# D3.4 Performance Results and Conclusions

Nabeeh Jumah Hisashi Yashiro Julian Kunkel

Workpackage: WP3 Evaluation
Responsible institution: Yashiro, Maruyama, Ludwig
Contributing institutions: RIKEN, IPSL
Date of submission: August 2019

## Contents

**Abstract**

In the project AIMES we investigate techniques to improve the programming of (icosahedral) models, and to optimize the use of the storage and I/O resources. Such research activities are done under the workpackages WP1 and WP2. Benchmarking mini-IGCMs have been developed and presented in a previous report under this workpackage (MP3). To evaluate the applicability and the impact of the investigated techniques to real global (icosahedral) models, we present in this report projected results based on applying the developed techniques to the mini-IGCMs.

Both, model programming techniques and storage optimization, are discussed in this report. Results show that the techniques are beneficial in terms of code quality, development costs, performance, performance portability, and optimal use of I/O and storage resources.

# 1 Introduction

The report contains the achieved performance results of the created solutions, observations and discussions. The following text is the description of the project proposal for this task and deliverable:

- *Packaging of Icosahedral Global Climate Models (mini-IGCMs)*
  *In this task, we develop the mini-apps by stripping non-essential components from full codes. Compared to the full model their code base is reduced, as we restrict the models scope and, thus, the range of experiments that can be conducted with the mini-IGCM. We build one common package containing all different model flavors. The package of mini-app contains settings for one or two scientific experiments, required input data, and documentation. We setup and document the package in such a way that users not involved in these models, such as vendors, can compile and execute the provided scientific experiments easily.*

- *Evaluation of mini-IGCMs*
  *Firstly, we evaluate the DSL versions on the kernels implemented in WP1 and identify the advantages and problems of our approach. Secondly, we evaluate the applicability of our DSL concepts to the models by integrating these kernels into the mini-IGCMs that are prepared in Task 3.3. On the mini-IGCMs, we also identify performance issues of file I/O and evaluate the compression schemes developed in WP2. Both, single-node performance and scalability are evaluated in this task.*

- *Estimating DSL-benefit for full-featured models*
  *It is not feasible and necessary to completely rewrite the models using the DSL, with our demonstrator we estimate the benefit and yield sufficient results to continue the discussion with the scientific community. By extrapolation of measured performance characteristics for the original models and the rewritten kernels, we predict the benefits the DSL would bring to the full models.*

- *I/O advances for full model*
  *For demonstrating the advances in file I/O performance, we perform real productive runs using the full featured models with a typical configuration and next-generation resolution.*

*The focus of the deliverable lies on the last two items: the estimation of the benefit for full models.*

# 2 Packaging of the Mini-IGCMs

The partners of the project AIMES colaborated to develop a package with three mini-IGCMs for the purpose of running experiments using icosahedral grids. Different grid structures are used in the different mini-IGCMs. Each mini-IGCM is developed based on one of the three icosahedral models ICON, DYNAMICO, and NICAM. The package is published online under `https://github.com/aimes-project/IcoAtmosBenchmark_v1`. The different mini-IGCMs are packaged as ready to run components. Input data files for the purposes of testing are provided, and compiation and running files are provided along with the source code. Documentation of the mini-IGCMs is alos provided within the package.

## 2.1 ICON-based Mini-IGCM

This mini-IGCM was developed based on kernels from the ICON model. However, for licensing reasons, we could not use the same kernels. So, we developed this code using a grid that resembles the icosahedral grid of the ICON model. This grid is a tessellation of the surface of the globe into triangles using an unstructured grid. The tessellation is described computationally using the connectivity information to enable the access to

the neighbors and related components. Fields data are located at the centers of the triangular cells, or on the edges between the cells, or at the vertices of the edges.
The mini-IGCM contains a set of kernels that update a set of fields. The selected kernels represent a set of PDE operators and some frequently-happening computational patterns. Within the mini-IGCM we have the folowing kernels

- Vertical Integration
- Divergence
- Gradient
- Laplacian
- 3D Variable Weighted by a Horizontal Parameter
- Computations with Vertical and Horizontal Neighbors and Different Kinds of Parameters

Some kernels traverse grid cells and some traverse grid edges. The kernels demonstrate the access to neighbors, edges of a cell, cells of an edge, etc.

## 2.2 DYNAMICO-based Mini-IGCM

This mini-IGCM was developed based on the DYNAMICO icosahedral model. DYNAMICO uses an icosahedral hexagonal C-grid for the surface. Some fields are located at the centers of of the hexagons, while others are located on the edges between the heaxagons and others are located at the vertices. The mini-IGCM contains the following set of kernels:

- compute pvort
- compute geopot
- compute caldyn horiz
- compute caldyn vert

Each of the kernels within this mini-IGCM is taken from a kernel within the original code of DYNAMICO. The mini-IGCM includes the necessary code along with the source code of the kernels, e.g. I/O handling routines, and the code to run the kernel as a standalone application. To support experiments using the kernels, input files and validation reference files are provided with each kernel. Using the 'Makefile' within the mini-IGCM simplifies compiling and running the kernels.

## 2.3 NICAM-based Mini-IGCM

This mini-IGCM was developed based on the NICAM icosahedral model. NICAM targets massively-parallel infrastructure. To enable high resolution simulations, the spherical triangles of the icosahedron are divided recursively. In each grid refinement step, the arc between two vertices is divided by a new vertex in the middle, dividing each triangle into four equally-sized smaller triangles. The fields are located at the centers of the triangles as an Arakawa-A grid. Hexagonal structure of the grid is constructed by connecting the centers of the triangles.
The mini-IGCM includes the following set of kernels:

- OPRT diffusion
- OPRT3D divdamp
- horizontal flux
- horizontal limiter thuburn
- vertical limiter thuburn
- vi rhow solver

# 3 Evaluation of the Mini IGCMs

The three mini-IGCMs are packaged such that they can be downloaded, compiled, and run on Linux machines in general. However, files are provided along with 'NICAM' and 'DYNAMICO' -based mini-IGCMs to run and test them on specific machines, Mistral at German Climate Computing Center (DKRZ) and K-Computer at the Riken Advanced Institute for Computational Science.
The mini-IGCMs include Multiple kernels, and the necessary I/O operations. Communication code for multiple-node cases is alos provided with the NICAM-based mini-IGCM. DYNAMICO and NICAM -based min-IGCMs provide a main program to test each computational kernel. This allows to test each kernel seperately and evaluate different parameters for that kernel. Detailed information on building and running the mini-IGCMs is provided in the deliverable report D3.3 (`https://wr.informatik.uni-hamburg.de/_media/research/projects/aimes/d3.3.pdf`). The output results of running a sample kernel 'comp_caldyn_horiz' from the DYNAMICO-based mini-IGCM is shown in Listing 1.

**Listing 1: Running the DYNAMICO kernel comp_caldyn_horiz**

```
[KERNEL] comp_caldyn_horiz
*** Start  initialize
              iim, jjm, llm:    23    25    19
            ij_begin, ij_end:    48   528
    ij_begin_ext, ij_end_ext:    24   552
            ll_begin, ll_end:     1    19
       t_right, t_rup, t_lup:     1    23    22
    t_left, t_ldown, t_rdown:    -1   -23   -22
       u_right, u_rup, u_lup:     0  1173   575
    u_left, u_ldown, u_rdown:    -1  1150   553
          z_rup, z_up, z_lup:   598     0   597
    z_ldown, z_down, z_rdown:   -23   575   -22
              caldyn_conserv:     1
                  boussinesq:     F
                           g:     9.80000000
+check[pk_prev         ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz_pre] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du_prev         ] max=  3.4399140149818440E-03,min= -3.0658810374294527E-03,sum=  5.5109794763703335E-01
+check[le              ] max=  1.3457165724385556E+05,min=  0.0000000000000000E+00,sum=  1.6031419146648201E+08
+check[Ai              ] max=  3.4618288017294556E+10,min=  0.0000000000000000E+00,sum=  1.7746401564746273E+13
+check[de              ] max=  4.5171816240714993E+06,min=  0.0000000000000000E+00,sum=  4.7785815753077912E+08
+check[Av              ] max=  4.1228713627140027E+11,min=  0.0000000000000000E+00,sum=  3.2428753277257527E+13
+check[Wee             ] max=  5.9893054722291683E-01,min= -5.8540209553487599E-01,sum=  2.4695023837951297E+01
*** Finish initialize
*** Start kernel
### check point iteration:           1
### Input ###
+check[u               ] max=  4.1278968179782127E-01,min= -4.1278968179782127E-01,sum=  1.6791131703073393E+01
+check[rhodz           ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[qu              ] max=  1.0339537867296609E-06,min= -8.4408169682701225E-07,sum=  3.9419811615778674E-04
+check[theta           ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[geopot          ] max=  3.8250620498369227E+05,min=  0.0000000000000000E+00,sum=  1.1718001851963627E+09
+check[pk_prev         ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz_pre] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du_prev         ] max=  3.4399140149818440E-03,min= -3.0658810374294527E-03,sum=  5.5109794763703335E-01
### Output ###
+check[pk              ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux           ] max=  3.1805763161244854E+07,min= -2.8604204589892026E+07,sum=  2.4131331333014986E+08
+check[convm           ] max=  1.0361970643226587E-03,min= -1.0359249303947807E-04,sum= -1.5233533963107249E-01
+check[dtheta_rhodz    ] max=  3.2251351666935379E-01,min= -3.3676276308628725E-02,sum= -5.3720539414185993E+01
+check[du              ] max=  3.4404317002518906E-03,min= -3.0804630348046005E-03,sum=  5.5048589972605033E-01
### final iteration:        1000
### Validation : point-by-point diff ###
+check[pk              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[hflux           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz    ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel

*** Computational Time Report
*** ID=001 : MAIN_comp_caldyn_horiz           T=     0.992 N=    1000
```

The test uses a small grid $23 \cdot 25 \cdot 19$. Different grid dimensions and other grid traversal variables and constants are printed during the initialization of the run. Involved fields that are used in the kernel are checked and the minimum, maximum and sum of each field are printed. Along with the mini-IGCM are provided validation files, which one can use to validate the tests. Finally, the test shows a computation time report.
Again, with NICAM-based mini-IGCM seperate kernels can be run to evaluate each kernel. Output of runni9ng the sample kernel 'dyn_diffusion' is shown in Listing 2.

**Listing 2: Running the NICAM kernel dyn_diffusion**

```
[KERNEL] dyn_diffusion
*** Start  initialize
*** Finish initialize
*** Start  kernel
### Input ###
+check[check_dscl      ] max=  6.1941315670898286E-08,min= -7.0374144752795210E-08,sum= -2.7407850230958588E-07
+check[check_dscl_pl   ] max=  2.2139244811324830E-08,min= -6.0170678327656930E-11,sum=  1.8274579608905828E-07
+check[scl             ] max=  1.6578626530298903E-11,min= -1.2670212860993856E-11,sum= -2.0289014286353776E-10
+check[scl_pl          ] max=  1.9358849576453664E-11,min= -8.2853331106472899E-12,sum=  1.1445754720677440E-09
+check[kh              ] max=  2.8341305529772246E+12,min=  6.7659597088284981E+10,sum=  6.0439053980501018E+17
+check[kh_pl           ] max=  2.8334094314435532E+12,min=  6.7659597088284981E+10,sum=  4.3486317454839525E+14
### Output ###
+check[dscl            ] max=  6.1941315670898286E-08,min= -7.0374144752795210E-08,sum= -2.7407850230958588E-07
+check[dscl_pl         ] max=  2.2139244811324830E-08,min= -6.0170678327656930E-11,sum=  1.8274579608905828E-07
### Validation : point-by-point diff ###
+check[check_dscl      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[check_dscl_pl   ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel

*** Computational Time Report
*** ID=001 : MAIN_dyn_diffusion               T=     0.028 N=      1
*** ID=002 : OPRT_diffusion                   T=     0.028 N=      1
```

The output shows the results of checking the fields that the kernel involves. At the end, a computation time report is shown for the run of the kernel, and for the operator itself.
Further on the ICON-based mini-IGCM is discussed in the next section, as that code was used mainly to evaluate the expected benefits of using the DSL in global models. For licensing reasons, ICON code is not used directly in this project. Therefore, we developed our mini-IGCM using code that uses grids and concepts like those in ICON model[1] . Furthermore, we used same techniques for indirect access to field data. Similar idea of using filling curve for the sufrace grid was also used.

# 4 Estimated DSL Benefits for Modeling

To estimate the benefits of using the DSL for modeling, we consider the ICON-based mini-IGCM. We have written the code of this mini-IGCM originally in GGDML DSL + C language. The code uses icosahedral grids covering the surface of the globe. The fields are localized at the cell centers, and on the edges of the cells. The grid of the globe surface is mapped to one dimensional array using Hilbert space-filling-curve. We used 1,048,576 grid points (and more points over multiple-node runs) to discretize the surface of the globe. The code is written with 64 vertical levels. The surface is divided into blocks. The kernels are organized into components, each of which resembles a scientific process.
The mini-IGCM is translated into C code using the GGDML source-to-source translation tool through configuration files that allow scaling the code over multiple nodes. This allows to emmitate real global models and real model runs with high resolution global grids over multiple nodes.
The GGDML code can be translated to target different hardware configurations and architectures. Therefore, the original code of the mini-IGCM can be translated to different architectures including multicore processors and GPU-accelerated machines. We carried out some experiments on Mistral at the German Climate Computing Center (DKRZ). We used dual socket Intel Broadwell nodes (Intel Xeon E5-2695 v4 @ 2.1GHz). We used OpenMPI version 1.8.4 and GCC version 7.1 to test the mini-IGCM using higher-resolution grids on multiple nodes. We carried out some performance measurements using different optimization alternatives.
To demonstrate the ability of the DSL to support global icosahedral models, we present in Figure 1 the results of an experiment using the ICON-based mini-IGCM[2]. In this experiment we use the global grid of the mini-IGCM and apply a Laplacian stencil. We varied the number of nodes that we use to run the code up to 48 nodes. The minimum number of the grid points we used is 1,048,576. We used this number of points for strong scalability analysis. However, more grid points are used for the analysis of the weak scalability.
The weak scalability of the code is approximately 100%. Thus increasing the resolution of the grids and running the code on more nodes is achieved efficiently. This is an important point as higher resolution grids are essential for recent and future global simulations.
Further analysis was done using OpenMP to parallelize the code over 36 cores on chip, and using MPI to parallelize code over multiple nodes (again up to 48). Results are shown in Figure 2.
We carried out experiments to run the same code on GPUs to evaluate scalability. Initial results were measured on a few nodes, where we had access on a cluster that uses GPUs. P100 GPUs were used for the measurements and results are shown in Figure 3.

[1] A C version of the code is available at `https://github.com/aimes-project/IcoAtmosBenchmark_v1/tree/master/kernels/ICOC`
[2] Refer to *Performance Portability of Earth System Models with User-Controlled GGDML code Translation (Nabeeh Jumah, Julian Kunkel), DOI: https://doi.org/10.1007/978-3-030-02465-9_50*

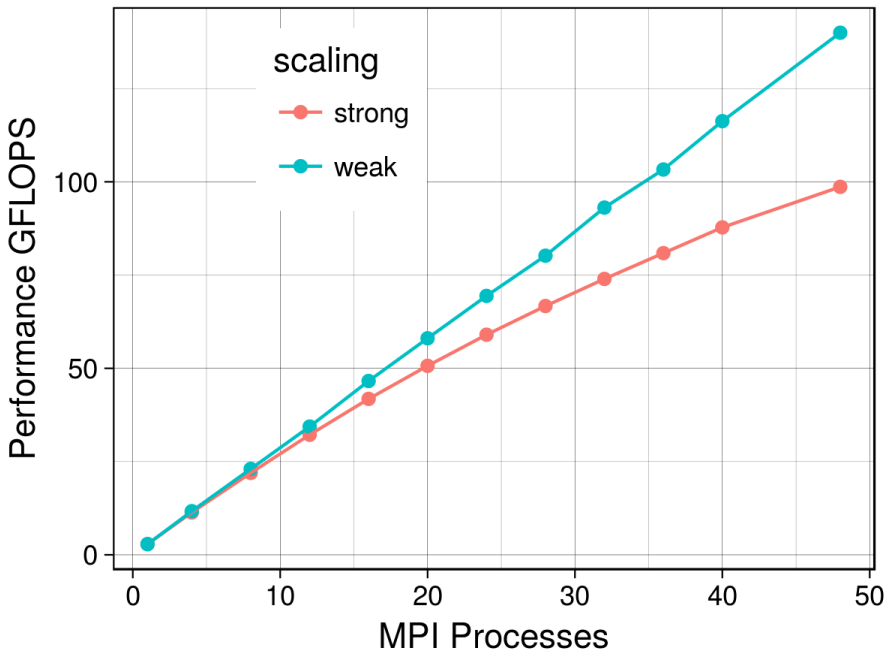


Figure 1: MPI process scalability

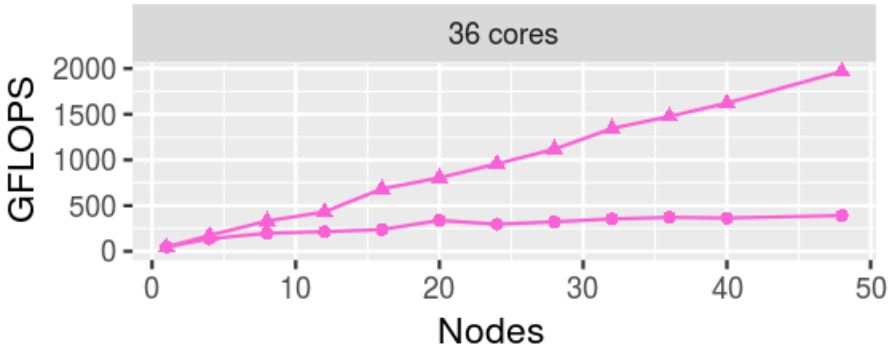


Figure 2: OpenMP + MPI scalability

Further improvements and experiements on larger scale with a code that was developed later were done and published in *Scalable Parallelization of Stencils using MODA (Nabeeh Jumah, Julian Kunkel), ISC 2019 Workshop: P3MA, Frankfurt, Germany, 2019.*
The GGDML language extensions allowed to drive multiple necessary optimization procedures. Memory layout trnasformations are doable in a flexible way using GGDML. This is a result of using the GGDML indices, which avoids memory-based semantics. The GGDML iterator allows transforming source code into optimized loops. Translation process includes a variety of optimization aspects, e.g. loop order or blocking. The optimization procedures that the tools apply throughout the code translation process allow to achieve near optimal use of the node resources.
Parallelization over multiple nodes scales GGDML-coded models to support higher resolution grids. Parallelization does not need any changes to the source code. This is an important feature of the DSL and its support to global climate modeling and other earth system modeling tasks, i.e. the source code is written once. No matter if the model is to be run on a single node or multiple nodes, on multi-core processors or GPUs, the same source code is used, and it does not need to do be changed.

**Estimating DSL Impact on Code Quality and Development Costs** To estimate the impact of using the DSL on the quality of the code and the costs of model development, we took two relevant kernels from each of the three icosahedral models, and analyzed the achieved code reduction. We rewrote the kernels (originally

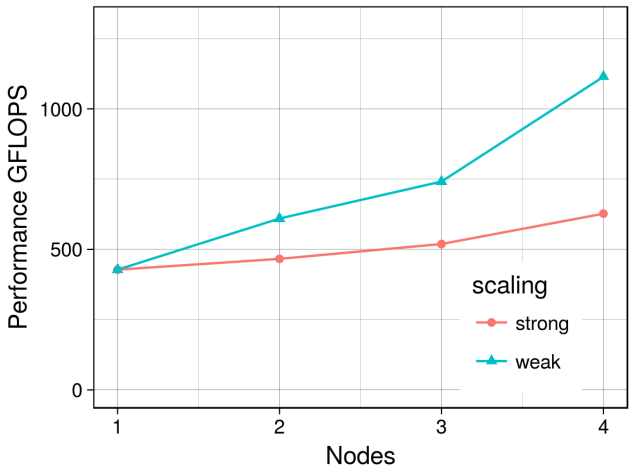


Figure 3: OpenACC + MPI scalability

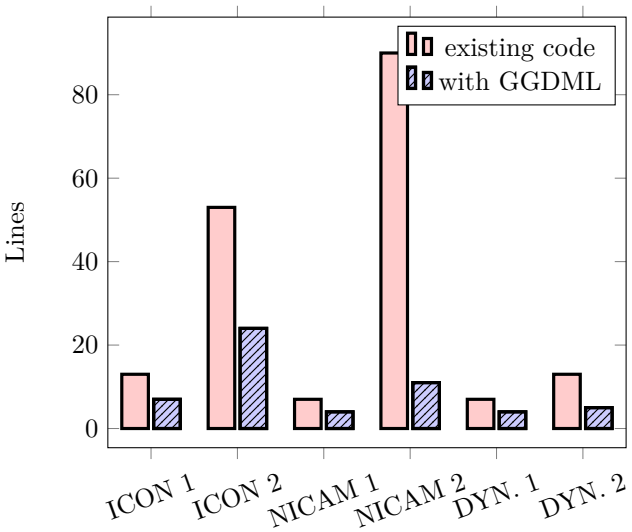


Figure 4: GGDML impact on LOC

written in Fortran) using GGDML + Fortran. Results are shown in Figure 4. [3]
The average reduction in terms of LOC is around (30%), i.e. LOC in GGDML+Fortran in comparison to original Fortran code. More reduction is noticed in some stencils (NICAM example No.2, reduced to 12.22%).
**Influence on readability and maintainability:** Using COCOMO as a model to estimate complexity of development effort and costs, we estimated in Table 1 the benefits as a result of the code reductions when applying GGDML to develop a model comparable to the ICON model. The estimations are based on a code with 400KLOC, where 300KLOC of the code are the scientific portion that allows for code reduction while 100KLOC are infrastructure.
From the predicted developed effort, it is apparent that the code reductions would be leading to a significant effort and cost reduction that would justify the development and investment in DSL concepts and tools.

| Development Style | Codebase | Effort Applied | Dev. Time (months) | People require | Dev. costs (M€) |
|---|---|---|---|---|---|
| **Semi-detached** | Fortran | 2462 | 38.5 | 64 | 12.3 |
| | DSL | 1133 | 29.3 | 39 | 5.7 |
| **Organic** | Fortran | 1295 | 38.1 | 34 | 6.5 |
| | DSL | 625 | 28.9 | 22 | 3.1 |

Table 1: COCOMO cost estimates [**?**]

[3]Results are published in *GGDML: Icosahedral Models Language Extensions (Jumah et. al.), DOI:http://dx.doi.org/10.15379/2410-2938.2017.04.01.01*

**Code Portability and Performance Portability** An important aspect of the solution we developed is performance portability. The scientists develop a single source code. The source does not need to be modified when target a new machine.
Besides to supporting icosahedral grids, GGDML is developed to serve the domain of earth system modeling and stencil computations in general. Therefore, it serves applications using other grids other than icosahedral grids. We used it to solve the shallow water equations using a structured rectangular grid. We prepared different configuration files to target different hardware configurations. Multi-core processors, GPUs, and vector engines were used to run the application. The code wass written with GGDML+C language. Numbers in Table 2 show a comparison of the sizes of the code versions. LOC in the source code (GGDML+C) in comparison to different generated optimized code versions. We show a set of configurations including using MPI vs. GASPI for communcation, and using different architectures. Also, a single node, vector engine, is shown. Two kernels (denoted Kernel1 and Kernel2) of different sizes are considered in the comparison besides to the application itself.

| Code | GGDML | MPI | GASPI | GPU(multiple nodes) | VE(single node) |
|---|---|---|---|---|---|
| Kernel1 | 5 | 20 | 20 | 19 | 14 |
| Kernel2 | 10 | 148 | 218 | 180 | 21 |
| Application | 161 | 779 | 946 | 748 | 531 |

Table 2: LOC of the GGDML application code vs. the generated GPL code for different target platforms

Regrading performance portability, the same code was run on the different architectures with about 80% of the maximum memory bandwidth of each archtecture (on Broadwell processor, P100 GPUs, and NEC Aurora vector engine). As the code is memory bound, the performance is considered near optimal on each of the used architectures. Using the same source code and getting the performance ratio to the expected performance on each architecture shows that the technique is successful to provide performance portability.

# 5 I/O Advances for Modeling

The I/O work was mainly done to improve the storage handling of the field data. Support for parallel code run on multiple nodes using suitable data formats for climate modeling and weather prediction applications are main concerns. To optimize the use of the storage resources and the I/O operations, lossy compression is provided through the 'SCIL' library. Compression algorithms are specified by the users. Also, parameters that determine the compression characteristics are also chosen by users to control the quality of the compression in order to control the tolerence based on the field characteristcs.
In our DSL-based mini-IGCM code we developed a small I/O component to handle I/O operations using NetCDF. To improve I/O operations of global icosahedral models, which need to be run on multiple nodes for modern models, we have developed the I/O components to use parallel I/O.
To estimate the impact of using our I/O support and compression library for global modeling using icosahedral grids, we executed some experiments with alternative compression algorithms and parameters. We used a high-resolution grid with 268 million points in the surface grid times 64 levels in the vertical dimension. The experiments were run on 128 nodes on the machine 'Mistral' in German Climate Computing Center 'DKRZ'. We run the application with MPI library, with one MPI process per node. A field with single precision values between -10 and +55 was written to the storage. Some measurements were recorded with directly writing field data to storage with NetCDF without using the 'SCIL' library. Other measurements were done with 'SCIL' library to do differnt compression cases. The results are shown in Table 3. The figure shows the write times in seconds, the data sizes in GB, a virtual throughput relative to the uncompressed write, and the speedup. The differnt cases to use 'SCIL' are done with the algorithms

- memcopy- which does not do any real compression, but allows to measure overhead of the library use
- lz4- the well-known compression algorithm
- abstol,lz4- which processes data elements based on the absolute value being processed, we control the tolerence by the parameter *absolute_tolerance*
- sigbits,lz4- which processes data elements based on a percentage of the value being processed, we control the tolerence by the parameter *relative_tolerance_percent*

| Compression method | Parameter | Write time | Data size | Throughput* | Speedup |
|---|---|---|---|---|---|
| | | in s | in GB | in MB/s | |
| No-compression | | 165.3 | 71.4 | 432 | 1.0 |
| memcopy | | 570.1 | 71.4 | 125 | 0.3 |
| lz4 | | 795.3 | 71.9 | 90 | 0.2 |
| abstol,lz4 | absolute_tolerance=1 | 12.8 | 2.7 | 5578 | 12.9 |
| | absolute_tolerance=.1 | 72.6 | 2.8 | 983 | 2.3 |
| sigbits,lz4 | relative_tolerance_percent=1 | 12.9 | 2.9 | 5535 | 12.8 |
| | relative_tolerance_percent=.1 | 18.3 | 3.2 | 3902 | 9.0 |

Table 3: Compression results of 128 processes on Mistral

Without compression the performance is quite poor: achieving only 432 MB/s on Mistral on 128 nodes, while an optimal benchmark can achieve 100 GB/s. The HDF5 compression is not yet optimally parallelized, and requires certain collective operations to update the meta-data. Internally, HDF5 requires additional data buffers. This leads to extra overhead in the compression slowing down the IO (see the memcopy and and LZ4 results which serve as baselines). By activating lossy compression and accepting an accuracy of 1% or 0.1%, the performance can be improved in this example up to 13x.

# 6 Summary and Conclusions

In this report we described at a higher-level look the impact of the work that is done in the project AIMES on the domain. Mainly, the work is focused on improvements on the model programming, and the I/O and storage. The main focus of this report is the estimation of the applicability to global (icosahedral) models, which face many challenges.
We reviewed briefly the packaging of the mini-IGCMs that were discussed in D3.3[4] to show what would be expected when running those mini-IGCMs. However, the main focus is the estimation of the benefits of applying the techniques that were investigated in this project to real global (icosahedral) models.
The estimations of the benefits of the programming techniques was discussed in terms of impact on code quality, cost of model code development, performance and portability. The techniques prove to be effecient to reduce development costs as shown. Performance and performance portability show efficiency to use a single code tree, written in high-level DSL, to generate optimized code for differnt architectures, with high percentage of the performance that can be achieved on each architecture.
I/O and storage also show improved data storage capabilities using the techniques that were developed in AIMES project. Data sizes were clearly reduced allowing to optimize the use of the storage resources based on characteristics of the data that is generated from those models. I/O throughput also is improved when the right compression algorithms and parameters are used.

# Acknowledgement

This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 „Software for Exascale Computing“ (SPPEXA).



[4] https://wr.informatik.uni-hamburg.de/_media/research/projects/aimes/d3.3.pdf