Towards Performance Portability for Atmospheric and Climate Models with the GGDML DSL

Nabeeh Jum'ah[1], Julian Kunkel[2], Zängl Günther[3], Hisashi Yashiro[4], Thomas Dubos[5], and Yann Meurdesoif[6]

1 Universität Hamburg Jumah@informatik.uni-hamburg.de, 2 Deutsches Klimarechenzentrum kunkel@dkrz.de, 3 Deutscher Wetterdienst, 4 RIKEN Advanced Institute for Computational Science, 5 École Polytechnique, 6 Laboratoire des sciences du climat et de l'environment

Abstract

Demand for high-performance computing is increasing in atmospheric and climate sciences, and in natural sciences in general. Unfortunately, automatic optimizations done by compilers are not enough to make use of target machines' capabilities. Manual code adjustments are mandatory to exploit hardware capabilities. However, optimizing for one architecture, may degrade performance for other architectures. This loss of portability is a challenge.

The **contributions** of this poster are:

- Introduction of a DSL for Icosahedral models
- Evaluation of code-reduction opportunity

In this poster, we focus on code quality and benefit from the developer perspective (not on performance).

Goals

Achieve high performance, portability and maintainability through a DSL and a lightweight compilation infrastructure fostering separation of concerns:

- Scientists from domain science provide the logic of the software to solve the problem. They don't need to provide any optimization details.
- The configuration details related to platform-dependent implementation are provided by scientific programmers. They provide the compilation tool the needed configuration to generate architecture-optimized code.

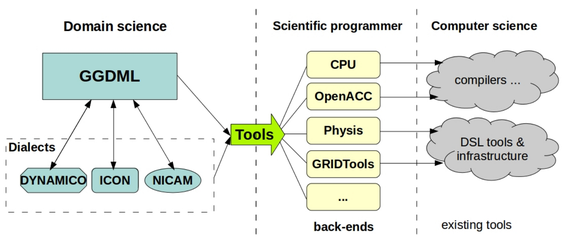


Fig. 1: Seperation of Concerns

Approach

- We started with the code of the three models; DYNAMICO, ICON, and NICAM.
- We adopted co-design approach that included domain scientists, each of whom is experienced with one of the models
- We Identified the most compute intensive parts of the code as they are the critical pieces and hence the key to achieve performance.
- An abstraction has been extracted by recognizing the domain concepts and operations in these compute intensive code parts. During this process, we tried to identify commonalities in the three models and create a representation that expresses all three models. Technical requirements for performance were considered during this abstraction process.
- We rewrote codes from the models accordingly to the suggestions.
- We discussed with scientists, who are experts with those models, the abstraction levels and code examples.

Design Overview

- We introduce GGDML (**General Grid Definition and Manipulation Language**) and examine the approach for icosahedral-grid based climate & atmospheric models. GGDML is a domain-specific language (DSL) that fosters separation of concerns between domain scientists and computer scientists.
- Our DSL **extends** Fortran language with concepts from domain science, apart from any technical descriptions such as hardware based optimization.
- The main concept is based on grids, and offers ways to define and manipulate grids and grid-bound variables.
- Fortran code extended with novel semantics from GGDML goes through the meta-DSL based compilation procedure. This generates high performance code – aware of hardware features, based on provided configurations.

Source-to-Source Translation

A lightweight translation tool –that ships with code repositories and integrates into build systems– translates model code that uses GGDML extensions into a target-architecture-optimized code (or intermediate language).

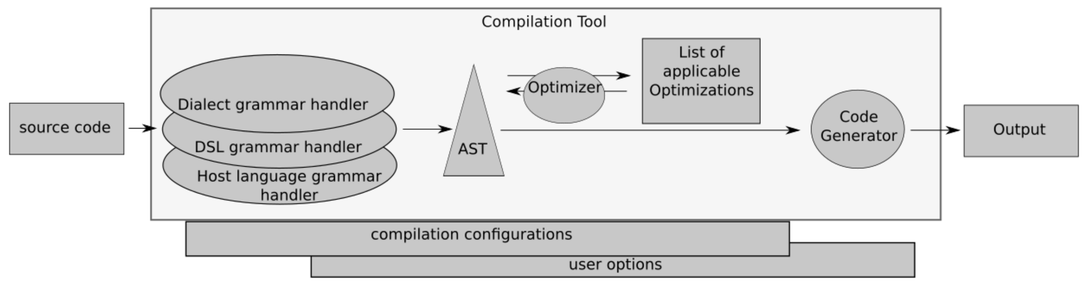
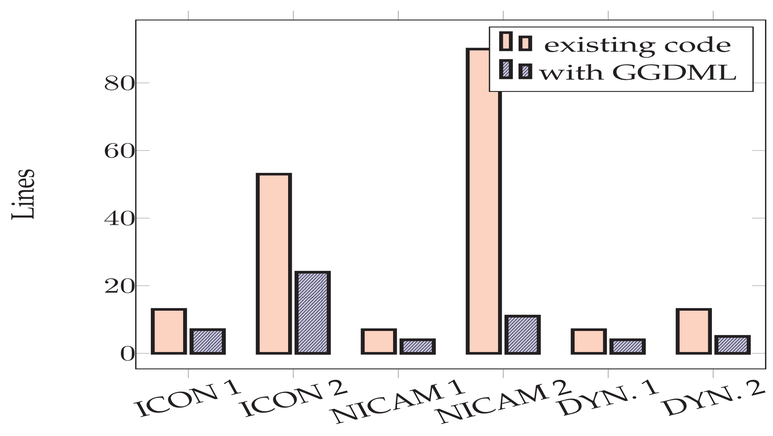


Fig. 2: Translation Process

Results

To evaluate the DSL, we took two relevant kernels from each of the three models, and analyzed the achieved code reduction. An overview of the results is shown in the table and figure below. The numbers demonstrate the impact on code length when porting code to GGDML.

Model, kernel	lines (LOC)		words		characters	
	before DSL	with DSL	before DSL	with DSL	before DSL	with DSL
ICON 1	13	7	238	174	317	258
ICON 2	53	24	163	83	2002	916
NICAM 1	7	4	40	27	76	86
NICAM 2	90	11	344	53	1487	363
DYNAMICO 1	7	4	96	73	137	150
DYNAMICO 2	13	5	30	20	402	218
total	183	55	911	430	4421	1991
percentage	30.05%		47.20%		45.04%	



- In average, we cut down the LOC to less than one third (30%) of the original code. Better reductions are achieved in stencil codes (NICAM example No.2, reduced to 12.22% of the original LOC).
- Influence on readability and maintainability: Reducing the important code metrics like **code duplication**, WTF/Minute – in code review, in some cases, boundary conditions could be removed thus reducing the cyclomatic complexity.
- Code reduction reduces development costs. By applying COCOMO, we estimate the benefit to be:

Software project	DSL?	Effort Applied	Dev. Time (months)	People required	dev. costs (M€)
Semi-detached	without	2462	38.5	64	12.3
	with	1133	29.3	39	5.7
Organic	without	1295	38.1	34	6.5
	with	625	28.9	22	3.1

Performance

- Carried out experiments on a prototype application, small kernels with a two-dimensional grid, with two grid-bound variables.
- Fed source code written with C extended with DSL.
- Used two configuration files.
- Switching memory layout (AoS to SoA) yields **double (2x) the performance**.
- Inlining and loop fusion yields **10% of the doubled performance** as additional enhancement.
- This result is comparable to the performance of a hand-tuned code version of the application.
- Machine: Intel(R) Core(TM) i7-6700 CPU Skylake
- Compiler: GCC 5.4 with -O3 optimization option was used to compile all codes

Code Example

The following example demonstrates the use of GGDML in the DYNAMICO model.

```
DO l=ll_begin,ll_end
!DIR$ SIMD
  DO ij=ij_begin,ij_end
   berni(ij,l) = .5*(geopot(ij,l)+geopot(ij,l+1)) + 1/(4*Ai(ij)) *
        (le(ij+u_right)*de(ij+u_right)*u(ij+u_right,l)**2 &
        +le(ij+u_rup)  *de(ij+u_rup)  *u(ij+u_rup,l)**2   &
        +le(ij+u_lup)  *de(ij+u_lup)  *u(ij+u_lup,l)**2   &
        +le(ij+u_left) *de(ij+u_left) *u(ij+u_left,l)**2  &
        +le(ij+u_ldown)*de(ij+u_ldown)*u(ij+u_ldown,l)**2 &
        +le(ij+u_rdown)*de(ij+u_rdown)*u(ij+u_rdown,l)**2 )
  ENDDO
ENDDO
```

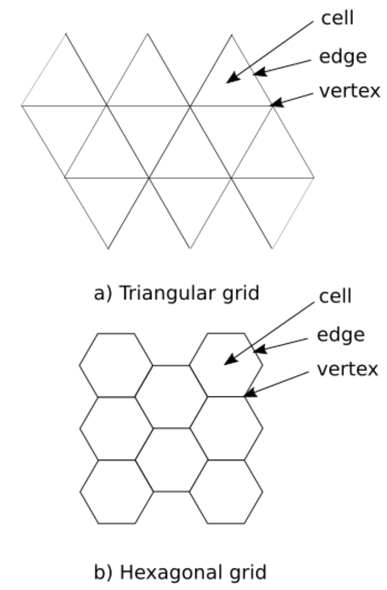
GGDML version of the code above

```
FOREACH cell IN grid
 berni(cell) = .5*(geopot(cell)+geopot(cell%above)) + 1/(4*Ai(cell%ij)) *
  REDUCE(+, N={1..6} le(cell%neighbour(N)%ij)*de(cell%neighbour(N)%ij)*u(cell%neighbour(N))**2)
END FOREACH
```

Language Extensions

Offer abstract grid concepts (e.g. cell, edge, vertex) especially for icosahedral models:

- Embedded into a general-purpose language
- Definition of grids
  - Various shapes, e.g., triangular, hexagonal



- Variable definition on grid elements
- Reference variables by grid elements
  - Named element relationships
    - to reference cell edge
    - to reference cell above/below
    - to reference a neighbour cell
    - ...
- Element ranges traversal
  - Specify/modify dimensions of ranges
  - Update data of variables while traversing
- Reduction operator

The compiler infrastructure is flexible to implement other language extensions on demand.

Summary

- We introduce GGDML, a Fortran language extension to improve readability and performance portability.
- We adopt collaborative bottom-up development of GGDML with re-engineered concepts top-down to provide a consistent perspective from a domain science point of view.
- Hardware-related information are eliminated from a code written with GGDML.
- GGDML reduces code size significantly and conforms to presumed requirements.
- We introduce also meta-compilation with a source-to-source translation tool.
- Compilation configurations guide the tool to generate machine-dependent optimizations such as memory layout.
- Specialized scientific programmers provide translation configurations to get performance.

Future Work

- Extend and improve GGDML.
- Enhance translation tools.
- Support various general-purpose languages.
- Support further optimization techniques.
- Furthermore, we aim to identify and analyze best-practices and coding strategies for achieving performance on different platforms.
- Support profile-guided optimization to utilize the flexibility of the DSL semantics.

Acknowledgements

This work was supported in part by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) (GZ: LU 1353/11-1).