Development of Aeolus 1.0
A Statistical-Dynamical Atmosphere Model

Dim Coumou, Alexey V. Eliseev
Vladimir Petoukhov, Stefan Petri

ModStrat Seminar 14 Nov. 2013
Content

• Intro Aeolus and Climber-4
• Key Modules
• Object-oriented Code Design
  Strict separation of Data Storage, Gridding and Computations
• Safety Mechanisms:
  Explicit Read-Write Functionality, Design-by-Contract, Automatic Range Checking
• Documentation
• Testing / Benchmarking
  Module-wise, Stand-alone, Fully-coupled
• Handy Tools:
  Totalview, valgrind, VC++ (or in general IDEs)
Hierarchy of Atmosphere Models

Atmosphere: Increasing complexity

- EBM
- Aeolus
- CM2C (low-res)
- AM2 (high-res)

Sea ice
Ocean: MOM5

Vegetation:
- LPJmL

Ice sheets:
- PISM-PIK

Land

FMS Coupler
Synoptic Scale Eddies

Atmospheric Power spectrum

\[ \bar{Z}^2 (10^2 m^2) \]

\begin{tabular}{c|c|c|c|c|c}
\hline
\textbf{Wavenumber} & \textbf{Period ( days )} & \textbf{Atmospheric Power spectrum} & \textbf{GCMs} & \textbf{EXPLICITLY SOLVED} & \textbf{HIGH RES in Space & Time} \\
\hline
\end{tabular}

\(~30^\circ, \sim 3000km, k > 7\)

NB: Interested in Climate rather than Weather!
Synoptic Scale Eddies

SDAMs

Atmospheric Power spectrum

$\overrightarrow{V} = \langle \overrightarrow{V} \rangle + \overrightarrow{V}'$

$\sim 30^\circ, \sim 3000\text{km}, k \sim 10$

Wavenumber

Period (days)

EXPLICITLY SOLVED

PARAMETERIZED
Aeolus

- Object-oriented C++ implementation
- ~45k code lines
- svn version control
- MPI Parallel Computing

Compute Modules:
- Dynamical Core
- 3-Layer Clouds
- Radiative Transfer
- Mass/Energy Transport

I/O:
- NetCDF
- namelist

Coupling:
- FMS
- SimENV
Grid: Core Building Blocks

Cell

- Coordinates
- Volume
- ID
- Boundary Flag (SPOLE / NPOLE / etc)
- Neighbors (Cells & Interfaces)

Interface

- Coordinates
- Area
- ID
- Normal vector
- Boundary Flag (SPOLE / NPOLE / etc)
- Neighbors (Cells)
Grids

SphericalGrid

vector<Cell>

... Functions for fast access of specific cells and interfaces

Numerical Stability

Finite Difference:
\[ \Delta t < \frac{2 \Delta x^2}{K} \]

Finite Volume:
\[ \Delta t < \frac{\Delta x}{|\vec{V}|} \]

‘Internally consistent, self-described grid’

Standard Spherical Grid

Reduced Spherical Grid
Data Storage

- **Variable**
  - data
  - name
  - unit

- **CellVariable**

- **InterfaceVariable**
  - ..., ..., ...

Primary variables:
- Temperature, Humidity, ...
- Fluxes, velocities, etc
- Specific sub-regions
Explicit Read / Write access

Aeolus

DataStorage
- Cloud variables
- Temperature field
- Water vapor
- .... variables
- .... variables

Read access only

Read & Write access

Compute Modules

3-Layer Clouds

Dynamical Core

Variables:
- Temperature field
- Water vapor
- Wind field
Explicit Read / Write access

Only **one** `Compute` class has write access to **one** `DataStorage` class. This is done using specific `Access-functions` and `friend` definition.

**Example:** 1 layer cloud scheme

```cpp
class Clouds
{

public:

    Clouds(const SphericalGrid&);
    ~Clouds(void);

    // Provide read functionality
    inline const Variable& cumulus_() const { return cumulus; ;
    inline const Variable& stratus_() const { return stratus; ;
    inline const Variable& total_() const { return total; ;

    // Give specific write access to one compute class
    friend class Cloudiness;

private:
    Variable cumulus, stratus, total;
};
```
Data Storage – Write-access:

**Compute classes**

- class Cloudiness_3Layers
- class LargeScaleDynamics
- class SynopticScaleDynamics
- class PlanWave
- class HeatTransfer
- class WaterVaporTransfer
- class RadiativeTransfer
- class PlanetaryBoundaryLayerPhysics
- class Surface Fluxes

**DataStorage classes**

- class Clouds_3Layers
- class LargeScaleWind
- class SynopticScaleField
- class PlanetaryWave
- class Temperature
- class Humidity
- class RadiativeFluxes
- class PlanetaryBoundaryLayer
- class SurfaceLayer
Design-by-Contract (on the function-level)

```c++
void FunctionBlaBla (const Variable& v1, const Variable& v2, Variable& v3)
{
    // Valid input?
    v1.IsValid();
    v2.IsValid();

    // Calculate v3:
    ...

    // Valid output?
    v3.IsValid();
}
```

→ Checks done in *Debug-mode* only, turned-off in *Release-mode*
Content

• Intro Aeolus and Climber-4
• Key Modules
• Object-oriented Code Design
  Strict separation of Data Storage, Gridding and Computations
• Safety Mechanisms:
  Explicit Read-Write Functionality, Design-by-Contract, Automatic Range Checking
• Documentation
• Testing / Benchmarking
  Module-wise, Stand-alone, Fully-coupled
• Handy Tools:
  Totalview, valgrind, VC++ (or in general IDEs)
Documentation

• **Peer reviewed**
  - Synoptic Dynamics (Coumou et al, 2011)
  - 3-Layer Clouds (Eliseev et al, 2013)
  - Dynamical Core (in progress...)

• **Technical reports (Mathematical Model Description)**
  Aim: One for each *Compute* module

• **Doxygen**
  Generates html/LaTeX documentation directly from source-code
Class Hierarchy

Go to the graphical class hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Atm
- Atmosphere
- Cell
- Clouds
- Compute
  - Cloudiness
  - HeatTransfer
  - LargeScaleDynamics
  - LargeScaleWindComputations
  - PlanetaryBoundaryLayerPhysics
  - PlanWave
  - RadiativeTransfer
  - SurfaceFluxes
  - SynopticScaleDynamics
  - SynScale
  - WaterVaporTransfer

Private Attributes

- `const LargeScaleWind & wind` constant reference to read large scale wind field
- `const SynopticScaleField & syn` constant reference to read synoptic field
- `const Temperature & temp` constant reference to read large scale wind field
- `const Humidity & hum` constant reference to read large scale wind field
- `const SurfaceLayer & sf_layer` constant reference to read equilibrium specific humidity at the surface
- `Parameter< double > a1` reference to store cloud field
- `Parameter< double > a2` stratus cloud parameters
- `Parameter< double > a3` Parameter< double > a3
- `Parameter< double > a5` Parameter< double > a5
- `Parameter< double > ncm` Parameter< double > ncm
- `Parameter< double > b1` Parameter< double > b1
- `Parameter< double > b2` Parameter< double > b2

Call-graphs / Dependency-graphs

```
HeatTransfer::MoistAdiabaticLapseRate
  -> HorizontalVariable::Read
  -> SphericalGrid::SurfaceCellsBegin
  -> SphericalGrid::SurfaceCellsEnd
  -> HorizontalVariable::Write
```
Can get somewhat complex...
Content

• Intro Aeolus and Climber-4
• Key Modules
• Object-oriented Code Design
  Strict separation of Data Storage, Gridding and Computations
• Safety Mechanisms:
  Explicit Read-Write Functionality, Design-by-Contract, Automatic Range Checking
• Documentation
• Testing / Benchmarking
  Module-wise, Stand-alone, Fully-coupled
• Handy Tools:
  Totalview, valgrind, VC++ (or in general IDEs)
Tuning: Module-wise

Parameter Values

Compute Module

- Satellite data
- Reanalysis
- Direct observations
- GCM data

Output: Some Atmospheric Variables

Tune

Compare

Satellite data

Reanalysis
Tuning: Example Cloudiness

3-Layer Cloudiness

- Temperature, Humidity, Wind (Reanalysis)
- $\alpha, \beta, \gamma, \ldots$

Compare with:
- Satellite data
- Reanalysis
- Direct observations

3 Layer stratiform clouds
Convective clouds
Precipitation
Iterative optimization

For example using **SimENV**

Temperature, Humidity, Wind (Reanalysis)

3-Layer Cloudiness

- 3 Layer stratiform clouds
- Convective clouds
- Precipitation

α, β, γ, ...

Cost / Skill Function

Compare with
- satellite data
- reanalysis
- direct observations

Dim Coumou, Earth System Analysis
Tuning: Module Calibration

Improves low precipitation in major desert regions

Conclusion: automated calibration is very useful
...but only when a proper Skill function can be defined!

Eliseev et al, 2013
Tuning: Stand-alone *Aeolus*

**Next Step:** Sensitivity-analysis using SimENV

Hadley & Ferrel Cell strengthen and widen for stronger T-gradients

(Erik Peukert)
Tuning: Fully-Coupled  \((\text{MOM5 - LAD - Aeolus})\)

with Georg & Stefan:

- Technical coupling completed
- Latest version runs for \(~50\) years
- Model drift: small but non-zero cooling

![CLIMBER-4 Diagram]
Tools: Visual C++ (IDE)
Tools: Totalview (Debugger)