A user-controlled GGDML code translation technique for Performance Portability of Earth System Models

Nabeeh Jum'ah[1] & Julian Kunkel[2]

1 Universität Hamburg Jumah@informatik.uni-hamburg.de, 2 Deutsches Klimarechenzentrum kunkel@dkrz.de

Abstract

Demand for high-performance computing is increasing in earth system modeling, 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. Our approach involves the use of the GGDML language extensions to write a higher-level modeling code, and use a user-controlled source-to-source translation technique. Translating the code results in an optimized version for the target machine.

The **contributions** of this poster are:

- The use of a highly-configurable code translation technique to transform higher-level code into target-machine-optimized code
- Evaluation of code transformation for multi-core and GPU based machines, both single and multi-node configurations

Goals

Achieve high performance and portability besides to improving code readability and maintainability through a slight language modification and a lightweight compilation infrastructure fostering separation of concerns:

- Scientists from the domain science develop the code of the model to solve the problem from a scientific perspective. The machine-specific optimization is not needed.
- The configuration details related to machine-specific optimization are provided by scientific programmers. They provide the translation tool the needed configuration information to generate architecture-optimized code.

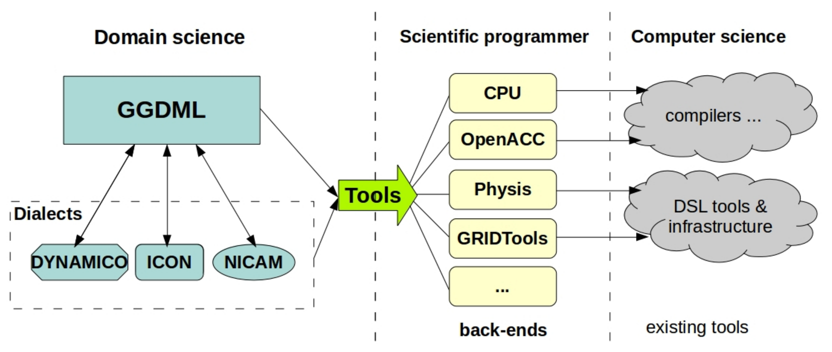


Fig. 1: Separation of Concerns

Approach

- The modeling language, e.g. C or Fortran, along with the GGDML language extensions are used to write the source code of a model
- Machine-specific configuration information are written to transform the source code into an optimized target-specific version
- The translation tool uses both
  - the semantics of the language extensions
  - and the configuration information

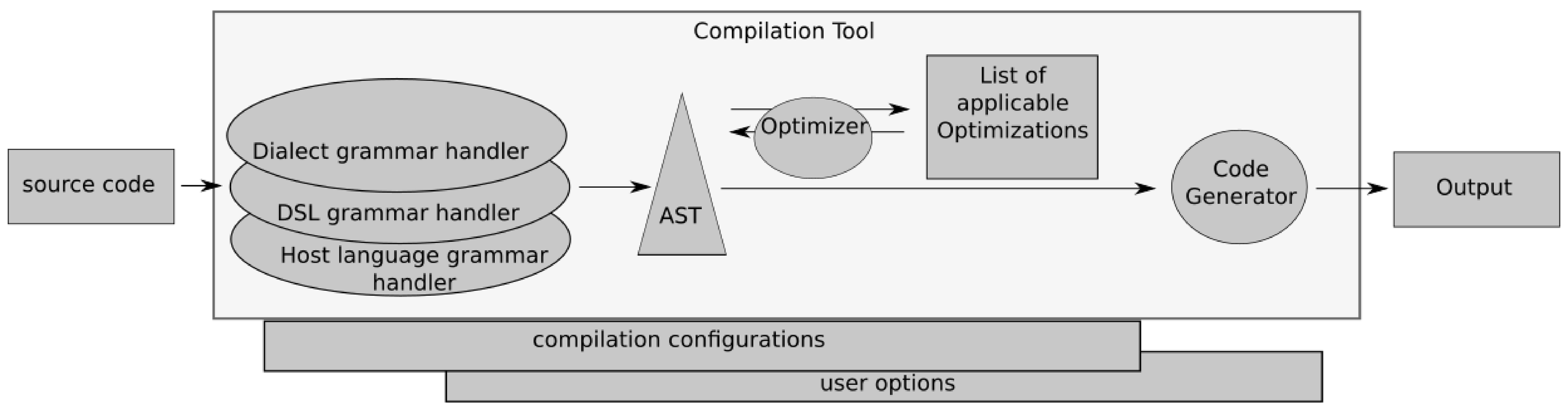
  to translate the source code and apply the optimization procedures

Configuration Specification

- The machine-specific configuration information allows the user to control the source code translation (and optimization) process
- The set of language extensions is dynamically extensible through the configuration specification that is fed to the translation tool
- The declaration specifiers are defined through the configuration specification
  - The specifiers are defined in groups
    - e.g. dimension group: 2D or 3D
  - They control how the variable is handled
- The allocation and deallocation of the model's variables is guided by a specific section
- The definition of the grids that the model uses is handled by a specific section that allows to describe the model's global domain
- The parallelization of the code is handled by the configuration specification
- The memory layout is completely controlled
  - Flexible array index transformations
- The halo exchange is initialized and accomplished in a controlled manner

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.



Test Setup

- An icosahedral-grid-based modeling testbed, 1024x1024x64 grid
- Two different memory layouts
  - A three-dimensional array with three-index addressing
  - A one-dimensional array with transformed one-index transformed addressing
- Two test machines
  - DKRZ Mistral: dual socket Intel Broadwell nodes (Intel Xeon E5-2695 v4 @ 2.1GHz), OpenMPI version 1.8.4 and GCC version 7.1
  - NVIDIA's cluster: Haswell CPUs (Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz), P100 and V100 GPUs, OpenMPI version 1.10.7 and the PGI compiler version 17.10

GPU Experiments

The table below shows the performance and the memory throughput (measure with NVIDIA's 'nvprof' tool) of the code when run on a single P100/V100 node.

	Serial	P100			V100		
		performance GF/s	Memory throughput GB/s		performance GF/s	Memory throughput GB/s	
			read	write		read	write
3D	1.97	220.38	91.34	56.10	854.86	242.59	86.98
3D-1D	1.99	408.15	38.75	43.87	1240.19	148.49	57.12

- Changing the memory layout reduces the amount of the data that needs to be read from the memory, thus performance improves.

To evaluate the scalability of the testbed code on multiple nodes with GPUs, we have translated the code with MPI and we have run it on 1-4 P100 GPU nodes.

- The figure shows the performance achieved in both cases when measuring the strong and the weak scalability. The performance is measured with and without halo exchange. The gap reflects the cost of data movement from/into the GPU's memory and the communication time.

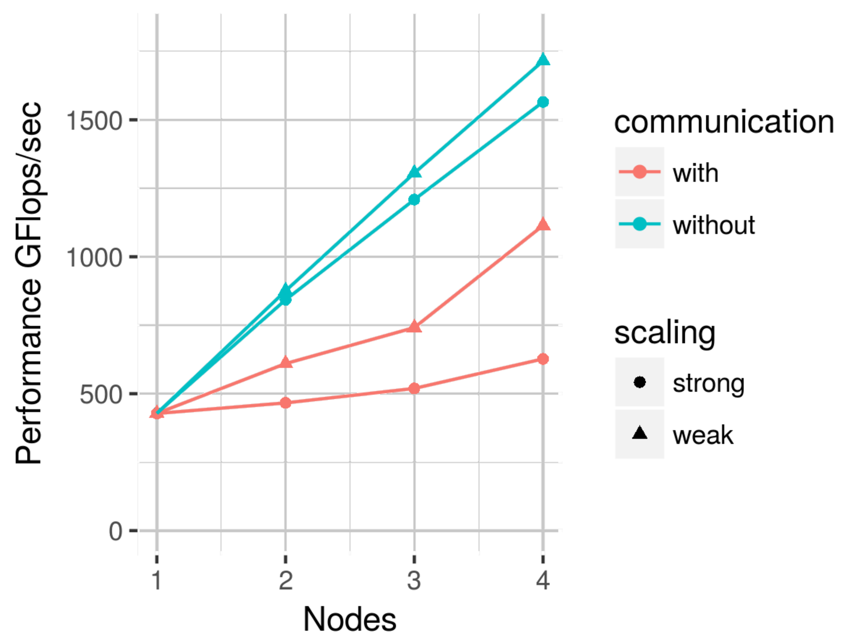
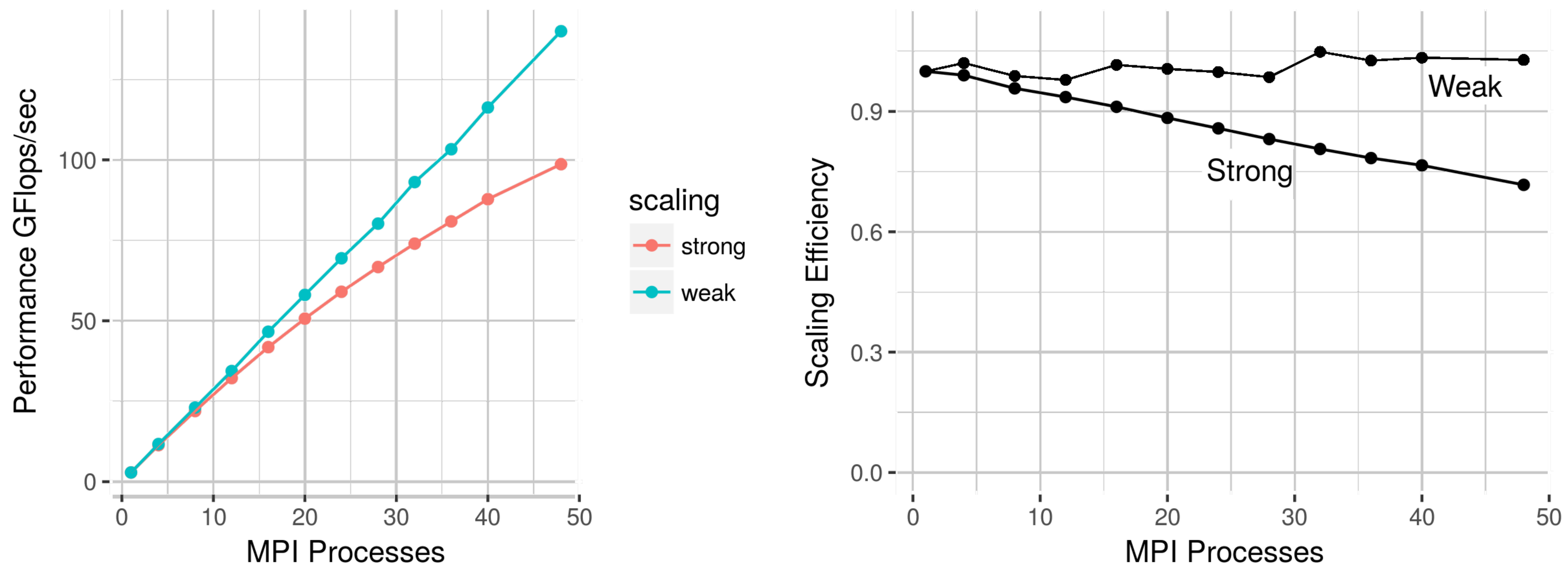


Fig. 2: P100 scalability

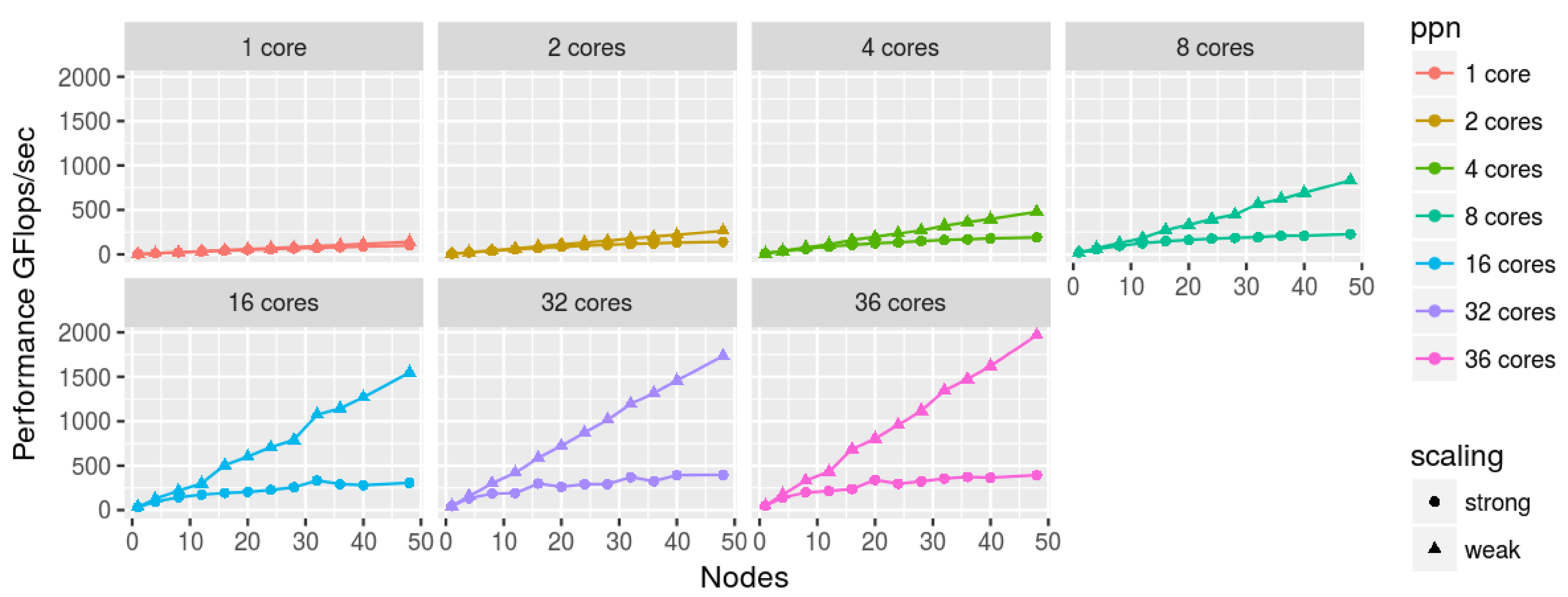
Multicore Experiments

To evaluate the scalability of the translated code with multiple MPI processes on CPU nodes, we have run it on 1,4,8,12,16,20,24,28,32,36,40, and 48 nodes.



Both the strong and the weak scalability efficiency are shown in the figure. The efficiency is still around 100% up to 48 MPI processes for the weak scaling measurements. The Strong scaling measurements decrease from 100% at one process to about 70% at 48 processes in a linear trend.

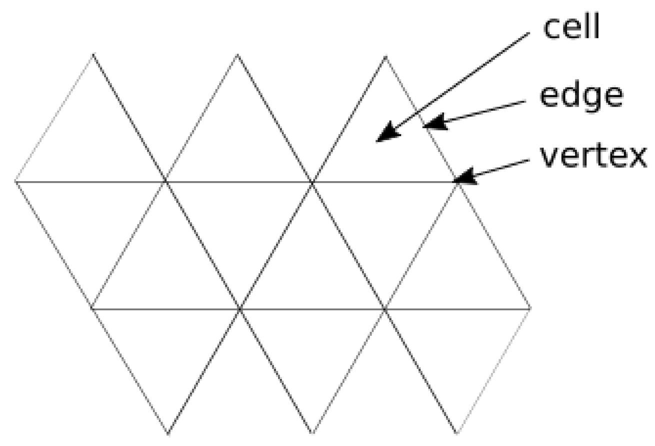
The performance of the translated code that uses OpenMP with the MPI is also evaluated. We have run the code on 1,4,8,12,16,20,24,28,32,36,40 nodes and 1,2,4,8,16,32, and 36 cores per node.



Language Extensions

With the GGDML language extensions, the test code is written with higher-level semantics. GGDML abstracts grid concepts (e.g. cell, edge, vertex ...) especially for icosahedral models:

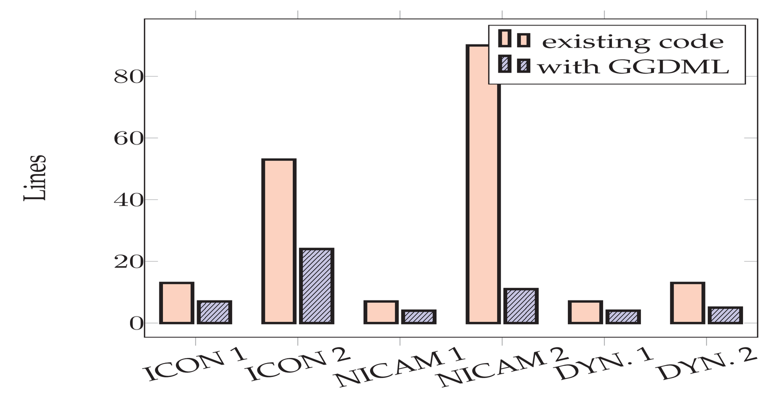
- Extends a general-purpose language
  - Can be used with different languages
- Declaration of models' variables
- Variable allocation and deallocation
- Expressions to specify grids



- Reference variables by grid elements
  - Named element relationships
    - to reference cell edge
    - to reference cell above/below
    - to reference a neighbour cell
    - ...
- Grid elements traversal
  - Apply stencil operation
  - Update data of variables while traversing
- Reduction expressions

Code Quality

GGDML achieves code reduction to 30% of the original code (LOC). The figure shows six kernels from three icosahedral models (two each).



Summary

- We introduce a user-controlled code translation technique of higher-level GGDML-based code to provide performance portability for earth system modeling.
- The model's code is written with GGDML from a scientific perspective (not machine).
- The higher-level code is translated by a source-to-source translation tool.
- Besides to the semantics of the GGDML language extensions, the translation process is driven by user-provided configuration information, which depends on the target machine.
- Compilation configurations guide the tool to generate machine-dependent optimizations such as memory layout.
- The experiments show the success of the technique to run the test application on multi-core machines and GPUs, both on single and multiple node configurations.

Future Work

- Investigate performance improvement opportunities in kernel computation.
- Investigate further kernel code analysis for inter-kernel optimization possibilities.
- Investigate the use of optimized libraries.
- Investigate communication optimization.

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).