The atmospheric and climate sciences and the natural sciences in general are increasingly demanding for higher performance computing. Unfortunately, the models that describe the state of the atmosphere and climate are extremely complex and require an efficient use of hardware resources. This is where the GGDML (General Grid Definition and Manipulation Language) comes into play. GGDML provides a way to abstract away the details of the hardware architecture, allowing scientists to focus on the scientific problems rather than the technical details of the underlying hardware.

GGDML has been developed in a co-design approach in collaboration with domain scientists. The goal was to provide performance portability to the icosahedral climate modeling we have developed a set of higher-level language extensions that we call GGDML E. The extensions provide semantically-higher-level constructs with which scientists can express their scientific problem with scientific concepts.

In order to provide performance portability to the icosahedral climate modeling we have developed a set of higher-level language extensions that we call GGDML. The extensions provide semantically-higher-level constructs with which scientists can express their scientific problem with scientific concepts. This allows the need to explicitly provide lower-level machine-dependent constructs so scientists can use the general-purpose language.

The parts of the code in which a scientist uses the GGDML extensions are translated by source-to-source translation tool that optimizes the generated code to a specific machine. The translation process is driven by configurations that are provided independently from the source code.

In this poster we review some GGDML extensions and we focus mainly on the configurable code translation of the higher-level code.

**Goals**

With the approach that we suggest we aim at an enhanced and more productive software development process through which a single source code that is easily maintainable can be developed. The source code is mainly developed with the general-purpose language that the developer scientists choose for modeling. The code which would eventually run on a machine will exploit its performance-supporting capabilities. The software development process fosters operation of concerns:

- Scientists from the domain science provide the problem logic in terms of scientific concepts.
- The configuration that is responsible for platforms-dependent implementation is provided by scientific programmers.

** GGDM Extensions**

GGDML (General Grid Definition and Manipulation Language) provides abstract grid concepts that support unstructured grids like icosahedral models besides regular grids.

GGDML has been developed in a co-design approach in collaboration with domain scientists. The set of GGDML extensions:

- Extends a general-purpose language – It extends the grammar of the language
- The concept applies to the different languages
- Allows for the definition of grids
- Abbreviations are e.g., triangular, hexagonal

- Allows to define variables on the grid
- Allows to reference variables by grid elements
- Linked element relationships
- to reference cell edge
- to reference cell above/below
- to reference a neighbour cell

- Provides an iterator to traverse the grid
- Specify/dimensions of ranges
- Update data of variables while traversing
- Provides a reducer operation

**Translation Configuration**

- Allows to control the way of the variable declaration on the grid.
- This is handled by specifying the extensions within the configuration file
- Groups of alternatives that are provided by the configuration control the variable declaration
- 2/3D group to control the dimensionality of the grid
- CELL/EDGE/VERTEX to control the grid parts where the variable is measured/computed
- The groups and the alternatives are dynamic, they can be changed/expanded on need

**Code Example**

The following example demonstrates the use of GGDML for vertical integration:

```plaintext
DECLARE: SPECIFIED (tile=CELL/EDGE) SPECIFIED (dim=3D;2D)

ALLOCATION:
CASE loc=CELL & dim=3D:
  cell_name = (data_type=restrict)malloc(c ->cellSize * 
  g->cellIndex + (height) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 
  (grid->neighbors) * g->cellIndex + 
  (grid->neighbors) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 
  (global_domain.neighbors) * g->cellIndex + 

  Provides the way to specify the default dimensions of the grid and its components
  - Can serve to define structured and unstructured grids
  - The different components of the grid are configurable, for example
  - The set of the cells of the 3D grid (same for edges, vertices, whatever component needed)
  - The set of the cells on the surface (2D)
  - The default grid specifications can be overridden in an iterator for a specific kernel

GLOBALDOMAIN:
  CELL: 0 TO g->height

INDEXEXSORT:
  index = (dim=3D)
  above(): height=height+1

INDEXEXSORT:
  allows to define the memory layout of the variables declared with the extensions
  - The configuration allows different layouts by changing
  - the memory layout when allocating the variables
  - generating the right indices when accessing a variable in an iterator
  - The configuration allows index transformation with general-purpose language expressions
  - even transformation functions (e.g. Hilbert filling curve)
  - The configuration also provides the ways to exchange data with fixed memory layout

INDEXEXSORT:
  allows to control code annotation
  - This allows the configuration to guide annotating code with OpenMP or OpenACC for example

MEMORY LAYOUT AND PERFORMANCE IMPACT

The figure (right) shows the impact of changing the memory layout of an application with various optimization options with intel compiler(on an Intel(R) Core(TM) i7-860 @ 2.80 GHz):

- 3.6x improvement with OpenMP
- 3.4x improvement with OpenACC
- 3.3x improvement with OpenMP and OpenACC

The table below shows the impact of changing the memory layout of a stencil code of 3x, 7-point stencil on CPUs (lgy Bridge E5-2690 v2, 3.0GHz) and GPUs (Nvidia K80 and F100) –I/GP compiler.

**Code Quality**

We have already taken two relevant kernels from each of the three icosahedral models: ICON,Nicam, and Dynamico, and analyzed the achieved code reduction. The figure below gives an indication for that:

- In average, we cut down the LOC to (30%) of the original code. Better reductions are achieved in stencil codes (NICAM example No.2, reduced to 12.22% of the original LOC).
- Code reduction reduces development time and costs. By applying COCCOMO to a case model we estimated a cost reduction from 12.3 to 5.7 M€ for a project with semi-detached team and from 6.5 to 3.1 M€ for organic team.

**Summary**

- GGDML extensions provide a way to improve climate/atmospheric models development
- GGDML extensions lift the model development process to a higher level that enables improved code maintainability & readability while providing performance portability.
- GGDML and the translation technique eliminate the need for lower-level architecture-specific details in the source code.
- GGDML significantly reduces the size of the source code and model development costs.
- A target-specific configuration (independent of the source code) offers the generation of a machine-dependent optimized code.
- Scientists do not need to care about computing details, scientific programmers write the configuration that leads the optimization process.
- The whole process is controlled and driven by the users, thanks to the configuration flexibility which allows to define/language the extensions.

**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 1553/11-1).

**References**
