Using Higher-Level Language Extensions to Support Earth-System Modeling

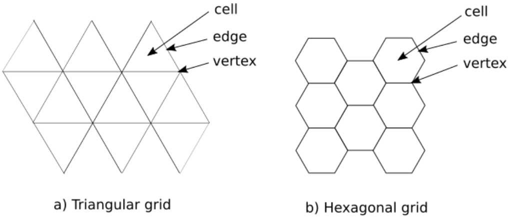
Nabeeh Jum'ah

Scientific Computing
Department of Informatics
University of Hamburg

Leogang Workshop 2018
27-02-2018

Earth-System Modeling

- Simulates the natural processes within the earth system
- Comprises variables that represent different quantities
    - Measured or computed over a specific domain
        - Global
        - Local
- Discretizes the domain into a grid
    - Different types of grids: regular, icosahedral ...
    - The model's variables are measured with respect to grid
        - At the centers of the cells of the grid
        - On the edges of the grid's cells
        - At the vertices of the grid's cells



a) Triangular grid b) Hexagonal grid

Earth-System Modeling

Need for Performance

- Models run many computational kernels within time steps
  - Kernels apply stencil operations to compute model's variables
  - The stencil operation is repeatedly applied over the grid
- The higher resolution grids lead to better simulation accuracy
- The higher resolution grids need more computational resources

Impact on Model Development

- The need to exploit the hardware features is challenging
  - The complexity of the architectural features
  - The diversity of the architectures
- Technical knowledge is needed
  - Hardware features, tools, programming models...

Earth-System Modeling

General-Purpose Languages

- The semantical nature of the languages limit the compilers ability to exploit some optimization opportunities
- Scientists need to manually optimize code
  - Need to learn how to deal with machine features
  - Need to learn new tools and programming models
- Model development for multiple different architectures even complicates the development further in terms of
  - Learning optimization techniques (for multiple architectures)
  - Code duplication
  - Model's maintainability

Improvement Opportunities

Improvement Aspects

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

A Slight Language Shift

- Bypass the shortcomings of the general-purpose languages
- Extend the modeling programming language
  - Based on scientific concepts
  - Hiding lower level details (e.g., architecture, memory layout)

Approach

Separation of Concerns

- Domain scientists formulate scientific logic in source code
- Scientific programmers specify hardware configurations

- Model development with extended language
  - Scientific perspective
    - not machine perspective
  - The need for optimization is dropped from the source code
  - Code is developed once
    - performance is achieved for different configurations
- Hardware configurations define software performance
  - Written by programmers with more experience in platform
  - Comprise information on 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 with 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

DSL Development

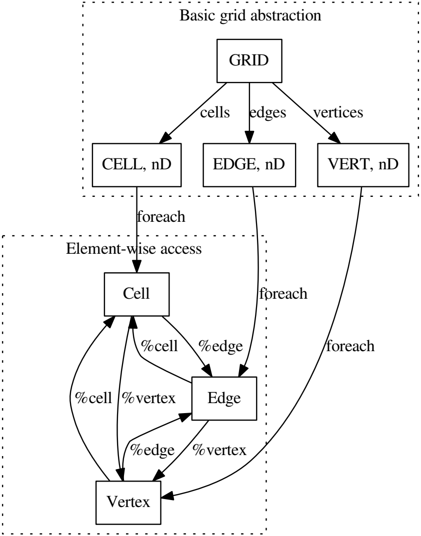
- Iterative development
  - Feedback from scientists

GGDML

- **GGDML**:*__G__eneral __G__rid __D__efinition and __M__anipulation __L__anguage*
- Grid definition
- Variable declaration, allocation and deallocation
- Variable 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
```

Translation Configuration Information

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

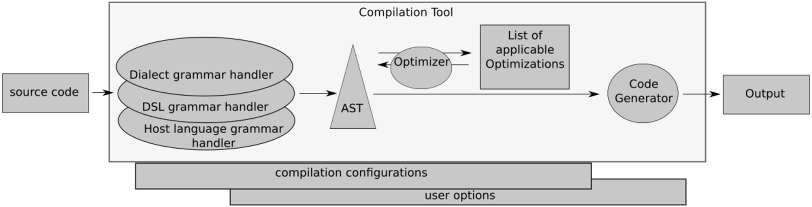
Translation Configurations

- Define language extensions
    - access specifiers
    - access operators
- Control memory allocation/deallocation
- Define grids
- Control code parallelization
- Control memory layout
- Control halo exchange in multi-node configurations

Translation Configuration Information

- Access specifers are defined in groups
  - A group allows multiple alternatives for one attribute
    - e.g. Dimension specifier group: 2D and 3D
- Access operators are defined by the user
  - Simplifies definition of grid connectivity
    - e.g. cell.neighbor, cell.edge
  - Allows the user to add any needed operators
  - Allows the user to control the bahavior of the operator
- The grids of the model are defined in the configuration
  - Global domain is defined
  - Grids relationships are defined through access operators
- Memory layout is completely controlled by user
  - Memory allocation
  - Index transformations including mathematical transformations
- Communication is controlled by user
  - Initializing communication libraries
  - Communicating the halo

Translation process

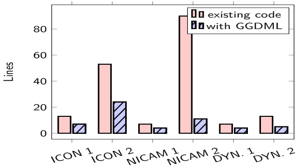


GGDML Impact on the Source Code

The DSL reduces development and maintenance effort

- LOC statistics

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
relative size with dsl	30%		47%		45%	



- Applying the DSL to 300k code of ICON
  - 100k infrastructure (does not change with the DSL)
  - Remaining code reduced according to our test kernels
  - COCOMO estimations

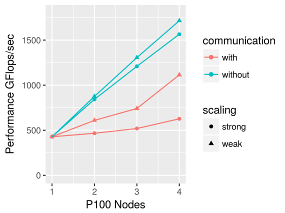
Software project	Version	Effort Applied	Dev. Time (months)	People require	dev. costs (M€)
Semi-detached		2462	38.5	64	12.3
	DSL	1133	29.3	39	5.7
Organic		1295	38.1	34	6.5
	DSL	625	28.9	22	3.1

Performance Evaluation

- Current tool's implementation can transform code into
    - GPU code with OpenACC
    - MPI code on multi-node configurations (MPI+OpenACC)
- 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

- Results for:
  - Multi-core processor code with OpenMP
  - MPI code on multi-node configurations (MPI+OpenMP)

Testcode performance on Broadwell processors