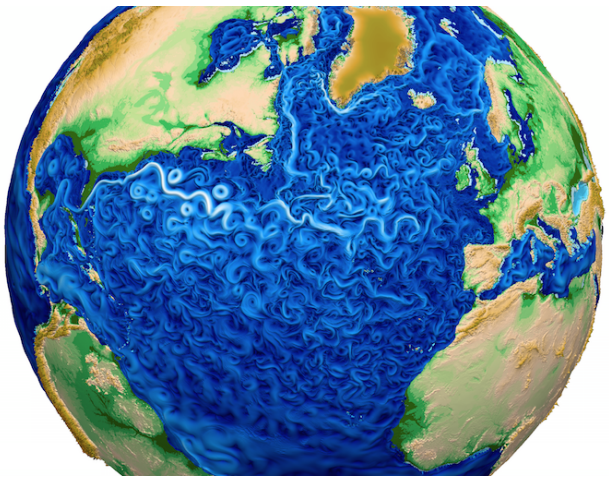
Let's make an Earth System Modeling Language!

July 2, 2021

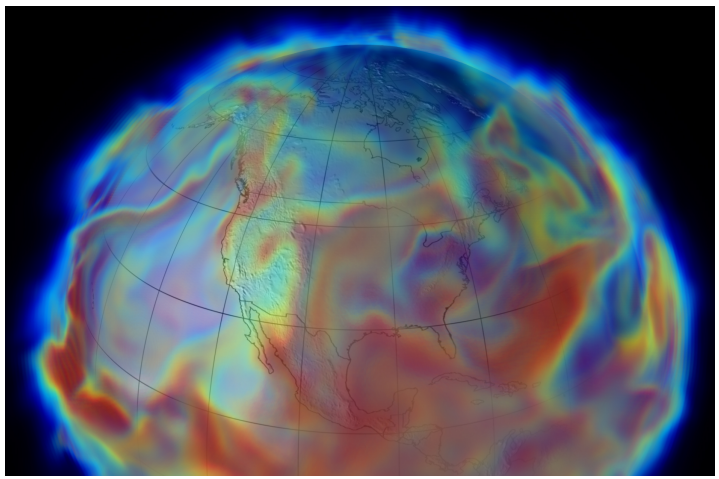
Earth System Modeling is...

Simulation from first-principles(-ish) of the entire Earth for use in e.g. climate predictions, including global circulation models for:

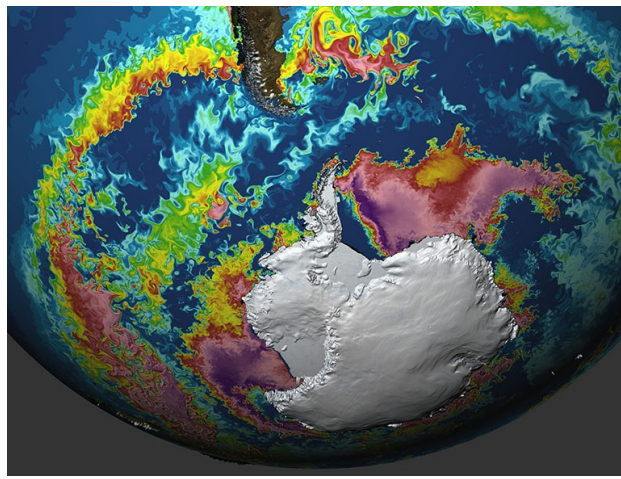
Ocean



Atmosphere



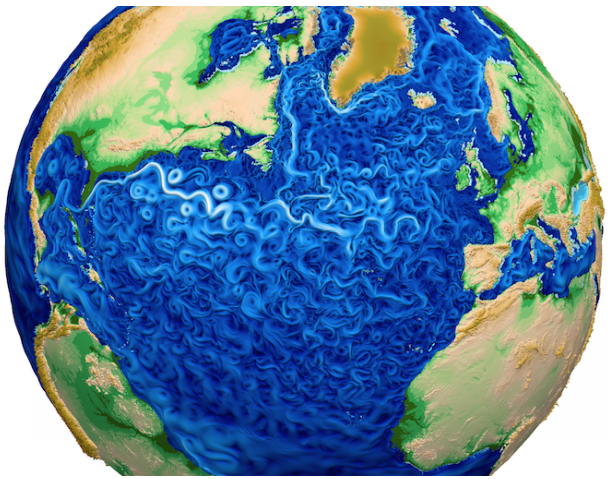
Biogeochemistry



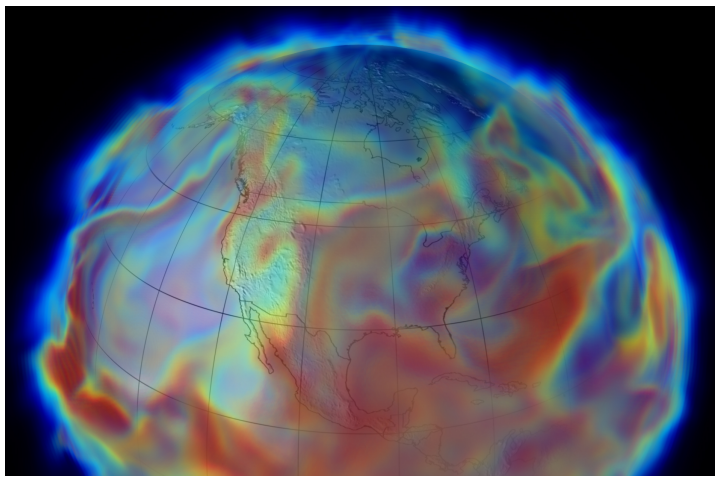
Earth System Modeling is...

Simulation from first-principles(-ish) of the entire Earth for use in e.g. climate predictions, including global circulation models for:

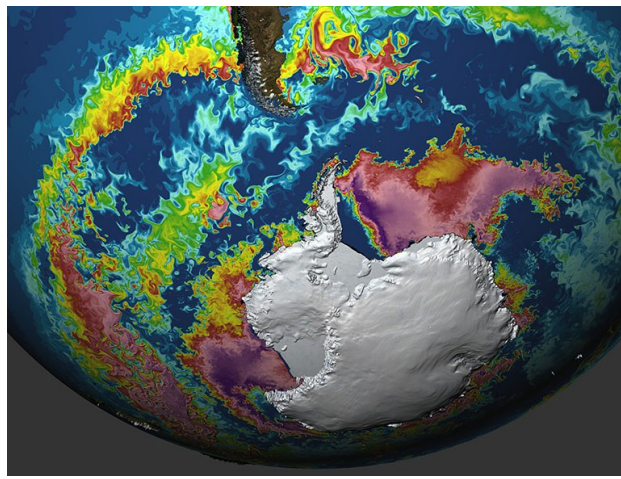
Ocean



Atmosphere



Biogeochemistry

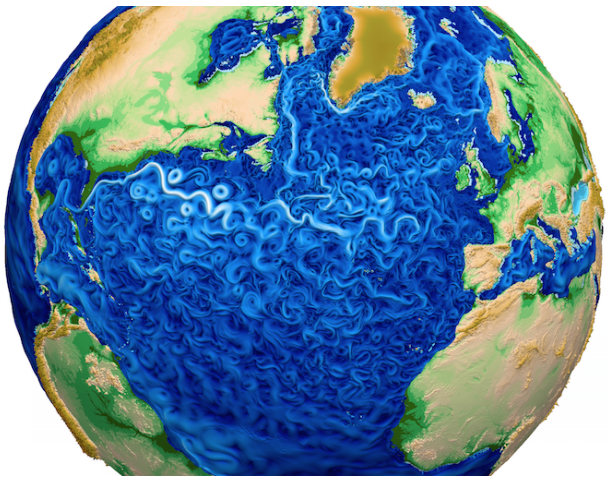


- Resolution: $\approx$10km $\times$ 10km grid cells: effect of Relevant sub-grid effects (down to cm-scale) through parameterized approximations.

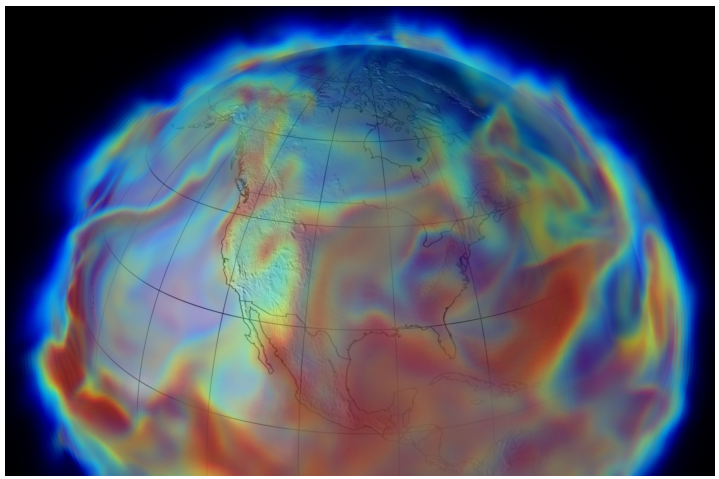
Earth System Modeling is...

Simulation from first-principles(-ish) of the entire Earth for use in e.g. climate predictions, including global circulation models for:

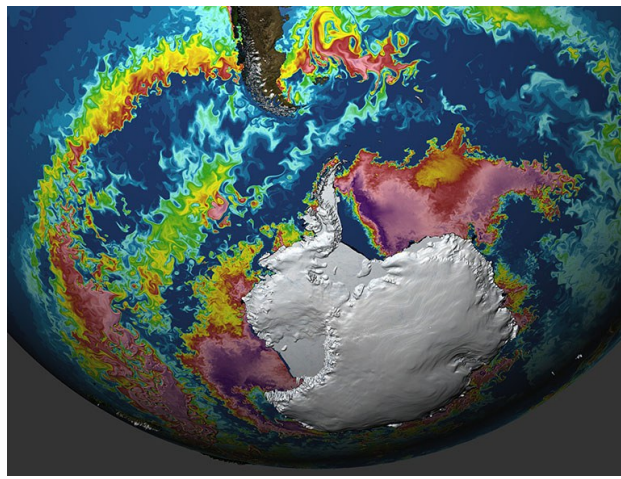
Ocean



Atmosphere



Biogeochemistry

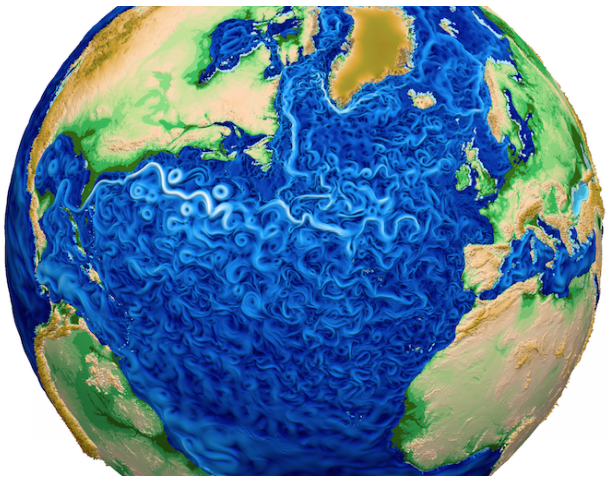


- Resolution: $\approx$10km $\times$ 10km grid cells: effect of Relevant sub-grid effects (down to cm-scale) through parameterized approximations.
- Time integration step size: hours, storing e.g. daily means.

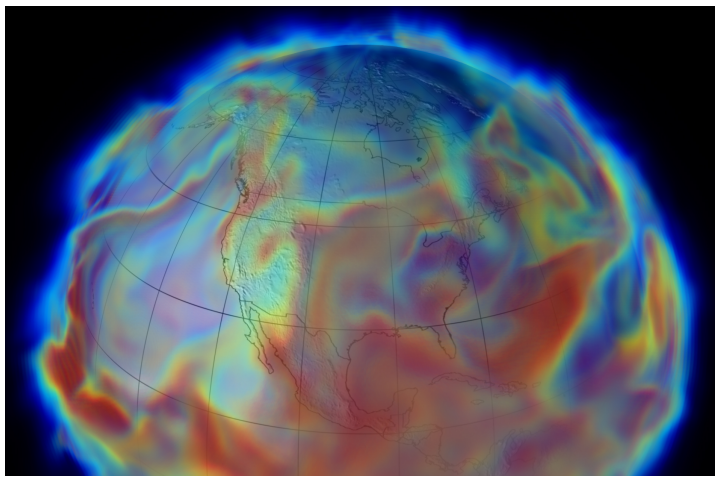
Earth System Modeling is...

Simulation from first-principles(-ish) of the entire Earth for use in e.g. climate predictions, including global circulation models for:

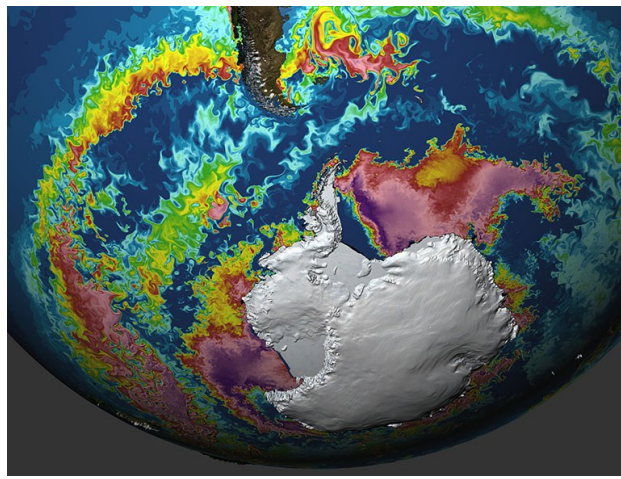
Ocean



Atmosphere



Biogeochemistry



- Resolution: $\approx$10km $\times$ 10km grid cells: effect of Relevant sub-grid effects (down to cm-scale) through parameterized approximations.
- Time integration step size: hours, storing e.g. daily means.
- Simulation run times: up to tens of thousands years for climate models to match paleoclimate.

How is it currently done?

Example: *Community Earth System Model* CESM

`https://www.cesm.ucar.edu/`

comprises about one million lines of code (mainly Fortran90).

How is it currently done?

Example: *Community Earth System Model* CESM

`https://www.cesm.ucar.edu/`

comprises about one million lines of code (mainly Fortran90).

- No one has full overview.

How is it currently done?

Example: *Community Earth System Model* CESM

`https://www.cesm.ucar.edu/`

comprises about one million lines of code (mainly Fortran90).

- No one has full overview.
- Extremely difficult to verify correctness.

How is it currently done?

Example: *Community Earth System Model* CESM

`https://www.cesm.ucar.edu/`

comprises about one million lines of code (mainly Fortran90).

- No one has full overview.
- Extremely difficult to verify correctness.
- Adding new physics to test a hypothesis takes a full PhD.

How is it currently done?

Example: *Community Earth System Model* CESM

`https://www.cesm.ucar.edu/`

comprises about one million lines of code (mainly Fortran90).

- No one has full overview.
- Extremely difficult to verify correctness.
- Adding new physics to test a hypothesis takes a full PhD.
- Changing numerics to better convergent or stable methods: no-go, would require total rewrite.

What's the Task? The equations of motion for the CESM/POP ocean model:

$$\frac{\partial}{\partial t}u + \mathcal{L}(u) - (uv\tan\phi)/a - fv = -\frac{1}{\rho_0 a\cos\phi}\frac{\partial p}{\partial\lambda} + \mathcal{F}_{Hx}(u,v) + \mathcal{F}_V(u) \quad (2.1)$$

$$\frac{\partial}{\partial t}v + \mathcal{L}(v) + (u^2\tan\phi)/a + fu = -\frac{1}{\rho_0 a}\frac{\partial p}{\partial\phi} + \mathcal{F}_{Hy}(u,v) + \mathcal{F}_V(v) \quad (2.2)$$

$$\mathcal{L}(\alpha) = \frac{1}{a\cos\phi}\left[\frac{\partial}{\partial\lambda}(u\alpha) + \frac{\partial}{\partial\phi}(\cos\phi v\alpha)\right] + \frac{\partial}{\partial z}(w\alpha) \quad (2.3)$$

$$\mathcal{F}_{Hx}(u,v) = A_M\left\{\nabla^2 u + u(1-\tan^2\phi)/a^2 - \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial v}{\partial\lambda}\right\} \quad (2.4)$$

$$\mathcal{F}_{Hy}(u,v) = A_M\left\{\nabla^2 v + v(1-\tan^2\phi)/a^2 + \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial u}{\partial\lambda}\right\} \quad (2.5)$$

$$\nabla^2\alpha = \frac{1}{a^2\cos^2\phi}\frac{\partial^2\alpha}{\partial\lambda^2} + \frac{1}{a^2\cos\phi}\frac{\partial}{\partial\phi}\left(\cos\phi\frac{\partial\alpha}{\partial\phi}\right) \quad (2.6)$$

$$\mathcal{F}_V(\alpha) = \frac{\partial}{\partial z}\mu\frac{\partial}{\partial z}\alpha \quad (2.7)$$

Parallel Ocean Program Reference Manual p.6+7. Next 6 chapters: Discretization.

What's the Task? The equations of motion for the CESM/POP ocean model:

Continuity equation:

$$\mathcal{L}(1) = 0 \tag{2.8}$$

Hydrostatic equation:

$$\frac{\partial p}{\partial z} = -\rho g \tag{2.9}$$

Equation of state:

$$\rho = \rho(\Theta, S, p) \rightarrow \rho(\Theta, S, z) \tag{2.10}$$

Tracer transport:

$$\frac{\partial}{\partial t}\varphi + \mathcal{L}(\varphi) = \mathcal{D}_H(\varphi) + \mathcal{D}_V(\varphi) \tag{2.11}$$

$$\mathcal{D}_H(\varphi) = A_H \nabla^2 \varphi \tag{2.12}$$

$$\mathcal{D}_V(\varphi) = \frac{\partial}{\partial z}\kappa\frac{\partial}{\partial z}\varphi, \tag{2.13}$$

Parallel Ocean Program Reference Manual p.6+7. Next 6 chapters: Discretization.

Example: advection.F90 in CESM ocean code

Everything is mixed together “to make it fast”:

Example: advection.F90 in CESM ocean code

Everything is mixed together "to make it fast":

- Defining differential equations / equations of motion

Example: advection.F90 in CESM ocean code

Everything is mixed together "to make it fast":

- Defining differential equations / equations of motion
- Boundary conditions

Example: advection.F90 in CESM ocean code

Everything is mixed together "to make it fast":

- Defining differential equations / equations of motion
- Boundary conditions
- Discretization of space and space-differentiation $\partial^n / \partial x_i^n$

Example: advection.F90 in CESM ocean code

Everything is mixed together "to make it fast":

- Defining differential equations / equations of motion
- Boundary conditions
- Discretization of space and space-differentiation $\partial^n / \partial x_i^n$
- Time evolution scheme (approximation to time-integration).

What do we want instead?

User Code

Model Equations

$$\frac{\partial}{\partial t}u + \mathcal{L}(u) - (uv\tan\phi)/a - fv = -\frac{1}{\rho_0 a\cos\phi}\frac{\partial p}{\partial\lambda} + \mathcal{F}_{Hx}(u,v) + \mathcal{F}_V(u) \quad (2.1)$$

$$\frac{\partial}{\partial t}v + \mathcal{L}(v) + (u^2\tan\phi)/a + fu = -\frac{1}{\rho_0 a}\frac{\partial p}{\partial\phi} + \mathcal{F}_{Hy}(u,v) + \mathcal{F}_V(v) \quad (2.2)$$

$$\mathcal{L}(\alpha) = \frac{1}{a\cos\phi}\left[\frac{\partial}{\partial\lambda}(u\alpha) + \frac{\partial}{\partial\phi}(\cos\phi v\alpha)\right] + \frac{\partial}{\partial z}(w\alpha) \quad (2.3)$$

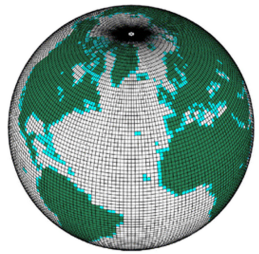
$$\mathcal{F}_{Hx}(u,v) = A_M\left\{\nabla^2 u + u(1-\tan^2\phi)/a^2 - \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial v}{\partial\lambda}\right\} \quad (2.4)$$

$$\mathcal{F}_{Hy}(u,v) = A_M\left\{\nabla^2 v + v(1-\tan^2\phi)/a^2 + \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial u}{\partial\lambda}\right\} \quad (2.5)$$

$$\nabla^2\alpha = \frac{1}{a^2\cos^2\phi}\frac{\partial^2\alpha}{\partial\lambda^2} + \frac{1}{a^2\cos\phi}\frac{\partial}{\partial\phi}\left(\cos\phi\frac{\partial\alpha}{\partial\phi}\right) \quad (2.6)$$

$$\mathcal{F}_V(\alpha) = \frac{\partial}{\partial z}\mu\frac{\partial}{\partial z}\alpha \quad (2.7)$$

Domain



Boundary Conditions

Discretization of $\frac{\partial^k}{\partial x_j^k}$

(Autogenerate from stencils)

Timestep-integration

(Autogenerate from Butcher)

Simulation setup

What do we want instead?

User Code

Model Equations

$$\frac{\partial}{\partial t}u + \mathcal{L}(u) - (uv\tan\phi)/a - fv = -\frac{1}{\rho_0 a\cos\phi}\frac{\partial p}{\partial\lambda} + \mathcal{F}_{Hx}(u,v) + \mathcal{F}_V(u) \quad (2.1)$$

$$\frac{\partial}{\partial t}v + \mathcal{L}(v) + (u^2\tan\phi)/a + fu = -\frac{1}{\rho_0 a}\frac{\partial p}{\partial\phi} + \mathcal{F}_{Hy}(u,v) + \mathcal{F}_V(v) \quad (2.2)$$

$$\mathcal{L}(\alpha) = \frac{1}{a\cos\phi}\left[\frac{\partial}{\partial\lambda}(u\alpha) + \frac{\partial}{\partial\phi}(\cos\phi v\alpha)\right] + \frac{\partial}{\partial z}(w\alpha) \quad (2.3)$$

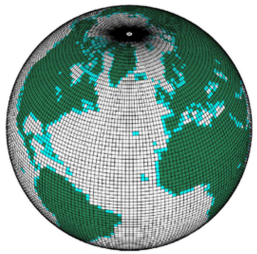
$$\mathcal{F}_{Hx}(u,v) = A_M\left\{\nabla^2 u + u(1-\tan^2\phi)/a^2 - \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial v}{\partial\lambda}\right\} \quad (2.4)$$

$$\mathcal{F}_{Hy}(u,v) = A_M\left\{\nabla^2 v + v(1-\tan^2\phi)/a^2 + \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial u}{\partial\lambda}\right\} \quad (2.5)$$

$$\nabla^2\alpha = \frac{1}{a^2\cos^2\phi}\frac{\partial^2\alpha}{\partial\lambda^2} + \frac{1}{a^2\cos\phi}\frac{\partial}{\partial\phi}\left(\cos\phi\frac{\partial\alpha}{\partial\phi}\right) \quad (2.6)$$

$$\mathcal{F}_V(\alpha) = \frac{\partial}{\partial z}\mu\frac{\partial}{\partial z}\alpha \quad (2.7)$$

Domain



Boundary Conditions

Discretization of $\frac{\partial^k}{\partial x_j^k}$

(Autogenerate from stencils)

Timestep-integration

(Autogenerate from Butcher)

Simulation setup

ESML Compiler

Static Model Checking:

Conservation laws,
Error bounds,
Numerical stability, etc.

PAT:

Ctrl&Data-flow analysis
Fusion, Streaming,
Temp elim., etc.

Code-generation:

Discretization code opt.
Mem accesses pattern opt.
Transfer latency hiding, etc.

What do we want instead?

User Code

Model Equations

$$\frac{\partial}{\partial t}u + \mathcal{L}(u) - (uv\tan\phi)/a - fv = -\frac{1}{\rho_0 a\cos\phi}\frac{\partial p}{\partial\lambda} + \mathcal{F}_{Hx}(u,v) + \mathcal{F}_V(u) \quad (2.1)$$

$$\frac{\partial}{\partial t}v + \mathcal{L}(v) + (u^2\tan\phi)/a + fu = -\frac{1}{\rho_0 a}\frac{\partial p}{\partial\phi} + \mathcal{F}_{Hy}(u,v) + \mathcal{F}_V(v) \quad (2.2)$$

$$\mathcal{L}(\alpha) = \frac{1}{a\cos\phi}\left[\frac{\partial}{\partial\lambda}(u\alpha) + \frac{\partial}{\partial\phi}(\cos\phi v\alpha)\right] + \frac{\partial}{\partial z}(w\alpha) \quad (2.3)$$

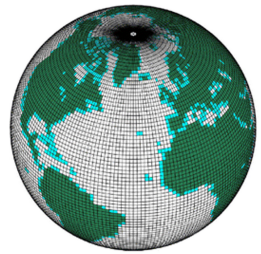
$$\mathcal{F}_{Hx}(u,v) = A_M\left\{\nabla^2 u + u(1-\tan^2\phi)/a^2 - \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial v}{\partial\lambda}\right\} \quad (2.4)$$

$$\mathcal{F}_{Hy}(u,v) = A_M\left\{\nabla^2 v + v(1-\tan^2\phi)/a^2 + \frac{2\sin\phi}{a^2\cos^2\phi}\frac{\partial u}{\partial\lambda}\right\} \quad (2.5)$$

$$\nabla^2\alpha = \frac{1}{a^2\cos^2\phi}\frac{\partial^2\alpha}{\partial\lambda^2} + \frac{1}{a^2\cos\phi}\frac{\partial}{\partial\phi}\left(\cos\phi\frac{\partial\alpha}{\partial\phi}\right) \quad (2.6)$$

$$\mathcal{F}_V(\alpha) = \frac{\partial}{\partial z}\mu\frac{\partial}{\partial z}\alpha \quad (2.7)$$

Domain



Boundary Conditions

Discretization of $\frac{\partial^k}{\partial x_j^k}$

(Autogenerate from stencils)

Timestep-integration

(Autogenerate from Butcher)

Simulation setup

ESML Compiler

Static Model Checking:
Conservation laws,
Error bounds,
Numerical stability, etc.

PAT:
Ctrl&Data-flow analysis
Fusion, Streaming,
Temp elim., etc.

Code-generation:
Discretization code opt.
Mem accesses pattern opt.
Transfer latency hiding, etc.

Efficient HPC Code

How Could an Ocean Model Look: Python

```
from ESML import *
from VerOS2.constants import
  mean_sea_radius as a,
  mean_water_density as
  rho0, vertical_mixing_mu as mu,
  coriolis_constant as Omega,
  ocean_viscosity as A_M,
  gravity_constant as g

# Set up spatial and field variables
lam,phi,z = SpatialCoordinates(3) # Longitude, latitude and depth
t         = TimeCoordinate(1)
u,v,w     = Field([lam,phi,z], 3) # Water density velocities along lam, phi, and z
p, rho    = Field([lam,phi,z])    # Pressure and water density

# Auxiliary definitions for defining equations
f = 2*Omega*Sin(phi)            # Coriolis parameter f
advection   = lambda alpha: 1/(a*Cos(phi)) * (D(u*u, lam) + D(Cos(phi)*v*alpha)) + D(w*u,z)        # Eq. (2.3)
laplacian   = lambda alpha: 1/((a*Cos(phi))**2) * D(alpha,lam,2) + 1/(a*a*Cos(phi))*D( Cos(phi)*D(alpha,phi), phi) # Eq. (2.6)

friction_Hx = A_M*( laplacian(u) + u*(1-Tan(phi)**2)/a**2 - 2*Sin(phi)/((a*Cos(phi)**2) * D(v,lam))) # Eq. (2.4)
friction_Hy = A_M*( laplacian(u) + v*(1-Tan(phi)**2)/a**2 - 2*Sin(phi)/((a*Cos(phi)**2) * D(u,lam))) # Eq. (2.5)
friction_V  = lambda alpha: D(mu*D(alpha, z), z)                                                    # Eq. (2.7)

# Fundamental equations of motion: Momentum equations for the Ocean
EOM1 = D(u,t) == -advection(u) + (u*v)*Tan(phi)/a + f*v - 1/(rho0 * a * Cos(phi))*D(p,lam) + friction_Hx + friction_V(u) # Eq. (2.1)
EOM2 = D(v,t) == -advection(v) + (u*u)*Tan(phi)/a + f*u - 1/(rho0 * a * Cos(phi))*D(p,lam) + friction_Hy + friction_V(v) # Eq. (2.2)

velocities_eq  = [u,v,w] == Gradient(rho)
hydrostatic_eq = D(p,z) == -rho*g
```

How Could an Ocean Model Look: C++

```
#include <ESML>
#include <Veros2/constants.hh>

using namespace ESML;
using namespace Veros2;

constexpr real_t
 a     = mean_sea_radius,
 rho0  = mean_water_density,
 mu    = vertical_mixing_mu,
 Omega = coriolis_constant,
 A_M   = ocean_viscosity,
  g    = gravity_constant;

// Set up spatial and field variables
ESML::Model M;
Variable lam, phi, z, t, u, v, w, p, rho;

M.setSpatialCoordinates({lam,phi,z});
M.setTimeCoordinate(t);
M.setFields({lam,phi,z}, {u,v,w});  // Water density velocities along lam, phi, and z
M.setFields({lam,phi,z}, {p,rho});  // Pressure and water density

// Auxiliary definitions for defining equations
f = 2*Omega*Sin(phi);            // Coriolis parameter f
auto advection = [&](Variable &alpha) { return 1/(a*Cos("phi")) * (D(u*u, lam) + D(Cos(phi)*v*alpha)) + D(w*u,z);  };                    // Eq. (2.3)
auto laplacian = [&](Variable &alpha) { return 1/((a*Cos("phi"))^2) * D(alpha,lam,2) + 1/(a*a*Cos(phi))*D( Cos(phi)*D(alpha,phi), phi); }; // Eq. (2.6)

Expression friction_Hx = A_M*( laplacian(u) + u*(1-Tan(phi)^2)/a^2 - 2*Sin(phi)/((a*Cos(phi)^2) * D(v,lam))); // Eq. (2.4)
Expression friction_Hy = A_M*( laplacian(u) + v*(1-Tan(phi)^2)/a^2 - 2*Sin(phi)/((a*Cos(phi)^2) * D(u,lam))); // Eq. (2.5)
auto friction_V =[&](Variable &alpha) { return D(mu*D(alpha, z), z); };                                          // Eq. (2.7)

// Fundamental equations of motion: Momentum equations for the Ocean
Equation EOM1 = D(u,t) == -advection(u) + (u*v)*Tan(phi)/a + f*v - 1/(rho0 * a * Cos(phi))*D(p,lam) + friction_Hx + friction_V(u); // Eq. (2.1)
Equation EOM2 = D(v,t) == -advection(v) + (u*u)*Tan(phi)/a + f*u - 1/(rho0 * a * Cos(phi))*D(p,lam) + friction_Hy + friction_V(v); // Eq. (2.2)

Equation velocities_eq  = [u,v,w] == Gradient(rho);
Equation hydrostatic_eq = D(p,z) == -rho*g;

M.setEquations({EOM1,EOM2,velocities_eq,hydrostatic_eq})
```

Spatial Discretization

Do we need to implement a library of the endless differential operator discretizations of various orders? No: All FD differential operators are just convolutions with different coefficients. Easy to generate fast code for any choice of stencil!

Derivative	Accuracy	−5	−4	−3	−2	−1	0	1	2	3	4	5
1	2					−1/2	0	1/2				
	4				1/12	−2/3	0	2/3	−1/12			
	6			−1/60	3/20	−3/4	0	3/4	−3/20	1/60		
	8		1/280	−4/105	1/5	−4/5	0	4/5	−1/5	4/105	−1/280	
2	2					1	−2	1				
	4				−1/12	4/3	−5/2	4/3	−1/12			
	6			1/90	−3/20	3/2	−49/18	3/2	−3/20	1/90		
	8		−1/560	8/315	−1/5	8/5	−205/72	8/5	−1/5	8/315	−1/560	

⋮

Linear combinations of differential operators ⇒ just sum into one stencil.

Unified Time Integration: Butcher Tableaus

Do we need to implement a library of the endless numerical integration / time-evolution methods? No! Only three: Explicit, Simple Implicit, Implicit Predictor/Corrector. The rest is data.

Specifically, Butcher-tableau data:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

$$k_i = f\left(t_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j\right)$$

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}
c_2		$\ddots$		
$\vdots$				
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss}
	b_1	b_2	$\cdots$	b_s

Unified Time Integration: Butcher Tableaus

Do we need to implement a library of the endless numerical integration / time-evolution methods? No! Only three: Explicit, Simple Implicit, Implicit Predictor/Corrector. The rest is data.

Specifically, Butcher-tableau data:

$$
\begin{aligned}
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i \\
y^*_{n+1} &= y_n + h \sum_{i=1}^{s} b^*_i k_i \\
e_{n+1} &= y_{n+1} - y^*_{n+1}
\end{aligned}
$$

$$
k_i = f\left(t_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j\right)
$$

$$
\begin{array}{c|ccc}
c_1 & a_{11} & \cdots & a_{1s} \\
c_2 & & \ddots & \\
\vdots & & & \\
c_s & a_{s1} & \cdots & a_{ss} \\
\hline
 & b_1 & \cdots & b_s \\
 & b^*_1 & \cdots & b^*_s
\end{array}
$$

Unified Time Integration: Butcher Tableaus

Explicit Euler

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Implicit Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Symplectic Midpoint

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

Crank-Nicolson

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

Runge-Kutta 4

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Runge-Kutta 3/8

$$\begin{array}{c|cccc} 0 & & & & \\ 1/3 & 1/3 & & & \\ 2/3 & -1/3 & 1 & & \\ 1 & 1 & -1 & 1 & \\ \hline & 1/8 & 3/8 & 3/8 & 1/8 \end{array}$$

Adaptive order-4 Gauss-Legendre

$$\begin{array}{c|cc} \frac{1}{2}-\frac{1}{6}\sqrt{3} & \frac{1}{4} & \frac{1}{4}-\frac{1}{6}\sqrt{3} \\ \frac{1}{2}+\frac{1}{6}\sqrt{3} & \frac{1}{4}+\frac{1}{6}\sqrt{3} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \\ & \frac{1}{2}+\frac{1}{2}\sqrt{3} & \frac{1}{2}-\frac{1}{2}\sqrt{3} \end{array}$$

. . . et cetera ad nauseum.

For another talk. . .

- Automatic error bounds
- Automatic numerical stability analysis
- Other static & runtime analysis opportunities
- How to autogenerate fast discretizations
- . . .