Design and Interfaces for CliMA's Next-Generation Performance-Portable Earth System Model

Valeria Barra, Ph.D.
Assistant Professor, San Diego State University
valeriabarra.org
Co-authors: Simon Byrne, Akshay Sridhar, Shriharsha Kandala, Lenka Novak, Julia Sloan, Dennis Yatunin, Charles Kawczynski, Gabriele Bozzola, Tapio Schneider (PI), and the entire CliMA team

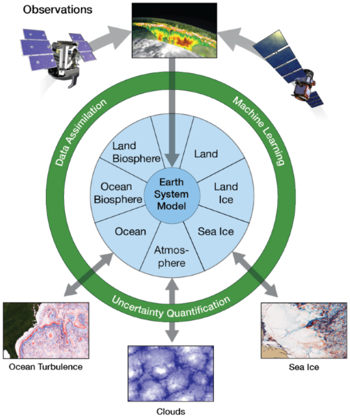
June 5th, 2024

Overview

About CliMA

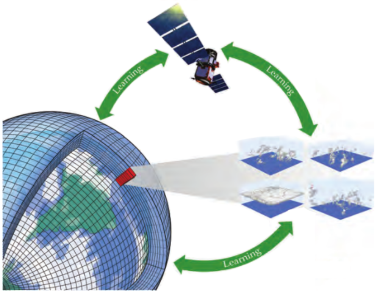
The Climate Modeling Alliance (CliMA) is a coalition of scientists, engineers, and applied mathematicians from **Caltech**, **MIT**, and the **NASA Jet Propulsion Laboratory**, building the first Earth System Model (ESM) in the Julia programming language that automatically learns from diverse data sources to produce more accurate climate predictions with quantified uncertainties.

Goals



[Source: courtesy of Tapio Schneider (Caltech)]

- The Earth System Model (ESM) will be grounded in physics (using sub-grid scale, cloud-resolving modeling) and designed for automated calibration of parameters using machine learning.
- High-resolution Large-Eddy Simulations (LES) are used to inform parametrizations of the global circulation model (GCM), which in turn, can be used for large-scale forcings to force the LES.



[Source: Physics Today - June 2021, pg. 44-51]

Technical and Scientific Aims

- Support CPUs and GPUs using a common open-source (single-source) code base written in the high-level, dynamic Julia programming language (familiar syntax, similar to Python and Matlab).
- Julia has an interactive REPL, is Just-In-Time (JIT) compiled (triggered by first evaluation of function). Allows polymorphism via multiple dispatch (at compile or run time).
- Can write generic code, compiler will specialize on types of calling arguments, e.g., `f(x::AbstractArray)` where `AbstractArray` can be `Array` of `Float32`, `Float64` or a `CuArray`.
- Be accessible and extensible by a mixture of users.

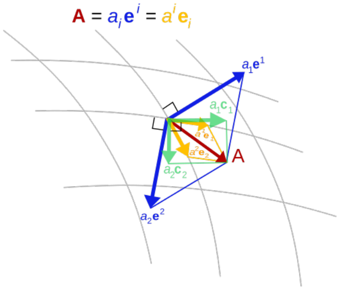
- For the atmosphere model, support both Large-Eddy Simulation (LES) and General Circulation Model (GCM) configurations (i.e., Cartesian and spherical geometries).
- Allow specification of any governing equations and boundary conditions by composing operators.
- Support uniform/non-uniform structured and unstructured meshes.

Why Julia?

- User-friendliness
- Package management
- Debuggability
- Performance portability
- Reproducibility!
- Use the Julia ecosystem:
  - Documentation: Documenter.jl
  - Unit testing: `julia> include("test/runtests.jl")`
  - Data structures and Optimizations
  - Profiling (e.g, `@time`)
  - Plotting
  - I/O (NetCDF, HDF5)
- Duality: can be used in instructional (Jupyter Notebooks) or operational settings
- Attracts young talent

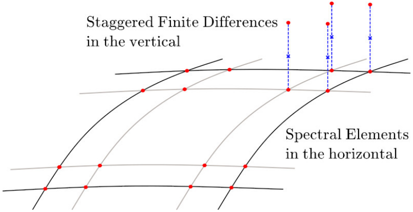
ClimaCore.jl

ClimaCore.jl — a new dynamical core (*dycore*).



A library (suite of tools) for constructing flexible space discretizations.

- Geometry:
    - Supports different geometries (Cartesian & spherical).
    - Supports covariant/contravariant vector representation for curvilinear, non-orthogonal systems and Cartesian vectors for Euclidean spaces.
- Space Discretizations:
    - Horizontal: Supports both Continuous Galerkin (CG) and Discontinuous Galerkin (DG).
    - Vertical: staggered or unstaggered Finite Differences (FD).



ClimaCore.jl: API

ClimaCore.jl's composable `Operators` and Julia broadcasting:

- Julia broadcasting:
    - apply a vectorized function point-wise to an array. Scalar values are "broadcast" over arrays; Fusion of multiple operations.
    - User-extensible API: can be specialized for custom functions or argument types (e.g., `CuArray` compiles and applies a custom CUDA kernel).
- `Operators` (`grad`, `div`, `interpolate`) are "pseudo-functions": act like functions when broadcasted over a `Field`, but can't be called on a single value; can be composed and fused w/ function calls. Matrix-free, i.e., no assembly; specify action of operator.

```
# apply f to each element of X
f.(X)

# fuse multiple operations
# and assign to existing array
# without intermediate temporaries
Y .= X0 .+ ϵ .* f.(X)

# expression internally calls
materialize!(Y,
  broadcasted(+, X0,
    broadcasted(*, ϵ,
      broadcasted(f, X))))
```

```
grad = Operators.Gradient()
wdiv = Operators.WeakDivergence()
diff = @. -wdiv(grad(u))
```

Performance/optimization

1. Julia Optimizations:
   - Disable bounds checking to facilitate vectorized (SIMD) instructions via `@inbounds`
   - Ensure type stability using tools, e.g., JET.jl that allows to do static type checking
   - Profiling to eliminate dynamic memory allocations
2. Fused CUDA kernels:
   - MultiBroadcastFusion.jl: allows users to fuse multiple broadcast expressions into a single CUDA kernel launch via the annotation `@fused_direct`
3. I/O:
   - NetCDF and parallel HDF5 support

Examples: Shallow-water equations

The shallow water equations
(in vector-invariant form):

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\boldsymbol{u}) = 0 \tag{1a}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla(\Phi + \tfrac{1}{2}\|\boldsymbol{u}\|^2) = (\boldsymbol{u} \times (f + \nabla \times \boldsymbol{u})) \tag{1b}$$

where f is the Coriolis term and $\Phi = g(h + h_s)$.

Written in terms of a curvilinear, non-orthogonal basis:

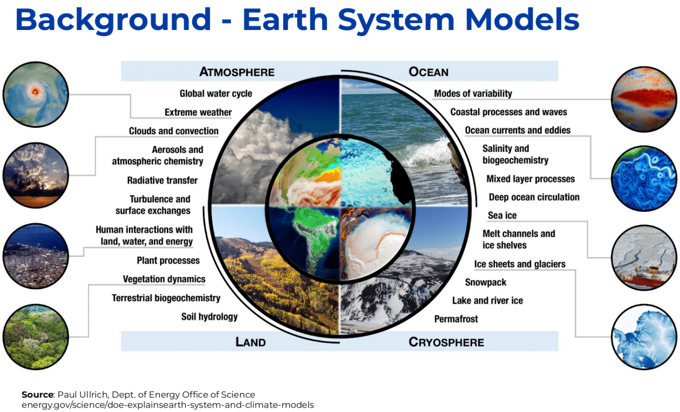
$$\frac{\partial h}{\partial t} + \frac{1}{J}\frac{\partial}{\partial \xi^j}\left(hJu^j\right) = 0 \tag{2a}$$

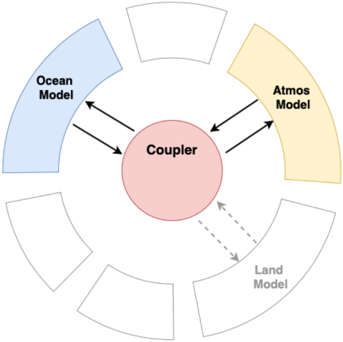
$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial \xi^i}(\Phi + \tfrac{1}{2}\|\boldsymbol{u}\|^2) = E_{ijk}u^j(f^k + \omega^k) \tag{2b}$$

```julia
∇ = Operators.Gradient()
∇⋅ = Operators.WeakDivergence()
curl = Operators.Curl()

@. begin
    dydt.h = - ∇⋅(y.h * y.u)
    dydt.u = -∇(g * (y.h + h_s) - norm(y.u)^2 / 2) + y.u × (f + curl(y.u))
end
```

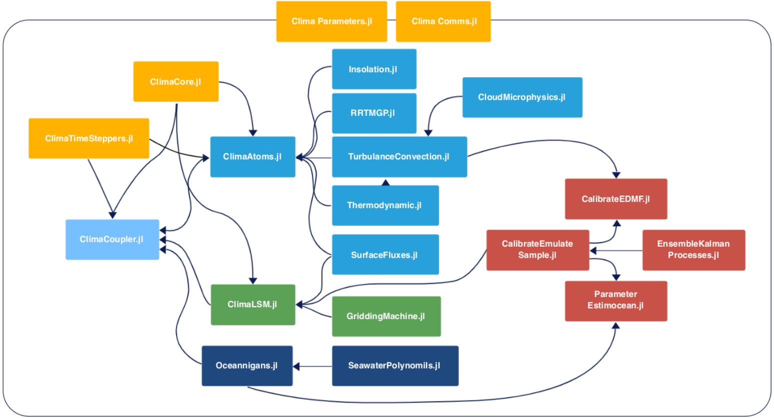
ESM





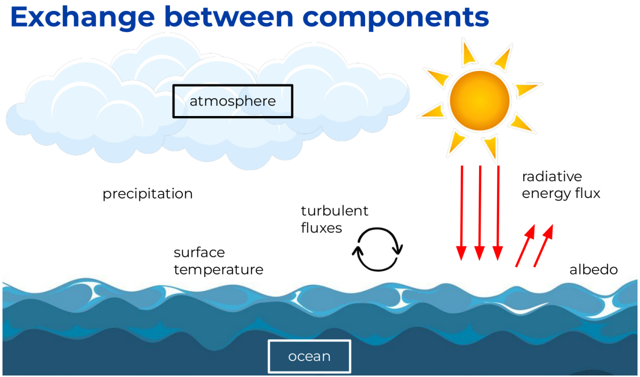
[Source: ClimaCoupler.jl docs]

ClimaCoupler.jl in the CliMA ESM ecosystem



[Source: courtesy of Lenka Novak (CliMA, Caltech)]

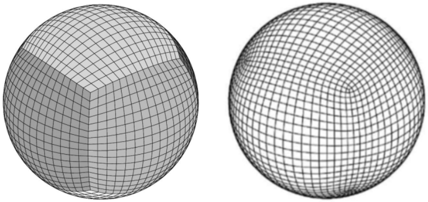
Role 1: Exchange quantities between components



[Source: courtesy of Julia Sloan (CliMA, Caltech)]

Role 2: Regridding/Remapping and Time-stepping

Cubed sphere types:

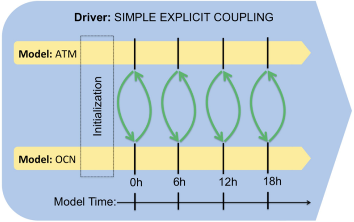


equiangular cubed sphere

Ullrich 2014

conformal cubed sphere

Rančić et al. 1996

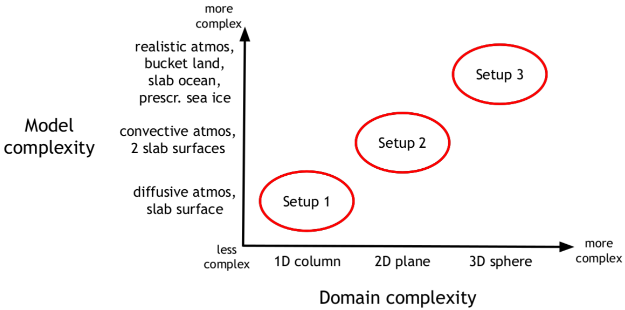


→ Sequential or concurrent

```
"""
    step_model_sims!(model_sims, t)

Iterates `step!` over all component model simulations saved in `cs.model_sims`.
"""
function step_model_sims!(model_sims, t)
    for sim in model_sims
        step!(sim, t)
    end
end
```

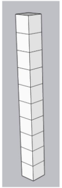
Role 3: Allowing different hierarchies



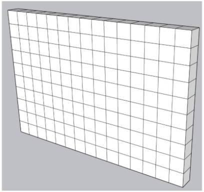
[Source: courtesy of Julia Sloan (CliMA, Caltech)]

Process-based Hierarchy: e.g., Geometry Hierarchies

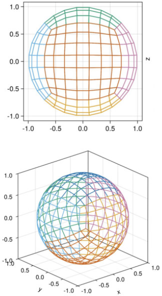
Domain visualizations



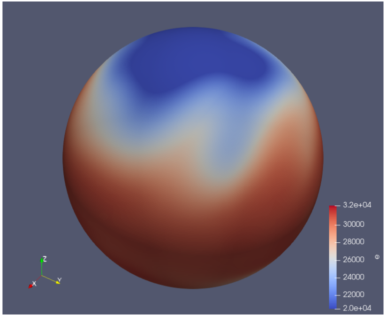
1D column



2D plane

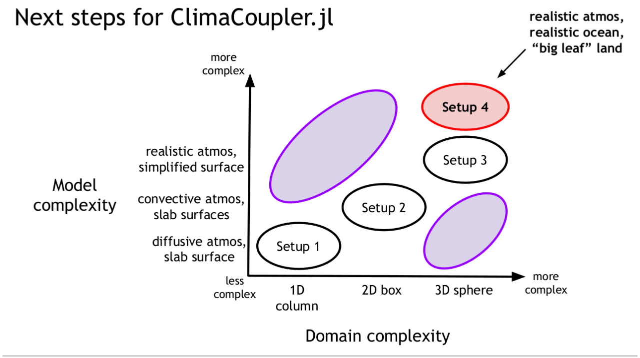


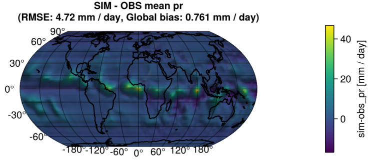
3D cubed sphere



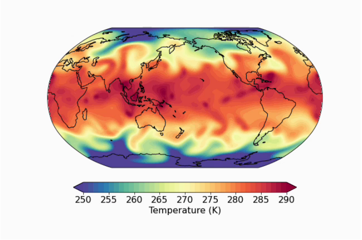
Held-Suarez 180-days simulation.

Generality-based Hierarchy: Model Hierarchies





AMIP (with diagnostic EDMF and topography) simulation—total bias.



Preliminary AMIP (w/o EDMF and topography) simulation—temperature.

Conclusions and Outlook

- We have introduced ClimaCore.jl (the dycore for atmos and land components) and ClimaCoupler.jl, part of the CliMA's ESM ecosystem
- We have showed their flexible and user-friendly APIs, allowing for high-performance composable solvers and different complexities/hierarchies
- Current preliminary performance:
  - For atmospheric component alone: 1 SYPD for `helem=30`, `Nq=4`, `zelem=63` (64 faces), 64 MPI processes, and 1 A100 GPUs
  - For Coupler setup: 1 SYPD for `helem=30`, `Nq=4`, `zelem=63` (64 faces), `coupled_dt=2min`, 64 MPI processeses, and 4 A100 GPUs
- Outlook:
  - Improve prognostic EDMF and atmospheric chemistry for more realistic atmosphere component
  - Include full 3D land component
  - Calibrate

Acknowledgements

Many thanks to all ClimaCore.jl's and ClimaCoupler.jl users and contributors.
In particular, for the latest CliMA dycore efforts a special mention goes to:

- Tapio Schneider[1] (PI), Paul Ullrich[2], Oswald Knoth[3], Simon Byrne[1], Jake Bolewski[1], Charles Kawczynski[1], Sriharsha Kandala[1], Zhaoyi Shen[1], Jia He[1], Kiran Pamnany[1], Ben Mackay[1], Akshay Sridhar[1], Dennis Yatunin[1], Lenka Novak[1], Toby Bischoff[1], Daniel (Zhengyu)Huang[1], Andre Souza[4], Yair Cohen[1]
  1: Caltech, 2: UC Davis, 3: TROPOS, 4: MIT

Our funders:

SCHMIDT FUTURES   NSF   DARPA   HEISING-SIMONS FOUNDATION   Audi Environmental Foundation   CHARLES TRIMBLE