“Analyze This” – The Earth System Modeling Framework’s (ESMF) component model

Chris Hill, MIT cnh@plume.mit.edu

AD Workshop – Large Scale Apps, 2005, Shrivenham, UK
Talk Outline

• **Background**
  – AD in Earth system modeling
  – ESMF goals

• The ESMF Component Model
  – Component hierarchies
  – Dataflow between components
  – Controlling components

• Under the hood

• **AD on this sort of large-scale, multi-component application**
I) Model data synthesis

- **ECCO uses DA to estimate full depth ocean state on interannual, decadal time scales.**
  - DA is adjoint based 4dvar, employs automatic differentiation (AD) to generate adjoint
  - Cost function is 10 year misfit from a spectrum of ocean observations

\[
J = \frac{1}{2} \left[ (\bar{\eta} - \bar{\eta}_{lp})^{T} W_{\text{geoid}} (\bar{\eta} - \bar{\eta}_{lp}) + (\eta' - \eta'_{lp})^{T} W_{\eta_{lp}} (\eta' - \eta'_{lp}) + (\delta \eta_{x})^{T} W_{\eta_{x}} (\delta \eta_{x}) + \cdots \right]
\]

Used to make estimates of planetary scale phenomena on decadal time scales

McKinley Follows

Model CO2 Flux Estimates (mol m\(^{-2}\) y\(^{-1}\))

Stammer et al
Kohl et al
II) Sensitivity Analysis

Analyzing model air-sea gas exchange of a tracer subject to time-dependent transport.

Using the tracer equation

\[ \frac{\partial C}{\partial t} = -\mathbf{U} \cdot \nabla C + \nabla \cdot (K \nabla C) + f(C) - \mu C + S \]

we can derive global maps of mean residence time for a tracer injection in the ocean. Adjoint computational work is approx. 8000 years simulated time. Non-adjoint would require 108 million simulated years!
ESMF Goals

GOAL: Build a software superstructure and infrastructure that supports technical interoperability and sharing of code between Earth system modeling groups.

PRODUCTS:
- Coupling superstructure and utility infrastructure software

Interoperability

- Efficient
- Parallel
- Scalable code from different applications and institutions all running under ESMF

Achieving interoperability requires some commonality in programming models – in ESMF, it is the component model that provides commonality.
Talk Outline

• Background
  – AD in Earth system modeling
  – ESMF goals

• The ESMF Component Model
  – Component hierarchies
  – Dataflow between components
  – Controlling components

• Under the hood

• AD on this sort of large-scale, multi-component application
The ESMF Component Model

Provides machinery to code an ESMF application == a hierarchy of interacting
- coupler components, e.g. ocm2atm_coupler and
- gridded components, e.g. atm_phys_comp

Machinery includes
- general-purpose mechanisms for developers to code “wirings” between components (ESMF_State, regrid())
- mechanisms to create components and to control a components lifecycle (SetServices, Init(), Run(), Final() ...)

Example: hypothetical ESMF application

Graph of hierarchy in hypothetical ESMF application in left lower panel.
Numbers correspond to numbers on left

Components are nodes. Flow of data between components is shown by edges.
The ESMF Component Model. Wiring I: ESMF_State

- **ESMF_state** handles data flows **between** components.
- Component model defines a special data type, **ESMF_State** for these transfers.
- Components exchange data (with their parent in the graph) by encoding/decoding **ESMF_State** data type arguments.
- An **ESMF_State** holds different sorts of information, includes meta-data needed to decode it.

Typical sequence from highlighted fragment
1. atm_comp calls atm_phys_comp. Inputs for atm_phys_comp encoded in ESMF_State.
2. atm_phys_comp results encoded in another ESMF_State.
3. atm_comp calls phy2dyn_coupler using results in 2 as input.
4. phy2dyn_coupler returns its results, another ESMF_State, to atm_comp etc...
The ESMF Component Model Wiring II: regrid() and redist()
The ESMF Component Model.
Creating and controlling components

**Component model** defines a standard set of interfaces that a component must support.

Key ones `Initialize(IState,EState,Clock,RC)`, `Run(IState,EState,...)` `Finalize(IState,EState,...)`

Typical component life cycle is

1. Parent creates a component variable
   \[
   \text{child}=\text{GridCompCreate()}
   \]

2. Parent gets component to initialize its interface list
   \[
   \text{CALL GridCompSetServices(child,childSS)}
   \]

3. Parent initializes and then executes the component (usually repeatedly)
   \[
   \text{CALL GridCompInitialize(child,
   imp,exp,clk,rc)}
   \]
   \[
   \text{CALL GridCompRun(child,imp,exp,clk,rc)}
   \]

A component also carries private internal state pointers. These enable “child” to remember private state between calls.
The ESMF Component Model. Synopsis

- Data flows from component to component hierarchically.
- An ESMF application makes a tree is an acyclic graph.
- Data is transferred between components through ESMF State type “container variables”.
- Functions such as regrid/redist are used in coupler components to succinctly remap data within components.
- Every component registers Init(), Run(), Finalize() so that it can be controlled from a parent.
- Components may use an internal state pointer to maintain private state between calls.
- The component model provides everyone with a common model of time and alarms at the component interface.

These aspects of the ESMF component model are being used both to develop interoperable applications and to develop new applications. So far no AD has been applied.
Talk Outline

• ESMF Goals
• The ESMF Component Model
  – Component hierarchies
  – Dataflow between components
  – Controlling components
• Under the hood

• AD on this sort of large-scale, multi-component application
Visualizing the component architecture

1. ESMF provides an environment for assembling components

   Application driver
   
   Grided components
   Coupler components

2. ESMF provides a toolkit that components use to

   • ensure interoperability
   • abstract common services

   Component run(), checkpoint():
   Field: halo(), import(), export() + I/O
   Grid: regrid(), transpose() + Metrics
   DELayout, PEList, Machine model

3. Gridded components and coupler components are user written

   (c)
Creating a component

**Phase 1.** *Creation* creates components and registers user-code procedures for methods called in later phases. Create steps also are called for gridded component `COMPONENT1` and coupler component `COUPLER21` (not shown).

```fortran
COMPONENT1 = ESMF_GridCompCreate(“Example Component 1”, DELAYOUT_GC1)
CALL ESMF_GridCompSetServices(COMPONENT1, component1_register)
COUPLER12 = ESMF_CplCompCreate(“Example Coupler12”, DELAYOUT_CPL12)
```

Figure 4. Simplified Fortran-like pseudo code for two gridded components. `COMPONENT1` and `COMPONENT2` communicate with one another through two coupler components, `COUPLER12` and `COUPLER21`. 
Initializing a component

Calls one of the functions registered in the function pointer table. Function has standard “container” args IMPORT1, EXPORT1.

Phase 2. Initialization calls the user-code initialization procedures registered in phase 1. Initialize steps are also called for COMPONENT2, COUPLER12, COUPLER21 (not shown):

CALL ESMF_GridCompInitialize(COMponent1, IMPORT1, EXPORT1, CLOCK, RC)

Figure 4. Simplified Fortran-like pseudo code for two gridded components. COMPONENT1 and COMPONENT2 communicate with one another through two coupler components, COUPLER12 and COUPLER21.

IMPORT1, EXPORT1 are liked lists in which contain pointers to a number of ESMF types for holding actual data.
Phase 3. Run calls the user code’s run procedures, normally within one or more control loops (not shown). At each loop iteration, gridded component COMPONENT1 receives import and export ESMF state objects, IMPORT1 and EXPORT1. The run procedure of coupler component COUPLER12 maps between EXPORT1, the export ESMF state object of COMPONENT1, and IMPORT2, the import ESMF state object of gridded component COMPONENT2. The coupler component COUPLER21 acts in the opposite sense. It maps the COMPONENT2 export ESMF state, EXPORT2, onto IMPORT1, the import ESMF state object of COMPONENT1. The run procedures use values set (not shown) in the ESMF clock object CLOCK.

CALL ESMF_GridCompRun(COMPONENT1, IMPORT1, EXPORT1, CLOCK, RC)
CALL ESMF_CplCompRun(COUPLER12, EXPORT1, IMPORT2, CLOCK, RC)
CALL ESMF_GridCompRun(COMPONENT2, IMPORT2, EXPORT2, CLOCK, RC)
CALL ESMF_CplCompRun(COUPLER21, EXPORT2, IMPORT1, CLOCK, RC)
Talk Outline

• ESMF Goals

• The ESMF Component Model
  – Component hierarchies
  – Dataflow between components
  – Controlling components

• Under the hood

• AD challenges on this sort of large-scale, multi-component application
“Analyze this” – AD challenges for component arch.

From user perspective
Clean abstraction and modularity for person wiring together an application.

From AD perspective
Cross-component data flows and dependency are buried in specialized types.
Cross-component data flows are almost always linear steps (not dependent on local state) e.g. permute, average..

Q - What is the best approach to expose information to an AD tool?