The Interface between the MPAS Land Ice Dynamic Core and the Velocity Solver Requirements and Design

MPAS Development Team

June 8, 2011
Contents

1 Summary 2

2 Requirements 4
  2.1 Requirement: MPAS Supplies Grid and Geometry Input Needed by Velocity Solver 4
  2.2 Requirement: MPAS Supplies State Variable Input Needed by Velocity Solver 4
  2.3 Requirement: Velocity Solver Returns Output Needed by MPAS 5

3 Algorithmic Formulations 6
  3.1 Design Solution: Identification of Grid and Geometry Input Needed by Velocity Solver 6
  3.2 Design Solution: Identification of State Variable Input Needed by Velocity Solver 6
  3.3 Design Solution: Identification of Velocity Solver Output 7

4 Design and Implementation 8
  4.1 Implementation: Grid and Geometry Input from MPAS to Velocity Solver 8
  4.2 Implementation: State Variable Input from MPAS to Velocity Solver 8
  4.3 Implementation: Output from Velocity Solver to MPAS 9
  4.4 Implementation: Format of the Grid and Geometry Interface 10
  4.5 Implementation: Format of the State Variable Interface 10

5 Testing 13
  5.1 Testing and Validation: Passing Dummy Data Through the Interface 13
Chapter 1

Summary

The MPAS system is developing a new land ice dynamic core. The geometry of a given region is modeled by a grid of cells which are organized into vertical layers or sheets. The values of physical quantities of interest, called state variables, are to be computed at specific locations within each cell, and over a sequence of time values. Thus a typical computation involves some initial setup, associated with defining the grid, followed by a sequence of timestep calculations which compute estimates for the state variables at the next time.

A particular state variable needed at each time step is the instantaneous velocity field. There are a number of approaches to computing this quantity, and the MPAS land ice development team has decided that the computation should be carried out by a separate procedure, here called the velocity solver module.

It is desirable that the internal details of the velocity solver module be left free to the developers. This is so because:

- it may be desirable to provide a choice of solvers corresponding to different models;

- velocity solvers may be designed by independent groups;

- it may be preferable to apply the finite element method to model the velocity equations.

Thus, the land ice dynamic core, as it carries out a time step, must be prepared to transfer information about the current state to the velocity solver module, and then to accept back the velocity field as it has been computed.
Given these considerations, the appropriate approach is to carefully define an interface that controls the flow of state information, geometry, and other conditions to the velocity solver, and the return of information that defines the velocity field. If this interface is carefully and clearly defined, then the velocity solver itself can be considered a “black box”. As long as designers carry out the required calculations, and respect the communication interface, they will then be free to implement the computation according to their interests.

This document specifies the details of the interfaces needed in order for the MPAS land ice dynamic core to successfully interact with a velocity solver module. This entails the information needed to specify the calculation, and the format of the desired results. Furthermore, it lays out an explicit interface, that is, a formal function call, with a parameter list, whose types, dimensions, and meanings are determined.

This information allows developers of the MPAS dynamic core and those working on velocity solvers to proceed independently on code design, without having to wait for each other. In particular, a group working on one side of the interface can test their code by supplying a simple dummy function to stand on the other side of the interface and supply “canned data”.

A first step towards judging the success of the interface is to run the full program on one side, with a dummy function on the other, which simply goes through the motions of computing data, but actually simply sends a standard set of test data. Thus, the velocity solver can be partially tested by sending it data from a dummy version of MPAS, while the MPAS land ice dynamic core can be tested by having a dummy velocity solver return canned velocities back through the interface.
Chapter 2

Requirements

2.1 Requirement: MPAS Supplies Grid and Geometry Input Needed by Velocity Solver

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

MPAS stores the geometry, grid, boundary conditions, and state variables for the entire computation. The velocity solver needs some of this information once, at the beginning of the computation, in order to set up its own geometry model. Thus, it is required that MPAS supplies to the velocity solver the grid and geometry information, before any time steps are computed, using an agreed interface.

2.2 Requirement: MPAS Supplies State Variable Input Needed by Velocity Solver

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

MPAS stores and updates state variables for the entire computation. The velocity solver will need to receive updated information on the current values of the state variables and boundary conditions in order to correctly set up the calculation for the corresponding velocity at the new time. Thus, it is required that MPAS supplies to the velocity solver the changing state information at each time step, using an agreed interface.
2.3  Requirement: Velocity Solver Returns Output Needed by MPAS

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

During each time step, once the velocity solver has determined the velocity field, this information must be returned to MPAS. The velocity solver can also determine certain related quantities derived from the velocity information, and needed by MPAS. This information must be supplied in a suitable format, and at appropriate geometric positions, as required by MPAS. Thus, it is required that the velocity solver returns information to MPAS during each time step, using an agreed interface.
Chapter 3

Algorithmic Formulations

3.1 Design Solution: Identification of Grid and Geometry Input Needed by Velocity Solver

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

The velocity solver needs the location of the cell centers of the MPAS mesh, the number of layers, and various connectivity arrays in order to recreate the discretized geometry. Assuming the basic structure of the mesh does not vary over the time steps, then this information should only need to be transferred once.

3.2 Design Solution: Identification of State Variable Input Needed by Velocity Solver

Date last modified: 2011/06/01
Contributors: John Burkardt, Mauro Perego

In order to set up the equations for the velocity, the velocity solver needs, on each time step, the state variables and boundary condition information that affect this computation.
3.3 Design Solution: Identification of Velocity Solver Output

Date last modified: 2011/06/01
Contributors: John Burkardt, Mauro Perego

Once the velocity solver has computed a solution, MPAS will require velocity values, as well as certain other derived quantities, at specific locations in the mesh. The velocity solver may carry out interpolation or other suitable procedures in order to produce these values at the desired locations.
Chapter 4

Design and Implementation

4.1 Implementation: Grid and Geometry Input from MPAS to Velocity Solver

Date last modified: 2011/06/02
Contributors: John Burkardt, Mauro Perego

It is assumed that the geometry of the MPAS grid is fixed throughout the computation.

The velocity solver needs mesh information from MPAS in order to construct the grid. This information includes:

- \((x_{Cell}, y_{Cell}, z_{Cell})\), the coordinates of the MPAS cell centers; under discussion is the idea of using 2D coordinates from a polar stereographic projection;
- \(\text{cellsOnVertex}\), the indices of cells incident on a vertex;
- \(\text{indexToVertexID}\), the tags for vertices;
- \(\text{verticesOnEdge}\), indices of the two vertex endpoints of a cell edge;
- \(n_{VertLevels}\), the number of vertical layers.

4.2 Implementation: State Variable Input from MPAS to Velocity Solver

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego
As each new time step begins, MPAS updates certain variables that define the state of the system, and which the velocity solver needs in order to properly set up the system defining the velocities. These items include:

- **temperature**, the temperature at each cell center, for each layer;
- **thickness**, the thickness, associated with each cell center;
- **elevation**, the elevation, associated with each cell center;
- **?**, the sigma layers (*no name suggested; not clear on the meaning*);
- **beta**, the sliding coefficient, and other basal parameters, associated with each cell center.
- **flowfactor**, the flow factor \( A(T) \) of each layer in each cell;
- **emptyCell**, a logical variable that indicates whether a cell is empty (*this is a suggestion for how to handle the “empty cell” issue.*).

### 4.3 Implementation: Output from Velocity Solver to MPAS

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

The velocity solver determines the velocity vector field. MPAS requires the velocity solver to return certain velocity components at selected points, as well as related information:

- \( u, v \), normal and tangential velocities at the midpoint of each edge face of each layer;
- **heat dissipation**, the tensor product of the full stress tensor and the strain tensor integrated over each layer of each cell;
- **viscosity**, possibly, for diagnostics, the viscosity of each cell;
- **?**, possibly, for diagnostics, terms of the stress and strain rate tensors; (*no name specified; not clear on details.*);
4.4 Implementation: Format of the Grid and Geometry Interface

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

Before the time step calculation begins, the MPAS routine `mpas_core_init` in module `mpas_core` sets up the geometry and the grid. The velocity solver needs a copy of some of this information in order to build the corresponding geometric model for its calculations. The information can be transferred by a call from `mpas_core_init` with the following proposed format:

```fortran
call velocity_solver_grid ( nCells, nEdges, nVertices, &
   nVertLevels, cellsOnVertex, indexToVertexID, &
   verticesOnEdge, xCell, yCell, zCell )
```

4.5 Implementation: Format of the State Variable Interface

Date last modified: 2011/06/07
Contributors: John Burkardt, Mauro Perego

Each time integration step of the land ice calculation is carried out in module `time_integration`. Within this module, subroutine `timestep()` controls the computation of data associated with the new time by calling the routine `rk4()`, which uses a fourth-order Runge Kutta method to advance the solution data in time.

Since the velocity solver, instead of `rk4()`, will be computing the new velocities, then `rk4()` must be modified so that it no longer attempts to compute velocity updates, and `timestep()` must now call both `rk4()` and the velocity solver. The generic format of this portion of `timestep()` might look something like this:

```fortran
if (trim(config_time_integration) == 'RK4') then
   call rk4 ( domain, dt )
else
   write(0,*), 'Unknown time integration option ' &
   // trim ( config_time_integration )
   write(0,*), 'Currently, only ' 'RK4' ' is supported.'
```

stop
end if

call velocity_solver_time ( nCells, nEdges, &
nVertLevels, beta, elevation, emptyCell, flowfactor, &
temperature, thickness, heatIntegral, u, v, viscosity )
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Dimension</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>double precision</td>
<td>beta</td>
<td>(nCells)</td>
<td>the sliding coefficient.</td>
</tr>
<tr>
<td>integer</td>
<td>cellsOnVertex</td>
<td>(nVertices,3)</td>
<td>the indices of cells incident on a vertex.</td>
</tr>
<tr>
<td>double precision</td>
<td>elevation</td>
<td>(nCells)</td>
<td>the elevation at each cell center.</td>
</tr>
<tr>
<td>logical</td>
<td>emptyCell</td>
<td>(nCells)</td>
<td>true if cell is empty.</td>
</tr>
<tr>
<td>double precision</td>
<td>flowfactor</td>
<td>(nCells,nVertLevels)</td>
<td>the flow factor at cell centers.</td>
</tr>
<tr>
<td>integer</td>
<td>indexToVertexID</td>
<td>(nVertices)</td>
<td>the unique integer tag for each vertex.</td>
</tr>
<tr>
<td>double precision</td>
<td>heatIntegral</td>
<td>(nCells,nVertLevels)</td>
<td>heat dissipation integrated over each cell.</td>
</tr>
<tr>
<td>integer</td>
<td>nCells</td>
<td>1</td>
<td>the number of cells.</td>
</tr>
<tr>
<td>integer</td>
<td>nEdges</td>
<td>1</td>
<td>the number of cell edges.</td>
</tr>
<tr>
<td>integer</td>
<td>nVertices</td>
<td>1</td>
<td>the number of cell vertices.</td>
</tr>
<tr>
<td>integer</td>
<td>nVertLevels</td>
<td>1</td>
<td>the number of vertical layers (typically 10 or 11).</td>
</tr>
<tr>
<td>double precision</td>
<td>temperature</td>
<td>(nCells,nVertLevels)</td>
<td>temperature at cell centers.</td>
</tr>
<tr>
<td>double precision</td>
<td>thickness</td>
<td>(nCells)</td>
<td>layer thickness at cell center.</td>
</tr>
<tr>
<td>double precision</td>
<td>u</td>
<td>(nEdges,nVertLevels)</td>
<td>the normal velocity at the midpoint of each edge face of each layer.</td>
</tr>
<tr>
<td>double precision</td>
<td>v</td>
<td>(nEdges,nVertLevels)</td>
<td>the tangential velocity at the midpoint of each edge face of each layer.</td>
</tr>
<tr>
<td>integer</td>
<td>verticesOnEdge</td>
<td>(nEdges,2)</td>
<td>the pair of vertices that are the endpoints of a cell edge.</td>
</tr>
<tr>
<td>double precision</td>
<td>viscosity</td>
<td>(nCells,nVertLevels)</td>
<td>the viscosity of each cell.</td>
</tr>
<tr>
<td>double precision</td>
<td>xCell</td>
<td>(nCells)</td>
<td>the x coordinate of the cell center on the unit sphere.</td>
</tr>
<tr>
<td>double precision</td>
<td>yCell</td>
<td>(nCells)</td>
<td>the y coordinate of the cell center on the unit sphere.</td>
</tr>
<tr>
<td>double precision</td>
<td>zCell</td>
<td>(nCells)</td>
<td>the z coordinate of the cell center on the unit sphere.</td>
</tr>
</tbody>
</table>

Table 4.1: Glossary of MPAS/Velocity Solver Interface Variables
Chapter 5

Testing

5.1 Testing and Validation: Passing Dummy Data Through the Interface

As soon as a reasonably detailed interface has been agreed upon, it will be possible for developers on either side of the interface to test their code, and the interface, by creating a dummy version of the program that would normally be running on the other side of the interface.

Thus, the MPAS developers will be able to write a simple procedure for producing “plausible” velocity values and related quantities, and to call that procedure through the interface.

Similarly, developers of the velocity solver functions can set up main programs that pass in a test case geometry and time evolution data.

Such trial runs enable each developer to make simple test runs of their code in cases where the other “half” of the code is not yet available; it also allows developers to spot situations in which the interface does not include certain information that is required for the calculations to proceed properly.