The ECMWF OpenIFS atmospheric model in a Docker container

Marcus O. Köhler[1], Theofilos Ioannis Manitaras[2]

[1] European Centre for Medium-Range Weather Forecasts, Reading, United Kingdom
[2] Swiss National Supercomputing Centre, Lugano, Switzerland

We describe the motivation and step-by-step approach in containerizing the OpenIFS global 3D atmospheric forecasting model, developed by ECMWF. The model is used for training and research and the availability of a containerized version will enhance its portability. The use of MPI and OpenMP posed a challenge that needed resolving for porting to the HPC system. Finally, we tested the installation through a forecasting experiment on a Linux workstation and on a Cray XC50 system to evaluate scalability.

Overview of the OpenIFS Model

The OpenIFS atmospheric model has been developed at the European Centre for Medium-Range Weather Forecasts (ECWMF) since 2011. The aim is to provide the scientific community in academia and at research institutions with a version of the ECMWF's integrated weather forecasting system (IFS) which can locally be used for teaching and research on numerical weather prediction or closely related topics. OpenIFS consists of the atmospheric forecasting model used in the IFS and therefore provides the same forecasting capabilities, however without the data assimilation system and the ocean model. The OpenIFS version used for implementation in a container is based on IFS cycle CY43R3. The model can be used on a wide variety of hardware platforms, ranging from laptops and small Linux workstations to large supercomputers at national computing facilities.

The OpenIFS model code requires specific settings for its computing environment. This can present a challenge during workshops and training events, specifically away from ECMWF, when it can be time consuming to setup the model or when the software requirements may not be available. Creating a containerized version of the model code will greatly enhance its portability. This would make the provision of training less dependent on locally available computer hardware. It further will result in a highly portable and scalable binary executable version of the model code which will open new possibilities.

Containerization Approach

Requirements and Starting Point

The development of OpenIFS has been kept intentionally flexible in order to enable its use on a wide variety of hardware platforms. The model is supported on a variety of Linux systems using compilers, such as gfortran, ifort and Cray-cce. Based on the code stability experienced with the gnu compilers this build environment was chosen for this project.

Beyond gcc and gfortran the model requires Perl for the FCM configuration manager, MPI libraries (e.g. OpenMPI or MPICH), LAPACK and BLAS libraries, also Python and netCDF libraries are recommended. Further the model requires the ECMWF ecCodes library in order

to enable the processing of the Grib data format. ecCodes is available from ECMWF and requires CMake, optionally also JPEG, PNG and Jasper libraries.

Description of test cases

The standard OpenIFS installation comes with a short test case, using a T21 low resolution grid, which conducts a forecast experiment over 3 days. The test can be run with multiple OpenMP threads, with several MPI tasks and in mixed OpenMP/MPI mode. In a further acceptance test the model calculates internal statistical norms which are compared against reference data and verifies that the model produces numerical answers within acceptable limits.

In addition, a further six-day deterministic forecast experiment was prepared which simulates a storm event in the North Atlantic (NAWDEX: Tropical Cyclone Karl) which had low predictability and was associated with forecasting errors. This case study was used during the 5th OpenIFS Workshop in 2019 and hence has previously been extensively studied with model output available from numerous earlier experiments. This experiment makes also use of the wave model and the experiment setup and input data are consistent with a typical use of OpenIFS for a training event.

Steps porting the code to a Docker container

- The initial development took place on a Linux PC (SuSE distribution). We selected an Ubuntu-based Docker image and the first challenge was presented through the requirement to define web proxies in order to be able to access the Internet from within the container. This was done through providing environment variables in the build process.
- In a second step we installed the build environment, gnu compilers, and the required libraries by using the Ubuntu package manager. For the MPI libraries we initially opted for the OpenMPI distribution.
- In a third step we downloaded the ecCodes library directly from the ECMWF repository (using wget) and compiled from sources with Fortran, Python and netCDF support enabled. We did not provide a destination prefix in the hope that the default location would help avoiding potential problems at a later stage. We opted for a removal of the sources from the container.
- In the next layer the OpenIFS 43r3 source code was provided through a tar-ball. OpenIFS requires several environment variables which had to be adjusted for the compilation process. After a complete build process, we successfully carried out the T21 parallel and acceptance tests.
- Subsequently through a refactoring of the Dockerfile we reduced the number of total layers (by grouping items).
- The mpirun launcher required the option `--allow-run-as-root` as the default container user has root permissions. This was considered a potential risk and therefore a new user (named “oifs_user”) was created. This step however required assigning rw permissions to the OpenIFS source tree as the user.

In order to test the scalability of the installation the docker image was ported to the Piz Daint Cray XC50 HPC system. The Sarus software, designed to run Linux containers on HPC systems, can import docker images. Piz Daint uses the Slurm batch job scheduler and Cray-MPICH.

- The existing OpenIFS scripts made use of the aprun or mpirun launchers which are not compatible with Slurm on Piz Daint. Therefore, a different method had to be devised to use the model with MPI in a container.
- Sarus can detect the use of the MPICH library in containerized software and will instead make use of the corresponding MPICH library installed on Piz Daint. This results in the srun job launcher to spawn several container instances (as per number of tasks) which communicate with the abi compatible MPICH available on Piz Daint.
- In order to use this feature, it is necessary to replace OpenMPI in the Docker image with the MPICH 3.1.4 library which is ABI compatible with the version on Piz Daint. The library was compiled from sources and installed in its default location so that it can be found by the Sarus software.

After the T21 acceptance test was successful on Piz Daint we moved to the second test case.

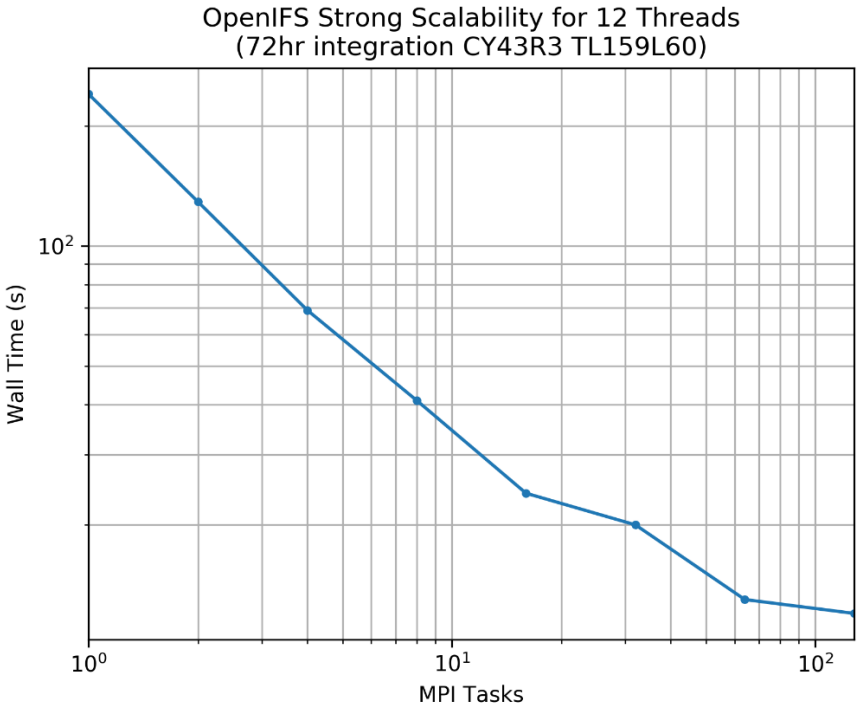
- The experiment directory for the TC Karl case study contained initial data for the atmospheric and wave model, climatological boundary conditions, namelist files and job & service scripts. In order to maintain portability linked directory structures had to be copied into subdirectories. The entire experiment directory was moved to Piz Daint and remained outside the container.
- The `oifs_run` script had to be shortened and customised for use in a containerized model installation. This affected especially how the model executable was called.
- The experiment directory was mounted with rw permissions into the container at run time to provide access to the input data and to allow the model to generate Grib output. The experiment is started by calling the script `oifs_run` from the experiment directory. The srun launcher therein submits the batch job.
- The TC Karl experiment completed successfully on the Workstation and on Piz Daint.

Scalability testing

Scalability was tested with several configurations, using up to 128 nodes, 1536 MPI tasks, and 12 OpenMP threads (no hyper-threading). The table and graph below indicate the speed up in wall clock time. Experiment #7 and #9 show that there is no time difference if only MPI tasks are used compared with mixed MPI/OpenMP mode. This is no longer the case for experiments #8 and #10 where the MPI configuration was slower than the mixed mode.

Performance table: *72 hr integration with OpenIFS 43r3 (T159L60) on Piz Daint*

#	Nodes	Tasks	Threads	Wall time (sec)
1	1	1	12	240
2	2	2	12	129
3	4	4	12	69
4	8	8	12	41
5	16	16	12	24
6	32	32	12	20
7	64	64	12	13
8	128	128	12	12
9	64	768	1	13
10	128	1536	1	16



Conclusion

This document summarises our work during a 3-day workshop at the Swiss National Supercomputing Centre in Lugano, December 2019. Therein, several working groups, each supported by a mentor, worked on the task to containerize their environmental modelling codes. The approach to provide high-level guidance through an agenda, while in practice allowing each group to determine their own model-specific goals and deliverables, resulted not only in a very focused and goal orientated work effort, but also kept the motivation going during the more difficult coding challenges. The possibility to work always with an experienced mentor made the entire learning and developing process practice-oriented and very efficient. As a suggestion to further improvement of this workshop I would recommend providing the attendees before the workshop with a collection of web links for tutorials or documentation which would help with the (optional) preparation for those with no experience in containers.