}} = S_{\mathrm{w}} = 1$, while $S_{\mathrm{s}} < 0.5$ would indicate a detrimental effect of adding computational resources.

Values of T_{f} needed in calculating S_{s} and S_{w} are provided by the simulation log files. These time measurements, which are part of the model infrastructure, are taken for the integration within the time loop that includes the computations for all processes (dynamics, transport, radiation, cloud microphysics, vertical diffusion and land processes), as well as other operations required to combine the results from these components, to communicate between the domains of the parallelizations, to send data to the output processes, etc. But for benchmarking we are mostly interested in the performance of the time loop integration and the above-mentioned processes. The benchmarking should show that the GPU port provides a substantial speed-up on GPU compared to CPU, and it should characterize the strong and weak scaling behavior of the ICON model on the compute systems available in this study.

Two model versions have been used in the benchmarking, as listed in Table **??**. Version (1) resulted from the GPU-porting on *Piz Daint*, and version (2) from the further developments made when porting to *Juwels-Booster*. This latter version was also used on *Levante*.

Table 2. Model revisions and their usage.

#	Computer	Revision	Comments
(1)	*Piz Daint*	icon-cscs:7de52b43701a5f56b5f82c41f651a290edb3950c	480 g-points, clear-sky computation
(2)	*Juwels-Booster*, *Levante*	icon-cscs:baf28a514c0f6d8143e1f4e2ebce7fe02bec479d	240 g-points, no clear-sky computation

6.2 Optimization parameters

The computational performance of benchmark experiments can be optimized by the choice of the blocking length nproma, the radiation sub-blocking length rcc, and the communication method. As discussed already in Sect. **??**, the most important point for execution on GPU is to have all data in a single block on each MPI domain, thus including the grid points for which computations are needed (Table **??**, column n_{gpp}) as well as the halo points needed as input for horizontal operations in dynamics and transport. At the same time the GPU memory must be sufficient to store the local data given the (large) block size.

For *Piz Daint* the 1024 nodes setup for the R2B9 grid ~~then has~~ has 20480 horizontal grid points per domain and a nproma value of ca. ~~20000. Based on this the~~ 21000. The single node benchmarking ~~was made on~~ uses the R2B4 grid with ~~160 resolution that has~~ nproma$= n_{\text{gpp}} =$ 20480 ~~cells~~because no halo is needed. Strong scaling tests are based on ~~a 64-times larger grid,~~ the R2B7 grid ~~with 20 resolution,~~ using 64 to 1024 nodes. Thus the initial 64 node setup uses practically the same amount of memory per node as the small single node test, while the largest setup has a ca. 16 times smaller block size. The weak scaling tests consist of the same R2B7 setup on 64 nodes used for strong scaling and the 16 times larger R2B9 setup on 1024 nodes, which ~~is the size used for QUBICC experiments. This largest setup has therefore again a block size~~ therefore have comparable block sizes of ca. ~~20000.~~ 21000.

A second performance parameter consists in the size of the sub-blocking used for radiation, rcc, which was introduced to reduce the memory requirement of the radiation and thus to allow the usage of single blocks for all other components of the model. For setups on *Piz Daint* with nproma close to ~~20000~~ 21000, tests showed that the maximum ~~length for the sub-blocking~~ rcc is ~~800.~~ 800 (Table **??**), thus splitting a data block into 26 sub-blocks for the radiation calculation ($20480 = 25 \times 800 + 480$). In the strong scaling series, ~~where the~~ the increasing number of nodes reduces the the grid size per node ~~is 16 times smaller at 1024 nodes,~~ and nproma~~for single blocks is accordinglysmaller and thus the radiation~~ accordingly, which allows to increase rcc. Only for 512 nodes it showed that having $rcc = 1280$, which amounts to two sub-blocks of equal size, was more efficient to compute the radiation than having a higher rcc, which triggers two radiation calls with quite unequal sub-block ~~size can be increased to 1280, so that no sub-blocking is needed when reaching~~ sizes as for instance 2000 and 560. Finally, for 1024 ~~nodes.Both blocking parameters are compiled in Table ??.~~ nodes, $rcc = 1280$ covers all computational cells so that a single radiation call is sufficient, i.e. no sub-blocking takes place.

Further optimizations can be exploited in the communication. Choosing direct GPU to GPU communication instead of CPU to CPU communication (cf. Sect. ?? results in a speed-up of ca. 10% on *Piz Daint*. Unfortunately the GPU to GPU communication on *Piz Daint* caused random crashes seemingly related to the MPICH implementation, and therefore all scaling tests and experiments on *Piz Daint* use the slower CPU communication. On *Juwels-Booster* no such problems were encountered so that the GPU to GPU communication is used in all experiments.

On *Juwels-Booster* more GPU memory ~~(160 GB as compared to~~ is available compared to *Piz Daint*(160 GB vs. 16 GB per node)~~is available compared to~~. This allows scaling tests with the same R2B7 and R2B9 grids on a minimum of 8 and 128 nodes, respectively, with a blocking length `nproma` close to 42000 and a sub-blocking length ~~starting at 5120.~~ `rcc`starting at 5120, or 8 sub-blocks of equal size. This larger sub-blocking length is possible not only because of the larger GPU memory, but also because the newer ICON code that is used on *Juwels-Booster* has only half of the g-points of the gas optics in the radiation, 240 instead of 480, which reduces the memory for local arrays in radiation accordingly. On *Juwels-Booster* the strong scaling tests extend from 8 to 256 nodes, thus from 32 to 1024 GPUs. On ~~the 128 and 256 nodessetups~~ 64 and more nodes, `nproma` is ~~reduced to 2999 and 1589~~small enough to set `rcc`to the full number of computational grid points in the domain, which allows to compute the radiation scheme without sub-blocking. Further it should be pointed out that the reduction of the number of g-points constitutes a major computational optimization by itself, as this reduces the computing costs of this process by a factor 2 without physically significant effects on the overall results of the simulations.

On *Levante*, where no GPUs are used and the CPUs have a comparatively large memory, the best `nproma` is 32 for all grids and number of nodes, and no sub-blocking is necessary for the radiation~~.~~, i.e. also `rcc`is 32 for all cases. An additional optimization concerns the parallelization between MPI processes and OpenMP threads. In all tests on *Levante* we use 2 CPU/node × 16 process/CPU × 4 thread/process = 128 thread/node.

6.3 ~~Time-to-solution~~ Single node CPU-to-GPU speed-up on *Piz Daint*

6.3.1 ~~Single node GPU/CPU speed-up on~~

On *Piz Daint* the achievable speed-up of a small R2B4 model setup on a single GPU versus a single CPU was an important metric. Single node tests give a clear indication of the performance speed-up achievable on GPUs vs. CPUs without side effects from parallelization between nodes. Only a speed-up clearly larger than two would be an improvement for a node hosting one CPU and one GPU versus a node with two CPUs. To achieve this goal, the speed-up must be favorable especially for the model components which dominate the time-to-solution of the integration.

Figure ~~?? shows the ratio of the computing time on GPU~~ ?? therefore shows in panel (a) the relative costs of the model components on the GPU as percentage of the time-to-solution of the integration in the time loop, and in panel (b) the CPU-to-GPU speed-up for the integration and the model components. Concerning the relative costs it is clear that dynamics and radiation are the dominating components, each taking between 30 and ~~CPU for the major components of the test simulation. The experiments show that~~ 40 %. All other components consume less than 10 % of the integration time. Also the CPU-to-GPU speed-up varies between the components. The highest value is achieved by radiation: 7.4, and the lowest by the land scheme:

Table 3. Time-to-solution, T_f; percent of time spent on dynamical solver (Dyn); strong and weak scaling, S_s and S_w respectively; and temporal compression, τ, for experiments on *Piz Daint*, *Juwels-Booster*, and *Levante* with code version (Code) from Table **??**, grid name, number of grid points, n_gp, number of computing nodes, n_cn, number of grid points per MPI process, n_gpp, and optimization parameters, `nproma` and `rcc`. Scaling values shown in bold are used in the extrapolation for a 1 SYPD simulation at ca. 1 km resolution, see Sect. **??**. The temporal compression is only shown for the R2B9 setup, to which the chosen time-step (40 s) corresponds.

Code	Grid	n_gp	n_cn	n_gpp	`nproma`	`rcc`	T_f / s	Dyn / %	S_s	S_w	τ / SDPD
	Piz Daint 1×P100 GPU per compute node, 1 MPI process per GPU										
(1)	R2B7	1310720	64	20480	21464	800	132.8	35.5			
(1)	”	”	128	10240	10944	1200	73.43	38.9	**0.904**		
(1)	”	”	256	5120	5621	1600	49.77	38.7	**0.738**		
(1)	”	”	512	2560	2999	1280	37.18	38.9	0.669		
(1)	”	”	1024	1280	1589	1280	31.20	31.9	0.596		
(1)	R2B9	20971520	1024	20480	21706	800	147.4	33.6		**0.974**	48.85
	Juwels-Booster 4×A100 GPU per compute node, 1 MPI process per GPU										
(2)	R2B7	1310720	8	40960	42338	5120	49.58	39.5			
(2)	”	”	16	20480	21464	”	31.07	37.2	**0.798**		
(2)	”	”	32	10240	10944	10240	22.66	38.7	**0.686**		
(2)	”	”	64	5120	5621	5120	16.13	33.8	0.703		
(2)	”	”	128	2560	2999	2560	14.40	30.9	0.560		
(2)	”	”	256	1280	1589	1280	14.77	26.2	0.487		
(2)	R2B9	20971520	128	40960	42690	4096	54.13	37.5		**0.978**	133.0
(2)	”	”	256	20480	21706	5120	33.97	35.5	0.797	0.978	212.0
	Levante 2×Milan CPU per compute node, 16 MPI processes per CPU, 4 OMP threads per process										
(2)	R2B7	1310720	8	5120	32	32	484.3	46.5			
(2)	”	”	16	2560	”	”	241.2	47.5	1.004		
(2)	”	”	32	1280	”	”	118.6	48.6	1.017		
(2)	”	”	64	640	”	”	63.60	47.5	0.932		
(2)	”	”	128	320	”	”	34.46	44.7	**0.923**		
(2)	”	”	256	160	”	”	18.60	39.3	**0.926**		
(2)	”	”	512	80	”	”	11.14	34.3	0.835		
(2)	R2B9	20971520	128	5120	”	”	491.1	46.4		0.997	14.7
(2)	”	”	256	2560	”	”	249.7	46.8	0.984	0.991	28.8
(2)	”	”	512	1280	”	”	127.3	46.7	0.980	0.982	56.5
(2)	”	”	1024	640	”	”	69.45	44.7	0.917	**0.978**	103.7

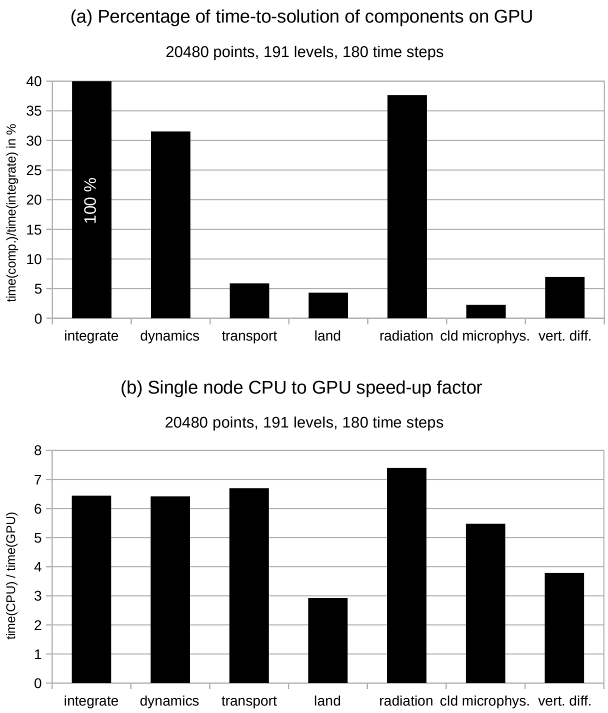

Figure 4. ~~Single node speed-up on GPU compared to CPU for~~ For a small setup with 20480 grid points and 191 levels integrated over 180 time steps~~.~~: (a) Time-to-solution on GPU of the model components as percentage of the time for the "integrate" ~~includes the full~~ timer that measures to whole time loop, ~~which includes~~ and (b) the ~~main~~ CPU-to-GPU speed-up for the whole time loop as well as the model components~~"dynamics", "transport" and "atm.physics" and "land physics". "radiation" is the most costly component of "atm. physics".~~

2.9. All together, the ~~time to solution reduces by more than a factor of six. Among the different components, radiationshows the greatest increase in throughput, which is~~ speed-up of the full integration is 6.4, as a result of the high scaling of the most expensive components and the very low costs of the components with a lower speed-up.

The high speed-up of radiation is attributed to the higher computational intensity and more time invested in optimizations as compared to other, less costly components. The land physics stands out for its poor performance, which is attributed to the very small GPU kernels, so that the launch time is often comparable to the compute time. But for the same reason (small computational cost), this has little effect on the speed-up of the full model. The ~~roughly 6.5~~6.4-fold speed-up of the code of the whole integration is considered satisfactory, given that the ICON model is bandwidth limited and the GPU bandwidth to CPU bandwidth ratio on *Piz Daint* is of approximately the same order.

6.3.1 ~~Load distribution among components~~

6.4 Scaling

~~Full~~ On each compute system the R2B7 and R2B9 setups are run for successive doublings of $n_{\rm cn}$, starting from the minimum value of $n_{\rm cn}$ ($n_{\rm cn,min}$) that satisfies the memory requirements of the model, and proceeding to the largest $n_{\rm cn}$ for which we could obtain an allocation. Blocking sizes are optimized for each value of $n_{\rm cn}$. The smaller memory requirements of R2B7 allow it to be run over a much larger range of $n_{\rm cn}$. In each case the full time-to-solution ~~$T_{\rm f}$ for a fixed period of simulated time and resolution allows quantification of the relative costs of the main components, which can also help to guide future optimization efforts.~~ $T_{\rm f}$ ~~for the time loop, i.e. for the “integrate" timer, are tabulated~~ measured for the model integration is provided in Table **??**~~, for different configurations. The time to solution~~. (The time-to-solution per grid column and time step can be calculated straightforwardly from these data as $T_{\rm i} = T_{\rm f}/n_{\rm gp}/180$.) The strong and weak scaling parameters are then calculated from $T_{\rm f}$ and Eq. (**??**). First we discuss the R2B7 benchmarks made for the strong scaling analysis, followed by the weak scaling analysis based on R2B7 and R2B9 benchmarks.

~~The~~

6.4.1 Time-to-solution and strong scaling of the model integration

The full time-to-solution $T_{\rm f}$ and the cumulative strong scaling $S_{\rm s,cum}$ of the model integration on the R2B7 grid, as measured by the "integrate" timer, are displayed in Figure **??**. The time-to-solution for the smallest setups clearly shows that the GPU machines allow to get more quickly to the solution than the CPU machine, when small node numbers are used. And as expected the more modern *Juwels-Booster*machine is faster than the older *Piz Daint*machine. The benefit of repeatedly doubling the GPU node count decreases, as visible in the flattening of the time-to-solution series for *Piz Daint*and *Juwels-Booster*. Generally only 2 doubling steps are possible if $S_{\rm s,cum}$ should be higher than 0.5. For *Juwels-Booster*, where the allocation allowed a fifth doubling step, the time-to-solution of the last step, at 256 nodes, is actually higher than for 128 nodes. For *Levante*, the time-to-solution essentially halves for each doubling of nodes, except for a small degradation building up towards the highest node counts. This makes already clear that the strong scaling of this experiment differs substantially between the GPU machines *Piz Daint*and *Juwels-Booster*, and the CPU machine *Levante*.

Indeed the scaling panel shows that the GPU machines have a cumulative strong scaling $S_{\rm s,cum}$ that decays rather quickly, almost linearly with the number of node doubling steps. Correspondingly also the single step strong scaling $S_{\rm s}$ decreases with increasing node counts, as can be seen in Table **??** for *Piz Daint*and *Juwels-Booster*. For *Levante*we find, however, that $S_{\rm s,cum}$ even slightly increases in the first two doubling steps, before showing a weak degradation, so that $S_{\rm s,cum}$ exceeds 0.6 even at the highest tested parallelization on 512 nodes with a 64 times increased node count, where only 80 grid columns are left per MPI process. This results from favorable $S_{\rm s}$ values even at high node counts, with $S_{\rm s}$ staying above 0.9 up to 256 nodes, see Table **??**.

6.4.2 Time-to-solution of components

The measurement of $T_{\rm f}$ of the model components in the strong scaling benchmarks allows quantification of the relative costs of the main components. Typically the most time-consuming components not only dominate the total time ~~and often also~~ but also often determine the scaling behavior of the model. Thus knowing the relative costs and the scaling of the components

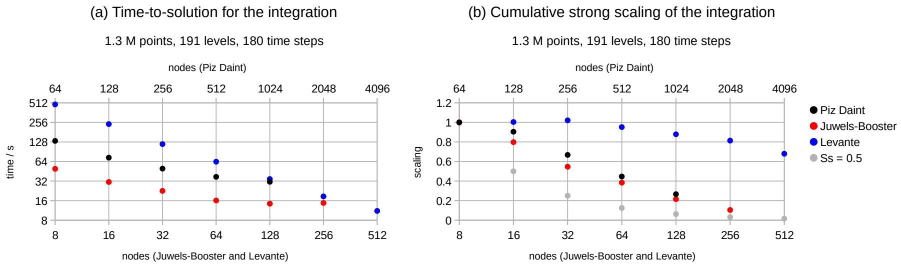

Figure 5. Time-to-solution (~~left column~~a) and cumulative strong scaling $S_{\mathrm{s,cum}}$ (~~right column~~b) ~~for~~ from the model integration on *Piz Daint* (~~top row~~black), *Juwels-Booster* (~~middle row~~red), and *Levante* (~~bottom row~~blue). The ~~time used and the cumulative strong~~ scaling ~~are shown for the full time loop (integrate) and the contributing processes:~~ panel additionally shows in ~~blue the resolved processes dynamics, transport; in red the atmospheric parameterizations~~ grey $S_{\mathrm{s,cum}}$ for ~~radiation, vertical diffusion and cloud microphysics, and in brown the land physics. The solid horizontal line shows the perfect scaling = 1 and the dashed line the strong scaling~~ $S_{\mathrm{s}} = 0.5$ for ~~the case that the time-to-solution is~~ a constant time-to-solution.

contributes to understanding the behavior of the full model and can give guidance for future improvements. For these purposes ~~the contribution to~~ T_{f} from the dynamical core~~and~~ , tracer transport, radiative transfer, turbulent mixing processes, cloud microphysics~~,~~ , and land processes ~~are~~ is displayed in the left column of Fig. ~~??~~ **??** for all three compute systems. Among these components the dynamics generally dominates, taking about 40 % of the compute time on the GPU systems, and closer to 50 % of the time on *Levante* (see Table **??** for precise percentages). The fraction of the compute time spent on the dynamics decreases for high node counts as the model stops scaling. On *Piz Daint* radiation is the second most computationally expensive component, while on *Juwels-Booster* transport is substantially more costly. This difference results from the changed setup of the radiation code used on *Juwels-Booster* and *Levante*, which includes (1) the reduction of g-points from 480 to 240 and (2) avoiding the extra computation of clear sky fluxes. In the smallest R2B7 setup on *Juwels-Booster*, the first step reduces the radiation time by 43%, and both steps together yield a reduction of 60%. On *Levante*, where also the faster radiation is used, the compute time spent in transport and radiation is almost equal.

The ranking of the less costly processes partly depends on the scaling of the components, which makes the radiation scheme relatively cheaper and the land processes relatively more expensive for higher number of nodes. ~~The~~ On the GPU machines *Piz Daint*and *Juwels-Booster*, the least amount of time is spent for cloud microphysics ~~, on all systems and~~ for all numbers of nodes~~.~~ , while on the CPU machine *Levante*the same is true for the land processes. This points again at the poor CPU-to-GPU speed-up of the land scheme and in addition also at a poor strong scaling. However, the total cost of the microphysical complexity is much larger, as in the absence of microphysics there would be no need for tracer transport, which is computed here for six tracers, and the vertical diffusion would be computed only for temperature and wind.

6.5 ~~Scaling~~

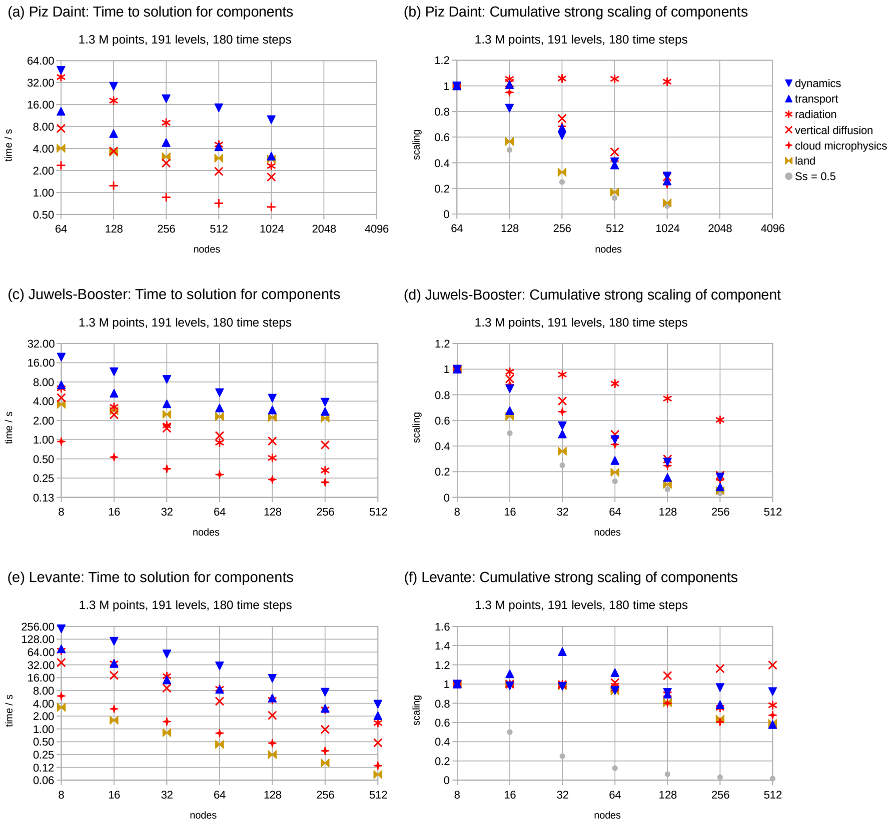

Figure 6. Time-to-solution (left column) and cumulative strong scaling $S_{\mathrm{s,cum}}$ (right column) for *Piz Daint*(top row), *Juwels-Booster*(middle row), and *Levante*(bottom row). The time used and the cumulative strong scaling are shown for the model components: in blue the resolved processes dynamics, transport; in red the atmospheric parameterizations for radiation, vertical diffusion and cloud microphysics, and in brown the land physics. The scaling panels additionally shows in grey $S_{\mathrm{s,cum}}$ for $S_{\mathrm{s}} = 0.5$ for a constant time-to-solution.

~~On each compute system the R2B7 and R2B9 setups are run for successive doublings of n_{cn}, starting from the minimum value of n_{cn} ($n_{\mathrm{cn,min}}$) that satisfies the memory requirements of the model, and proceeding to the largest n_{cn} for which we could obtain an allocation. Blocking sizes are optimized for each value of n_{cn}. The smaller memory requirements of R2B7 allow it to be run over a much larger range of n_{cn}. In each case T_{f} measured for the model integration is provided in Table ??. The strong and weak scaling parameters are then calculated from T_{f} and Eq. (??).~~

6.4.1 Strong scaling of components

~~6.5 Strong and weak scaling~~

~~Figure ?? presents the~~ The cumulative strong scaling ~~,~~ $S_{\mathrm{s,cum}}$ ~~, from the R2B7 benchmarks, which includes all successive doublings from $n_{\mathrm{cn,min}}$ to n_{cn} nodes, and the weak scaling $S_{\mathrm{w,16}}$ measured from the R2B7 and R2B9 benchmarks with a~~

~~factor 16 in node count and grid points and thus equal cell count per process. If S_s was independent of n_cn the cumulative scaling should decay exponentially in $\log_2(n_\mathrm{cn})$ with $S_\mathrm{s,cum} = S_\mathrm{s}^{(\log_2(n_\mathrm{cn}/n_\mathrm{cn,min}))}$, as shown for the special case of constant time of solution, for which $S_\mathrm{s} = 0.5$. A more linear decay of $S_\mathrm{s,cum}$ is indicative of reductions in S_s with increasing n_cn.~~

~~It is found that ICON exhibits very good weak-scaling for a 16-fold increase in node count. And for and , where $S_\mathrm{w,16}$ was evaluated for different node counts, $S_\mathrm{w,16}$ remains higher than 0.9 (and S_w above 0.97) for all node counts. The strong scaling is less impressive, particularly on the GPU machines. In general $S_\mathrm{s} < S_\mathrm{w}$ and it also decreases much more rapidly with the number of doublings of nodes.This leads to a more linear decrease in S_s versus $n_\mathrm{cn}/n_\mathrm{cn,min}$, and implies that the model will eventually yield less throughput as node counts are increased further, a limit that is reached in the last doubling (for $n_\mathrm{cn} = 256 \rightarrow 512$) with the R2B7 configuration on . On the strong scaling is much better. S_s and $S_\mathrm{s,cum}$ actually exceed the ideal case as n_cn increases from 8 to 32, and remain above 0.9 and 0.8, respectively up until $n_\mathrm{cn} = 256$. Only for $n_\mathrm{cn} = 512$, where only 80 grid columns are left per MPI process do they diminish substantially (decreasing to 0.83 and 0.68, respectively).~~

~~Cumulative strong and weak scaling, $S_\mathrm{s,cum}$ and $S_\mathrm{w,16}$ respectively, on (PD, red), (JB, blue) and (Le, black). The strong scaling is shown for 1.3 M grid points with respect to the smallest node number N_{min} (squares, PD: $N_{min} = 64$, JB and Le: $N_{min} = 8$). The dashed line show the strong scaling from N_{min} to N nodes for the case of a constant $S_\mathrm{s} = 0.5$ and thus a constant time-to-solution. The weak scaling is shown for the noted fixed number of points per process and a 16-fold increase of the grid size and number of nodes.~~

~~For the practical employment of the ICON-A model, here in the QUBICC configuration, the scaling results have the following consequences. (1) On the GPU machines the possibilities to speed-up the turnover rate along the strong scaling line is rather limited. Starting from the smallest setup only a doubling or quadrupling of the number of compute nodes would be reasonable. (2) On the CPU machine the high strong scaling allows to increase the turnover with little scaling loss as long as the number of grid columns stays roughly above 100. (3) The excellent weak scaling would allow to increase the horizontal resolution to the largest grids fitting in these systems. Thus while a 2.5 km (R2B10) QUBICC simulation would still be efficient on and on , on also a 1.25 km (R2B11) simulation would be an effective use of computational resources.~~

6.4.1 ~~Strong scaling of components~~

of the components is shown in the right column of Fig. ??. Most components show similar scaling behavior to the model as a whole (Fig. ~~??~~??b), with some noteworthy exceptions. On the GPU machines *Piz Daint* and *Juwels-Booster* the important exceptions to this rule are the land and radiation. Land shows very poor strong scaling, ~~while radiation~~ quickly approaching the cumulative strong scaling for $S_\mathrm{s} = 0.5$, shown in grey symbols, where the time-to-solution is constant for all node numbers. Radiation, however, achieves a scaling better than 1 on *Piz Daint*, and remains close to 1 on *Juwels-Booster* in the first and second doubling of nodes, only starting to decay for larger increases in n_cn.

The strong scaling on GPUs depends sensitively on the ability to maintain sufficient work for each node as the node count is increased. In the case of land, which is computationally inexpensive and spatially sparse, this is not possible. For radiation the workload can be increased through the optimization of the sub-blocking parameter, `rcc`. On *Piz Daint* this is possible up to $n_{\mathrm{cn}} = 512$, reaching the case of no sub-blocking in the last step only, on 1024 nodes, see Table **??**. On *Juwels-Booster* the initial sub-blocking size is substantially larger from the beginning, owing to the larger available memory, and thus the largest workload is reached already on 32 nodes. From this step onward no sub-blocking is needed and the work load decays with the decreasing number of grid points on the processor.

~~On the exceptions are the transport and~~ Not critical but noteworthy is the scaling of transport. Generally the scaling of transport resembles that of dynamics, as both schemes have horizontal dependencies. But for the first node doubling on *Piz Daint*and the second node doubling on *Levante*, transport shows a remarkably higher strong scaling than dynamics. Further, on *Levante* the ~~vertical diffusion, for which the~~ cumulative strong scaling ~~exceeds the ideal scaling (=~~of the vertical diffusion takes a value of 1 ~~)~~ from 8 to 64 ~~and from 64 to~~ nodes and increases for higher node counts up to 1.2 for 512 nodes~~, respectively.~~. The reasons for these behaviors have not been investigated.

6.4.1 Weak scaling

The weak scaling S_{w} derived from pairs of R2B7 and R2B9 benchmarks with a factor 16 in node count and grid points is shown in Table **??**. Generally we found a very good weak scaling, whether on GPUs or on CPUs. For *Juwels-Booster*and *Levante*, where S_{w} was evaluated for more than one pair of R2B7 and R2B9 experiments, S_{w} remains higher than 0.97 for all cases.

6.4.2 Scaling evaluation

For the practical employment of the ICON-A model, here in the QUBICC configuration, the scaling results have the following consequences. (1) On GPU machines the possibility to speed-up the turnover rate along the strong scaling line is rather limited. Starting from the smallest setup only a doubling or quadrupling of the number of compute nodes would be reasonable. (2) On the CPU machine the high strong scaling allows to increase the turnover with little scaling loss as long as the number of grid columns stays roughly above 100. (3) The excellent weak scaling would allow to increase the horizontal resolution to the largest grids fitting in the memory of these systems. Thus a resolution of 2.5 km (R2B10 grid) would be possible on *Piz Daint*and *Juwels-Booster*. And *Levante*is large enough for a resolution of 1.25 km (R2B11 grid).

6.5 Outlook for 1 simulated year per day at 1 km resolution on a global grid

6.5.1 Temporal compression of benchmarks

The temporal compression, τ, of a model setup on an available compute systems is important for determining what kind of scientific questions the model may be used for. Here it is measured as a unit-less parameter, of simulated days per day (SDPD), and only calculated for the R2B9 benchmark simulations, for which the correct physical configuration and time step are used. Achieving a full simulated year per day (1 SYPD = 365.25 SDPD), for kilometer-scale configurations, is a target for centennial scale climate simulations, and still a major challenge.

The temporal compression of the R2B9 benchmarks is shown in Fig. **??** and also tabulated in Table **??**. On *Piz Daint* the R2B9 experiment on 1024 nodes achieves a temporal compression of 48 SDPD~~, considering~~. Considering the poor strong

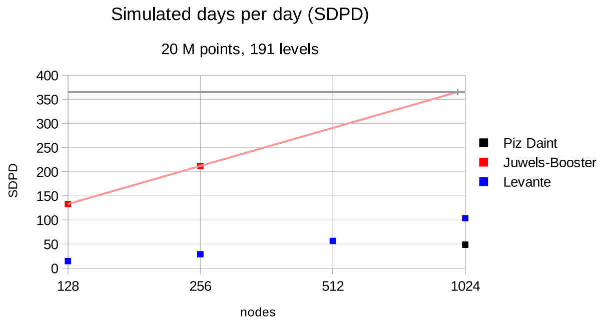

Figure 7. Simulated days per day R2B9 (20 M points) based on the "integrate" timer on *Piz Daint*~~(PD)~~, *Juwels-Booster*~~(JB)~~, and *Levante*~~(Le)~~. The black horizontal line indicates 1 simulated year per day. The thin ~~straight~~ red line is a linear fit in log(nodes) for the *Juwels-Booster* simulations. The vertical markers on the 1 sim. year per day line indicate the estimated number of nodes for the integration of a full year on *Juwels-Booster*: 984 nodes.

scaling, 1 ~~SDPD~~ SYPD is well beyond reach. On *Levante*, where the experiment has been run on 128, 256, 512 and 1024 nodes, the turnover grows from 14.7 to 103.7 SDPD. Hence, using the entire machine gets closer, but still falls about 25% short, of 1 SYPD. On *Juwels-Booster* a turnover of 133 and 212 SDPD was achieved on 128 and 256 nodes respectively. The linear extrapolation in log(n_{cn}), shown as thin red line in Fig. **??**, indicates that ca. 984 nodes could return about 1 SYPD. Thus the entire *Juwels-Booster* system that has 936 nodes would get close to 1 SYPD with the model setup for QUBICC experiments.

6.5.2 Computational demands for 1 SYPD

Based on the results above, we extrapolate to assess the computation requirements for a global simulation using an R2B11 ($1.25\,\mathrm{km}$) mesh with a temporal compression of 1 SYPD. We base our estimates on reference calculations using the QUBICC configuration, anticipating that its increased number of vertical levels would be commensurate with the target system. Further we assume that a four times smaller timestep is stable on the R2B11 grid.

Our calculations of weak-scaling allows us in a first step to estimate the performance of an R2B11 system using a 16 fold enlarged setup on an enlarged ~~(16 fold)~~ version of one of the reference compute systems (Ref → 16×Ref). Then we use the strong scaling factor for the reference system to estimate the increase in the temporal compression for a four-fold larger ~~system~~ setup (16×Ref → 64×Ref). For this strong scaling calculation we use the two S_{s} values starting from the R2B7 setup with the same number of grid points per node as used in the R2B9 setup enlarged 16-fold in the weak scaling step. S_{w} and S_{s} values used for the extrapolation are shown in bold in **??**. The parameter γ_1, measures the gap, i.e., the additional factor of temporal compression required to reach 1 SYPD (Table **??**), alternatively it can be understood as the number of days required to simulate one year for a given configuration.

Table 4. ICON R2B11 configurations and their expected turnover. Compute system with processor type, its total number of nodes and the R_{max} LINPACK benchmark, and ICON code version, ICON grid, horizontal resolution (Δx), time step, number of nodes, fraction of nodes with respect to the total number of nodes, temporal compression τ (in SDPD), the gap factor, γ_1 for 1 simulated year per day, defined as $\gamma_1 = 365.25/\tau$, and the required computational power $P_1 = R{\cdot}\gamma_1$, where $R = n_{\mathrm{cn}}/n_{\mathrm{cn,tot}} \cdot R_{\mathrm{max}}$, and given in units of EFlop/s required to simulate one year per day of the indicated model on the indicated configuration of the machine, assuming ideal strong scaling for a speed-up of γ_1. Because we use the Linpack references for R_{max}, the $n_{\mathrm{cn,tot}}$ for Levante is not its present node-count but the number used in the Nov 2021 benchmarks.

System	Grid	Δx / km	Δt / s	n_{cn}	$n_{\mathrm{cn}}/n_{\mathrm{cn,tot}}$	τ / SDPD	γ_1	P_1 / EFlop/s
Piz Daint, 1×P100 GPU per node, $n_{\mathrm{cn,tot}} = 5704$, $R_{\mathrm{max}} = 21.2\,\mathrm{PFlop/s}$, code-base (1)								
Ref	R2B9	5.00	40	1024	0.18	48.85	7.48	0.028
16×Ref	R2B11	1.25	10	16384	2.87	11.01	33.2	2.024
64×Ref	”	”	”	65536	11.5	29.37	12.4	3.033
Juwels-Booster, 4×A100 GPU per node, $n_{\mathrm{cn,tot}} = 936$, $R_{\mathrm{max}} = 44.1\,\mathrm{PFlop/s}$, code-base (2)								
Ref	R2B9	5.00	40	128	0.14	133.0	2.75	0.017
16×Ref	R2B11	1.25	10	2048	2.19	30.46	12.0	1.158
64×Ref	”	”	”	8192	8.75	66.66	5.48	2.116
HLRE5 4×NG-100 GPU per node, code-base (2)								
	R2B11	1.25	10	2048		131.0	2.79	
	”	”	”	8192		286.6	1.27	
Levante 2×AMD EPYC Milan per node, $n_{\mathrm{cn,tot}} = 2048$, $R_{\mathrm{max}} = 7.0\,\mathrm{PFlop/s}$, code-base (2)								
Ref	R2B9	5.00	40	1024	0.5	103.7	3.52	0.0123
16×Ref	R2B11	1.25	10	16384	8	23.73	15.4	0.862
64×Ref	”	”	”	65536	32	81.15	4.50	1.008

It shows that for *Piz Daint*, where our reference system uses only 1024 of its 5704 nodes, a system roughly 12 fold larger ~~system~~ than *Piz Daint* would still fall a factor 12 short of the desired compression. ~~Thus the required system, based on this technology would have to be 142 times the size of for the targeted ICON simulation~~This factor is too large to be achieved by further strong scaling. We get closer with the A100 chip-set, as a system roughly nine times larger than *Juwels-Booster* ($n_{\mathrm{cn,tot}} = 936$) leads to less than a factor of six shortfall in temporal compression. ~~Based on this technology a system 48 times the size of~~ Also this factor is out of reach given the strong scaling found on *Juwels-Booster*~~is required~~. The system gets closer not just by being bigger, but also because of the better usage of the compute power, characterized here by the LINPACK R_{max}. For the Ref and 16×Ref setups the minimum required compute power, as measured by P_1, is 1.7× reduced on *Juwels-Booster* compared to *Piz Daint*. Even if the radiation costs on *Piz Daint* were reduced by 60% due to less g-points and computing no clear sky radiation (and assuming the scaling remains unchanged), P_1 on *Juwels-Booster* would still be 1.5× reduced. This higher efficiency is only partly explained by the 30% increase in the counter-gradient versus LINPACK performance of *Juwels-Booster* versus *Piz Daint*.

For the CPU chip-set of *Levante* it is found that the 64×Ref setup would fall a factor 4.5 short of the targeted compression, and the required system would be 104 times the size of *Levante*. The estimate for 16×Ref on *Levante* may be compared to that of **?**, which was based on similar ICON simulations ($\Delta x = 5\,\mathrm{km}$, 90 levels, $\Delta t = 45\,\mathrm{s}$), albeit with a different implementation of the physical processes. Their benchmarks were performed on the *Mistral* computer at DKRZ (now being replaced by *Levante*), using the partition with two Broadwell CPUs per node. Scaling of their performance of 26 SDPD on 256 *Mistral*-Broadwell nodes (their Fig. 4), and assuming an $R_{\max}$ of 1.8715 $\mathrm{PFlop/s}$ for the 1714 node Broadwell partition[1], yields an estimate[2] of $P_1 = 0.655\ \mathrm{EFlop/s}$. This is somewhat better than what is realized on *Levante*, a difference that might be related to an initial, and hence sub-optimal, *Levante* implementation, as well as slight differences in the configuration of physical processes, for instance the treatment of radiative transfer. However, it seems reasonable to conclude that we are not seeing a large reduction in P_1 in transitioning from the Broadwell based *Mistral* machine to the Milan based *Levante*. This stands in contrast to the reduction in P_1 in transitioning from the P100 *Piz Daint* to the A100 *Juwels-Booster*, and while the P_1 values for both GPU machines remain higher than for the CPU machines, the trend is more favorable for the GPU machines, something consistent with changes in memory bandwidth[3] for the different architectures.

Strong scaling limits how much we could translate a larger machine into a reduction in γ_1. For example, from Table **??** the envisioned 16×Ref version of *Juwels-Booster* is 2.19 times larger than the existing machine, and thus would correspond to an $R_{\max} = 96.5\ \ \mathrm{PFlop/s}$, which is still far from the required $R_{\max} = 1.158\ \mathrm{EFlop/s}$ to achieve 1 SYPD. Were one to simply increase the size of *Juwels-Booster*, poor strong scaling would create the need for a proportionally larger machine, something that is measured by the increase in P_1 from 1.158 $\mathrm{EFlop/s}$ for the 16×Ref implementation to 2.116 $\mathrm{EFlop/s}$ for 64×Ref (Table **??**). Using these numbers we see that 1 SYPD at R2B11 is likely not attainable with the present implementation of ICON on existing GPU architectures. The present situation is somewhat more favorable on the CPU architectures. The currently second most performant computer, *Fugaku*, with ($R_{\max} = 442$ (**?**), would have $\gamma_1 = 2.3$, if ICON operated on *Fugaku* at the same P_1 value as for the 64×Ref setup based on *Levante*. A factor 2.3 larger CPU machine with ($R_{\max} = 1017$)seems technically within reach.

The situation for the GPU machines becomes more favorable when we look toward the future. Realizing a factor of 4.3×A100 in transitioning to a next generation GPU (NG-100 in Table **??**), as found in the ICON-A benchmarks in the transition from the P100 to the A100 GPU, would imply $\gamma_1 = 2.78$ for a 2048 node 4×NG-100 system rated at $R_{\max} = 415$

[1] Top500 for Nov. 2015 reported $R_{\max}$ = 1.1392 $\mathrm{PFlop/s}$ for 1556 Haswell nodes, and Top500 for Nov. 2016 reported $R_{\max}$ = 3.0107 $\mathrm{PFlop/s}$ for 1557 Haswell nodes + 1714 Broadwell nodes. The difference, attributed here to 1714 Broadwell nodes, is 1.8715 $\mathrm{PFlop/s}$

[2] Here we assume perfect weak scaling, starting from the 26 SDPD for R2B9 on 256 nodes, which yields a ~~$\gamma_1 = \frac{365.25}{26/4}\left(\frac{45}{40}\right)$~~ $\gamma_1 = \frac{365.25}{26/4}\left(\frac{45}{40}\right) = 63.2$ for R2B11, with the latter (45/40) factor accounting for differences in time steps. The weak-scaling to R2B11 (using the same $S_{\mathrm{w}} = 0.978$ as for *Levante*) inflates the size to $16\frac{191}{90} \times 256/(0.978)^4 = 9492$ nodes

[3] Memory bandwidth per core decreased by approximately 25% on *Levante* relative to *Mistral* but increased by 10% on *Juwels-Booster* as compared to *Piz Daint*.

$\mathrm{PFlop/s}$. For such a system, even without improvements in strong scaling, the $64\times$Ref benchmark on *Juwels-Booster* implies a $\gamma_1 = 1.3$ and an $R_{\mathrm{max}} = 1660\ \mathrm{PFlop/s}$. This indicates that for the GPU architectures, performance improvement of $4.3\times$ over the A100 would begin to out perform the CPU performance for the same R_{max}.

The recently announced Nvidia Hopper GPU, promises a performance increase in this range and perhaps even larger (**?**). In addition, *Fugaku*, an exceptionally efficient CPU machine, still uses twice as much electrical power as *Juwels-Booster* when normalized by R_{max}, which further favors GPU based implementations of ICON in the future. The upshot of these calculations is that the goal of 1 SYPD at roughly a 1 km scale is well within reach.

6.5.3 Anticipated increases in τ from hardware

General circulation models typically processes many grid points with often relatively little compute load, which results in the bandwidth limitation encountered in the single node speed-up tests. This means that improved memory bandwidth allow for a better exploitation of the compute power of the GPUs, which helps explain the considerably improved performance of ICON-A on the A100 versus the P100 GPUs

Another typical characteristics is the organization of the work, which happens in many separated loops often with not very many operations. On the GPUs this results in a non-negligible amount of time spent for the preparation of the parallel regions. If this amount remains constant while the computation decreases with increasing parallelization, then the benefit of stronger parallelization will be limited. Similarly, if a newer system has increased compute speed but still spends the same time for the overhead for GPU kernels, then ICON cannot profit that much. Thus, in the absence of refactoring, to realize the benefits of an acceleration of the GPU compute speed would require a commensurate speedup of the overhead time for the GPU parallel regions.

6.5.4 Anticipated increases in τ from software

Because the scaling and computational throughput are limited by the dynamical core, algorithmic improvements to this component of the code, followed by the transport scheme, stand the best chance of increasing τ. This part of the model is, however, already relatively well optimized – given the limits of what can be accomplished with openACC. Further improvements in performance would require a refactoring to exploit special features of the processors. For instance, exposing solvers to the application of AI arithmetic, i.e., matrix-multiply and accumulates, where possible, could substantially improve throughput. Obviously the very poor scaling of the land scheme is also a matter of concern, though it is unclear how the underlying problems - little computational work - can be resolved.

Another, and perhaps the best, possibility to speed up the code concerns the precision of the variables and computations. ICON uses by default 64-bit variables and arithmetic, although the ICON model can be used in a mixed precision mode, in which mostly the dynamics is computed with 32-bit arithmetic, while the remaining model components still work with 64-bit arithmetic. Because this mixed mode has not been validated on GPUs, the mixed mode is not used in this study, neither on GPU nor on CPU, although on the latter ICON applications frequently make use of this option. Hence we see considerable potential to speed up simulations by using 32-bit variables and arithmetic. Even as few as 16 bits can be sufficient if round-off errors are kept under control, as shown in **?** for a shallow water model. This would not only speed up the simulations, but also reduce the memory footprint so that a certain turnover can be achieved with significantly fewer nodes. However, more model development would be needed to explore the potential compute time or memory savings, and the effects on the simulations.

An open question is if a different parallelization would improve the turnover for a given number of nodes. The current parallelization is organized as a single geographical domain decomposition used for all model processes (and a separate decomposition for the output scheme). Thus all model processes in a domain are computed in a specified sequence by the same processor. The practical sequence of the computations of processes is determined by the order of the processes in the coupling scheme of the model. This method balances the total work to be done in each domain relatively well. But the single processes can be quite different in their work load and as seen above this results in a quite uneven strong scaling behavior on GPUs. Would this be better in a process-specific parallelization, in which each process has its own domain decomposition? Such a parallelization would focus the resources on the more expensive components, and it would avoid higher parallelization of processes where the scaling limit is reached. It would also require a more complex communication scheme, and an ability to compute different processes in parallel, the latter can make it difficult to maintain physical limits in tendencies, for instance to maintain positivity of tracers. While these ideas have not yet been developed or tested in ICON, they could provide a substantial speed-up in the future.

7 First QBO experiments

Finally it is important to verify the utility of the new model code for the QUBICC experiments, for which a small selection of results is presented here relating to two experiments performed on *Piz Daint*. (A more detailed analysis will be published elsewhere.) The main questions to be addressed in the beginning are the following:

1. Is the model stable over at least 1 month?
2. Are there obvious biases which need to be corrected before scientific experiments can begin?
3. Can the model simulate a reasonable tropical precipitation and the downward propagation of the QBO jet as a result of wave-meanflow interaction?

The initial series of experiments ~~(*hc experiments*)~~ was integrated over 1 month starting from four initial dates (1.4.2004, 1.11.2004, 1.4.2005, 1.11.2005), which were selected based on the protocol for the QBOi experiments (**?**). These dates spread across a fairly normal QBO cycle, that is used here as well as in the QBOi project (**?**). A second series of experiments (*dy experiments*) was made to investigate the issues identified in the ~~hc experiments~~first one.

A key learning from the ~~(*hc experiments*)~~ first series of experiments was that all experiments remained stable over 1 month. Therefore, the first experiment with start date 1.4.2004 was continued until it crashed after 2 months and 6 days with too high wind speed at the model top. The analysis showed that not only this experiment, but also the other three ~~(*hc experiments*)~~ experiments of this series had a tendency towards a nearly vertical axis of the polar night jet with a very strong wind maximum at the top of the model was found. A vertical jet axis with a wind maximum at ca. $80\,\mathrm{km}$ height is in disagreement with

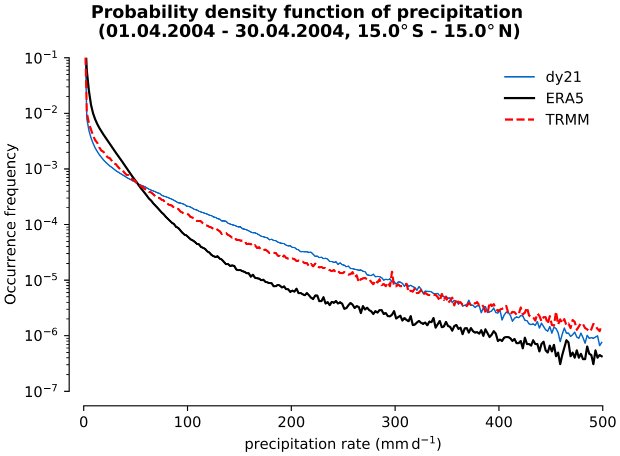

Figure 8. Occurance frequency of the 3h mean precipitation rate between 15° S and 15° N from 1.4.2004 to 30.4.2004 in the QUBICC simulation dy21, in ERA5, and in TRMM, using 250 bins of 2 mm/day width.

observations, and the high wind speeds pose a threat for numerical stability. This issue was addressed in a number of short experiments of the second series. These experiments showed that the Rayleigh damping of the vertical wind in the uppermost ca. 30 km of the model domain was too strong. An increase of the start level of the Rayleigh damping from 42 to 50 km and a reduction of the strength to 20% of the original value lead to a more realistic wind maximum of the polar night jet at heights of 50 to 60 km.

Another important finding from the first series of experiments was that the parameterized vertical diffusion was clearly too strong. This not only slowly damped the QBO jet~~instead of simulating a downward propagation~~, but also affected many aspects of the tropospheric circulation including the distribution and intensity of precipitation and convection. In the investigation of this problem it was found that the maximum mixing length within the vertical diffusion scheme was implemented in the code at a much larger value, 1000 m, than described in the original description, 150 m (Pithan et al., 2015). The too high value was reset in the experiment dy21, which ultimately reduced many large biases. Figure **??** shows the occurrence frequency for equatorial 3-hourly precipitation in the dy21 experiment, now fitting reasonably well to TRMM data (**?**). In comparison ERA5 (**?**) has more frequent weak precipitation and less frequent strong precipitation.

In contrast to the ~~he~~ first series of experiments, the dy21 experiment also shows a downward propagation of the zonal mean zonal wind in the equatorial stratosphere, thus a downward progression of the westerly and easterly QBO jets, which are initially centered at ca. 30 and 40 km height, respectively (Fig. **??**). However, in comparison to ERA5, the downward propagation of the easterly jet in dy21 is clearly faster, which results in growing differences in the zonal mean zonal wind in the upper half of this region. Nevertheless, the key result is that a downward propagation of the QBO jets is simulated.

The processes which cause this downward propagation of the QBO jets include the interaction of vertically propagating waves in equatorial latitudes with these jets and their vertical advection. The contribution of these processes to the tendency

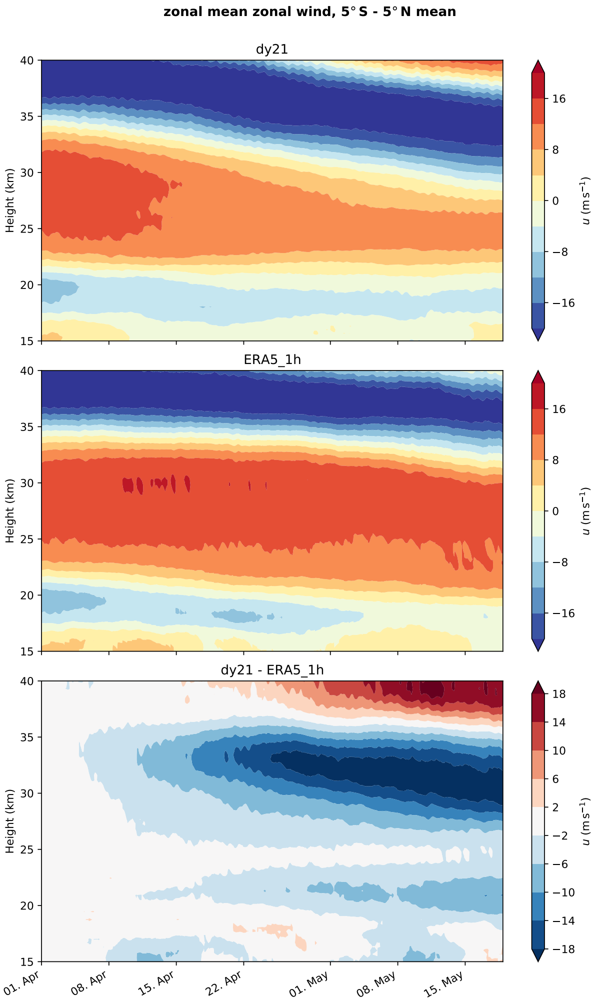

Figure 9. Zonal mean zonal wind $\overline{u}$ averaged from 5° S to 5° N from 1.1.2004 to 20.5.2004 in the simulation dy21 (top) and ERA5 (middle), and the difference between dy21 and ERA5.

of the zonal mean zonal wind $\overline{u}$ can be estimated from the divergence of the Eliassen-Palm flux (EP-flux) and the residual circulation (v^{*}, w^{*}) in the meridional plain (see for example **?**). For this diagnostics the simulation data as well as the ERA5 data were first interpolated to a Gaussian grid with 1024 latitudes × 512 latitudes. Figure **??** shows the tendency of $\overline{u}$ and the contributions from the EP-flux divergences and the advection terms averaged over the first month.

Over this first month the profiles of the zonal mean zonal wind ($\overline{u}$) remain quite similar, except for the easterly jet centered at 40 km altitude that extends already further down in dy21. The total tendency ($d\overline{u}/\mathrm{d}t$) profile is also comparable up to 27 km, but diverges above with a large negative tendency in dy21 maximizing at 33 km height, where the easterly jet has started its decent, while ERA5 shows a weaker negative tendency centered higher at 36 km height.

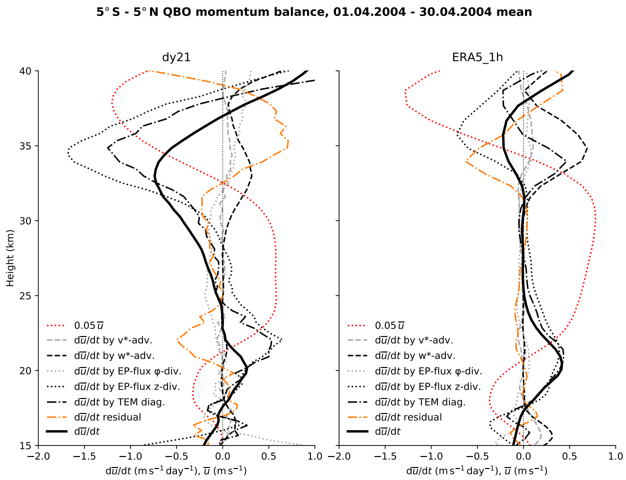

Figure 10. Zonal mean zonal wind (dotted red line) and its tendencies averaged between 5° S and 5° N and between 1.4.2004 and 30.4.2004 from 15 to 40 km height in the simulation dy21 and in ERA5, with the total tendency (bold line) and contributions diagnosed in the transformed Eulerian mean framework for advection by the residual meridional v^* and vertical wind w^* (dashed lines), the meridional and vertical divergence of the Eliassen-Palm flux (dotted lines), the sum of these four terms (dot dashed line) and the residual (orange dot dashed line).

Below 30 km altitude the total tendency $d\overline{u}/\mathrm{d}t$ in ERA5 is mostly explained by the vertical divergence of the EP-flux, ($d\overline{u}/\mathrm{d}t_{\mathrm{EP},z}$), which is almost identical to the total contribution by the EP-flux divergence and the advection terms ($d\overline{u}/\mathrm{d}t_{\mathrm{TEM}}$). These tendencies appear in similar shape in dy21. Thus for the wave meanflow interaction and the advection, as captured by this diagnostics, the simulation is close to the reanalysis. A difference exist however in the residual term ($d\overline{u}/\mathrm{d}t_{\mathrm{res}}$), which at these altitudes is negligible in ERA5 but significant in dy21, opposing the diagnosed vertical divergence of the EP-flux. This residual term is the main reason for the difference in the total tendency between dy21 and ERA5.

Above 30 km altitude, where the easterly jet has propagated further downward in dy21 than in ERA5, $d\overline{u}/\mathrm{d}t_{\mathrm{EP},z}$ is negative and peaks in the lower shear layer of the easterly jet, with a stronger amplitude in the simulation. The upward vertical wind, which creates a positive tendency $d\overline{u}/\mathrm{d}t_{w^*}$ in the same shear layer, however, is stronger in ERA5 than in dy21. Stronger differences exist however in the residuals, which play here a role also in ERA5. The residual tendencies make a substantial contribution to dy21 and ERA5, albeit with a vertical downward shift in the simulation. The meridional EP-flux divergence as well as the meridional advection play only minor roles.

The initial sets of experiments thus led to a model version in which the wave meanflow interaction and the advection by the residual meridional circulation play an important role. The nature of the residual terms is not yet known. But these simulations build the base for further research on the factors that influence the processes of the QBO. Eventually, with sufficient resources, this will also allow the simulation of full QBO cycles.

8 Conclusions

With the scientific motivation to conduct a first direct simulation of the QBO relying only on explicitly resolved convection and gravity waves, the ICON atmosphere model has been ported to GPUs with all components needed for such a simulation at a horizontal resolution of 5 km and with 191 levels up to a height of 83 km. The initial GPU port of ICON on *Piz Daint* at CSCS is based on OpenACC directives. Benchmark experiments showed a single node CPU-to-GPU speed-up of ~~ca. 6~~ 6.4 corresponding to the ratio of the GPU bandwidth to the CPU bandwidth. This memory bandwidth limitation of the ICON code is a typical characteristic for general circulation models. The strong scaling tests showed that a minimum of ca. 10k grid columns is needed on the GPU to remain efficient, which limits the possibilities to profit from strong scaling. On CPUs the limit is near 100 grid columns, which increases the strong scaling to larger processor counts. The weak scaling of ICON-A is very good (typically 0.98) over the tested 16-fold increase in grid size and node count, on both GPU and CPU architectures, making even higher resolved global simulations possible, albeit with the throughput limited by the strong scaling and the required reduction in the model timestep.

For the model setup used in the QBO simulations, a turnover of 48 SDPD and 133 SDPD was achieved on the GPU systems *Piz Daint* at CSCS and *Juwels-Booster* at FZJ, respectively, while 103 SDPD were achieved on the CPU system *Levante* at DKRZ. Extrapolations show that ICON simulations at 1.25 km resolution and 1 SYPD turnover will be possible on the next generation of supercomputers.

The GPU port of ICON-A made the first series of experiments related to the QBO processes possible. These experiments led to a better tuning of the damping and diffusion schemes, which in the end allowed a first simulation showing downward propagating QBO jets driven by wave-meanflow interaction in a model where the tropical wave-spectrum depends entirely on explicitly simulated convection. However, further research is needed to understand why the downward propagation of the easterly jet was too fast. As in the case of the QBO also other scientific problems in climate research which depend on scales from a few km or smaller to the global scale will need enormous computational resources. Having now a code that can be used on the largest supercomputers using GPUs will open up new opportunities in this direction.

Code availability. The codes (1) and (2) listed in Table **??** and the run scripts for *Piz Daint*, *Juwels-Booster* and *Levante* are available in the primary data set (**?**). These code versions are not standard release versions, but the related GPU developments are merged in the release candidate for the upcoming release version icon-2.6.5. Release versions of the code are available to individuals under licenses (https://mpimet.mpg.de/en/science/modeling-with-icon/code-availability). By downloading the ICON source code, the user accepts the license agreement.

Author contributions. Marco Giorgetta guided the presented work from the experimental side, made the benchmark and QUBICC simulations on *Piz Daint* and *Juwels-Booster* and led the writing of the manuscript as a whole. Will Sawyer, Dmitry Alexeev, Robert Pincus, and Reiner Schnur wrote parts of the manuscript. Marco Giorgetta and Henning Franke worked on the evaluation of the QUBICC experiment.

Will Sawyer developed the GPU port of most parts of the dynamical core and the transport scheme, with additional contributions from Daniel Reinert. Xavier Lapillonne and Philippe Marti ported most of the atmospheric physics parameterization. Monika Esch implemented and ported the "graupel" cloud microphysics . Valentin Clément and Reiner Schnur developed the GPU port of the land physics and the CLAW tool needed for this purpose. Robert Pincus and Sebastian Rast contributed with the RTE+RRTMGP radiation scheme and its interfaces to ICON, while Matthew Norman, Ben Hillman, and Walter Hannah contributed to the initial developement of the RTE+RRTMGP GPU implmentation. Dmitry Alexeev refined and optimized the GPU enabled ICON code especially with regard to the communication and specifics of the compilers. He also optimized the GPU port for *Juwels-Booster*. Remo Dietlicher developed the tolerance test procedure. Luis Kornblueh, Uwe Schulzweida, Jan Frederik and Panos Adamidis contributed to the I/O scheme, and Luis Kornblueh prepared the initial and external parameter files for the experiments. Claudia Frauen made the benchmark runs on the *Levante* computer at DKRZ. Bjorn Stevens initiated the PASC ENIAC project for the GPU port, contributed to the analysis of Sect. **??**, and to the writing of the manuscript as a whole.

Competing interests. The authors have no competing interests.

Acknowledgements. P. Marti, and V. Clément received funding from the "Enabling the ICON model on heterogeneous architectures" (ENIAC) project funded by the Platform for Advanced Scientific Computing (PASC) of ETH (reference number 2017-8). We acknowledge PRACE for awarding us access to *Piz Daint* based in Switzerland at the Swiss National Supercomputing Centre and *Juwels-Booster* based in Germany at the Forschungszentrum Jülich, under allocation no. 2019215178 for the project "Quasi-biennial oscillation in a changing climate". Thomas Schulthess at CSCS generously supported the development of the GPU port of ICON with staff and additional computational resources on *Piz Daint* during, especially during the transition from *Piz Daint* to *Juwels-Booster*. Some portions of the work on RTE+RRTMGP were funded by the US Department of Energy (grant number DE-SC0021262) and by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We are grateful to Italo Epicoco and an anonymous reviewer for their comments in EGUspheres, and to Leonidas Linardakis for the internal review at MPI-M.