Advanced Computation and I/O Methods for Earth-System Simulations (AIMES)

Julian Kunkel[1], Nabeeh Jumah[2], Anastasiia Novikova[3], Thomas Ludwig[4], Hisashi Yashiro[5], Naoya Maruyama[5], Mohamed Wahib[6] and John Thuburn[7]

[1]University of Reading, UK
[2]Universität Hamburg, Germany
[3]Fraunhofer IGD, Rostock, Germany
[4]Deutsches Klimarechenzentrum (DKRZ), Hamburg, Germany
[5]RIKEN, AICS, Japan
[6]AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL), Tokyo, Japan
[7]University of Exeter, UK

Abstract. Dealing with extreme scale Earth-system models is challenging from the computer science perspective, as the required computing power and storage capacity are steadily increasing. Scientists perform runs with growing resolution or aggregate results from many similar smaller-scale runs with slightly different initial conditions (the so-called ensemble runs). In the fifth Coupled Model Intercomparison Project (CMIP5), the produced datasets require more than three Petabytes of storage and the compute and storage requirements are increasing significantly for CMIP6. Climate scientists across the globe are developing next-generation models based on improved numerical formulation leading to grids that are discretized in alternative forms such as an icosahedral (geodesic) grid. The developers of these models face similar problems in scaling, maintaining and optimizing code. Performance portability and the maintainability of code are key concerns of scientists as, compared to industry projects, model code is continuously revised and extended to incorporate further levels of detail. This leads to a rapidly growing code base that is rarely refactored. However, code modernization is important to maintain productivity of the scientist working with the code and for utilizing performance provided by modern and future architectures. The need for performance optimization is motivated by the evolution of the parallel architecture landscape from homogeneous flat machines to heterogeneous combinations of processors with deep memory hierarchy. Notably, the rise of many-core, throughput-oriented accelerators, such as GPUs, requires non-trivial code changes at minimum and, even worse, may necessitate a substantial rewrite of the existing codebase. At the same time, the code complexity increases the difficulty for computer scientists and vendors to understand and optimize the code for a given system.
Storing the products of climate predictions requires a large storage and archival system which is expensive. Often, scientists restrict the number of scientific variables and write interval to keep the costs balanced. Compression algorithms can reduce the costs significantly but can also increase the scientific yield of simulation runs.
In the AIMES project, we addressed the key issues of programmability, computational efficiency and I/O limitations that are common in next-generation icosahedral earth-system models. The project focused on the separation of concerns between domain scientist, computational scientists, and computer scientists.

The key outcomes of the project described in this article are the design of a model-independent Domain-Specific Language (DSL) to formulate scientific codes that can then be mapped to architecture specific code and the integration of a compression library for lossy compression schemes that allow scientists to specify the acceptable level of loss in precision according to various metrics. Additional research covered the exploration of third-party DSL solutions and the development of joint benchmarks (mini-applications) that represent the icosahedral models. The resulting prototypes were run on several architectures at different data centers.

Keywords: Earth system modeling, DSL, Icosahedral grid, Lossy compression

1 Introduction

The problems on the frontier of science requires extreme computational resources and data volumes across the disciplines. Examples of processes include the understanding of the earth mantle [13], plasma fusion [25], properties of steel [8], and the simulation of weather and climate. The simulation of weather and climate requires to model many physical processes such as the influence of radiation from the sun and the transport of air and water in atmosphere and ocean [11]. As these processes are complex, scientists from different fields collaborate to develop models for climate and weather simulations.

The mathematical model of such processes is discretized and encoded as computer model using numerical methods [53]. Different numerical methods can be used to approximate the mathematical models. A range of different numerical methods are used, including finite difference, finite volume, and finite element. All of these methods partition the domain of interest into small regions and apply stencil computations to approximate operations such as derivatives.

The necessary computations include variables (fields) like temperature or pressure distributed spatially over some surface or space – the problem domain. Simple techniques divide a surface into rectangular smaller regions covering the whole domain. Such rectangular grids have a simple regular structure. Those grids fit computations well as the grid structure simply corresponds to the array notation of the programming languages. However, applying this grid to the globe leads to variable sizes of grid cells, e.g., the equator region has a coarse grid while the polar regions are a singularity. With such a shortcoming, rectangular grids are well suited for regional models but not for a global model.

Therefore, recent models targeting global simulations are developed using different grids. Moving to such alternative grids allows to solve the cell area problem for global models, but the formulation of the models is more complicated. Icosahedral grids are examples of such alternatives. An icosahedral grid results from projecting an icosahedron onto the surface of the globe. The surface of the globe is then divided into twenty spherical triangles with equal areas. Grid refinement is achieved with recursive division of the spherical arcs into halves. The resulting points of the division form four smaller spherical triangles within each spherical triangle. Such refinement is repeated until the needed resolution is reached. Icosahedral grids can be used with the triangles as the basic cell, but also hexagons can be synthesized.

Icosahedral grids have approximately uniform cell area and can be used for global models avoiding the cell area differences in contrast to the rectangular grids. However, complications arise when thinking of the technical side, where we need to know how to map the field data into data structures. Such technical details are challenging with the performance demand for the models.

The values of a field in the simulation is localized with respect to the grid cell depending on the numerical formulation. In one method, values of a field are localized at the centers of the cells – this can be a single value or multiple values with higher order methods. However, other methods localize values on the vertexes, while others reside on the edges separating the cells (see Figure 1). How the cells are connected to each other, i.e., the neighbors and orientation of the cells, is defined in the connectivity. A problem domain can be organized in a regular fashion into so-called structured grids – following an easy schema to identify (left, right, ...) neighbors. Unstructured grids can follow a complex connectivity, e.g., using smaller and larger cells, or covering complex surfaces but require to store the connectivity information explicitly. The modern models explore both schemes due to their benefit; for instance, the structured grid can utilize compiler optimizations more effectively, while unstructured grids allow local refinement around areas of interest.

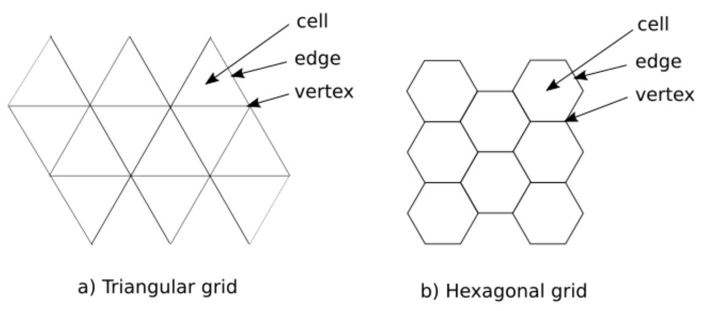


Fig. 1: Icosahedral grids and variables

General-purpose languages (GPL), e.g., Fortran, are widely used to encode the discretized computer model. The simulation of the earth with a high resolution like $1km^2$ that is necessary to cover many smaller-scale physical processes, requires a huge computation effort and likewise storage capacity to preserve the results for later analysis. Due to uncertainty, scientists run a single experiment many times, multiplying the demand for compute and storage resources. Thus, the optimization of the codes for different architectures and efficiency is of prime importance to enable the simulations.

With existing solutions, scientists rewrite some code sections repeatedly with different optimization techniques that utilize the capabilities of the different machines. Hence, scientists must learn different optimization techniques for different architectures. The code duplication brings new issues and complexities concerning the development and the maintainability of the code.

Thus, the effort from the maintainers and developers of the models, who are normally scientists and not computer scientists, is substantial. Scientists' productivity is an important point to consider as they do activities that should not be their focus. Maintaining model codes throughout the lifecycle of the model is a demanding effort under all the mentioned challenges.

The structure of the icosahedral grids brings complications not only to the computation, but also to the storage of the field data. In contrast to regular grids, where multi-dimensional array notation fits to hold the data, icosahedral grids do not necessarily map directly to simple data structures. Besides the challenge of file format support, modern models generate large amounts of data that impose pressure on storage systems. Recent models are developed with higher-resolution grids, and include more fields and processes. Simulations writing terabytes of data to the storage system push towards optimizing the use of the storage by applying data-reduction techniques.

The development of simulation models unfolds many challenges for the scientific community. Relevant challenges for this project are:

– **Long life**: The lifecycle of earth system models is long in comparison to the turnover of the computer technology mainly in terms of processor architectures.

- **Performance and efficiency**: The need for performance and the optimal use of the hardware resources is an important issue.
- **Performance-portability**: Models are run on different machines and on different architectures. They must use the available capabilities effectively.
- **Collaboration**: Another point is the collaborative efforts to develop models – involving PhD students to contribute pieces of science code to a large software project – that complicates the maintenance and software engineering of the models.
- **Data volume**: the large amounts of data must be handled efficiently.

The AIMES Project To address challenges facing earth system modeling, especially for icosahedral models, the project *Advanced Computation and I/O Methods for Earth-System Simulations* (AIMES) investigated approaches to mitigate the aforementioned programming and storage challenges.

The AIMES project is part of the SPPEXA program and consisted of the consortium:

- University of Hamburg, Germany
- Institut Pierre Simon Laplace (IPSL), Université Versailles Saint-Quentin-en-Yvelines, France
- RIKEN Advanced Institute for Computational Science, Japan
- Global Scientific Information and Computing Center, Tokyo Institute of Technology, Japan

The project started in March 2016 with plans for three years.

The main objectives of the project were: 1) Enhance programmability; 2) Increase storage efficiency; 3) Provide a common benchmark for icosahedral models. The project was organized in three work packages covering these aspects and a supplementary project management work package to achieve the three project objectives. The strategy of the work packages is layed out in the following.

Under the first work package, higher-level scientific concepts are used to develop a dialect for each of three icosahedral models: DYNAMICO [18], ICON [54], and NICAM [48]. A domain-specific language (DSL) is the results of finding commonalities in the three dialects. Also, a light weight source-to-source translation tool is developed. Targeting different architectures to run the high-level code is an important aspect of the translation process. Optimizations include applying parallelization to the different architectures. Also, providing a memory layout that fits the different architectures is considered.

Under the second work package, data formats for icosahedral models are investigated to deal with the I/O limitations. Lossy compression methods are developed to increase storage efficiency for icosahedral models. The compression is guided with user-provided configuration to allow to use suitable compression according to the required data properties.

Under the third work package, relevant kernels are selected from the three icosahedral models. A mini-IGCM is developed based on each of the the three models to offer a benchmark for icosahedral models. Developed code is used to evaluate the DSL and the compression of icosahedral global modeling.

The **key outcomes** of the project described in this article are: 1) the design of an effective light-weight DSL that is able to abstract the scientific formulation from architecture-specific capabilities. 2) the development of a compression library that separates the specification of various qualities that define the tolerable error of data from the implementation. We provide some further large-scale results of our compression library for large scale runs extending our papers about the library ([37,38]). 3) The development of benchmarks for the Icosahedral models that are mini-applications of the models.

Various additional contributions were made that are summarized briefly with citations to the respective papers:

– We researched the impact of lossless data compression on energy consumption in [5]. In general, the energy consumption increases with the computational intensity of the compression algorithm. However, there are algorithms that are efficient and less computational intense that can improve the energy-efficiency.
– We researched compilation time of code that is generated from the DSL using alternative optimization options on different compilers [22]. Different optimization options for different files allow different levels of performance, however, compilation time is also an important point to consider. Results show that some files need less optimization focus while others need further care. Small performance drops are measured with considerable reduction in compile times when the suitable compilation options are chosen.
– We researched annotating code for instrumentation automatically by our translation tool, to identify resource consuming kernels [21]. Instrumentation allows to better find where to focus the optimization efforts. We used the DSL translation tool to annotate kernels and make generated code ready for instrumentation. As a result performance measurements were recorded with reduced effort as manual preparations are not needed anymore.
– We researched applying vector folding to icosahedral grid codes in a bachelor thesis [50]. Vector folding allows to improve use of caches by structuring data in a way accounting for caches and data dimensionality. Results show that vector folding was difficult to apply manually to icosahedral grids. Performance was raised but not significantly as a result of the needed effort that should be invested to rewrite kernels with this kind of optimization.
– Involved ASUCA and the use of Hybrid Fortran [44] to port original CPU code to GPUs, to look at a different model with different requirements.

This article is structured as follows: First, the scope of the state-of-the-art and related work is sketched in Section 2. In Section 3, an alternative development approach for code-modernization is introduced. Various experiments to evaluate the benefit of the approach are shown in Section 4. The compression strategy is described in Section 5 and evaluated in Section 6. The benchmarks for the icosahedral models are discussed in Section 7. Finally, the article is concluded in Section 8.

2 Related Work

The related work covers research closely related to domain-specific languages in climate and weather and the scientific data compression.

2.1 Domain-Specific Languages

DSLs represent an important approach to provide performance portability and support model development. A DSL is always developed having a particular domain in mind. Some approaches support multiple layers of abstraction. A high-level abstraction for the finite element method is provided with Firedrake [45]. The ExaStencils pipeline generally addresses stencil codes and their operations [19,34] and many research works introduce sophisticated schemes for the optimization of stencils [10,12].

One of the first DSLs which were developed to support atmospheric modeling is Atmol [3]. Atmol provided a DSL to allow scientists to describe their models using partial differential equations operators. Later, Liszt [15] provided a DSL for constructing mesh-based PDE solvers targeting heterogeneous systems.

Multi-target support was also provided by Physis [43]. Physis is a C-based DSL which allows developing models using structured grids.

Another form of DSL is Icon DSL [51]. Icon DSL was developed to apply index interchanges based on described swapping on Fortran-based models.

Further work based on C++ constructs and generic programming to improve performance portability is Stella [24] and later GridTools [2]. Computations are specified with a C++-based DSL and the tools generate code for CPUs or GPUs. GridTools are used to port some kernels from the NICAM model in our project AIMES to evaluate using existing DSLs.

Although C++ provides strong features through generic programming allowing to avoid performance portability issues, scientists are reluctant to utilize alternative programming languages as the existing codes are huge. Normally scientists prefer to keep using preferred languages, e.g. Fortran, rather than moving to learning C++ features.

Other forms of DSLs used directives to drive code porting or optimization. Hybrid Fortran [44], HMPP [17], Mint [52], CLAW [1] are examples of directive-based approach. Such solutions allow adding directives to code to guide some optimization. Scientists write code in some form and add directives that allow tools to provide specific features, e.g., CLAW allows writing code for one column and allows using directives to apply simulations over the set of columns in parallel.

In the MetOffice's LFRic model [4], a DSL is embedded into the Fortran code that provides an abstraction level suitable for the model. The model ships with the PSyclone code-generator that is able to transform the code for different target platforms. In contrast to our lightweight solution, these DSLs are statically defined and require a big translation layer.

2.2 Compression

Data-reduction techniques related to our work can be structured into: 1) algorithms for the lossless data compression; 2) lossy algorithms designed for scientific (floating-point) data and the HPC environment; 3) methods to identify necessary data precision and for large-scale evaluation.

Lossless algorithms: There exist various lossless algorithms and tools, for example, the LZ77 [55] algorithm which utilizes dictionaries. By using sliding windows, it scans uncompressed data for two largest windows containing the same data and replaces the second occurrence with a pointer to the compressed data of the first. The different lossless algorithms vary in their performance characteristics and ability to compress data depending on its characteristics. A key limitation is that users have to pick an algorithm depending on the use case. In [27], we presented compression results for the analysis of typical climate data. Within that work, the lossless compression scheme MAFISC with preconditioners was introduced. With MAFISC, we also explored the automatic selection of algorithms by compressing each block with two algorithms, the best compression chain so far and one randomly chosen. It compresses data 10% more than the second best algorithm (e.g., standard compression tools).

Lossy algorithms for floating-point data: SZ [16] and ZFP [42] are the de-facto standard for compressing floating-point data in lossy mode. Both provide a way of bounding the error (either bit precision or absolute error quantities) but only one quantity can be selected at a time. ZFP [42] can be applied up to three dimensions. SZ is a newer and effective HPC data compression method; it uses a predictor and the lossless compression algorithm GZIP. Its compression ratio is typically significantly better than the second-best solution of ZFP. In [26], two lossy compression algorithms (GRIB2, APAX) were evaluated regarding the loss of data precision, compression ratio, and processing time on synthetic and climate dataset. These two algorithms have equivalent compression ratios and depending on the dataset APAX signal quality may exceed GRIB2 and vice versa.

Methods: The application of lossy techniques to scientific (floating-point) datasets is discussed in [14,40,39,28,23,41]. A statistical method to predict characteristics (such as proportions of file types and compression ratio) of stored data based on representative samples was introduced in [35] and the corresponding tool in [36]. It can be used to determine compression ratio by scanning a fraction of the data, thus reducing costs.

Efforts for determination of appropriate levels of precision for lossy compression methods for climate and weather data were presented in [7] and in [6]. The basic idea is to compare the statistics derived from the data before and after applying lossy compression schemes; if the scientific conclusions drawn from the data are similar and indistinguishable without the compression, the loss of precision is acceptable.

3 Towards Higher-Level Code Design

Computations in earth system modeling run hundreds or thousands of stencil computations over wide grids with huge numbers of points. Such computations are time consuming and are sensitive to optimal use of computer resources. Compilers usually apply a set of optimizations while compiling code. Semantical rules of the general purpose language (GPL) can be applied to the source code and used within compilers to translate the code to semantically equivalent code that runs more efficiently. However, often the semantical information extracted from the source code are not enough to apply all relevant optimizations as some high-level optimizations would alter the semantics – and the rules of the GPL forbid such changes. As mostly code from GPLs is at a lower semantical level than the abstraction level of the developers, opportunity of optimization is lost. Such lost opportunities are a main obstacle to develop software in a performance-portable way in earth system modeling.

To address the lost opportunities of optimization, different techniques are applied by the scientists directly to the source code. Thus, it is the responsibility of the scientists who develop the models to write the code with those optimizations decisions and guidelines in mind. Drawbacks of this strategy include pushing scientists to focus on machine details and optimal code design to use hardware resources. Scientists need to learn the relevant optimization techniques from computer science such as, e.g., cache blocking.

3.1 Our Approach

To solve the issues with the manual code optimization, we suggest using an additional set of language constructs which exhibit higher-level semantics. This way, tools can be used to apply optimizations based on those semantics. Optimization responsibilities are moved again from scientists to tools.

As usually, the source code is developed by scientists, however, in our approach, instead of coding on low-level and caring for optimization strategies, a DSL is used as abstraction. Machine-specific or computer scientific concepts are not needed to write the source code of a model. This is enabled by increasing the abstraction level of a GPL by providing a language extension with semantics based on the scientific concepts from the domain science itself.

The DSL implements a template mechanism to simplify and abstract the code. For the purpose of this project, it was designed to abstract climate and weather scientific concepts but other domains and models could be supported. Therefore, stencils and grids are used as the basis in the DSL. The language extensions hide the memory access and array notations. They also hide the details of applying the stencils and the traversal of the grids. The grid structure itself is hidden in the source code. A model's code uses the grid without specifying the locations of the grid points in memory or how neighbors are accessed. All such details are specified in the configuration files. Lower details are neglected at the DSL level.

Translation tools handle the processing of the higher-level source code and converting it to compatible GPL code. The semantics of the added language extensions are extracted from the source code and are used to apply further high-level transformations to the code. Applied transformations are guided by configuration files that allow users to control the optimization process and may convert the code to various back-end representations that, in-turn, can be converted to code that is executed on a machine. A key benefit of this strategy is that it increases the performance-portability: A configuration can apply optimization techniques exploiting features of a specific architectures. Therefore, a single scientific code base can be used to target multiple different machines with different architectures.

Model-specific configuration files are provided separately to guide the code translation and optimization process. Those files are developed by scientific programmers rather than domain scientists. In contrast to domain scientists, the scientific programmers must have an intermediate understanding of the scientific domain but also understand the hardware architecture of a target machine. They use their experience to generate code that uses the machine's resources, and write the configurations that serve the purpose of optimal use of that specific machine to run the selected model.

Our approach offers separation of concerns between the parties. Scientific work is done by scientists and optimization is done by scientific programmers. The concept of the approach is illustrated in fig. 2. For the icosahedral models, a single intermediate domain language is derived that can be adjusted for the needs of each model individually (dialects). From this single source various code representations (back-ends) could be generated according to configuration, e.g., MPI+OpenMP or GASPI.

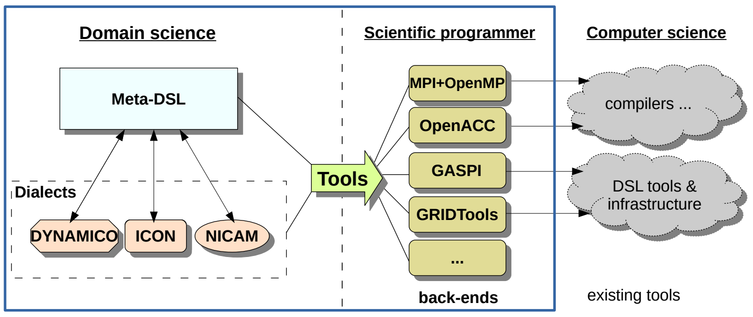


Fig. 2: Separation of Concerns

3.2 Extending Modeling Language

An important point we consider in our approach is keeping the general-purpose language, e.g. Fortran, that the scientists prefer to use to develop their model. We add additional language constructs to the GPL. This simplifies the mission to port existing models which could include hundreds of thousands of lines of code. The majority of the code is kept, while porting some parts incrementally; by providing templates in the configuration files, code can be simplified while it still produces equivalent GPL constructs: Changes are replacements of loop details and field access into alternative form using the added extensions. A drawback of the incremental approach is that the full beauty of optimizations like memory adjustments requires to have ported the complete model.

Our plan to develop such language extensions was to start with the three existing modern icosahedral models of the project partners. In the first phase, special dialects were proposed to support each model. Then, we identified common concepts and defined a set of language extensions that support the domain with domain-specific language constructs. We collected requirements, and worked in collaboration with scientists from the three models to reach at the language extensions, in detail:

1. The domain scientists suggested compute-intensive and time-consuming code parts.
2. We analyzed the chosen code parts to find out possibilities to use scientific terms instead of existing code. We always kept in mind that finding a common representation across the three models leads to domain-specific language extensions.
3. We replaced codes with suggested extensions.
4. We discussed the suggestions with the scientists. Discussions and improvements were done iteratively. The result of those discussions lead to the GGDML language extensions.

Extensions and Domain-Specific Concepts The *General Grid Definition and Manipulation Language (GGDML)* language extensions provide a set of constructs for:

– Grid definition
– Declaration of fields over grids
– Field access and update
– Grid traversal to apply stencils
– Stencil reduction expressions

GGDML code provides an abstraction including the order of the computation of elementary operators. Therefore, optimizations can result in minor changes of the computed result of floating-point operations; this is intentional as bit-reproducibility constraints the optimization potential.

In code that is written with GGDML, scientists can specify the name of the grid that should be traversed when applying a stencil. The definitions of the grids are provided globally through the configuration files. GGDML allows to specify a modified set of grid points rather than the whole set of grid points as provided through the grid definition grid, e.g., traversing a specific vertical level. Such possibilities are offered through naming a grid and using operators that allow adding, dropping, or modifying dimensions of that grid. Operators could change the dimensionality of the grid or override the existing dimensions.

Fields are declared over different grids through declaration specifiers. GGDML provides a flexible solution to support application requirements. A basic set of declaration specifiers allows to control the dimensionality of the grid, and the localization of the field with respect to the grid. Such declaration specifiers allow applications to deal with surfaces and spaces, and also supports using staggered as well as collocated grids.

Access specifiers provide tools the necessary information that will be used to allocate/deallocate and access the fields. Field access is an important part of stencil operations. GGDML provides an iterator statement to apply the stencil operations over a set of grid points. The GGDML iterator statement replaces loops and the necessary optimization to apply stencils. It provides the user an index that refers to the current grid point. Using this index, scientists can write their stencils without the need to deal with the actual data structures that hold the field data. The iterator applies the body to each grid point that is specified in the grid expression, which is one part of the iterator statement. This expression is composed from the name of a grid, and possibly a set of modifications using operators as mentioned above.

The iterators index alone is not sufficient to write stencil operations, as stencils include access to neighboring points. For this purpose, GGDML uses access operators, which represent the spatial relationships between the grid points. This allows to access the fields that need to be read or written within a stencil operation using spatial terms instead of arrays and memory addresses. To support different kinds of grids, GGDML allows users to define those access operators according to the application needs.

Repetitions of the same mathematical expressions over different neighbors is common in stencil operations. To simplify writing stencils, GGDML provides a reduction expression. Reduction expressions apply a given sub-expression over multiple neighbors along with a mathematical operator applied to the set of the subexpressions.

3.3 Code Example

To demonstrate the code written with extensions, in Listing 1.1 is a sample code from the NICAM model written with Fortran. As we can see from the original NICAM code, a pattern is repeated in the code: The same field is accessed multiple times over multiple indices. Optimization is limited as firstly, the memory layout is hardcoded in the fields cgrad and scl, secondly the iteration order is fixed. Integrating blocking for cache optimization in this schema would increase the complexity further.

Listing 1.1: NICAM Fortran code

```
do d = 1, ADM_nxyz
  do l = 1, ADM_lall
!OCL PARALLEL
    ! support indices to address neighbors
    do k = 1, ADM_kall
       do n = OPRT_nstart, OPRT_nend
          ij     = n
          ip1j   = n + 1
          ijp1   = n     + ADM_gall_1d
          ip1jp1 = n + 1 + ADM_gall_1d
          im1j   = n - 1
          ijm1   = n     - ADM_gall_1d
          im1jm1 = n - 1 - ADM_gall_1d

          grad(n,k,l,d) = cgrad(n,l,0,d) * scl(ij     ,k,l) &
                        + cgrad(n,l,1,d) * scl(ip1j   ,k,l) &
                        + cgrad(n,l,2,d) * scl(ip1jp1 ,k,l) &
                        + cgrad(n,l,3,d) * scl(ijp1   ,k,l) &
                        + cgrad(n,l,4,d) * scl(im1j   ,k,l) &
                        + cgrad(n,l,5,d) * scl(im1jm1 ,k,l) &
                        + cgrad(n,l,6,d) * scl(ijm1   ,k,l)
       enddo
       grad(          1:OPRT_nstart-1,k,l,d) = 0.0_RP
       grad(OPRT_nend+1:ADM_gall     ,k,l,d) = 0.0_RP
    enddo
  enddo
enddo
```

The same semantics rewritten with the DSL is shown in Listing 1.2. Instead of iterating across the grid explicitly, a FOREACH loop specifies to run on each element of the grid, the coordinates are encoded in the new cell variable. We reduced the repeated occurrences of the fields with the indices with a 'REDUCE' expression. The Fortran indices are replaced with DSL indices that made it possible to simplify the field access expressions.

Listing 1.2: NICAM DSL code

```
FOREACH cell in grid | g{OPRT_nstart..OPRT_nend}
  do d = 1, ADM_nxyz
        grad(cell,d) = REDUCE(+,N={0..6},
            cgrad(cell%g,cell%l,N,d) * scl(cell%neighbor(N)) )
  enddo
END FOREACH

FOREACH cell in GRID%cells | g{1..OPRT_nstart-1 , OPRT_nend+1 .. gall}
  do d = 1, ADM_nxyz
           grad(cell,d) = 0.0_PRECISION
  enddo
END FOREACH
```

3.4 Workflow and Tool Design

Model code that is based on the DSL along with code of the model in general-purpose language goes through a source-to-source translation process. This step is essential to make the higher-level code ready for processing by the compilers. Our tool is designed in a modular architecture. By applying a configuration, it translates code in a file or a source tree and generates a version of the transformed code. The generated code is the optimized version for a specific machine/architecture according to the configuration. Inter-file optimizations in code trees can also be detected and applied.

The tool is implemented with Python3. Users call the main module with parameters that allow them to control the translation process, e.g. to specify a language module. The code tree is provided to the main module also as an argument.

The main module loads the other necessary modules. The DSL handler module constructs the necessary data structures according to the user-provided configuration file. The source code tree is then parsed into abstract syntax trees (AST). The generated ASTs can be analyzed for optimization among the source files. After all the optimizations/transformations are applied, the resulting code tree is serialized to the output file. Figure 3 shows the design of the translation process.

3.5 Configuration

Configuration files include multiple sections, among which some are essential and others can be added only if needed. Optimization procedures are driven by those configuration sections. The translation tool uses defaults in case optional sections are missing.

Blocking Among the important optimizations that help improve the performance of stencil computations is the optimal use of the caches and memory access. Reusing the data in the caches eliminates the need to read the same data elements repeatedly from memory. Often, data locality can be exploited in stencil computations, allowing for performance improvements.

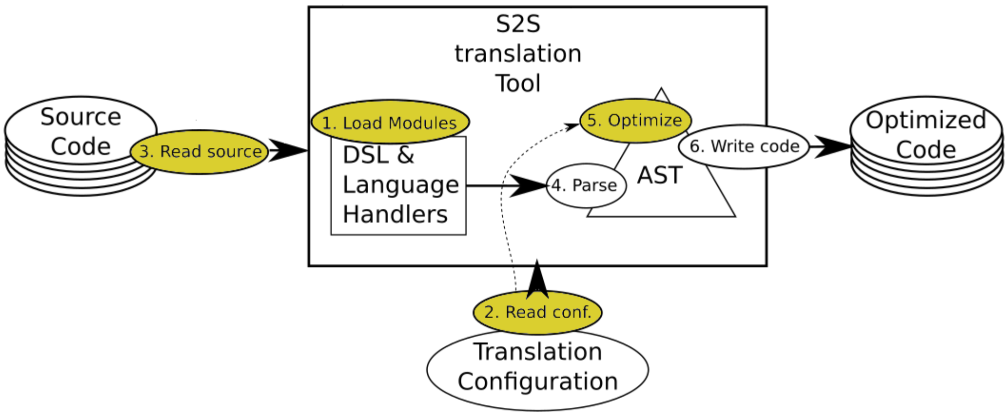


Fig. 3: Translation Process (Yellow components are influenced by the user options)

Cache blocking is a technique to improve the data reuse in caches. Our translation process can apply cache blocking based on the scientific programmer's recommendations. One configuration section is used to allow to specify cache blocking information. The default when this section is missing in a configuration file is to not apply cache blocking.

An example kernel using GGDML in the C programming language is shown in Listing 1.3.

Listing 1.3: Example kernel using C with GGDML

```
foreach c in grid
{
  float df=(f_F[c.east_edge()]-f_F[c.west_edge()])/dx;
  float dg=(f_G[c.north_edge()]-f_G[c.south_edge()])/dy;
  f_HT[c]=df+dg;
}
```

Applying cache blocking using a configuration file defining 10000 elements per block generates the loop code shown in Listing 1.4. The first loop handles completely occupied blocks with 10k elements and the second loop the remainder. In both cases, the loop body (not shown) contains the generated stencil code.

Listing 1.4: Example loop structure for blocking

```
for (size_t blk_start = (0); blk_start < (GRIDX); blk_start+=10000){
    size_t blk_end = GRIDX;
    if ((blk_end - blk_start) > 10000)
        blk_end = blk_start + 10000;
    // Generated loop body
}
#pragma omp simd
for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++) {
  // Generated loop body
}
```

Memory Layout Another important point to optimize memory bandwidth is to optimize the data layout in memory. The temporal and spacial locality of data should lead to access of data in complete cache lines such that it can be prefetched and cached effectively. Thus, data that is accessed in short time frames should be stored closer in memory. To exploit such possibilities, our translation tool provides a flexible layout transformation procedure. The DSL itself abstracts from data placement, however, the translation process generates the actual data accesses. This layout specification is described in configuration files.

Besides to data layout control, the loop nests that access the field data are also subject to user control. The order of the loops that form a nested loop is critical for the optimal data access. Loop order of loops that apply the stencils is also controlled by configuration files.

Listing 1.5 illustrates the resulting code after using a data layout transformation. In this case, a 2D grid is stored in a single-dimensional array.

Listing 1.5: Example code generated with index transformation

```
[...]
#pragma omp for
for (size_t YD_index = (0); YD_index < (local_Y_Cregion); YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++){
    float df = (f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index + 1)+1]
        - f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1]) * invdx;
    float dg = (f_G[((YD_index + 1)+1) * (GRIDX + 3) + (XD_index)+1]
        - f_G[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1]) * invdy;
    f_HT[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1] = df + dg;
   }
}
[...]
```

Inter-Kernel Optimization Cache blocking and memory layout allow improving the use of the caches and memory bandwidth at the level of the kernel. However, the optimal code at the kernel level does not yet guarantee optimal use of caches and memory bandwidth at the application level. Consider the example where two kernels share most of their input data but compute different outputs independently from each other. These kernels could be fused together and benefit from reusing cached data. Note that the benefit is system specific, as the size of the cache and application kernel determine the optimal size for blocking and fusion.

The inter-kernel optimization allows exploiting such data reuse across kernels. To exploit such potential, our translation tool can run an optimizer procedure to detect such opportunities and to apply them according to user decision of whether to apply any of those optimizations or not.

Therefore, the tool analyzes the calls among the code files within the code tree. This analysis leads to a list of call inlining possibilities. The inlining could lead to further optimization through loop fusions. The tool runs automatic analysis including data dependency and code consistency. This analysis detects possible loop fusions that still keep computations consistent. Such loop fusion may lead to optimized use of memory bandwidth and caches. We tested the technique experimentally (refer to Section 4.5) to merge kernels in the application. We could improve the use of caches and hence the performance of the application with 30-48% on different architectures.

The tool provides a list of possible inlining and loop fusion cases. Users choose from the list which case to apply – we anticipate that scientific programmers will make an informed choice for a target platform based on performance analysis tools. According to the choice that the user makes, the tool applies the corresponding transformations automatically.

Listing 1.6 shows two kernels to compute the two components of the flux.

Listing 1.6: Example code with two kernels to compute flux components

```
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion); YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
      XD_index++) {
      f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
      (f_H[YD_index][XD_index] +f_H[YD_index][XD_index -1])/2.0;
   }
}
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion); YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++){
      f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
      (f_H[YD_index][(XD_index] + f_H[YD_index -1][XD_index])/2.0;
   }
}
[...]
```

Listing 1.7 shows the resulting code when the two kernels are merged.

Listing 1.7: Merged version of the flux computation kernels

```
[...]
#pragma omp parallel for
for (size_t YD_index = 0; YD_index < (local_Y_Eregion); YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++) {
    f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
    (f_H[YD_index][XD_index] + f_H[YD_index][(XD_index) + (-1)])/2.0;

    f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
    (f_H[YD_index][(XD_index] + f_H[(YD_index) + (-1)][XD_index])/2.0;
  }
}
[...]
```

Utilizing Distributed Memory Beyond parallelization on the node resources, our techniques allow scaling the same source code that uses GGDML over multiple nodes utilizing distributed memory. This is essential to run modern models on modern supercomputer machines.

The GGDML code is unaware of underlying hardware, and does not need to be modified to run on multiple nodes. Rather, configuration files are prepared to translate the GGDML code into code that is ready to be run on multiple nodes. Configuration files allow domain decomposition to distribute the data and the computation over the nodes. Necessary communication of halo regions is also enabled through configuration files. Scientific programmers can generate simple parallelization schemes, e.g., MPI using blocking communication or sophisticated alternatives like non-blocking communication. When using non-blocking communication, a further optimization is to decouple the computation of the inner region that can be calculated without needing updated halo data and outer regions that require data from another process to be computed.

The translation tool extracts neighborhood information from the GGDML extensions. Such extracted information is analyzed by the tool to decide which halo regions should be exchanged between which nodes. Decisions and information from configuration files allow to generate the necessary code that handles the communication and the synchronization. This guarantees that the necessary data are consistent on the node where the computation takes place.

The parallelization is a flexible technique. No single library is used to handle the parallelization, rather, the communication library is provided through configuration files. Thus, different libraries or library versions can be used for this purpose. We have examined the use of MPI and GASPI as libraries for parallelization.

Listing 1.8 shows the resulting communication code of halo regions between multiple processes – in this case without decoupling of inner and outer area, code with decoupled areas is longer. In this listing, dirty flags are generated to communicate only necessary data. Flags are set global to all processes, and can be checked by each process such that processes that need to do communication can make use of them. This way we guarantee to handle communication properly.

Listing 1.8: Example generated code to handle communication of halo data

```
[...]
//part of the halo exchange code
if (f_G_dirty_flag[11] == 1) {
  if (mpi_world_size > 1) {
    comm_tag++;
    int pp = mpi_rank != 0 ? mpi_rank - 1 : mpi_world_size - 1;
    int np = mpi_rank != mpi_world_size - 1 ? mpi_rank + 1 : 0;
    MPI_Isend(f_G[0], GRIDX + 1, MPI_FLOAT, pp,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[0]);
    MPI_Irecv(f_G[local_Y_Eregion], GRIDX + 1, MPI_FLOAT, np,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[1]);
    MPI_Waitall(2, mpi_requests, MPI_STATUSES_IGNORE);
[...]

#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Cregion); YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++){
   float df = (f_F[YD_index][(XD_index) + (1)]
      - f_F[YD_index][XD_index]) * invdx;
   float dg = (f_G[(YD_index) + (1)][XD_index]
      - f_G[YD_index][XD_index]) * invdy;
   f_HT[YD_index][XD_index] = df + dg;
  }
}
[...]
```

3.6 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 ana-

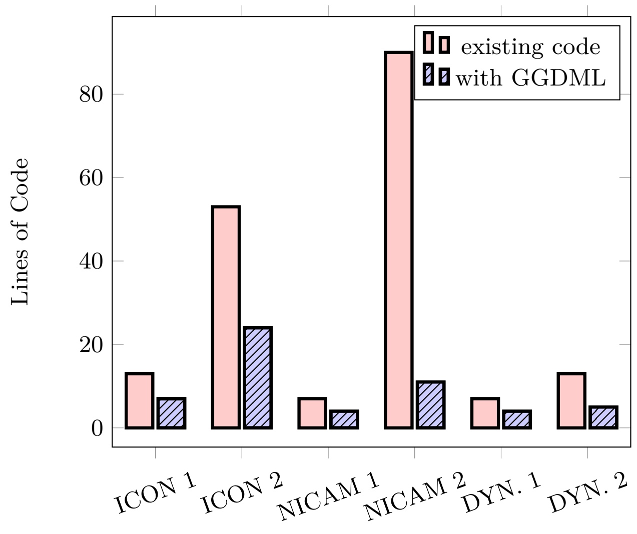


Fig. 4: GGDML impact on the LOC on several scientific kernels [33]

Development Style	Codebase	Effort applied (person month)	Dev. Time (months)	People required	Dev. costs (M€)
Embedded	Fortran	4773	37.6	127	23.9
	DSL	1133	28.8	72	10.4
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 [33]

lyzed the achieved code reduction in terms of lines of code (LOC)[33]. We rewrote the kernels (originally written in Fortran) using GGDML + Fortran. Results are shown in Figure 4.

The average reduction in terms of LOC is 70%, i.e. LOC in GGDML+Fortran in comparison to original Fortran code is 30%. More reduction is noticed in some stencils (NICAM example No.2, reduced to 12%).

Influence on readability and maintainability: Using COCOMO [9] 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 table shows the effort in person month, development time and average number of people (rounded) for three development modes: the embedded model is typically for large project teams a big and complex code base, the organic model for small code and the semi-detached mode for in-between. We assume the semi-detached model is appropriate but as COCOMO was developed for industry projects, we don't want to restrict the development 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.

4 Evaluating Performance of our DSL

To illustrate the performance benefits of using the DSL for modeling, we present some performance measurements that were measured for example codes written with the DSL and translated considering different optimization aspects (configurations). Two different testcodes were used to evaluate the DSL's support for different types of grids: One application uses an unstructured triangular grid, and the other uses a structured rectangular grid that could be applied for code in cubic sphere. Both were written using GGDML (our DSL) in addition to C as the host modeling language.

4.1 Test Applications

Laplacian Solver. The first application code uses an unstructured triangular grid covering the surface of the globe. The application was used in the experiments to apply the Laplacian operator of a field at the cell centers based on field values at neighboring cells. Generally, this code includes fields that are localized at the cell centers, and on the edges of the cells. The horizontal grid of the globe surface is mapped to a 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.

Shallow Water Model. The other code is a shallow water equation solver. It is developed with a structured grid. Structured grids are also important to study for icosahedral modeling, as some icosahedral grids can be structured. Fields are located at centers of cells and on edges between cells. This solver uses the finite difference method. The test code is available online[1]. As part of the testing, we investigate performance portability of code developed using the DSL.

4.2 Test Systems

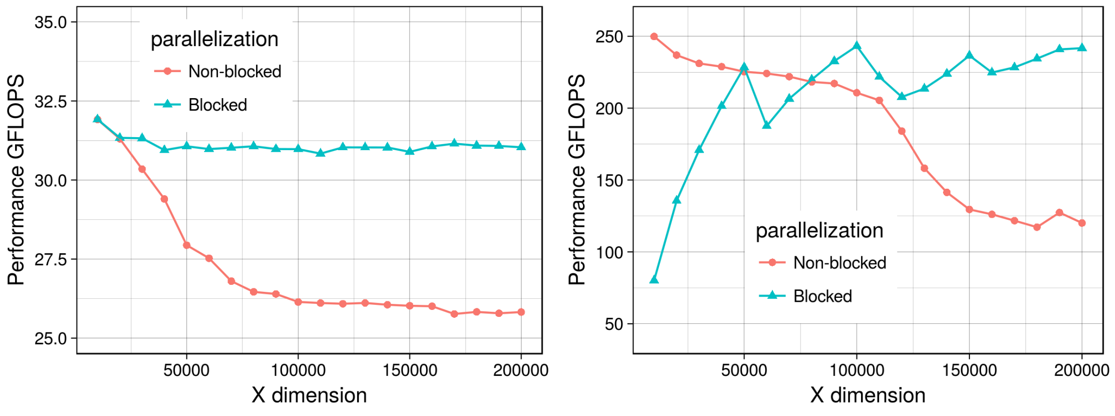
The experiments were executed in different times during the course of the project and used different machines based on availability and architectural features.

- Mistral
  The German Climate Computing Center provides nodes with Intel(R) Xeon(R) E5-2695 v4 (Broadwell) @ 2.1GHz processors.
- Piz Daint
  The Swiss supercomputer provides nodes equipped with two Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz processors and NVIDIA(R) Tesla(R) P100 GPUs.
- NEC test system
  Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz Broadwell processors with Aurora vector engines.

4.3 Evaluating Blocking

To explore the benefit of organizing memory accesses efficiently across architectures, experiments using the shallow water equation solver code were conducted on Piz Daint. First, we generated code versions with and without blocking for the Broadwell processor and the P100 GPU. An excerpt of results presented for different size of grids is shown in [30] and in Figure 5a. The

[1] `https://github.com/aimes-project/ShallowWaterEquations`



(a) Broadwell CPU (b) P100 GPU

Fig. 5: Impact of blocking: Performance measurements with variable grid width

experiment was done with grid width of 200K. In the paper, we investigated the influence of the blocking factor on the application kernel further revealing that modest block sizes are leading to best performance (256 to 10k for CPU and 2-10k for GPU). On both architectures, wider grids run less efficiently as a results of an inefficient use of caches. The GPU implies additional overhead for the blocked version, requiring to run with a sufficient large grid to benefit from it. This also shows the need to dynamically turn on/off blocking factors depending on the system capabilities.

4.4 Evaluating Vectorization and Memory Layout Optimization

As part of [29], we evaluated techniques to apply memory layout transformations. The experiments were done on two architectures, the Broadwell multi-core processors and Aurora vector engine on the NEC test platform using the shallow water equation solver code.

We used alternative layouts with different distances between data elements to investigate the impact of the data layout on the performance. The explored data alternatives were data accesses to

- contiguous unit stride arrays
- interleaved data with constant short distance separating data elements, 4 bytes separating each consecutive elements. This allowed to emulate array of structures (AoS) layouts.
- scattered data elements separated with long distances

The results are listed in brief in Table 2. Using memory profilers, we found that the contiguous unit-stride code allowed to use the memory throughput efficiently on both architectures. In the emulated AoS codes, the efficiency dropped on both architectures. The worst measurements were taken for the scattered data elements.

Besides to the impact on the optimization of the use of the memory bandwidth, vectorization is also affected by those alternatives. AVX2 was used for all kernels on Broadwell for the unit-stride code. Similarly, the vector units of the vector engine showed best results with this layout. Again the use of the vectorization was degraded with the emulated AoS, and was even worse with the scattered data elements.

Architecture	Scattered	Short distance	Contiguous
Broadwell	3 GFlops	13 GFlops	25 GFlops
NEC Aurora	80 GFlops	161 GFlops	322 GFlops

Table 2: Performance of memory layout variants on CPU and the NEC vector architecture

Architecture	Theoretical Memory bandwidth (GB/s)	Before merge		With Inter-Kernel Merging	
		Measured memory throughput (GB/s) and peak	GFlops	Measured memory throughput (GB/s) and peak	GFflops
Broadwell	77	62 (80%)	24	60 (78%)	31 (+30%)
P100 GPU	500	380 (76%)	149	389 (78%)	221 (+48%)
NEC Aurora	1,200	961 (80%)	322	911 (76%)	453 (+40%)

Table 3: Performance and efficiency of the kernel fusioning on all three architectures

4.5 Evaluating Inter-Kernel Optimization

To explore the benefit of kernel merging, experiments were done using the shallow water equation solver code on the NEC testsystem (for vector engines) and Piz Daint (for GPUs and CPUs) [30]. The performance of regular and merged kernels are shown in Table 3. The code achieves near optimal use of the memory bandwidth already before the merge and actually declines slightly after the merge. Exploiting the inter-kernel cache reuse allowed to reduce the data access in memory and increased the total performance of the application by 30-50%.

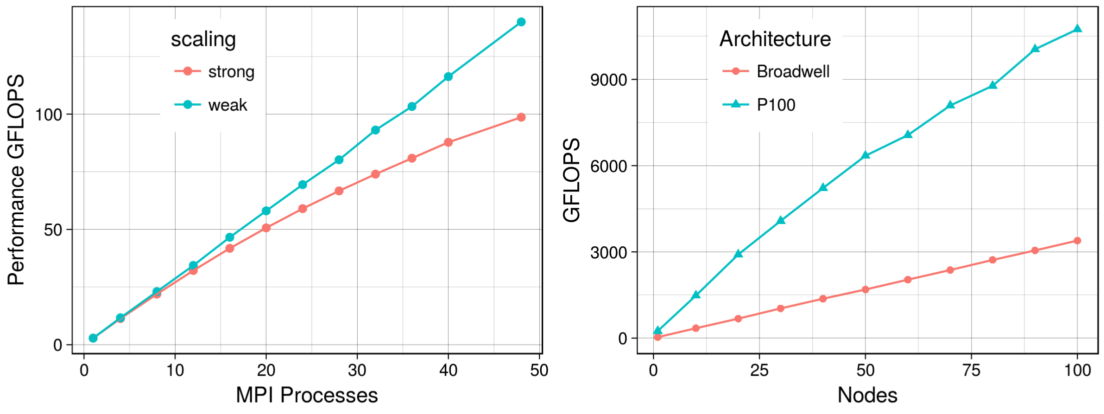
4.6 Scaling with Multiple-Node Runs

To demonstrate the ability of the DSL to support global icosahedral models, we carried out experiments using the two applications. Scalability experiments of unstructured grid code were run on Mistral. Shallow water equation solver code experiments were run on Mistral for CPU tests, and on Piz Daint for GPU tests.

In the experiment using the unstructured grid, we use the global grid of the application and apply a three-dimensional 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 the strong-scale analysis. Weak scalability experiments were based on this number of points for each node. Figure 6a shows the results.

We could do further numbers of nodes, however, we found that the code was scaling with the tested cases and further experiments needed resources and time to get jobs to run on the test machine. For the measured cases, the weak scalability of the code is close to optimal. 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.

We also carried out experiments to scale the shallow water equation solver on both Broadwell multi-core processors and on the P100 GPUs at Piz Daint. We generated code for Broadwell experiments with OpenMP as well as MPI, and for the GPUs with OpenACC as well as MPI. Figure 6b shows the measured results of scaling the code. On the Broadwell processor, we used 36 OpenMP threads on each node.



(a) Icosahedral code [31] (b) Shallow water solver [32]

Fig. 6: Scalability of the two different models (measured on Mistral; P100 on Piz Daint)

While the performance of the GPU code scaled well, it loses quite some efficiency when running on two processes as the halo communication involves the host code. In general, the code that the tools generate for multiple nodes shows to be scalable, both on CPUs and GPUs. The DSL code does not include any details regarding single or multiple node configuration, so users do not need to care about multiple node parallelization. However, the parallelization can still be applied by the tool and the users can still control the parallelization process.

4.7 Exploring the Use of Alternative DSLs: GridTools

The GridTools framework is a set of libraries and utilities to develop performance-portable weather and climate applications. It is developed at The Swiss National Supercomputing Center [2]. To achieve the goal of performance portability, the user-code is written in a generic form which is then optimized for a given architecture at compile-time. The core of GridTools is the stencil composition module which implements a DSL embedded in C++ for stencils and stencil-like patterns. Further, GridTools provides modules for halo exchanges, boundary conditions, data management and bindings to C and Fortran. GridTools is successfully used to accelerate the dynamical core of the COSMO model with improved performance on CUDA-GPUs compared to the current official version, demonstrating production quality and feature-completeness of the library for models on lat-lon grids [49]. Although GridTools was developed for weather and climate applications it might be applicable for other domains with a focus on stencil-like computations.

In the context of AIMES project, we evaluated the viability of using GridTools for the dynamical core of NICAM: namely NICAM-DC. Since NICAM-DC is written in Fortan, we first had to port the code to C++, which includes also changing the build systems. Figure 1.9 and Figure 1.10 show simple examples codes extracted from NICAM-DC and ported to GridTools, respectively. We ported the dynamical core using the following incremental approach. First, each operator was ported individually to GridTools, i.e. re-written from Fortran to C++. Next, we used a verification tool to assure the same input to the C++ and Fortran version give the same output. Next we move on to the following operator. Table 4 shows results from benchmarks extracted from NICAM-DC. It provides good speedup on GPU, and speed on CPU (in openMP) comparable to the hand-written version. Figure 7 shows results for running all operators of NICAM-DC on 10

Listing 1.9: Example of the diffusion operator extracted from NICAM-DC

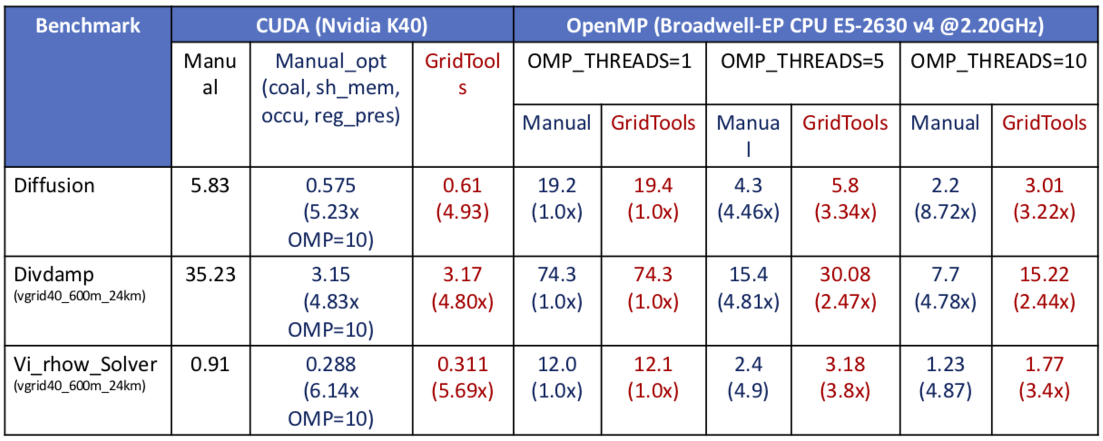
```
do d = XDIR, ZDIR
do j = jmin-1, jmax
do i = imin-1, imax
  vt(i,j,d) = (( + 2.0_RP * coef(i,j,d,1) &
                 - 1.0_RP * coef(i,j,d,2) &
                 - 1.0_RP * coef(i,j,d,3) ) * scl(i,j,d) &
             +  ( - 1.0_RP * coef(i,j,d,1) &
                 + 2.0_RP * coef(i,j,d,2) &
                 - 1.0_RP * coef(i,j,d,3) ) * scl(i+1,j,d) &
             +  ( - 1.0_RP * coef(i,j,d,1) &
                 - 1.0_RP * coef(i,j,d,2) &
                 + 2.0_RP * coef(i,j,d,3) ) * scl(i,j+1,d) &
            ) / 3.0_RP
enddo
enddo
enddo
```

Listing 1.10: Example of the diffusion operator ported to GridTools

```
template <typename evaluation>
  GT_FUNCTION
  static void Do(evaluation const & eval, x_interval) {
    eval(vt{}) =  (( + 2.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl{})
                 +  ( - 1.0 * eval(coef{})
                     + 2.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl{i+1})
                 +  ( - 1.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     + 2.0 * eval(coef{a+2}) ) * eval(scl{j+1})
                 ) / 3.0;
}
```

nodes. It is worth mentioning that the most time consuming operator is more than 7x faster on GPU versus CPU.

GridTools demonstrates good GPU performance and acceptable CPU performance. The functionalities and features included in GridTools were enough to support the regular mesh code of NICAM-DC without friction (i.e. no custom features were required in GridTools to support the requirements of NICAM-DC). In addition, GridTools, are transparent in the sense that no information about the platform is exposed to the end-user. On the other hand, following is a list of issues that requires one's consideration when using GridTools: First, the requirement of rewriting the entire dynamical core in C++ is not a trivial task, specially that C++ templates make the code more convoluted, in comparison to Fortran. Second, while GridTools as a stencil framework does a good job for the dynamical core, separate solutions are required for the physics



Benchmark	CUDA (Nvidia K40)			OpenMP (Broadwell-EP CPU E5-2630 v4 @2.20GHz)					
	Manual	Manual_opt (coal, sh_mem, occu, reg_pres)	GridTools	OMP_THREADS=1		OMP_THREADS=5		OMP_THREADS=10	
				Manual	GridTools	Manual	GridTools	Manual	GridTools
Diffusion	5.83	0.575 (5.23x OMP=10)	0.61 (4.93)	19.2 (1.0x)	19.4 (1.0x)	4.3 (4.46x)	5.8 (3.34x)	2.2 (8.72x)	3.01 (3.22x)
Divdamp (vgrid40_600m_24km)	35.23	3.15 (4.83x OMP=10)	3.17 (4.80x)	74.3 (1.0x)	74.3 (1.0x)	15.4 (4.81x)	30.08 (2.47x)	7.7 (4.78x)	15.22 (2.44x)
Vi_rhow_Solver (vgrid40_600m_24km)	0.91	0.288 (6.14x OMP=10)	0.311 (5.69x)	12.0 (1.0x)	12.1 (1.0x)	2.4 (4.9)	3.18 (3.8x)	1.23 (4.87)	1.77 (3.4x)

Table 4: Execution time (seconds) of different benchmarks extracted from NICAM-DC. This includes the regular regions only, using 1 region, a single MPI rank for a 130x130x42 grid.

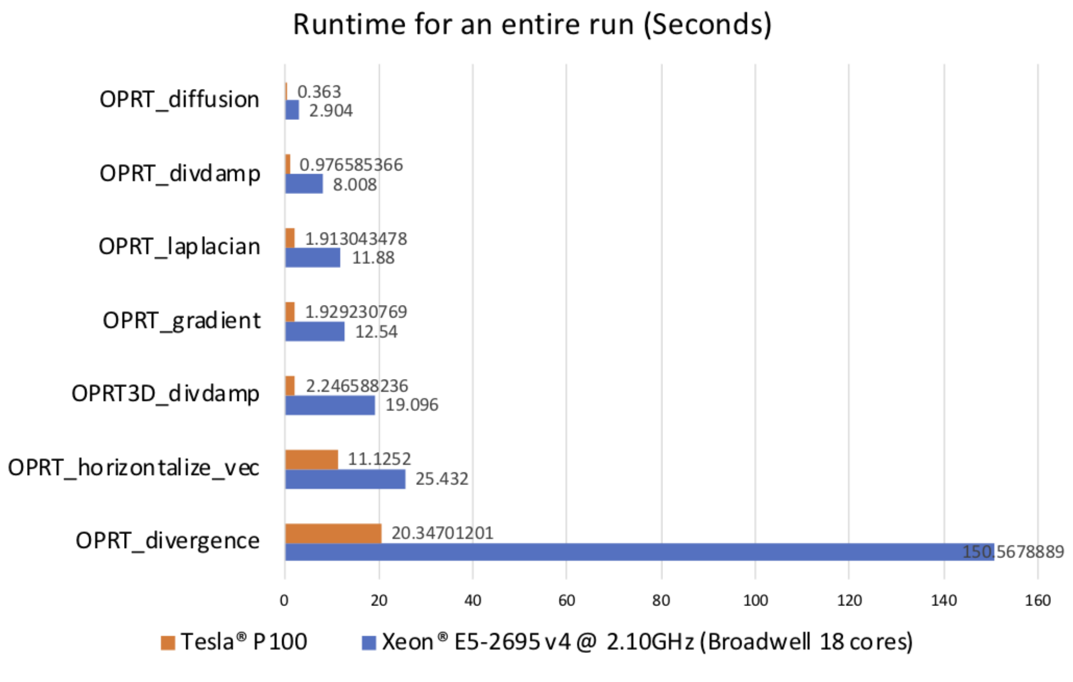


Fig. 7: NICAM-DC operators. Running on 10 nodes with one MPI rank per node. P100 is running GridTools generated kernels and Xeon uses the original Fortran code. Total grid is 32x32x10 using 40 vertical layers

modules, the communicator module, and the non-regular compute modules (e.g. polar regions). Using different solutions inside the same codebase typically increases the friction between code modules. Third, the interfacing between Fortran and C++ is non-trivial and can be troublesome considering that not all end-users are willing to change their build process.

5 Massive I/O and Compression

5.1 The Scientific Compression Library (SCIL)

The developed compression library SCIL [37] provides a framework to compress structured and unstructured data using the best available (lossless or lossy) compression algorithms according to the definition of tolerable loss of accuracy and required performance. SCIL acts as a meta-compressor providing various backends such as algorithms like LZ4, ZFP, SZ but also integrates some alternative algorithms.

The data path of SCIL is illustrated in Figure 8. An application can either use NetCDF4 [46][2], HDF5 [20] or directly the C-interface of SCIL. Based on the defined quantities, the values and the characteristics of the data to compress, the appropriate compression algorithm is chosen. SCIL also comes with a library to generate various synthetic test patterns for compression studies, i.e., well-defined multi-dimensional data patterns of any size. Further tools are provided to plot, to add noise or to compress CSV and NetCDF3 files.

5.2 Supported Quantities

There are three types of quantities supported:

Accuracy quantities define the tolerable error on lossy compression. When compressing the value v to $\hat{v}$ it bounds the residual error ($r = v - \hat{v}$):

- **absolute tolerance**: $v - \text{abstol} \leq \hat{v} \leq v + \text{abstol}$
- **relative tolerance**: $v/(1 + reltol) \leq \hat{v} \leq v \cdot (1 + reltol)$
- **relative error finest tolerance**: used together with rel tolerance; absolute tolerable error for small v's. If relfinest $> |v \cdot (1 \pm reltol)|$, then $v - \text{relfinest} \leq \hat{v} \leq v + \text{relfinest}$
- **significant digits**: number of significant decimal digits
- **significant bits**: number of significant digits in bits

SCIL must ensure that all the set accuracy quantities are honored regardless of the algorithm chosen, meaning that one can set, e.g., absolute and relative tolerance and the strictest of the criteria is satisfied.

Performance quantities define the expected performance behavior for both compression and decompression (on the same system). The value can be defined according to: 1) absolute throughput in MiB or GiB; or 2) relative to network or storage speed. It is considered to be the expected performance for SCIL but it may not be as strictly handled as the qualities – there may be some cases in which performance is lower. Thus, SCIL must estimate the compression rates for the data.

[2] HDF5 and NetCDF4 are APIs and self-describing data formats for storing multi-dimensional data with user-relevant metadata.

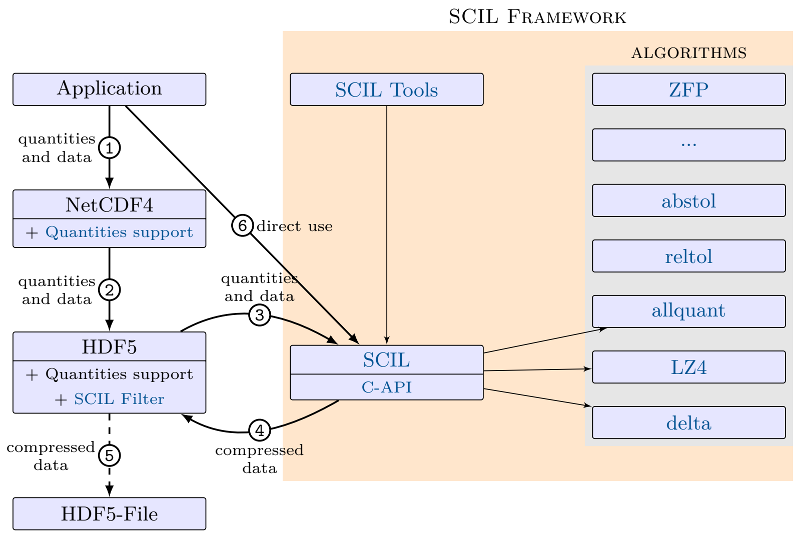


Fig. 8: SCIL compression path and components (extended)[37]

Supplementary quantities: An orthogonal quantity that can be set is the so called *fill value*, a value that scientists use to mark special data points. This value must be preserved accurately and usually is an specific high or low value that may disturb a smooth compression algorithm.

5.3 Compression chain

Internally, SCIL creates a compression chain which can involve several compression algorithms as illustrated in Figure 9. Based on the basic datatype that is supplied, the initial stage of the chain is entered. Algorithms may be preconditioners to optimize data layout for subsequent compression algorithms, converters from one data format to another, or, on the final stage, a lossless compressor. Floating-point data can be first mapped to integer data and then to a byte stream. Intermediate steps can be skipped.

5.4 Algorithms

SCIL comes with additional algorithms that are derived to support one or multiple accuracy quantities set by the user. For example, the algorithms Abstol (for absolute tolerance), Sigbits (for significant bits/digits), and Allquant. These algorithms aim to pack the number of required bits as tightly as possible into the data buffer but operate on each value independently. While Abstol and Sigbits just consider one quantity, Allquant considers all quantities together and chooses the required operation for a data point depending on the highest precision needed. We also consider these algorithms to be useful baselines when comparing any other algorithm. ZFP and SZ, for example, work on one quantity, too.

During the project, we explored the implementation for the automatic algorithm selection but only integrated a trivial scheme for the following reasons: If only a single quantity is set, we found out that the optimal parameter depends on many factors (features); the resulting optimal choice is embedded in a multi-dimensional space – this made it infeasible to identify the optimal algorithm. Once more than a single quantity is set, only one of the newly integrated algorithms can perform the compression, which eliminates any choice. As the decoupling of SCIL enables

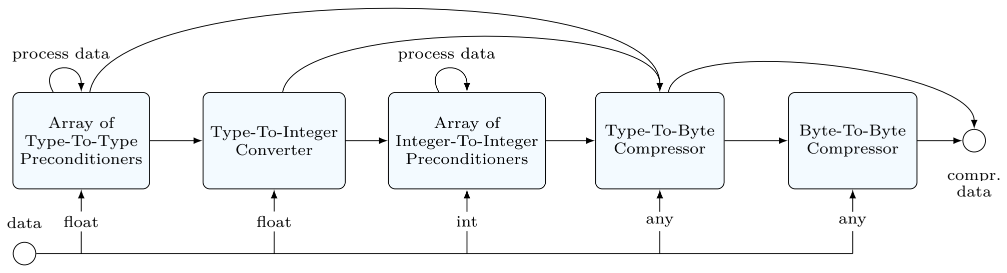

process data
process data
Array of Type-To-Type Preconditioners
Type-To-Integer Converter
Array of Integer-To-Integer Preconditioners
Type-To-Byte Compressor
Byte-To-Byte Compressor
compr. data
data
float
float
int
any
any

Fig. 9: SCIL compression chain. The data path depends on the input data type [37].

to integrate algorithms in the future, we hope that more algorithms will be developed that can then benefit from implementing the automatic selection.

6 Evaluation of the Compression Library SCIL

We evaluated the performance and compression ratio of SCIL against several scientific datasets and the synthetic test patterns generated by SCIL itself [37].

6.1 Single Core Performance

In the following, an excerpt of the experiments conducted with SCIL on a single core is shown. These results help to understand the performance behavior of compression algorithms and their general characteristics.

Four data sets were used each with precision floating-point data (32 bit): 1) The data created with the SCIL pattern library (10 data sets each with different random seed numbers). The synthetic data has the dimensionality of (300 x 300 x 100 = 36 MB). 2) The output of the ECHAM atmospheric model [47] which stored 123 different scientific variables for a single timestep as NetCDF. 3) The output of the hurricane Isabel model which stored 633 variables for a single timestep as binary[3]. 4) The output of the NICAM Icosahedral Global Atmospheric model which stored 83 variables as NetCDF.

The characteristics of the scientific data varies and so does data locality within the data sets. For example, in the Isabel data many variables are between 0 and 0.02 many between -80 and +80 and some are between -5000 and 3000.

We set only one quantity to allow using ZFP and SZ for comparison. Table 5 shows the harmonic mean compression ratio[4] for setting an absolute error of 1% of the maximum value or setting precision to 9 bit accuracy. The harmonic mean corresponds to the total reduction and performance when compressing/decompressing all the data.

The results for compressing 11 variables of the whole NICAM model via the NetCDF API are shown in Figure 10. The x-axis represents the different data files, as each file consists of several chunks, a point in the y-axis represents one chunk. It can be observed that generally the SZ algorithm yields the best compression ratio but Abstol+LZ4 yield second best ratio providing much better and predictable compression and decompression speeds.

Note that for some individual variables, one algorithm may supersede another in terms of ratio. As expected there are cases in which one algorithm is outperforming the other algorithms

[3] `http://vis.computer.org/vis2004contest/data.html`
[4] The ratio is the resulting file size divided by the original file size.

	Algorithm	Ratio	Compr. MiB/s	Decomp. MiB/s
NICAM	abstol	0.206	499	683
	abstol,lz4	0.015	458	643
	sz	0.008	122	313
	zfp-abstol	0.129	302	503
ECHAM	abstol	0.190	260	456
	abstol,lz4	0.062	196	400
	sz	0.078	81	169
	zfp-abstol	0.239	185	301
Isabel	abstol	0.190	352	403
	abstol,lz4	0.029	279	356
	sz	0.016	70	187
	zfp-abstol	0.039	239	428
Random	abstol	0.190	365	382
	abstol,lz4	0.194	356	382
	sz	0.242	54	125
	zfp-abstol	0.355	145	241

(a) 1% absolute tolerance

	Algorithm	Ratio	Compr. MiB/s	Decomp. MiB/s
NICAM	sigbits	0.439	257	414
	sigbits,lz4	0.216	182	341
	zfp-precision	0.302	126	182
ECHAM	sigbits	0.448	462	615
	sigbits,lz4	0.228	227	479
	zfp-precision	0.299	155	252
Isabel	sigbits	0.467	301	506
	sigbits,lz4	0.329	197	366
	zfp-precision	0.202	133	281
Random	sigbits	0.346	358	511
	sigbits,lz4	0.348	346	459
	zfp-precision	0.252	151	251

(b) 9 bits precision

Table 5: Harmonic mean compression performance for different scientific data sets

in terms of compression ratio which justifies the need for a metacompressor like SCIL that can make smart choices on behalf of the users. Some algorithms perform generally better than others in terms of performance. Since in our use cases, users define the tolerable error, we did not investigate metrics that compare the precision for a fixed compression ratio (e.g., the signal to noise ratio).

While a performance of 200 MB/s may look insufficient for a single core, with 24 cores per node a good speed per node can be achieved that is still beneficial for medium large runs on shared storage. For instance, consider a storage system that can achieve 500 GB/s. Considering that one node with typical Infiniband configuration can transfer at least 5 GB/s, 100 client nodes saturate the storage. By compressing 5:1 (or ratio of 0.2), virtually, the storage could achieve a peak performance of 2,500 GB/s, and, thus, can serve up to 500 client nodes with (theoretical) maximum performance.

Trading of storage capacity vs. space is an elementary issue to optimize bigger workflows. By separating the concerns between the necessary data quality as defined by scientists and compression library, site-specific policies could be employed that depend also on the available hardware.

6.2 Compression in HDF5 and NetCDF

We tested the compression library SCIL using the icosahedral grid code. The code can use NetCDF to output the generated data periodically to output the fields. In this experiment, a high-resolution grid with 268 million grid cells (single precision floating point) in the horizontal times 64 vertical levels was used and executed on the supercomputer Mistral. The code was run on 128 nodes, with one MPI process per node. It wrote one field to the output file in one timestep. The field values range between -10 to +55 which is important for understanding the impact of the compression.

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 6: Compression results of 128 processes on Mistral

The experiments varied the basic compression algorithms and parameters provided by SCIL. Compression is done with the algorithms

– memcopy: does not do any real compression, but allows to measure the overhead of the usage of enabling HDF5 compression and SCIL.
– lz4: the well-known compression algorithm. It is unable to compress floating-point data but slows down the execution.
– abstol,lz4: processes data elements based on the absolute value of each point, we control the tolerance by the parameter *absolute_tolerance*, after quantization an LZ4 compression step is added.
– sigbits,lz4: processes data elements based on a percentage of the value being processed, we control the tolerance by the parameter *relative_tolerance_percent*, after quantization an LZ4 compression step is added.

The results of this selection of compression methods is shown in Table 6, it shows the write time in seconds and resulting data size in GB, a virtual throughput relative to the uncompressed file size, and the speedup. 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 metadata. Internally, HDF5 requires additional data buffers. This leads to extra overhead in the compression slowing down the I/O (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.

Remember that these results serve as feasibility study. One of our goals was to provide a NetCDF backwards compatible compression method not to optimize the compression data path inside HDF5. The SCIL library can be used directly by existing models avoiding the overhead and leading to the results as shown above.

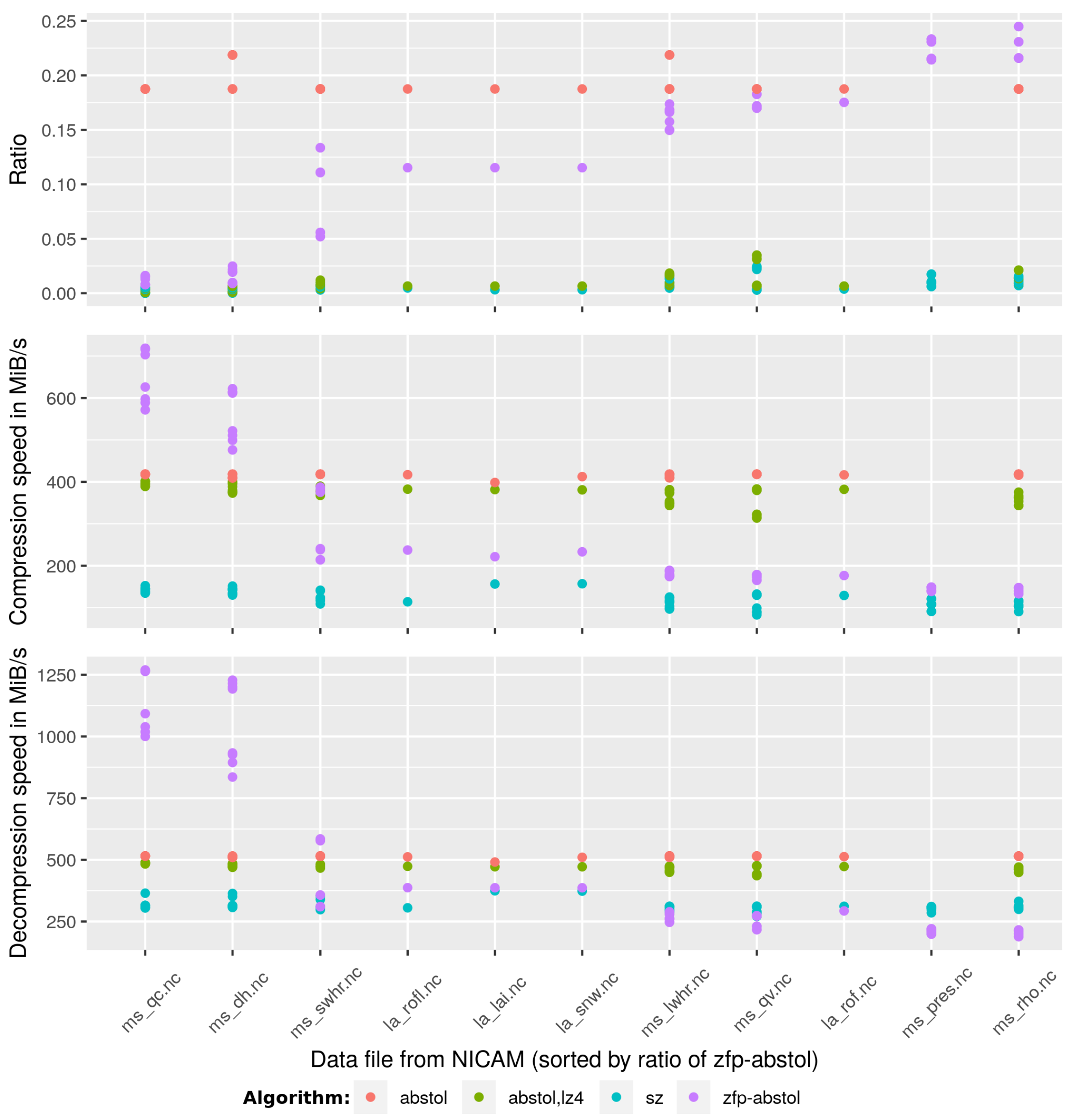


Fig. 10: Compressing various climate variables with absolute tolerance 1%

7 Standardized Icosahedral Benchmarks

Our research on DSL and I/O is more practical. We started with real-world applications, namely three global atmospheric models developed by the participating countries. The global atmospheric model with the icosahedral grid system is one of the new generation global climate/weather models. The grid-point calculations, which are less computational intense than the spherical harmonics transformation, are used in the model. On the other hand, patterns of the data access in differential operators are more complicated than the traditional limited-area atmospheric model with the Cartesian grid system.

There are different implementations of the dynamical core on the icosahedral grid: direct vs. indirect memory access, staggered vs. co-located data distribution, and so on. The objective of standardization of benchmarks is to provide a variety of computational patterns of the icosahedral atmospheric models. The kernels are useful for evaluating the performance of new machines and new programming models. Not only for our studies but also for the existing/future DSL studies, this benchmark set provides various samples of the source code. The icosahedral grid system is an unstructured grid coordinate, and there are a lot of challenging issues about data decomposition, data layout, loop structures, cache blocking, threading, offloading the accelerators, and so on. By applying DSLs or frameworks to the kernels, the developers can try the detailed, practical evaluation of their software. The benchmarks were used in the final evaluation of SCIL and they steered the DSL development by providing the relevant patterns.

7.1 IcoAtmosBenchmark v1: Kernels from Icosahedral Atmospheric Models

IcoAtmosBenchmark v1 is the package of kernels extracted from three Icosahedral Global Atmospheric models, NICAM, ICON, DYNAMICO. As shown in Figure 11, we prepared input data and reference output data for easy validation of results. We also arranged documentation about the kernels. The package is available online[5].

Documentation An excerpt to the NICAM kernel serves as an example. The icosahedral grid on the sphere of NICAM is the unstructured grid system. In NICAM code, the complex topology of the grid system is separated into the structured and unstructured part. The grids are decomposed into tiles, and one or more tiles are allocated to each MPI process. The horizontal grids in the tile are kept in a 2-dimensional structure. On the other hand, a topology of the network of the tiles is complex. We selected and extracted 6+1 computational kernels from the dynamical core of NICAM, as samples of the stencil calculation on the structured grid system. We also extracted a communication kernel, as a sample of halo exchange in the complex node topology. The features of each kernel are documented on the GitHub page.

All kernels are single subroutines and almost the same as the source codes in the original model, except to the ICON kernels. They are put into the wrapper for the kernel program. Values of input variables in the argument list of the kernel subroutine are stored as a data file, just before the call in the execution of the original model. They are read and given to the subroutine in the kernel program. Similarly, the values of output variables in the argument list are stored, just after the call in execution. They are read and compared to the actual output values of kernel subroutine. The differences are written to the standard output for validation. For easy handling of the input/reference data by both the Fortran program and C program, we prepared an I/O routine written in C.

[5] `https://aimes-project.github.io/IcoAtmosBenchmark_v1/`

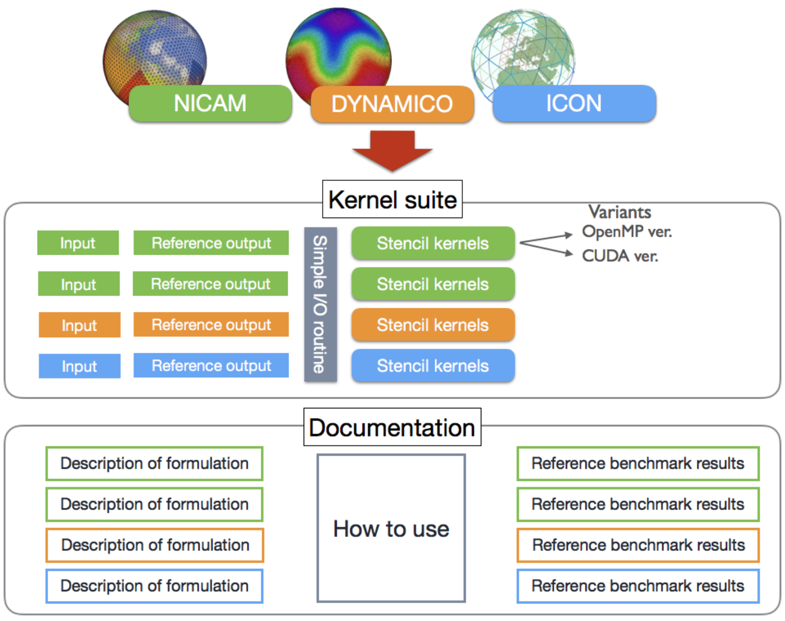


Fig. 11: Overview of IcoAtmosBenchmark v1

We provided a user manual, which contains the brief introduction of each model, the description of each kernel, usage of kernel programs, and sample results. This information is helpful for users of this kernel suite in the future.

8 Summary and Conclusion

The numerical simulation of climate and weather is demanding for computational and I/O resources. Within the AIMES project, we addressed those challenges and researched approaches that foster the separation of concerns. This idea unites our approaches for the DSL and the compression library. While a higher-level of abstraction can improve the productivity for scientists, most importantly the decoupling of requirements from the implementation allows scientific programmers to develop and improve architecture-specific optimizations.

Promising candidates for DSLs have been explored and with GGDML an alternative has been developed that covers the most relevant formulations of the three icosahedral models: DYNAMICO, ICON, and NICAM. The DSL and toolchain we developed integrates into existing code bases and suits for incremental reformulation of the code. We estimated the benefit for code reduction and demonstrated several optimizations for CPU, GPU, and vector architectures. Our DSL allows to reduce code to around 30% of the LOC in comparison to code written with GPL code.

With the semantics of GGDML, we could achieve near optimal use of memory hierarchies and memory throughput which is critical for the family of computations in hand. Our experiments show running codes with around 80% of achievable memory throughput on differnt archiectures. Furthermore, we could scale models to multiple nodes, which is essential for exascale computing, using the same code that is used for a single node. The separation of concerns in our approach allowed us to keep models code separate of underlying hardware changes. The single GGDML source code is used to generate code for the different architectures and on single vs. multiple nodes.

To address the I/O challenge, we developed the library SCIL, a metacompressor supporting various lossy and lossless algorithms. It allows users to specify various quantities for the tolerable error and expected performance, and allows the library to chose a suitable algorithm. SCIL is a stand-alone library but also integrates into NetCDF and HDF5 allowing users to explore the benefits of using alternative compression algorithms with their existing middleware. We evaluated the performance and compression ratio for various scientific datasets and on moderate scale. The results show that the choice of the best algorithm depends on the data and performance expectation which, in turn, motivates the need for the decoupling of quantities from the selection of the algorithm. A blocker for applying parallel large-scale compression in existing NetCDF workflows is the performance limitation of the current HDF5 stack.

Finally, benchmarks and mini-applications were created that represent the key features of the icosahedral applications.

Beside the achieved research in the AIMES project, the work done opens the door for further research in the software engineering of climate and weather prediction models. The performance portability, where we used the same code to run on different architectures and machines, including single and multiple nodes, shows that techniques to be viable to continue the research in this direction. The code reduction offered by DSLs promise to save million in development cost that can be used to contribute to the DSL development. During the runtime of the project, it became apparent that the concurrently developed solutions GridTools and PSYclone that also provide such features are preferred by most scientists as they are developed by a bigger community and supported by national centers. We still believe that the developed light-weight DSL solution provides more flexibility particularly for smaller-sized models and can be maintained as part of the development of the models itself. It also can be used in other contexts providing domain-specific templates to arbitrary codes.

In the future, we aim to extend the DSL semantics to also address I/O relevant specifications. This would allow to unify the effort towards optimal storage and computation.

References

1. CSCS Claw. `http://www.xcalablemp.org/download/workshop/4th/Valentin.pdf`.
2. CSCS GridTools. `https://github.com/GridTools/gridtools`.
3. Robert A van Engelen. Atmol: A domain-specific language for atmospheric modeling. *CIT. Journal of computing and information technology*, 9(4):289–303, 2001.
4. Samantha V Adams, Rupert W Ford, M Hambley, JM Hobson, I Kavčič, CM Maynard, T Melvin, Eike Hermann Müller, S Mullerworth, AR Porter, et al. LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models. *Journal of Parallel and Distributed Computing*, 2019.
5. Yevhen Alforov, Anastasiia Novikova, Michael Kuhn, Julian Kunkel, and Thomas Ludwig. Towards Green Scientific Data Compression Through High-Level I/O Interfaces. In *30th International Symposium on Computer Architecture and High Performance Computing*, pages 209–216. Springer, 02 2019.
6. Allison H Baker, Dorit M Hammerling, Sheri A Mickelson, Haiying Xu, Martin B Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, Christian N Gencarelli, John M. Dennis, Jennifer E. Kay, and Peter Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development, 9*, pages 4381–4403, 07 2016.
7. Allison H Baker, Haiying Xu, John M Dennis, Michael N Levy, Doug Nychka, Sheri A Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 203–214. ACM, 2014.
8. Thomas Josef Baron, Kirill Khlopkov, Thomas Pretorius, Daniel Balzani, Dominik Brands, and Jörg Schröder. Modeling of microstructure evolution with dynamic recrystallization in finite element simulations of martensitic steel. *steel research international*, 87(1):37–45, 2016.
9. Boehm Barry et al. Software engineering economics. *New York*, 197, 1981.
10. Peter Bastian, Christian Engwer, Jorrit Fahlke, Markus Geveler, Dominik Göddeke, Oleg Iliev, Olaf Ippisch, René Milk, Jan Mohring, Steffen Müthing, et al. Advances concerning multiscale methods and uncertainty quantification in EXA-DUNE. In *Software for Exascale Computing-SPPEXA 2013-2015*, pages 25–43. Springer, 2016.
11. Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47, 2015.
12. Simon Bauer, Daniel Drzisga, Marcus Mohr, U Rüde, Christian Waluga, and Barbara Wohlmuth. A stencil scaling approach for accelerating matrix-free finite element implementations. *SIAM Journal on Scientific Computing*, 40(6):C748–C778, 2018.
13. Simon Bauer, Markus Huber, S Ghelichkhan, Marcus Mohr, Ulrich Rüde, and B Wohlmuth. Large-scale simulation of mantle convection based on a new matrix-free approach. *Journal of Computational Science*, 31:60–76, 2019.
14. Tekin Bicer and Gagan Agrawal. A Compression Framework for Multidimensional Scientific Datasets. *Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 2250–2253, 05 2013.
15. Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, et al. Liszt: a domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 9. ACM, 2011.
16. Sheng Di and Franck Cappello. Fast Error-bounded Lossy HPC Data Compression with SZ. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 730–739. IEEE, 2016.
17. Romain Dolbeau, Stéphane Bihan, and François Bodin. Hmpp: A hybrid multi-core parallel programming environment. In *Workshop on general purpose processing on graphics processing units (GPGPU 2007)*, volume 28, 2007.
18. Thomas Dubos, Sarvesh Dubey, Marine Tort, Rashmi Mittal, Yann Meurdesoif, and Frédéric Hourdin. Dynamico, an icosahedral hydrostatic dynamical core designed for consistency and versatility. *Geoscientific Model Development Discussions*, 8(2), 2015.

19. Sara Faghih-Naini, Sebastian Kuckuk, Vadym Aizinger, Daniel Zint, Roberto Grosso, and Harald Köstler. Towards whole program generation of quadrature-free discontinuous Galerkin methods for the shallow water equations. *arXiv preprint arXiv:1904.08684*, 2019.
20. Mike Folk, Albert Cheng, and Kim Yates. HDF5: A file format and I/O library for high performance computing applications. In *Proceedings of supercomputing*, volume 99, pages 5–33, 1999.
21. Anja Gerbes, Nabeeh Jumah, and Julian Kunkel. Automatic Profiling for Climate Modeling, 04 2018.
22. Anja Gerbes, Julian Kunkel, and Nabeeh Jumah. Intelligent Selection of Compiler Options to Optimize Compile Time and Performance, 03 2017.
23. Leonardo A. Bautista Gomez and Franck Cappello. Improving floating point compression through binary masks. *2013 IEEE International Conference on Big Data*, 10 2013.
24. Tobias Gysi, Oliver Fuhrer, Carlos Osuna, Benjamin Cumming, and Thomas Schulthess. Stella: A domain-specific embedded language for stencil codes on structured grids. In *EGU General Assembly Conference Abstracts*, volume 16, 2014.
25. Mario Heene, Alfredo Parra Hinojosa, Michael Obersteiner, Hans-Joachim Bungartz, and Dirk Pflüger. EXAHD: An Exa-Scalable Two-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond. In *High Performance Computing in Science and Engineering'17*, pages 513–529. Springer, 2018.
26. Nathanael Hübbe, Al Wegener, Julian Kunkel, Yi Ling, and Thomas Ludwig. Evaluating Lossy Compression on Climate Data. In Julian Martin Kunkel, Thomas Ludwig, and Hans Werner Meuer, editors, *Supercomputing*, number 7905 in Lecture Notes in Computer Science, pages 343–356, Berlin, Heidelberg, 06 2013. Springer.
27. Nathanel Hübbe and Julian Kunkel. Reducing the HPC-Datastorage Footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression. *Computer Science - Research and Development*, pages 231–239, 05 2013.
28. Jeremy Iverson, Chandrika Kamath, and George Karypis. *Fast and effective lossy compression algorithms for scientific datasets*, volume 7484 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 843–856. Springer, 2012.
29. Nabeeh Jum'ah and Julian Kunkel. Automatic Vectorization of Stencil Codes with the GGDML Language Extensions. In *Workshop on Programming Models for SIMD/Vector Processing (WPMVP'19), February 16, 2019, Washington, DC, USA*, WPMVP, pages 1–7, New York, NY, USA, 2019. PPoPP 2019, ACM.
30. Nabeeh Jumah and Julian Kunkel. Optimizing Memory Bandwidth Efficiency with User-Preferred Kernel Merge. Lecture Notes in Computer Science. Springer, 07 2019.
31. Nabeeh Jum'ah and Julian Kunkel. Performance Portability of Earth System Models with User-Controlled GGDML code Translation. In Rio Yokota, Michele Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers*, number 11203 in Lecture Notes in Computer Science, pages 693–710. ISC Team, Springer, 01 2019.
32. Nabeeh Jumah and Julian Kunkel. Scalable Parallelization of Stencils using MODA. In *High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt/Main, Germany, June 20, 2019, Revised Selected Papers*, Lecture Notes in Computer Science. Springer, 07 2019.
33. Nabeeh Jumah, Julian Kunkel, Günther Zängl, Hisashi Yashiro, Thomas Dubos, and Yann Meurdesoif. GGDML: Icosahedral Models Language Extensions. *Journal of Computer Science Technology Updates*, pages 1–10, 06 2017.
34. Stefan Kronawitter, Sebastian Kuckuk, Harald Köstler, and Christian Lengauer. Automatic Data Layout Transformations in the ExaStencils Code Generator. *Modern Physics Letters A*, 28(03):1850009, 2018.
35. Julian Kunkel. Analyzing Data Properties using Statistical Sampling Techniques – Illustrated on Scientific File Formats and Compression Features. In Michaela Taufer, Bernd Mohr, and Julian Kunkel, editors, *High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P3MA, VHPC, WOPSSS*, number 9945 2016 in Lecture Notes in Computer Science, pages 130–141. Springer, 06 2016.

36. Julian Kunkel. SFS: A Tool for Large Scale Analysis of Compression Characteristics. Technical Report 4, Deutsches Klimarechenzentrum GmbH, Bundesstraße 45a, D-20146 Hamburg, 05 2017.
37. Julian Kunkel, Anastasiia Novikova, and Eugen Betke. Towards Decoupling the Selection of Compression Algorithms from Quality Constraints – an Investigation of Lossy Compression Efficiency. *Supercomputing Frontiers and Innovations*, pages 17–33, 12 2017.
38. Julian Kunkel, Anastasiia Novikova, Eugen Betke, and Armin Schaare. Toward Decoupling the Selection of Compression Algorithms from Quality Constraints. In Julian Kunkel, Rio Yokota, Michaela Taufer, and John Shalf, editors, *High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P3MA, VHPC, Visualization at Scale, WOPSSS*, number 10524 in Lecture Notes in Computer Science, pages 1–12. Springer, 2017.
39. S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N.F. Samatova. Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-Temporal Data. *European Conference on Parallel and Distributed Computing (Euro-Par), Bordeaux, France*, 08 2011.
40. D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener. Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics. *Super Computing*, 04 2013.
41. Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics 2012*, 08 2014.
42. Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 12(5):1245–1250, 2006.
43. Naoya Maruyama, Kento Sato, Tatsuo Nomura, and Satoshi Matsuoka. Physis: an implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12. IEEE, 2011.
44. Michel Müller and Takayuki Aoki. Hybrid fortran: High productivity gpu porting framework applied to japanese weather prediction model. *arXiv preprint arXiv:1710.08616*, 2017.
45. Florian Rathgeber, David A Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew TT McRae, Gheorghe-Teodor Bercea, Graham R Markall, and Paul HJ Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software (TOMS)*, 43(3):24, 2017.
46. Russ Rew and Glenn Davis. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications*, 10(4):76–82, 1990.
47. Erich Roeckner, G Bäuml, L Bonaventura, Renate Brokopf, Monika Esch, Marco Giorgetta, Stefan Hagemann, Ingo Kirchner, Luis Kornblueh, Elisa Manzini, et al. The atmospheric general circulation model ECHAM 5. PART I: Model description, 2003.
48. Masaki Satoh, Hirofumi Tomita, Hisashi Yashiro, Hiroaki Miura, Chihiro Kodama, Tatsuya Seiki, Akira T Noda, Yohei Yamada, Daisuke Goto, Masahiro Sawada, et al. The non-hydrostatic icosahedral atmospheric model: Description and development. *Progress in Earth and Planetary Science*, 1(1):18, 2014.
49. Felix Thaler, Stefan Moosbrugger, Carlos Osuna, Mauro Bianco, Hannes Vogt, Anton Afanasyev, Lukas Mosimann, Oliver Fuhrer, Thomas C. Schulthess, and Torsten Hoefler. Porting the cosmo weather model to manycore cpus. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '19, pages 13:1–13:11, 2019.
50. Jonas Tietz. Vector Folding for Icosahedral Earth System Models. Online `https://wr.informatik.uni-hamburg.de/_media/research:theses:jonas_tietz_vector_folding_for_icosahedral_earth_system_models.pdf`, 03 2018.
51. Raul Torres, Leonidas Linardakis, TL Julian Kunkel, and Thomas Ludwig. Icon dsl: A domain-specific language for climate modeling. In *International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colo.[Available at h ttp://sc13. supercomputing. org/sites/default/files/WorkshopsArchive/track139. html.]*, 2013.
52. Didem Unat, Xing Cai, and Scott B Baden. Mint: realizing cuda performance in 3d stencil methods with annotated c. In *Proceedings of the international conference on Supercomputing*, pages 214–224. ACM, 2011.

53. Kai Velten. *Mathematical modeling and simulation: introduction for scientists and engineers.* John Wiley & Sons, 2009.
54. Günther Zängl, Daniel Reinert, Pilar Rípodas, and Michael Baldauf. The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, 141(687):563–579, 2015.
55. Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.