The Programming Model for Concurrency in the Earth System Modeling Framework

SIMAC Workshop,
U of Chicago

Gerhard Theurich (NRL/ESMF/SAIC),
ESMF Core Team

December 10, 2012

http://www.earthsystemmodeling.org/
In climate research and numerical weather prediction...

... increased emphasis on detailed representation of individual physical processes; requires many teams of specialists to contribute components to an overall modeling system.

In computing technology...

... increase in hardware and software complexity in high-performance computing, as we shift toward the use of multi-paradigm parallel computing architectures.

In software...

... emergence of frameworks to promote code reuse and interoperability.

- ESMF is a focused community effort to tame the complexity of models and the computing environment.
- It leverages, unifies and extends existing software frameworks, creating new opportunities for scientific contribution and collaboration.
Phase 1: 2002-2005

NASA’s Earth Science Technology Office ran a solicitation to develop an Earth System Modeling Framework (ESMF).

A multi-agency collaboration (NASA/NSF/DOE/NOAA) won the award. The core development team was located at NCAR.

A prototype ESMF software package (version 2r) demonstrated feasibility.

Phase 2: 2005-2010

New sponsors included Department of Defense and NOAA.

Many new applications and requirements were brought into the project, motivating a complete redesign of framework data structures (version 3r).

Phase 3: 2010-2015

The core development team moved to NOAA/CIRES for closer alignment with federal models.

Basic framework development has been completed with version 5r (ports, bugs, feature requests, user support etc. still require resources).

Extensions continue: regridding, interoperability, I/O, and language bindings.

The focus is on increasing adoption and creating a community of interoperable codes.
Architecture – Overview I

- Base library of about 500,000 lines of source code.
- 60% in C/C++, 40% in Fortran (prior to autogen.)

- Complete Fortran user API
  - use ESMF
  - Derived types and methods

- Limited C user API
  - #include "ESMC.h"
  - Structs and methods

- Emerging Python user API
  - import ESMP
  - Classes with methods
Architecture – Overview II

- Unix/Linux and Windows (Cygwin/MinGW)
- Based on MPI (bypass mode “mpiuni” as option)
- OpenMP and Pthreads support
- OpenACC support
- I/O through NetCDF/HDF, Xerces, or PIO
- Sockets for web services and fault-tolerance extensions
- Highly portable: tested on > 40 different OS/Compiler/MPI combinations every night.
- Over 170,000 lines of example, unit and system testing code.
Search for Concurrency

• Domain specific (Earth System Modeling) parallel data structures:
  - Components: task-parallelism
  - Arrays/Fields: data-parallelism

• User directed parallelism:
  - OpenMP
  - OpenACC
Components in Earth System Modeling

ATM

OCN

LND

Sea-Ice

CPL

Exchange of state and flux variables

...
Concurrency between Components

- Technical requirement: separate sets of threads
- Scientific limitations: e.g. ATM ↔ OCN ≈ 12 sim.h

```
do
  call ESMF_GridCompRun(atm)
  call ESMF_GridCompRun(ocn)
  call ESMF_CplCompRun(cpl)
enddo
```
The PET Abstraction

- PET stands for “Persistent Execution Thread”
- Persists at least as long as the objects it operates on (Component scope)
- PETs can be implemented as *MPI tasks* or *Pthreads* – looks the same to the ESMF user
- PETs are used to manage processing elements (PEs) and to allow concurrent execution of Components:

  ```python
  atm = ESMF_GridCompCreate(petList=(/0..63/))
  ocn = ESMF_GridCompCreate(petList=(/64..127/))
  cpl = ESMF_CplCompCreate(petList=(/0..127/))
  ```
Concurrency within Components

- Domain decomposition: data parallelism
- Distributed data structures: Array/Field
- High level code or algorithms are written exclusively using the Array/Field abstraction, *not* directly using lower layer APIs (e.g. MPI):

  ```
  call ESMF_FieldRegrid(field)
  call ESMF_FieldGather(field)
  call ESMF_FieldSMM(field)
  call ESMF_FieldHalo(field)
  call ESMF_ArrayRedist(array)
  call ESMF_ArrayScatter(array)
  ```
Concurrency between communication-computation

call ESMF_FieldHalo(fieldB, & NBSTART, routehandle)

do

call ESMF_FieldHalo(fieldA, & NBSTART, routehandle)
call ESMF_FieldHalo(fieldB, & NBWAITFINISH, routehandle)

... work on fieldB ...
call ESMF_FieldHalo(fieldB, & NBSTART, routehandle)
call ESMF_FieldHalo(fieldA, & NBWAITFINISH, routehandle)

... work on fieldA ...
enddo
Mapping Distributed Data against OS and HW

Array/DistGrid

<table>
<thead>
<tr>
<th>A11</th>
<th>A12</th>
<th>A13</th>
</tr>
</thead>
<tbody>
<tr>
<td>A21</td>
<td>A22</td>
<td>A23</td>
</tr>
<tr>
<td>A31</td>
<td>A32</td>
<td>A33</td>
</tr>
<tr>
<td>A41</td>
<td>A42</td>
<td>A43</td>
</tr>
<tr>
<td>A51</td>
<td>A52</td>
<td>A53</td>
</tr>
<tr>
<td>A61</td>
<td>A62</td>
<td>A63</td>
</tr>
</tbody>
</table>

DELayout

PET 0 → PET 1 → PET 2

OS:
- processes + threads
  → persistent execution threads (PETs)
  → virtual address spaces (VASs)

Hardware:
- Proc. Elements (PEs)
- Compute units
- Single System Images
Components with Threads
Work Queue Balancing

- Some PETs implemented as Pthreads
- Threading level set by user
- Threading level set during Component creation

```
do i=1, localDeCount
    workDe = localDeToDeMap(i)
    reply = ESMF_DELayoutServiceOffer(delayout, de=workDe)
    if (reply == ACCEPT) then
        ... do work for workDe ... 
        call ESMF_DELayoutServiceComplete(delayout, de=workDe)
    endif
enddo
```
User Level Threading: OpenMP

- PETs associated with multiple PEs
- PET to PE mapping set by user
- PET to PE mapping set during Component creation

```
call ESMF_VMGet(peCount)
!$call omp_set_num_threads(peCount)
!$omp parallel do &
!$omp& shared (farrayPtr)

  do j = lbound(farrayPtr, 2), ubound(farrayPtr, 2)
    do i = lbound(farrayPtr, 1), ubound(farrayPtr, 1)
      ... work on farrayPtr ...
    enddo
  enddo
enddo
```
User Level GPU Code: OpenACC

- PETs placed to have access to node's GPU
- PET placement set by user
- PET placement set during Component creation

```plaintext
jl = lbound(farrayPtr, 2)
ju = ubound(farrayPtr, 2)
il = lbound(farrayPtr, 1)
iu = ubound(farrayPtr, 1)

!$acc kernels

    do j = jl, ju
        do i = il, iu
            ... work on farrayPtr ...
        enddo
    enddo

!$acc end kernels
```
Conclusions / Challenges

- ESMF is a domain-specific (Earth System Modeling) technical infrastructure software
- Concurrency based on componentization, domain decomposition and user directed fine grained parallelism
- Componentization and domain decomposition through a standard API
- High level code is shielded from some of the low level details of the compute architecture
- Fine-grained parallelism not efficiently abstracted (kernel compiler support?)
- Mitigation of data movement cost for GPU use (keep track of data location in ESMF Array/Field)
THANK YOU!