New Capabilities of the Jsbach Land Surface Programme for the ICON Climate Model

Akihiro Farivarzadeh

Federal Office of Meteorology and Climatology MeteoSwiss, Zurich, Switzerland

ABSTRACT

Emerging high performance computing systems have brought new opportunities to the development of many industries. On the climate front, to benefit from emerging high-performance computing systems, weather and climate models need to be tuned to run efficiently on different hardware architectures, such as accelerators. For existing community models, which represent very large codebases written in Fortran, this is a major challenge. In order to achieve high performance, we introduced Claw Compiler and Claw single Column Abstraction. We will report extensions to the CLAW-SCA to handle ELEMENTAL functions and subroutines. We demonstrate the new capabilities of the Jsbach land surface programme for the ICON climate model. With this extension, JSBACH can be automatically ported to OpenACC or OpenMP accelerators with little or no changes to the original code.

1. Introduction

Numerical Weather Prediction and Climate modeling can highly benefit from computer technologies advances by increasing resolution or model complexity. In the recent years, architectures such as Graphics Processing Unit (GPU) have emerged for scientific high performance computing offering new opportunities. Most of the current Numerical Weather Prediction and Climate models are large Fortran based community codes which require to be adapted or re-written to run on non traditional CPU architectures such as GPU. In order to prepare for heterogenous supercomputer architectures, the global weather and climate model ICON [2, 4] is being ported to accelerators. The major part of the porting is currently achieved using OpenACC [10] compiler directives. For many code sections simple insertion of OpenACC compiler directives is an appropriate porting approach. But, for time-critical components such as the physical parameterizations, code restructuring and optimizations it is necessary to obtain optimal performance [6].

In some cases, the required code restructuring and optimization for GPU architectures have a negative impact when running the same code on a CPU architecture. Multiple solutions to address this problem have been proposed to achieve performance portability across architectures [3, 5, 8]. Here we focus on CLAW [1], an

open-source source-to-source translator that allows to perform architecture-specific code transformations with minimal code modifications.

The CLAW SCA is designed to address the physical parameterizations of atmospheric models in Fortran. Physical parameterizations are typically horizontally independent so each vertical column can be computed separately. With the CLAW SCA, the physical parameterizations are written in Fortran only considering the vertical dependencies while the horizontal directions are abstracted out. The CLAW Compiler can transform the code for a specific target architecture and insert horizontal loops and compiler directives such as OpenMP [11] for accelerator (version ≥ 4.5) or OpenACC.

In this paper, we focus on an CLAW SCA incorporation that extends CLAW capability to address such performance portability issues without imposing disturbing changes to the code base. We introduce a special case of the CLAW SCA connected with ELEMENTAL functions and subroutines. We demonstrate the new feature on the JSBACH land surface scheme [7] used in the ICON climate model. This paper is structured as follows: Section 1 describes the JSBACH land surface scheme and its use of ELEMENTALs. In Sections 2 and 3 we present the CLAW Compiler and the extension of SCA for ELEMENTALs. In Sections 4 and 5 we discuss our performance results and some code metrics. Finally, in Section 5 we draw our conclusion and discuss future work.

2.1 Title

### 2. The JSBACH Land Surface Scheme and Elemental

In this section, we briefly describe the concept of ELEMENTAL subroutines and functions and its application in the JSBACH land surface scheme. Further, we summarize its structure.

In Fortran, an ELEMENTAL function or subroutine is defined as a scalar operator. Dummy arguments as well as potential return value must be scalars but it may be called with arrays of arbitrary dimensionality as actual arguments. In this case, the operations defined in the function or the subroutine are applied element-wise on the full arrays. The main benefit of ELEMENTAL functions or subroutines is that it allows the user to write more compact code and allows the compiler to parallelize function or subroutine execution.

A very basic ELEMENTAL function is implemented in Fig. 1 from line 18 to 22. First it is mentioned with scalar arguments at line 9, second, with arrays arguments at line 13.

JSBACH is the land surface scheme of the global ICON model in climate mode. It is designed to serve as a land surface boundary for the atmosphere in the coupled simulation but it can also be used offline as a standalone model. JSBACH simulates photosynthesis, phenology and land physics with hydrological and bio-geochemical cycles.

JSBACH code is written in 2008 Fortran object-oriented style. It is designed as a pipeline of tasks where each task is applied on various tiles of the soil depending on their properties. Each task retrieves a number of fields from the internal memory object and applies several ELEMENTAL functions and/or subroutines as well as vector operations to them.
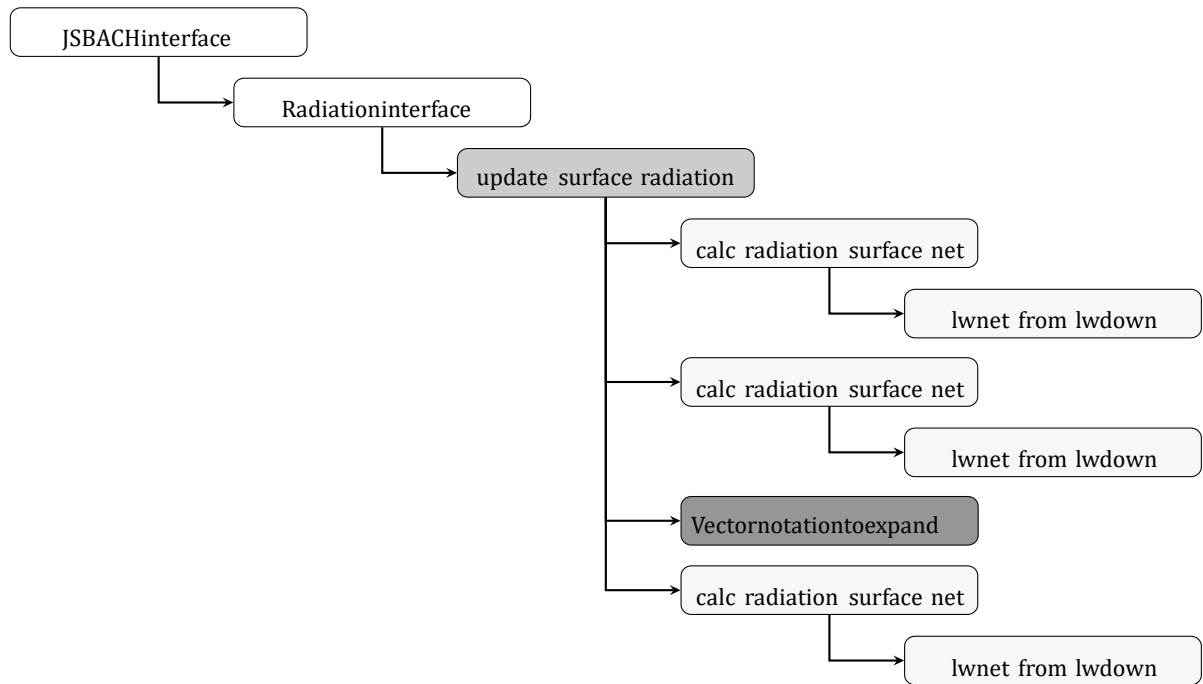
Figure 1. Example of a typical task workflow using ELEMENTALs in JSBACH - update radiation surface from the radiation interface

To hide some aspects of the design such as the memory access and pointers, JSBACH includes a non-Fortran Domain Specific Language (DSL). The code must be preprocessed by a Python script in order to obtain standard Fortran code.

subroutines. In some cases vector notation can be handled by compilers with the !$acc kernels construct. But in practice, we have experienced several cases where the compiler was not able to determine that the kernel can be run in parallel and thus generated a sequential version of the code resulting in a significantly slower GPU execution comparing to the CPU version. This was especially true with older version of OpenACC compatible compiler but can be solved having reasonable amount of vector notation in an !$acc kernels block.

In order to execute the code on a GPU, we need to either fundamentally refactor JSBACH or to take advantage of a source-to-source translator such as CLAW. CLAW can automatize the port while taking advantage of the current information we can extract from it. The latter solution is the one described in this paper and has the advantage to bring portability across compiler directives by supporting OpenACC and OpenMP simultaneosly.

Figure 2 is a typical call graph of a single task defined in JSBACH. update radiation surface is part of the radiation interface. This task is calling a single ELEMENTAL subroutine three times with different fields as arguments. This ELEMENTAL subroutine (calc radiation surface net) calls another ELEMENTAL function.

As all ELEMENTAL code needs to be transformed to accept compiler directives, everything in the task is transformed to be executed on the accelerator.

Table 1. JSBACH tasks with the number of automatically generated kernel and the number of 1 dimensional (1D) and 2 dimensional (2D) input/output fields for each task

Interface	Task name	Nb. kernels	1D in	1D out	2D in	2D out
Radiation	surface radiation	4	11	7	0	10
	radiation par	5	10	6	4	4

|	albedo	12	38	18	0	0
Phenology	phenology logrop	11	30	21	0	0
	fpc	3	2	1	0	10
Hydrology	snow _and skin _fraction	11	11	4	0	0
	soil properties	2	7	0	7	7
	evaporation	3	16	6	0	0
	surface hydrology	1	4	0	4	0
	soil hydrology	3	3	3	1	0
	canopy cond _unstressed	1	2	1	0	0
	canopy cond _stressed	1	7	5	5	0
	water _balance	5	7	3	2	3
Surface energy balance	surface energy _lake	7	16	16	0	0
	surface energy _land	18	23	14	0	0
	surface fluxes lake	1	11	6	0	0
	surface fluxes land	1	7	4	0	0
	asselin _land	3	3	2	0	0
	snowmelt correction	4	3	1	4	0
Soil snow energy	soil and _snow properties	14	2	4	2	2
	soil and _snow temperature	14	3	5	4	10
Assimilation	assimilation scaling factors	2	1	0	0	1
	canopy cond _unstressed	2	4	1	3	1
Turbulence	humidity scaling	10	14	3	3	0
	roughness	3	3	1	0	0

The full JSBACH model in its ICON configuration is composed of about 30 different tasks and represents roughly 30'000 lines of code. There is a total of 70 ELEMENTAL subroutines and functions to transform. Our approach is applied to all required tasks for a climate simulation.

Table 1 describes the list of tasks automatically ported to GPU. For each tasks, the number of kernels generated and their corresponding inputs and outputs are detailed. 1 dimensional fields includes only the horizontal dimension where 2 dimensional include the number of soil layer as their second dimension.

3. The Claw Cimpoler and the Single Column Abstraction

This section presents the CLAW Compiler and the Single Column Abstraction on which our work is based.

The CLAW Compiler is an extensible open-source source-to-source compiler for modern Fortran code. It is based on the OMNI Compiler Project [9].
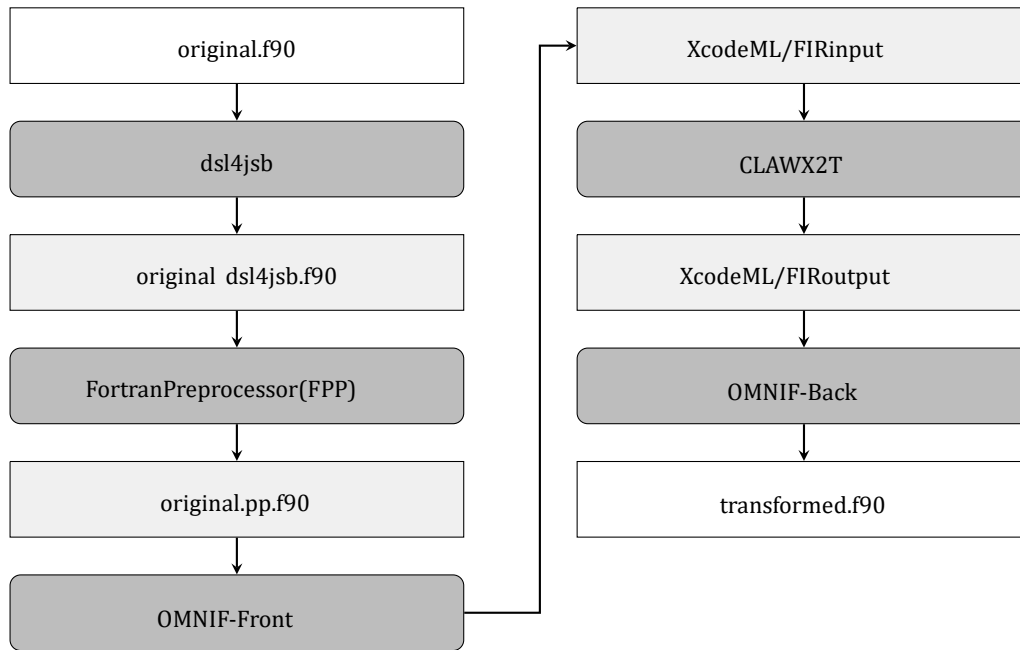
Figure 2. CLAW Compiler workflow including the JSBACH DSL preprocessing

Figure 2illustrates the internal workflow of the compiler. JSBACH code is pre-processed by a dedicated Python script which initially translates the JSBACH DSL to standard Fortran before entering the CLAW workflow. Fortran code is pre-processed and then parsed to the XcodeML/F Intermediate Representation (IR) [12]. This IR—represented as an Abstract Syntax Tree (AST)—is then manipulated by CLAW XcodeML to XcodeML Translator (X2T) to produce the refactored version of the code for a specific target with inserted directives. Finally, the IR is decompiled to standard Fortran code before being compiled by default compilers. Major contribution to extend its transformations for ELEMENTALs was introduced in the CLAW X2T.

In weather and climate models, the physical parameterizations are, in most cases, single column processes. Each column of the three-dimensional computational domain can be computed independently and does not have any data dependencies on its neighbors. The CLAW Single Column Abstraction DSL [1] was introduced to exploit this information. A CLAW SCA source code is fully standard Fortran code with the addition of specific CLAW directives. The code has no notion of the horizontal dimensions in the declaration of the fields as well as in the execution part. Loops iterating over horizontal dimension are omitted.

The SCA code is processed by the CLAW Compiler as shown in Fig. 4. The user defines the target architecture as well as the compiler directives to be used. Under the hood, the CLAW Compiler performs create a data dependency graph to decide where to create kernels and therefore which temporary fields have to be promoted. Based on this analysis it inserts the necessary iterations over the parallel dimensions and promotes the appropriate fields within the physical parametrization. In addition, the compiler inserts the required compiler directives to parallelize the loops and to manage data movement between the host and the device memory. The SCA transformation comes with various user-configurable options to manage promotion and data movement strategy, giving the end-user freedom to test various configurations.

4.   Extension for Elemental Subroutines and Functions

An ELEMENTAL function or subroutine can be seen as a specific case of the CLAW SCA [1]. The land surface scheme is a point-wise computation on the grid that can be viewed as a columnwise computation with a limited number of vertical level.The depth of the subroutine or function in the call graph determines the

transformation applied to it. In Fig. 2, the leaf function lwnet from lwdown can be considered as a function to be inlined but calc radiation surface net is a perfect candidate to apply SCA transformation rules.

When a field is promoted or when a parallel iteration is inserted back into the code by the compiler, the dimension information is taken from the SCA model configuration file. This TOML formatted file defines the dimensions omitted in ELEMENTAL as well as layouts to be applied during the promotion transformation. This allows the user to investigate different data layouts depending on the target architecture. The user can update the default layout in the configuration file or add a layout clause to the model−data directive with the desired layout. The SCA model configuration file used for JSBACH is shown in Fig. 6. This file is read by the compiler before applying transformation.

The CLAW SCA transformation applied to non-ELEMENTAL subroutine or function will modifies the source code for both CPU and GPU targets. Since the original code with ELEMENTALs is already suited for CPU target, the extension of the SCA has no effect for this target in this case. We rely here on the Fortran compiler to efficiently compile ELEMENTAL on CPU.

When applied to GPU, the following actions occur for non-leaf subroutine or function:

· The signature of the subroutine/function is updated and the ELEMENTAL or PURE specifiers are discarded.

· The flagged fields are promoted according to the specified layout. If no layout is specified, the default layout is assumed. The promotion and layout information come from the configuration file as shown in Fig. 6.

· Iterations over new dimensions specified in the layout are inserted.

· Data analysis is performed and temporary fields or scalars that need promotion are promoted.

Leaf ELEMENTAL subroutines and functions have simpler transformations applied to them. The code is processed as follows:

· As for other ELEMENTAL subroutines/functions, the signature is updated and the ELEMENTAL or PURE specifiers are discarded.

· Compiler directives are inserted to set the subroutine or function as an OpenACC routine or OpenMP target code.

Once the code is annotated with CLAW SCA directives, the CLAW Compiler is called the same way for file with SCA or SCA with ELEMENTALs. Listing 4 can be executed in the same way.

As mentioned in Section 1, vector notation is used widely in tasks. To automatize the port of these blocks of Fortran code to OpenACC and OpenMP, the CLAW expand directive is used.

5. Performance Comparison

In this section we present the performance results that have been achieved by applying the extended version of the CLAW Compiler to the JSBACH land surface scheme. The computational domain size (number of horizontal grid points × number of soil layers) used for the performance measurement is 20480×5. All

performance measurements in this section are obtained using −03 optimization flag or equivalent for each compiler. The code is transformed with the CLAW Compiler 2.0 and compiled with the PGI 18.10 Fortran compiler for the baseline CPU as well as the GPU OpenACC results. The CPU reference is obtained using multi-core OpenMP parallelism available in the original version of ICON. The GPU OpenMP target results were obtained with Cray Compiler CCE 8.7.4 and by running standalone version of the kernels since the full ICON model is not ready to have a code mixing OpenMP for multi-core and for accelerator.

Figure 10 shows the speedup achieved from the CLAW SCA transformed version with OpenACC directives and the CLAW SCA transformed version with OpenMP directives over the CPU reference. The original version of the code is exploiting the 12 cores available on the Intel

Xeon E5-2690 v3 Haswell CPU of Piz Daint at CSCS parallelized with multi-core OpenMP. The CLAW OpenACC and OpenMP versions are executed on one NVIDIA P100 GPU. The theoretical floating point peak performance of the Intel Haswell is 500 GFLOPS, while the NVIDIA P100 is 5.3 TFLOPS. In term of memory bandwidth, the Haswell has a theoretical peak at 68 GB/s while the P100 can reach up to 732 GB/s. For both compute and memory bandwidth limited problem one can expect a maximum speedup of 11x.

As shown, OpenACC and OpenMP results are very similar and expected. Kernels issued from the ELEMENTAL transformation are pretty simple to be handled by a compiler and we did not expect PGI and Cray to fundamentally generate different code for them. Depending on the size of the kernel and the tile it has to process, we are able to achieve speedup between 1.7x up to 8.6x for kernel like the albedo: calc sky view fractions . Tiles can be viewed as a collection of grid points from a specific type of land (e.g., lake or glacier) or as a mask for this specific type of land. Therefore tiles can have more or less work to be performed due to a different set of grid points. As GPUs need enough work to exploit massive parallelism, a reduced set of grid points is one of the reason the speedup can vary this much. Another factor is the size of the kernel. Some ELEMENTAL functions/subroutines are very small and the kernels generated from them are also small. Kernel's launch and synchronization is then a non-negligible part of the overall kernel execution time. These overhead can be hidden in the execution of the full model with asynchronous mechanism.

6. Code Metrics

This section briefly compares the original code with its CLAW SCA version before and after applying transformations. Unlike the original SCA the SCA for ELEMENTAL imposes less changes to the code. In order to comply with the specification, the user only annotates the ELEMENTAL subroutines and functions to be executed on GPU, and relies on the compiler from then on. There is no need to apply other code change in the ELEMENTALs.

The full JSBACH model is annotated with 40 CLAW SCA directives and about 130 CLAW expansion directives to convert vector notation to parallelizable loops. In the transformed code there are almost 2000 lines of OpenACC directives or OpenMP depending the user's choice. This is the amount of compiler directives a user would have had to write by hand to achieve the same goal. A hand-written version would also have to change the code structure significantly, for example by promoting fields and adding loops.

7. Conclusion

In this paper we propose an extension of the CLAW SCA DSL and its compiler to take advantage of the ELEMENTAL construct in Fortran code and automatize the port to heteroge- nous architecture. The ELEMENTAL construct as exploited in JSBACH provides the necessary abstraction to implement automatic code transformation and target GPU architectures without writing lots of compiler directives by hand.

Promotion, loop parallelization and compiler directive generation are handled by the CLAW Compiler automatically.

For this paper, the CLAW SCA extension was applied to a wide portion of the JSBACH land surface scheme, one of the physical parametrizations of the ICON global climate model. From a single simple source code multiple programming paradigm can be targeted. Performance results show up to 8.6x speedup against a 12-core parallel CPU version for a specific kernel. All kernels are a least 1.7x faster than the CPU version. The overall performance of the JSBACH model running on GPU is typically between 5x to 6x speedup depending on its configuration.

In the current implementation of the extension, a single parallel subroutine or function is generated from its ELEMENTAL counterpart. This is fine for the JSBACH use case as a single ELEMENTAL is always called with the same kind of arguments. Future improvement to generate several versions of a subroutine or a function if the type of argument used to call them is different.

As it is possible to generate any source code, we can imagine to take advantage of new compiler development such as exploiting the DO CONCURRENT construct from Fortran 2008 to target accelerators as it is investigated in latest version of PGI. Instead of generating compiler directives the CLAW Compiler could exploit this new Fortran feature.

References

Clement, V., Ferrachat, S., Fuhrer, O., et al.: The CLAW DSL: Abstractions for performance portable weather and climate models. In: Proceedings of the Platform for Advanced Scientific Computing Conference. pp. 2:1-2:10. PASC '18, ACM, New York, NY, USA (2018), DOI: 10.1145/3218176.3218226.

Crueger, T., Giorgetta, M.A., Brokopf, R., et al.: ICON-A, the atmosphere component of the ICON Earth system model: II. model evaluation. Journal of Advances in Modeling Earth Systems 10(7), 1638-1662, 2018, DOI: 10.1029/2017MS001233.

Fuhrer, O., Osuna, C., Lapillonne, X., et al.: Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. Supercomputing Frontiers and Innovations 1(1), 45-62, 2014, DOI: 10.14529/jsfi140103.

Giorgetta, M.A., Brokopf, R., Crueger, T., et al.: ICON-A, the atmosphere component of the ICON Earth system model: I. model description. Journal of Advances in Modeling Earth Systems 10(7), 1613-1637, 2018, DOI: 10.1029/2017MS001242.

Gysi, T., Osuna, C., Fuhrer, O., et al.: Stella: A domain-specific tool for structured grid methods in weather and climate models. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 41:1-41:12. SC '15, ACM, New York, NY, USA, 2015, DOI: 10.1145/2807591.2807627.

Lapillonne, X., Fuhrer, O.: Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science. Parallel Processing Letters 24(1), 2014, DOI: 10.1142/S0129626414500030.

Mauritsen, T., Bader, J., Becker, T., et al.: Developments in the MPI-M Earth System Model version 1.2 (MPI-ESM1.2) and Its Response to Increasing CO2. Journal of Advances in Modeling Earth Systems 11(4), 998-1038, 2019, DOI: 10.1029/2018MS001400.

Muller, M., Aoki, T.: Hybrid Fortran: High productivity GPU porting framework applied to Japanese weather prediction model. CoRR abs/1710.08616, 2017, http://arxiv.org/ abs/1710.08616.

Omni CompilerProject: Omni Compiler Project—An Infrastructure for Source-to-Source Transformation. http://omni-compiler.org, 2013-2019, accessed: 2019, 09, 02.

OpenACC Standard: The OpenACC application programming interface—versio 2.7. https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf, 2018, accessed: 2019, 09, 02.

OpenMP Architecture Review Board: OpenMP application programming interface – version 5.0. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf, 2018, accessed: 2019, 09, 03.

XcalableMP Specification Working Group: XcodeML/Fortran Specification. https //omni-compiler.org/download/xcodeml/stable/XcodeML-F-1.0.pdf, 2017, accessed: 2019, 09, 02.

Copyrights