Performance Portability of Earth System Models with User-Controlled GGDML code Translation

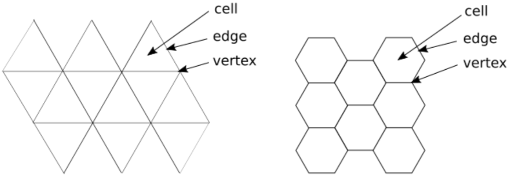
Nabeeh Jum'ah, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

3rd International Workshop on Performance Portable Programming Models for Accelerators 2018
28-06-2018

Earth-System Modeling

- Computations with fields over earth surface or parts of it
- Discretizes with different types of grids: regular, icosahedral ...
- Values at the centers of the cells, on the edges, at the vertices



a) Triangular grid b) Hexagonal grid

- Many kernels within time steps apply stencil operations

Earth-System Modeling

Modeling using general-Purpose Languages

- The semantical nature of the languages limit the compilers ability to exploit some optimization opportunities
- Scientists need to manually optimize code
- Challenging effort
  - The complexity of the architectural features
  - The diversity of the architectures
  - Various tools and programming models
- Code quality
  - Code duplication
  - Model's maintainability

Improvement Opportunities

- Code readability and maintainability
- Developers productivity
- Performance-portability

Modeling Language Extensibility

- Bypass the shortcomings of the general-purpose languages
- Still use the preferred modeling language
- Extend the modeling language
  - Based on scientific concepts
  - Hiding lower level details (e.g., architecture, memory layout)
- The semantical nature of the extensions allows optimization

Approach

Separation of Concerns

- Domain scientists formulate scientific logic in source code
- Scientific programmers write target configurations

- Model development with extended language
  - Scientific perspective
    - not machine perspective
  - Code is developed once
    - performance is achieved for different configurations
- Configurations define software performance
  - Written by programmers with more experience in platform
  - Fit the target run environment

Approach

- Higher-level code translation
  - A source-to-source translation tool is used
    - A lightweight tool
    - Easily ships with code repositories
    - Simply fits within build procedures, e.g. make
  - An optimized code is generated
    - With respect to a target-machine
- Multiple optimization procedures are applied during the code translation process

Translation Process Drivers

- The semantical nature of the language extensions
  - Exhibited by the source code
- Configuration information

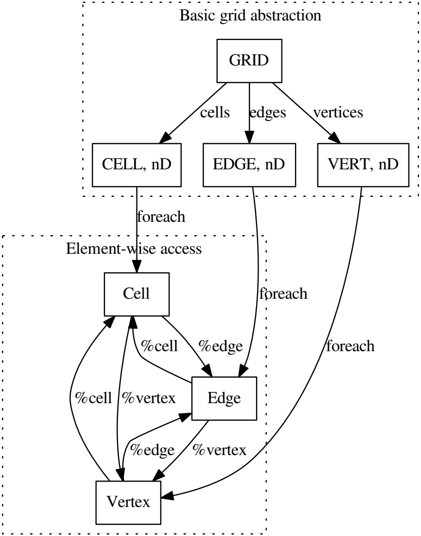
Higher-Level Coding with GGDML

GGDML

- **GGDML**:***G**eneral **G**rid **D**efinition and **M**anipulation **L**anguage*
- Grid definition
- Field declaration
- Field data access/update
    - Iterators
    - Access operators
- Stencil operations

- Hides memory locations and access details, data iteration
- Abstract higher concepts of grids, hiding connectivity details

Abstractions



Fortran vs. GGDML Code Example

```
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)) * REDUCE(+,N, le(cell%neighbour(N))*
     de(cell%neighbour(N))* u(cell%neighbour(N))**2)
END FOREACH
```

User-Controlled Code Translation

- The translation process is highly configurable
  - Users control the optimization procedures
  - The set of the language extensions can be easily extended

Translation Configurations

- Define language extensions
- Control memory allocation/deallocation of fields data
- Define grids
- Control code parallelization
- Control memory layout
- Control communication in multi-node configurations

User-Controlled Code Translation

Declaration Specifers

- NOT a static part of the language
  - Not built in compiler processing
- Defined in groups
- A group allows multiple alternatives for one attribute
- Example spaecifier group definition: SPECIFIER(dim=3D|2D)
  - Defines a dimension specifier group that informs whether the variable represents a 2D or 3D field
- Provide semantical information to the translation tool
  - The tool uses this information during optimization

User-Controlled Code Translation

Access operators

- The user defines
  - The syntax
  - The behavior
- Define grids relationships and connectivity
  - Simplify references to neighborhoods
  - Abstracts the machine notion of array indices with domain concepts, e.g. above, below, neighbor, right, edge...
- Example definition:
  - above(): height=$height+1
  - =>Allows access to the element directly above the current
- Comprises high semantical impact for optimization beside the impact on code quality
  - The translation tool uses the semantics for optimization

User-Controlled Code Translation

Problem Domain and Grids

- Multiple grids can be used
- The user defines the set of access operators that define the connectivity and relationships between the different grids
- Provide the translation tool information about the global problem domain (The whole space over all nodes)
- Allows the translation tool beside to the declaration specifiers to optimize field data access

User-Controlled Code Translation

Memory Layout

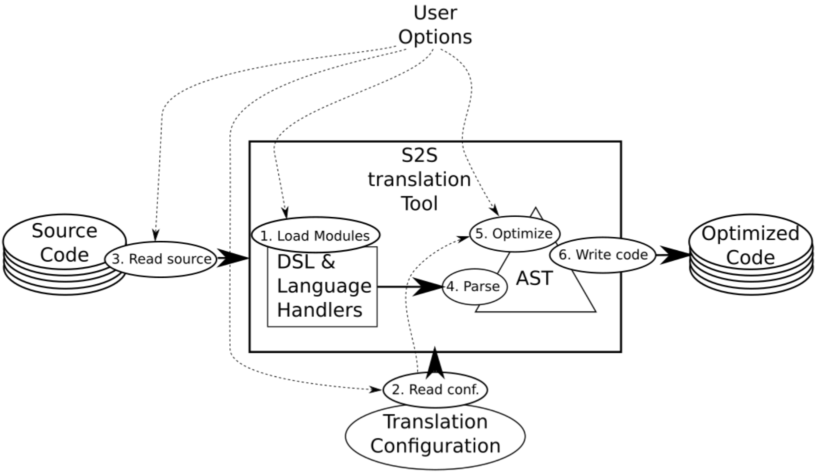
- Completely controlled by the user
  - Memory allocation
  - Array Indices
- The translation tool generates the needed memory layout of a field based on
  - The semantical information used to declare a field
  - The user-provided memory allocation configuration
- The indices are completely controlled by the user
  - Index reordering
  - More complicated formulae to apply mathematical transformations, e.g. Hilber filling curve

User-Controlled Code Translation

Parallelization

- Controlled by the user
  - Single-node and multiple-node configurations
  - Parallelization on node & Over multiple nodes
- The code parallelizaion was tested on
  - multi-core processors (using OpenMP)
  - GPUs (using OpenACC)
  - Multiplr-node MPI(+OpenMP/OpenACC)
- The parallelization on multiple-node configurations is possible
  - The user controls the communication library initialization
  - The user controls the halo exchange code
- The translation tool uses the semantics of the field access to generate the halo exchange code

Translation process



Performance Evaluation

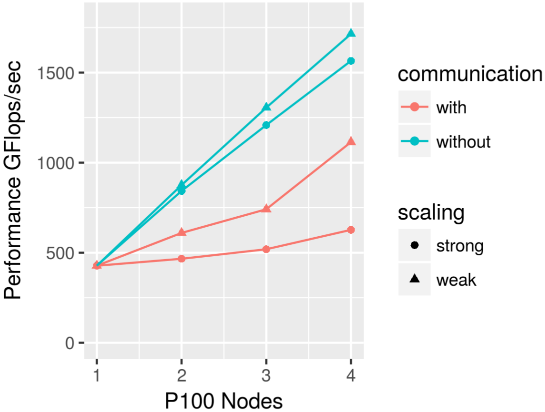
- GPU experiments with OpenACC and OpenACC+MPI
  - Tested on NVIDIA's PSG cluster, on Haswell (E5-2698 v3 @ 2.30GHz) nodes, with P100 and V100 GPUs
  - Testcode: Laplacian on icosahedral (triangles) grid (1024x1024 horizontal x 60 vertical levels)
- The table below shows impact of changing memory layout
  - On P100 and V100 GPUs
  - With 3D array, and a transformed 1D array

Testcode performance on P100 and V100 GPUs

	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

Performance Evaluation

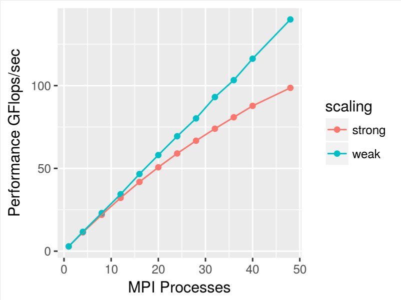
- Multiple-node configurations were tested for scalability
  - Both strong and weak scaling
  - Communication overhead was evaluated to estimate performance cost

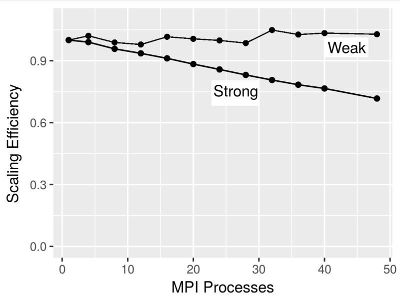


Performance Evaluation

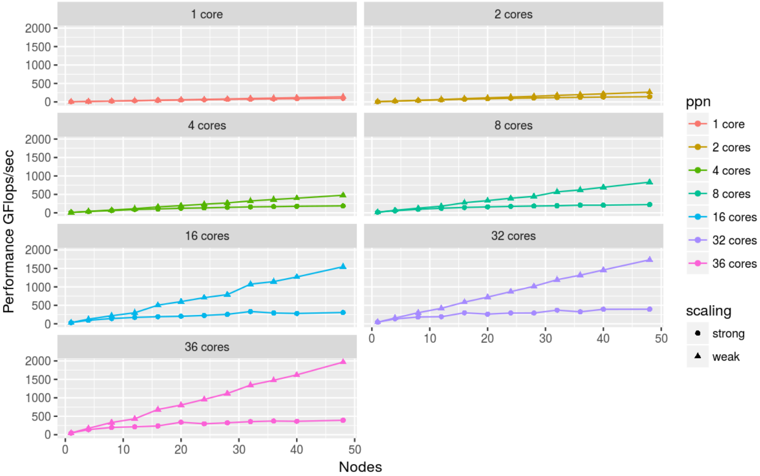
- Multi-core processor experiments with OpenMP and OpenMP+MPI
  - Tested on DKRZ Mistral, on Broadwell (E5-2695 v4 @ 2.1GHz) nodes
  - Same testcode as on GPUs

Testcode performance on Broadwell processors





Performance Evaluation



The testcode scalability under different numbers of MPI processes running different numbers of cores.

Conclusion

- The approach improves the software development process
- The technique is modeling-language neutral
- The extensions provide a slight language change
- Scientists role is restricted to scientific perspective
- Machine perspective is provided within separate configurations
- The translation technique allows users to control translation
- Code transformation supports multiple configurations

Future Work

- Explore applying further (including inter-kernel) optimizations
- Test optimization for other H.W. (e.g. NEC vector processors)
- Investigate high-level (Parallel) IO support with our tools
- Investigate calling optimized libraries by traslating user code

Acknowledgement

- DFG (German Research Foundation)
- DKRZ (German Climate Computer Center)
- NVIDIA