

# Feasibility Study for Achieving Performance Portable Global Weather and Climate Models on Icosahedral Grids using DSL Libraries

Xiaolin Guo

June 6, 2016

## **Abstract**

I am your abstract

## **1 Introduction**

The ICON model is a next generation model for weather and climate simulations that aims a global simulation at very high resolution. The new dynamical core of ICON solves the equations in a fully compressible non-hydrostatic mode, and its discretization in the grid point space of an icosahedral grid facilitates a quasi-isotropic horizontal resolution on the sphere, the coupling of ocean and land as well as local refinement in nested domains.

The model implemented in Fortran 90, has been implemented and optimized for x86 architectures. Current trunk of the software is lacking of a performance portable version of the code, capable of running efficiently in multiple architectures, like modern accelerators NVIDIA GPUs or Intel Xeon Phi.

The GridTools library is a modern library for stencil computations implemented in C++11. It provides a DSL specifically designed for weather and climate models for regional and global models. GridTools is the next generation of an already existing library named STELLA [?].

GridTools aims at achieving performance portable codes for weather and climate models and highly increase readability of the equations of the model by abstracting the implementation details, making use of a language at higher level than generic programming

languages like Fortran or C++. GridTools will support multiple architecture backends (x86, CUDA, XeonPhi,...) and multiple grid topologies (structured or curvilinear, icosahedral, octahedral, unstructured meshes, etc).

Contrary to the ICON (and most of the global models on icosahedral grids) that treats the icosahedral grid as an unstructured mesh using indirect addressing, GridTools is exploring a description and implementation of the grid in a structured way, in order to exploit the maximum performance of the accelerators.

Illustration Illustration shows the original domain decomposition in 10 diamonds, while Illustration Illustration shows the way GridTools indexes the cells in order to treat the icosahedral grid in a structured manner.

Currently the support for structured grids is quite advanced and will be finalized in 2016. A first implementation for icosahedral grids for x86 and cuda backend exists (performance optimizations will be implemented in the course of 2016) for a single trapezoid of the globe decomposition.

## 2 Icosahedral grid and the GridTools

### 2.1 C-type staggering

### 2.2 MPAS

### 2.3 The GridTools

## 3 Studies on operators

### 3.1 Orientation of edge normal and neighbor order

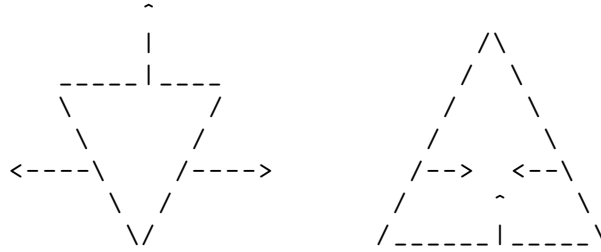
In ICON, we denote the direction of  $\mathbf{N}_l$  on a cell by a sign. It is equal to +1 if the normal to the edge is outwards from the cell, otherwise is -1.

The direction of  $\mathbf{N}_l$  does not need to follow a certain pattern. When we perform computations involving  $\mathbf{N}_l$ , like in a divergence operator in ICON, we just read the data field that stores this sign. But if we can fix this sign, then we can save the reading of this data field,

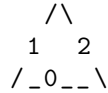
and simplify the computation. Here we propose fixing the sign by 1 on upward cells and  $-1$  on downward cells. We will evaluate the performance of this proposal in later sections.

## 3.2 Neighbor order

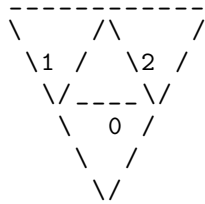
```
* Following specializations provide all information about the
  connectivity of the icosahedral/octahedral grid
* While ordering is arbitrary up to some extent, it must respect some
  rules that user expect, and that conform
* part of an API. Rules are the following:
* 1. Flow variables on edges by convention are outward on downward
  cells (color 0) and inward on upward cells (color 1)
* as depicted below
```



```
* 2. Neighbor edges of a cell must follow the same convention than
  neighbor cells of a cell. I.e. the following
```



```
* imposes
```



```
* 3. Cell neighbours of an edge, in the order 0 -> 1 follow the
  direction of the flow (N_t) on edges defined in 1.
```

```
* This fixes the order of cell neighbors of an edge
```

```
* 4. Vertex neighbors of an edge, in the order 0 -> 1 defines a
  vector N_l which is perpendicular to N_t.
```

```
* This fixes the order of vertex neighbors of an edge
```

### 3.3 div

We define the discrete divergence operator of a vector field  $\mathbf{v}$  as

$$\text{div}(\mathbf{v})_i := \frac{1}{A_i} \sum_{l \in \mathcal{E}(i)} v_{n_l} (\mathbf{N}_l \cdot \mathbf{n}_{i,l}) l \quad (1)$$

A most straightforward approach to compute this operator is

$$\text{div}(\mathbf{v})_i := \sum_{l \in \mathcal{E}(i)} v_{n_l} w_{i,l} \quad (2)$$

where  $w_{i,l} = \frac{1}{A_i} (\mathbf{N}_l \cdot \mathbf{n}_{i,l}) l$  is the weight for div and should be computed at the initialization phase.

The second approach takes into consideration the convention of normal orientation for downward and upward cells. The divergence operator is thus given by

$$\text{div}(\mathbf{v})_i = \frac{1}{A_i} \sum_{l \in \mathcal{E}(i)} v_{n_l} l \quad \text{for upward cells} \quad (3)$$

$$\text{div}(\mathbf{v})_i = -\frac{1}{A_i} \sum_{l \in \mathcal{E}(i)} v_{n_l} l \quad \text{for downward cells} \quad (4)$$

because on upward cells  $\mathbf{N}_l$  and  $\mathbf{n}_{i,l}$  are of the same direction whereas on downward cells they are of opposite directions. We have one more division/multiplication comparing to eq. (2).

The first two approaches are primarily based on cells. However, since two cells share an edge, we are computing  $v_{n_l} l$  twice for each  $l$ . The third approach aims to avoid this by iterating primarily on edges.

$$\begin{aligned} \text{div}(\mathbf{v})_{\varkappa(l,0)} &\leftarrow \text{div}(\mathbf{v})_{\varkappa(l,0)} + v_{n_l} l \\ \text{div}(\mathbf{v})_{\varkappa(l,1)} &\leftarrow \text{div}(\mathbf{v})_{\varkappa(l,1)} - v_{n_l} l \end{aligned} \quad (5)$$

$$\begin{aligned} \text{div}(\mathbf{v})_{\varkappa(l,0)} &\leftarrow \text{div}(\mathbf{v})_{\varkappa(l,0)} / A_{\varkappa(l,0)} \\ \text{div}(\mathbf{v})_{\varkappa(l,1)} &\leftarrow \text{div}(\mathbf{v})_{\varkappa(l,1)} / A_{\varkappa(l,1)} \end{aligned} \quad (6)$$

Here eq. (6) is primarily on edges, and assumes the convention that cell neighbors of an edge follow the direction of the  $\mathbf{N}_l$ , as well as the normal orientation convention.

Approach	Primal location	Operation on primal location
weights	Cells	6 read, 1 write, 3 mult, 3 plus
2	Cells	7 read, 1 write, 4 mult, 3 plus
3	Edges	4 read, 2 write, 1 mult, 2 plus + 2 read, 2 write, 2 mult

Approach	R02B05	R02B06	R02B07
1	0.0677755	0.317447	1.24704
2	0.0864071	0.423643	1.63169
3	0.101011	0.482151	1.90736

### 3.4 grad

The discrete gradient operator is given by

$$\text{grad}_n(\varrho)_l := \frac{\varrho_{i(l,2)} - \varrho_{i(l,1)}}{\hat{l}} \quad (7)$$

Without the flow convention