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

Nabeeh Jum’ah

Scientific Computing
Department of Informatics
University of Hamburg

Reading, 16-04-2018
AIMES Project

Address key issues of icosahedral earth-system models

- Enhance programmability and performance-portability
- Overcome storage limitations
- Shared benchmark for these models

Covered models

ICON
DYNAMICO
NICAM
Earth-System Modeling

Modeling with 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
Introduction

Higher-Level Language Extensions

Experiments

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**: General Grid Definition and Manipulation Language
- 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

Basic grid abstraction

GRID

CELL, nD

EDGE, nD

VERT, nD

cells -> edges -> vertices

foreach

Element-wise access

Cell

Cell

Edge

Vertex

foreach

foreach

foreach

foreach
Fortran vs. GGDML Code Example

```fortran
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 specifiers 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 behavior 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

source code → Dialect grammar handler → DSL grammar handler → Host language grammar handler → AST → Compilation Tool → List of applicable Optimizations → Code Generator → Output

compilation configurations

user options
GGDML Impact on the Source Code

The DSL reduces development and maintenance effort

- **LOC statistics**

<table>
<thead>
<tr>
<th>Model, kernel</th>
<th>lines (LOC) before DSL</th>
<th>lines (LOC) with DSL</th>
<th>words before DSL</th>
<th>words with DSL</th>
<th>characters before DSL</th>
<th>characters with DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICON 1</td>
<td>13</td>
<td>7</td>
<td>238</td>
<td>174</td>
<td>317</td>
<td>258</td>
</tr>
<tr>
<td>ICON 2</td>
<td>53</td>
<td>24</td>
<td>163</td>
<td>83</td>
<td>2002</td>
<td>916</td>
</tr>
<tr>
<td>NICAM 1</td>
<td>7</td>
<td>4</td>
<td>40</td>
<td>27</td>
<td>76</td>
<td>86</td>
</tr>
<tr>
<td>NICAM 2</td>
<td>90</td>
<td>11</td>
<td>344</td>
<td>53</td>
<td>1487</td>
<td>363</td>
</tr>
<tr>
<td>DYNAMICO 1</td>
<td>7</td>
<td>4</td>
<td>96</td>
<td>73</td>
<td>137</td>
<td>150</td>
</tr>
<tr>
<td>DYNAMICO 2</td>
<td>13</td>
<td>5</td>
<td>30</td>
<td>20</td>
<td>402</td>
<td>218</td>
</tr>
<tr>
<td>total</td>
<td>183</td>
<td>55</td>
<td>911</td>
<td>430</td>
<td>4421</td>
<td>1991</td>
</tr>
</tbody>
</table>

  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

<table>
<thead>
<tr>
<th>Software project</th>
<th>Version</th>
<th>Effort Applied</th>
<th>Dev. Time (months)</th>
<th>People require</th>
<th>dev. costs (M€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-detached</td>
<td>DSL</td>
<td>2462</td>
<td>38.5</td>
<td>64</td>
<td>12.3</td>
</tr>
<tr>
<td>Organic</td>
<td>DSL</td>
<td>1295</td>
<td>38.1</td>
<td>34</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>625</td>
<td>28.9</td>
<td>22</td>
<td>3.1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>P100</th>
<th>V100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Memory throughput</td>
<td>Memory throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GF/s</td>
<td>GB/s</td>
</tr>
<tr>
<td>read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D</td>
<td>1.97</td>
<td>220.38</td>
<td>91.34</td>
</tr>
<tr>
<td>3D-1D</td>
<td>1.99</td>
<td>408.15</td>
<td>38.75</td>
</tr>
</tbody>
</table>

![Graph showing performance comparison](image)
Performance Evaluation

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

Testcode performance on Broadwell processors
Performance Evaluation

- 1 core
- 2 cores
- 4 cores
- 8 cores
- 16 cores
- 32 cores
- 36 cores

- ppn
- strong
- weak
PhD

- "Language Extensibility and Configurability to Support Earth System Modeling"
  - Modeling-language extensibility
  - Modeling-language configurability
- Define a basic set of suggested extensions to support the development with domain-specific concepts
- Explore possibilities to support model-specific needs
- Investigate impact and performance gain in icosahedral models as a results of using the extensions with the meta-dsl translation technique.