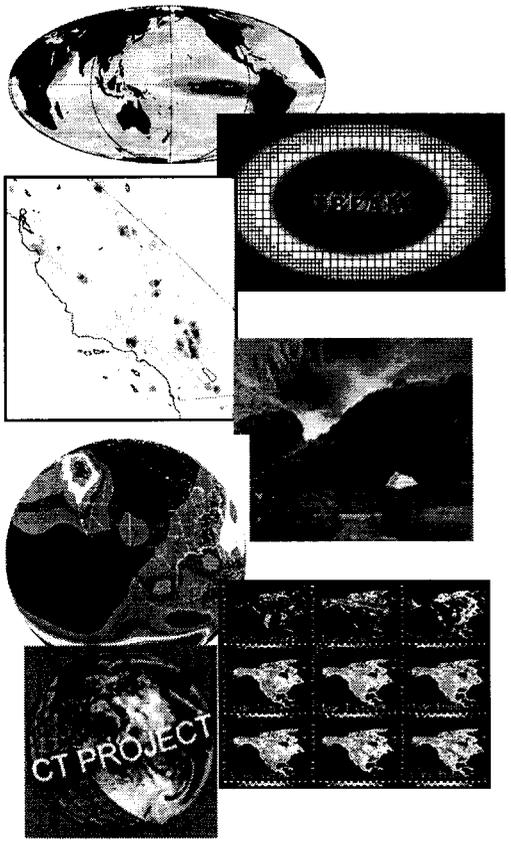


# Common Component Architecture (CCA) Experiences and Measurements

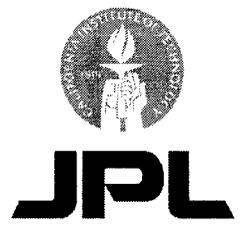


Daniel S. Katz, E. Robert  
Tisdale, Charles D. Norton,  
Craig Miller

Jet Propulsion Laboratory  
California Institute of Technology

[Daniel.S.Katz@jpl.nasa.gov](mailto:Daniel.S.Katz@jpl.nasa.gov)

Parallel Applications Technologies Group  
<http://pat.jpl.nasa.gov/>





# Topics

---

- ESTO CT
- CCA Demonstration Task  
(Completed Sept. 2002)
- Current CCA-Climate Work



# NASA Earth Science Technology Office (ESTO) Computational Technologies (CT) Project



**Jim Fischer/GSFC, Project Manager**

**Robert Ferraro/JPL, Associate Project Manager**

## **CT History:**

In a previous life, CT was known as the Earth and Space Science (ESS) Project within the High Performance Computing and Communication (HPCC) Program (from 1992-2000)

HPCC was a multi-agency federal program

Within NASA, ESS was funded by Codes S and Y, but run by code R

Eight Round 1 Science Teams (Grand Challenge Applications) demonstrated that NASA/university Earth and space science codes could work on parallel computers

Nine Round 2 Science Teams demonstrated that NASA/university Earth and space science codes could achieve good parallel performance on high-end parallel computers (up to 630 GFlops)



# NASA Earth Science Technology Office (ESTO) Computational Technologies (CT) Project



**Jim Fischer/GSFC, Project Manager**

**Robert Ferraro/JPL, Associate Project Manager**

## **CT Today:**

Eleven Round 3 Science Teams developing community frameworks to enable more realistic simulations of natural phenomena and interpretation of vast quantities of observational data on high-end computers.

In-house scientists to support the Science Teams

CT is funded and administered by Code Y

## **CT Project Goal:**

Demonstrate the power of high-end and scalable cost-effective computing environments to further our understanding and ability to predict the dynamic interaction of physical, chemical, and biological processes affecting the Earth, the solar-terrestrial environment, and the universe.



# CT Round-3 Awardees

*Managed via 144 negotiated milestones*



CAN-00-OES-01

"Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life and Microgravity Sciences"



## for the Earth System Modeling Framework

*\$9.8M over 3 years*

T. Killeen/NCAR

*Part I: Core Earth System Modeling Framework Development*

J. Marshall/MIT

*Part II: Modeling Applications for the Earth System Modeling Framework*

A. da Silva/GSFC

*Part III: Data Assimilation Applications for the Earth System Modeling Framework*

## in Earth Science

*\$6M over 3 years*

A. Donnellan/JPL

*Numerical Simulations for Active Tectonic Processes*

P. Houser/GSFC

*Land Information Systems*

C.R. Mechoso/UCLA

*Atmosphere-Ocean Dynamics and Tracer Transport*

J. Schnase/GSFC

*Biotic Prediction: HPCC Infrastructure for Public Health and Environmental Forecasting*

## in Space Science

*\$7M over 3 years*

T. Gombosi/U.Mich

*A High-Performance Adaptive Simulation Framework for Space-Weather Modeling (SWMF)*

P. Saylor/U.Illinois

*Development of an Interoperability Based Environment for Adaptive Meshes (IBEAM) with Applications to Radiation-Hydrodynamic Models of Gamma-Ray Bursts*

T. Prince/Caltech

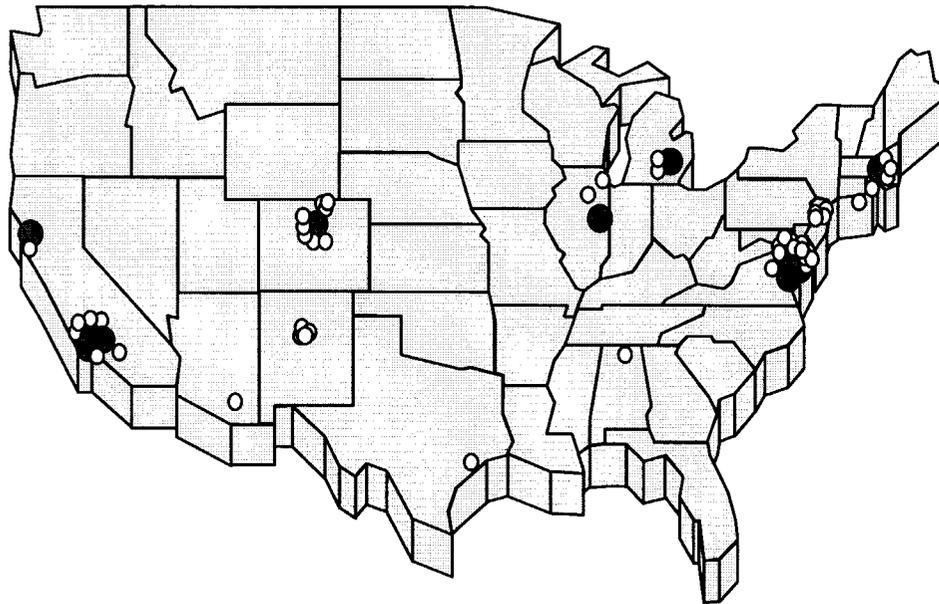
*High-Performance Cornerstone Technologies for the National Virtual Observatory*

P. Colella/DoE/LLNL

*A C++ Framework for Block-Structured Adaptive Mesh Refinement Methods*



# Round-3 P-I and Co-I Institutions



- **Principal Investigator**
  - **Co-Investigator**
- Dots represent institutions - multiple teams are at many institutions.





# Center-based Direct Support for Round-3 Teams

*Provided by Goddard and JPL*



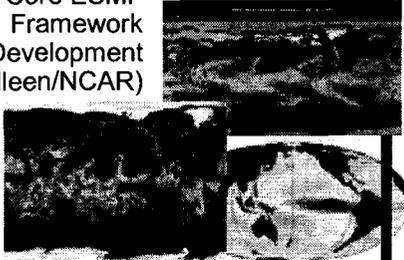
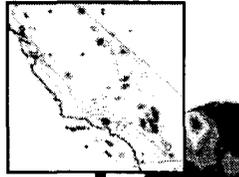
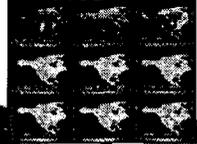
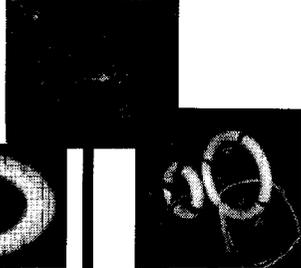
- Software engineering guidance
- ESMF risk assessment and mitigation
- Unique software product co-development
- Performance optimization
- High performance computing system access and data storage
- PC cluster access
- PC cluster pathfinding
- High end networking assistance
- Visualization services
- Summer school in high performance computational sciences
- Information officer



# Report Card on CT Round-3 Investigator Milestones

\$22.8M over 3 years

**Collaborations to develop software frameworks that enable more realistic simulations of natural phenomena and interpretation of vast quantities of observational data on high-end computers.**

<p>ESMF Data Assimilation Applications (A.daSilva/GSFC)</p> <p>Core ESMF Framework Development (T.Killeen/NCAR)</p>	<p>Simulation of Active Tectonic Processes (A.Donnellan/JPL)</p>	<p>Land Information Systems (P.Houser/GSFC)</p>	<p>Framework for Space-Weather Modeling (T.Gombosi/U.Michigan)</p>	<p>National Virtual Observatory Technology (T.Prince/Caltech)</p>
				
<p>ESMF Modeling Applications (J.Marshall/MIT)</p>	<p>Atmosphere-Ocean Dynamics and Tracer Transport (R.Mechoso/UCLA)</p>	<p>Biotic Prediction (J.Schnase/GSFC)</p>	<p>Radiation-Hydro Models of Gamma-Ray Bursts (P.Saylor/U.Illinois Urbana-Champaign)</p>	<p>C++ Framework for Block-Structured Adaptive Mesh Refinement Methods (P.Colella/LBNL)</p>

Signed Agreement																			
A -Software Engineering Plan																			
E -Code baseline																			submitted
F -1st Code Improvement																			submitted
G -2nd Code Improvement																			
H -Community agreement on Interoperability design policy																			
I -Interoperability Prototype Tested																			
J -Full Interoperability Implemented																			
K -Customer Delivery																			
B -1st Annual Report																			
C -2nd Annual Report																			
D -Final Report																			

(as of 11 April 2003)



# Earth System Modeling Framework Development

**JPL**



## Description

- Design and implement a software framework to allow climate model components from different researchers to interoperate on parallel computers

## PIs

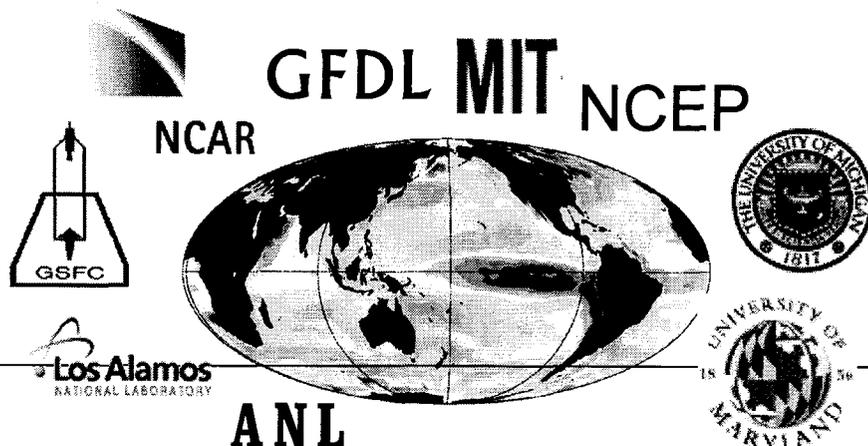
- Tim Killeen / NCAR
- John Marshall / MIT
- Arlindo da Silva / GSFC

## Approach

- Gather and analyze requirements for model interoperability from the U.S. climate and weather modeling communities, as well as for data assimilation systems used with major weather and climate models
- Design the required software framework and toolkit
- Engineer, test and validate the framework against the requirements
- Convert existing model and data assimilation components to comply with and use the framework
- Test scalability and overhead of the resulting system using real modeling applications

## Management

- 11 shared milestones valued at \$9.8M over 3 years



## The ESMF Will Assist the U.S. Climate Research Community to Advance by:

- Reducing redundant effort by scientists and software developers
- Strengthening communication and collaboration among diverse groups
- Strengthening the links between weather forecasting and climate modeling
- Increasing the portability and scalability of climate models
- Simplifying the construction of climate models and the exchange and incorporation of new submodels

<http://www.esmf.ucar.edu/>



# Earth System Modeling Framework Rollout



## 19 major Earth system modeling components

All compliant by April '04

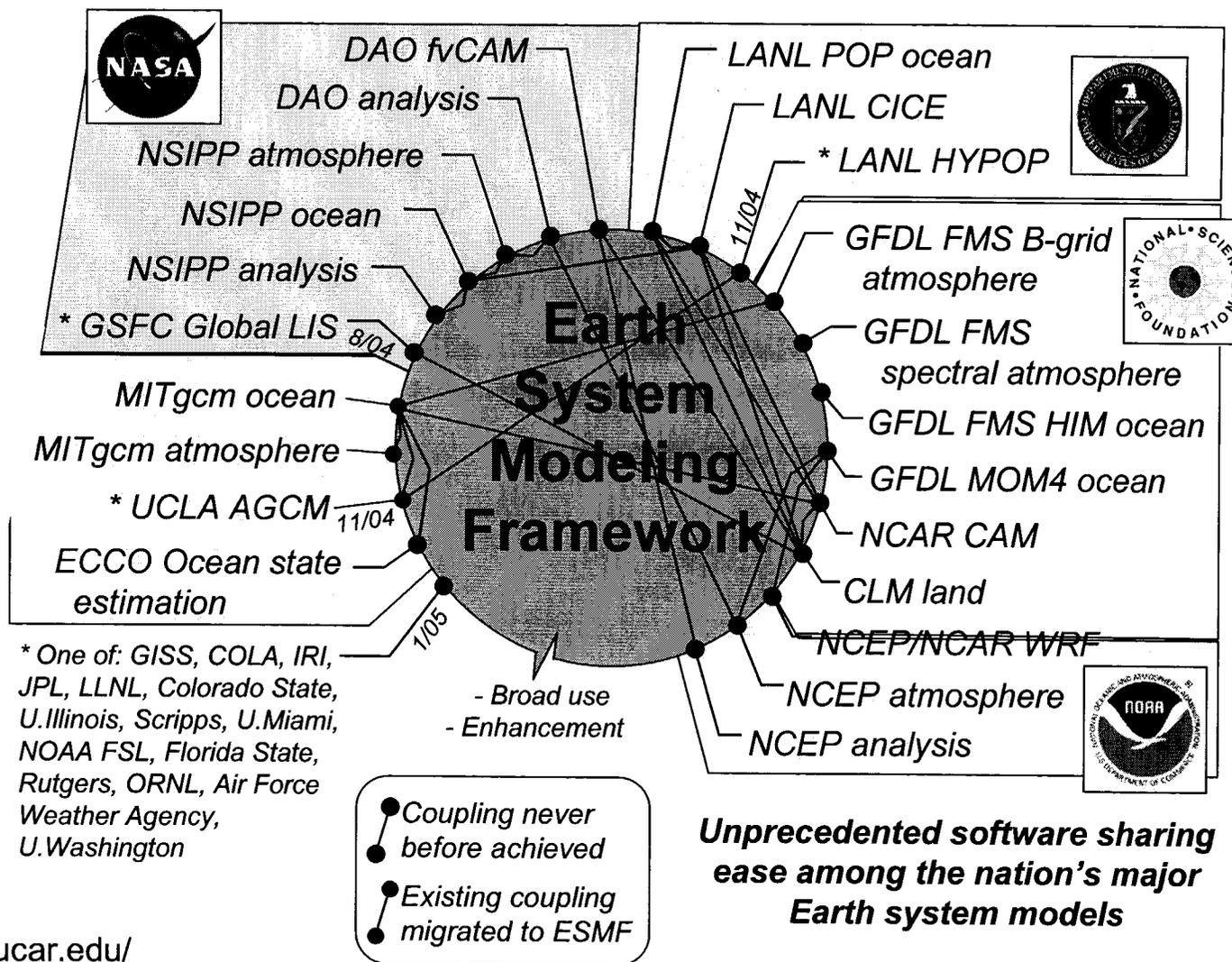
## 30 ESMF applications

15 research and operational

8 entirely new

7 synthetic samples

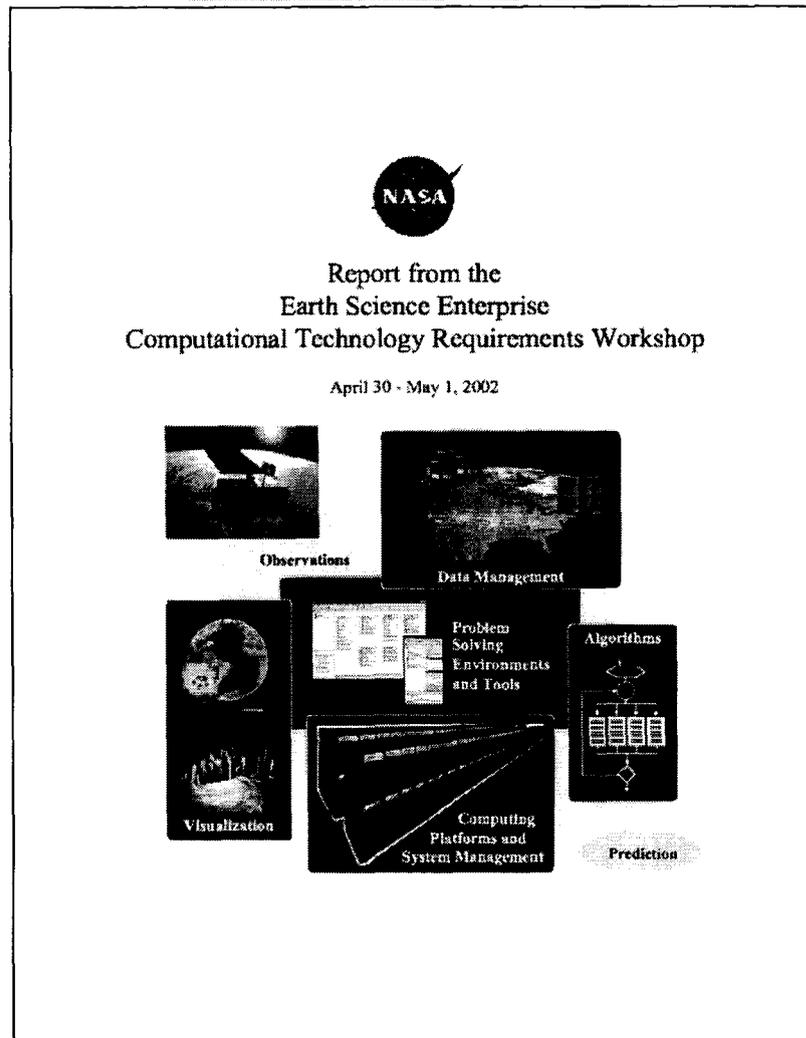
\* Early adopters of the ESMF



<http://www.esmf.ucar.edu/>



# Future Requirements



- Weather, Climate, and Solid Earth panels defined capabilities needed to achieve NASA prediction goals in 2010
- These capabilities were analyzed for stressing technology requirements
- Technology Cross-Cut of Gaps Identified
  - Computing Platforms
  - Data Management
  - Programming Environment and Tools
  - Distributed Computing
  - Other Requirements

Full report and presentation slides at:  
<http://esto.nasa.gov/programs/ct/ese-ct-results.html>



**JPL**



# Topics

---

- ESTO CT
- CCA Demonstration Task  
(Completed Sept. 2002)
- Current CCA-Climate Work



# Motivation for JPL's CCA Task

- ESTO-CT is a joint JPL-Goddard Space Flight Center (GSFC) Project
- Project includes in-house scientists at JPL and GSFC who:
  - Help the teams meet their milestones
    - **Advice/Consulting/Code Optimization**
  - Support the project and the teams by developing tools and libraries
- Some of the Round 3 science teams had mentioned CCA in their proposals as a mechanism useful for developing their applications by:
  - Enabling small groups to write parts of a larger application without understanding the full application's code
  - Allowing the scientists to concentrate on science while still working towards modern, reusable applications
  - Taking advantage of previously developed code
  - Supporting language interoperability
- This led the project to start a task to investigate the CCA
- The investigation task recommended a demonstration task



# What is a component architecture (CA)?

---

- A set of standards that allows:
  - Multiple groups to write units of software (components)
  - The groups to be sure that their components will work with other components written in the same architecture
- A framework that holds and runs the components
  - And provides services to the components to allow them to know about and interact with other components



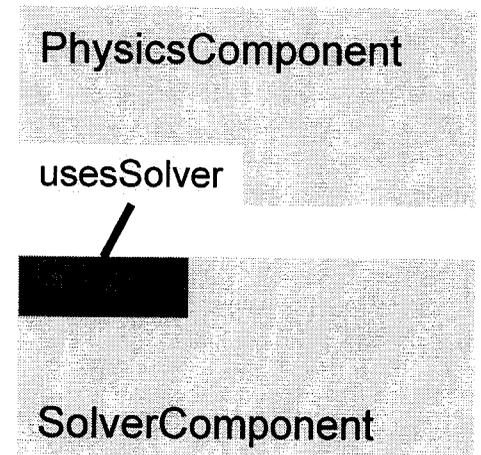
## CCA and CORBA/COM/DCOM?

- CCA is similar to CORBA/COM/DCOM because it is a CA
- All use an interface definition language (IDL)
- CCA is different from CORBA/COM/DCOM because it is written specifically for high-performance computing
  - The IDL (SIDL) includes array types
  - It is much faster than other CAs



# Common Component Architecture (CCA)

- A component model specifically designed for high-performance computing
- Supports both parallel and distributed applications
- Designed to be implementable without sacrificing performance
- Minimalist approach makes it easier to componentize existing software
- Components are peers
  - **No component assumes it is “in charge” of the others**
  - **Allows the application developer to decide what is important**
- Components interact through well-defined interfaces, or *ports*
  - **In OO languages, a port is a class**
  - **In Fortran, a port is a bunch of subroutines**
- A given component may *provide* a port – implement the class or subroutines
- The Go port is a special provides port - used to start the app’s first component
- Another component may *use* that port – call methods or subroutines in the port
- Links denote a caller/callee relationship, **not dataflow!**
  - **e.g., linSolve port might contain: *solve(in A, out x, in b)***





## Common Component Architecture (2)

- The framework provides the means to “hold” components and compose them into applications
- The framework is the application’s “main” or “program”
- Frameworks allow exchange of ports among components without exposing implementation details
- Frameworks may support sequential, distributed, or parallel execution models, or any combination they choose
- Frameworks provide a small set of standard services to components
- Steps to run an application:
  - Launch framework (use a GUI or a script)
  - Instantiate components required for app.
  - Connect appropriate provided and used ports
  - Start first component
    - **i.e., click Go port in the GUI or call the Go port in a script**
- CCA Forum is an open community working developing the CCA
  - Currently, mostly DOE and academic



## JPL's CCA Task

- The demonstration task ran from Feb. to Sept. 2002
- The task tried to answer two questions:
  - How usable is the CCA software? What work is involved for a scientist to take previously written software and turn it into components?
  - Once the components exist and are linked together, how does performance of the componentized version of the application compare with that of the original application?
- The task had two deliverables:
  - Report on completed sequential component demonstration of Pyramid AMR library and one application - 5/2002
  - Report on completed parallel component demonstration of Pyramid AMR library and one application - 9/2002



# PYRAMID:



## Parallel Unstructured Adaptive Mesh Refinement

Modern... Simple... Efficient... Scalable...

### Technology Description

An advanced software library supporting parallel adaptive mesh refinement in large-scale, adaptive scientific & engineering simulations.

### State-of-the-Art Design!

- Efficient object-oriented design in Fortran 90 and MPI
- Automatic mesh quality control & dynamic load balancing
- Scalable to hundreds of processors & millions of elements

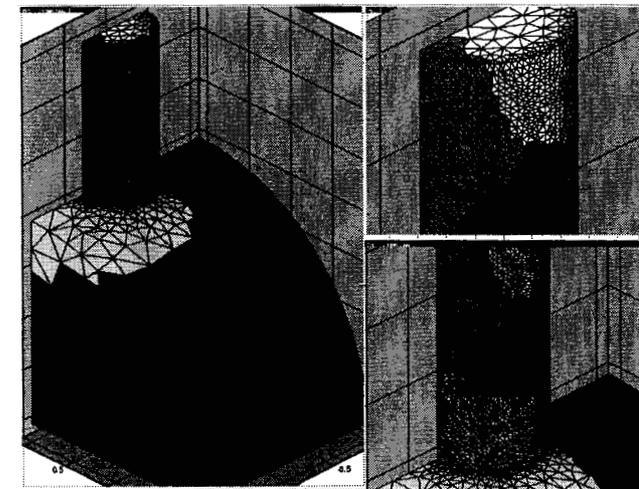
### Application Arena

- Computer Modeling & Simulation Applications with complex geometry
- Electromagnetic and semiconductor device modeling
- Structural/Mechanical/Fluid dynamics applications

**John Z. Lou, Charles D. Norton, & Thomas A. Cwik**

High Performance Computing Systems and Applications Group

<http://hpc.jpl.nasa.gov/APPS/AMR>





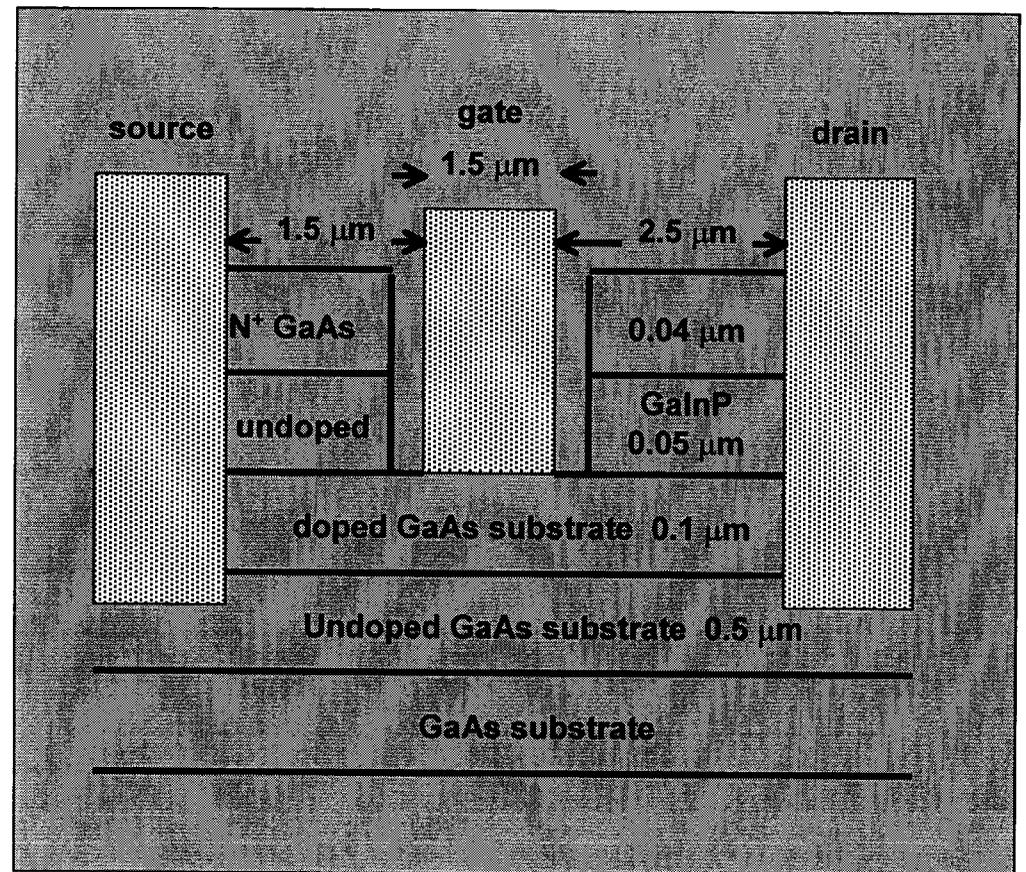
# Sample Application: Device Modeling

JPL



## MICROWAVE ACTIVE DEVICES

- Active devices have very thin layers with extended regions
- Charge resides in thin layers, but is driven by EM fields extending into bulk regions
  - This is a multi-scale problem
- This problem was examined using Pyramid in: T. Cwik, et. al, "Multi-Scale Meshes for Finite Element and Finite Volume Methods: Device Modeling," AP2000 Millennium Conference on Antennas & Propagation



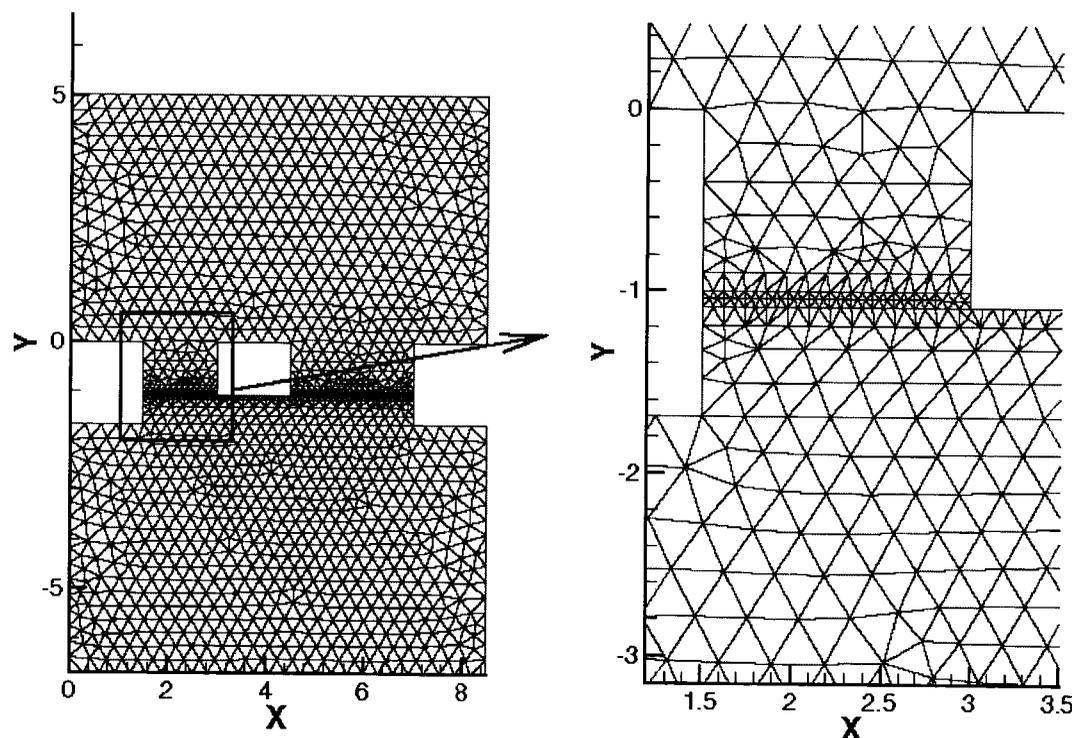
MESFET Model (not to scale)

Lin and Lu, IEEE Elec. Let. Sept 1996



# Multi-Scale Mesh: Geometry Driven

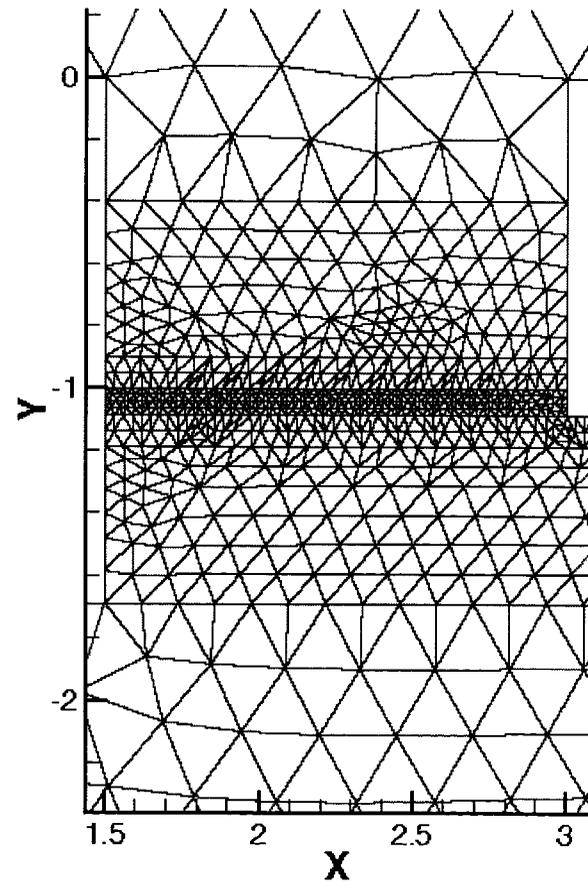
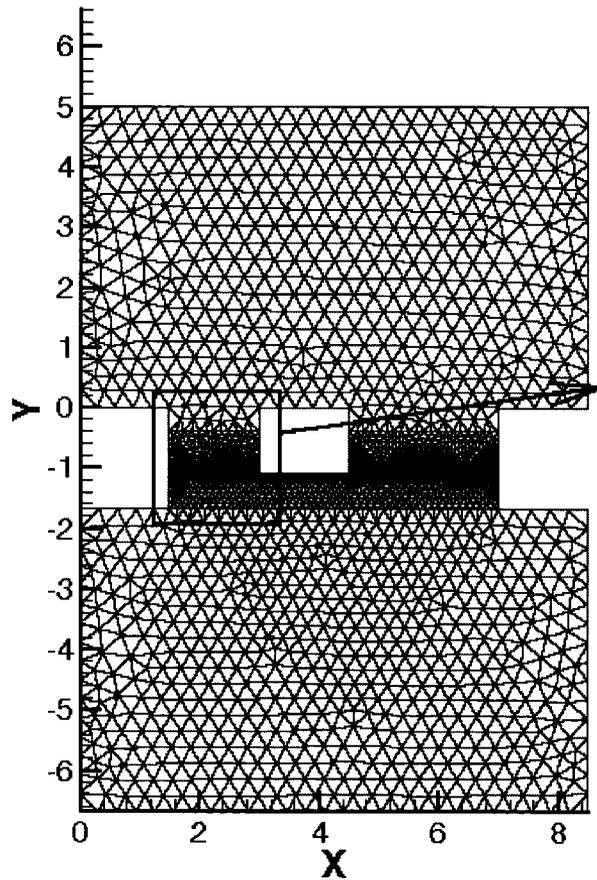
- Initial mesh, derived from a commercial mesh generator, contains large elements that just preserve the thin-layered geometry
- Pyramid library performs adaptive refinement of initial mesh in stages
- Problem solved using coupled Hydrodynamic/ Maxwell equations:
  - Irregular FDTD for EM updates
  - Box method for transport updates



- Our sample application is only concerned with building the mesh

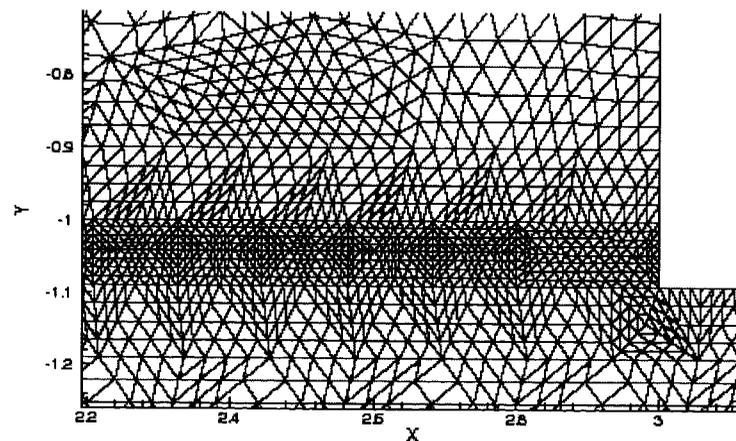
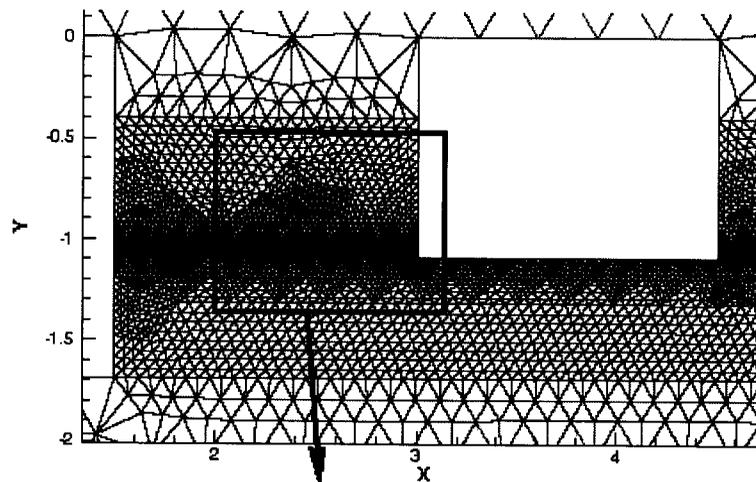
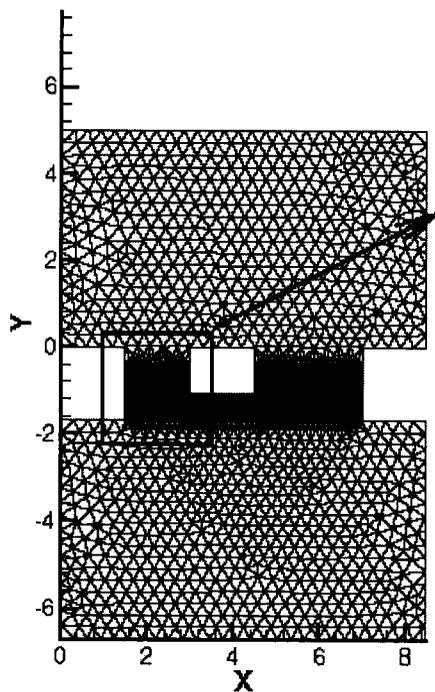


# Multi-Scale Mesh: Geometry Driven - Level 2



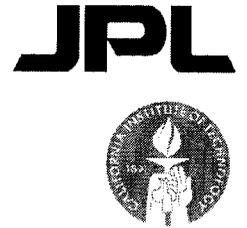


# Multi-Scale Mesh: Geometry Driven - Level 3





# Basic Component Examples: Hello World!



- Initially, we wrote 2 example applications
  - A single component Hello World! application
    - The hello component has only a Go port
      - The action of the Go port is to print “Hello World!” to standard out, then to exit
  - A two component Hello World! application
    - The hello component has a Go port, and a Uses port
      - The Uses port says that hello component will use a helloServer component
      - The action of the Go port is to instantiate a helloServer component, to call its returnString method, to print the returned string to standard out, then to exit
    - The helloServer component has a Provides port
      - This port provides a returnString method, which returns the string “Hello World!”



## Lessons from Basic Examples

- Learning the CCA software, then writing and running these examples took about 3 months of part-time effort for two people
  - Most of this effort was learning:
    - What are components?
    - What demonstration code is available?
    - How do we build and run the demos?
    - How do we extract the basics from the complex demos?
  - Very little work in actual writing
    - Create, build, and run our basic examples in C++



# Componentizing the Software

---

- Fairly short effort
  - About 3 weeks of part-time effort for two people
- We basically took what we learned from the simple examples (written in C++) and applied it to Pyramid
- However, Pyramid driver and library are Fortran 90
  - Understanding how to build components out of Fortran 90 code was our biggest challenge
  - Fortran 90 integration issues took a couple of weeks to work out
  - First step: examine interface between potential components...



## A Sample Pyramid Program

- A sample Pyramid program is Fortran90, and looks object-oriented
  1. Instantiate a mesh object
  2. Work with the mesh object, by calling method functions
- Calls to Pyramid are made with a first argument that is the mesh object to be worked on
  - call PAMR\_METHOD (input\_mesh, ...)
- In an object-oriented programming language, these calls would look like
  - input\_mesh.method (...)



## Fortran 90 Components?

---

- We observed that the main items passed across the interface are Fortran 90 pointers
- We chose to use the CCAFEINE framework, which required code to be written in C++
  - CCAFEINE now allows components that use BABEL, and thus permits code in C, C++, Fortran 77, Java, and Python
  - Other CCA frameworks exist, such as DCAFE
- We decided to write a C++ version of the driver code that could pass Fortran 90 pointers



# Details of Componentizing the Sequential Software



- First, we wrote a test program that used a Fortran 90 pointer
  - We compiled this into object code, to understand the routine names that the compiler was generating, so that these routines could later be called from C
  - Additionally, we compiled the code into assembler, which we studied to understand how a Fortran 90 pointer was stored and passed
- Once these two issues were clear, it was a simple matter to write a C main program, and to wrap the Pyramid library with a C wrapper
  - Neither of the main nor the wrapper are portable to other machines, OSes, or compilers, but the non-portable code is limited to two specific files, and can be rewritten for other environments
- Next, we wrote a C++ main program, and a C++ wrapper for the C-wrapped Fortran 90 library
- Once this was working, it was a simple matter to use the knowledge gained in the two-component Hello World example to turn the main and the wrapped library into components, and run them in the CCAFEINE framework



# Sequential Timing Results

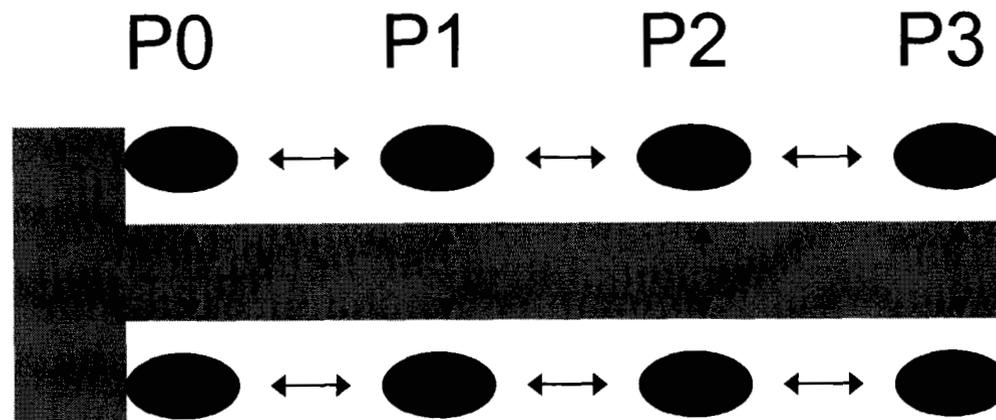
- Overall result - the overhead of componentization is negligible

	Original F90 application	Componentized application
User time, as returned by the Unix time call	19.51 s	19.49 s
Wall clock time, as measured from the first Pyramid call to the last Pyramid call	20.37 s	20.43 s
	- 0.87 s system time	- 0.91 s system time
	= 19.50 s	= 19.52 s
One call to Pyramid made one million times	98.83 s	102.24 s



# Parallelizing the Componentized Program

- Parallelizing the componentized program was trivial
  - One copy of the framework runs on each parallel processor
  - Each process of each parallel component communicates with the equivalent process of the other other component through the equivalent process of the framework
  - The processes in a parallel component communicate with each other through MPI



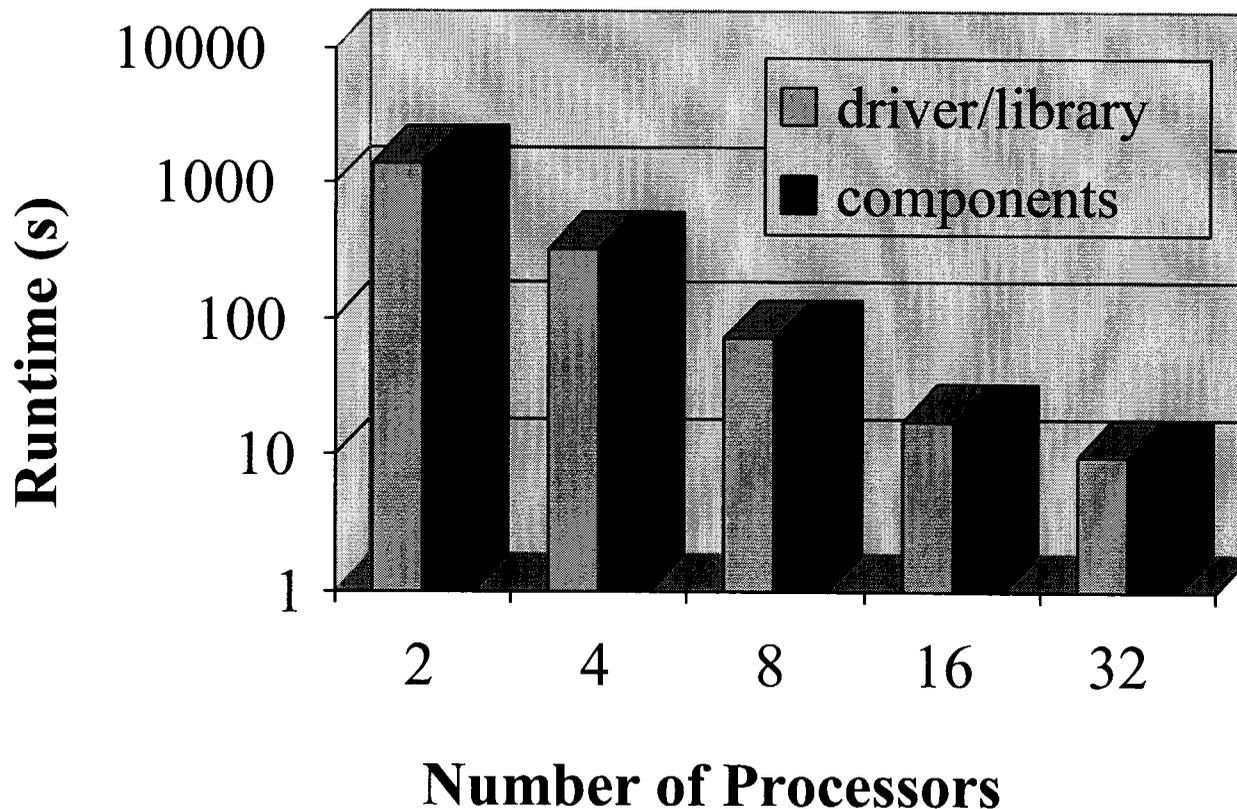
Components: Blue, Red

Framework: Gray



# Parallel Timing Results

- Overall results
  - The overhead of componentization is negligible
  - Componentization doesn't hurt scalability





## Lessons Learned (as of Sept. 2002)

---

- There was a fair amount of learning associated with use of the CCA Forum's technology, including the CCAFEINE framework
  - It may take 1-3 months for a computational scientist to be able to componentize an initial application
  - A second should be able to be componentized fairly quickly
- The lack of a means to write Fortran 90 components is a serious shortcoming for many science applications
  - It is possible to get around this shortcoming
  - This introduces additional work for the componentizer
  - This adds the chance for additional errors to come into the application
- Once an application is componentized, if the amount of work done in each component call is large when compared with the time needed to make a function call, it is likely that the componentized version of the application will perform well



# Topics

---

- ESTO CT
- CCA Demonstration Task  
(Completed Sept. 2002)
- Current CCA-Climate Work



## Current Work

---

- Aimed at infusing CCA into CT
  - Examine if CCA is useful for ESMF
- Working with the CCA Forum on Fortran 90 issues
- We are now working with a coupled climate simulation from UCLA...



# Development of an Earth System Model (ESM): Atmosphere-Ocean Dynamics and Tracers Chemistry

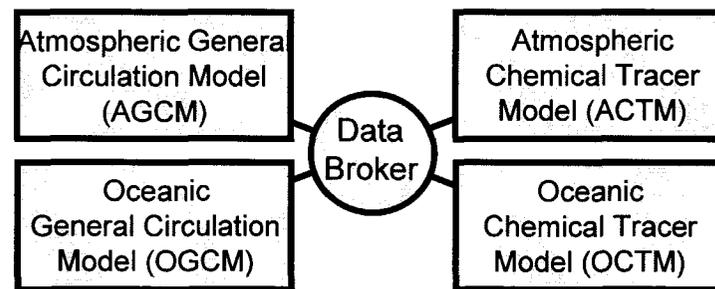
PI: Carlos R. Mechoso, UCLA <http://www.atmos.ucla.edu/esm>

JPL



**Goal:** *Develop and apply to problems of climate change a model that describes the coupled global atmosphere - global ocean system, including chemical tracers.*

**Ultimate goal:** *Have an ESM capable of performing ensembles of century-long simulations.*



**UCLA Earth System Model**

**Four model elements:**

- UCLA AGCM
- JPL version of LANL Parallel Ocean Program (POP)
- UCLA ACTM (which can include up to 64 species)
- JPL Ocean Chemical Transport Model

**Distributed Data Broker developed to couple all models**

**Each model designed for high performance parallel internal execution**

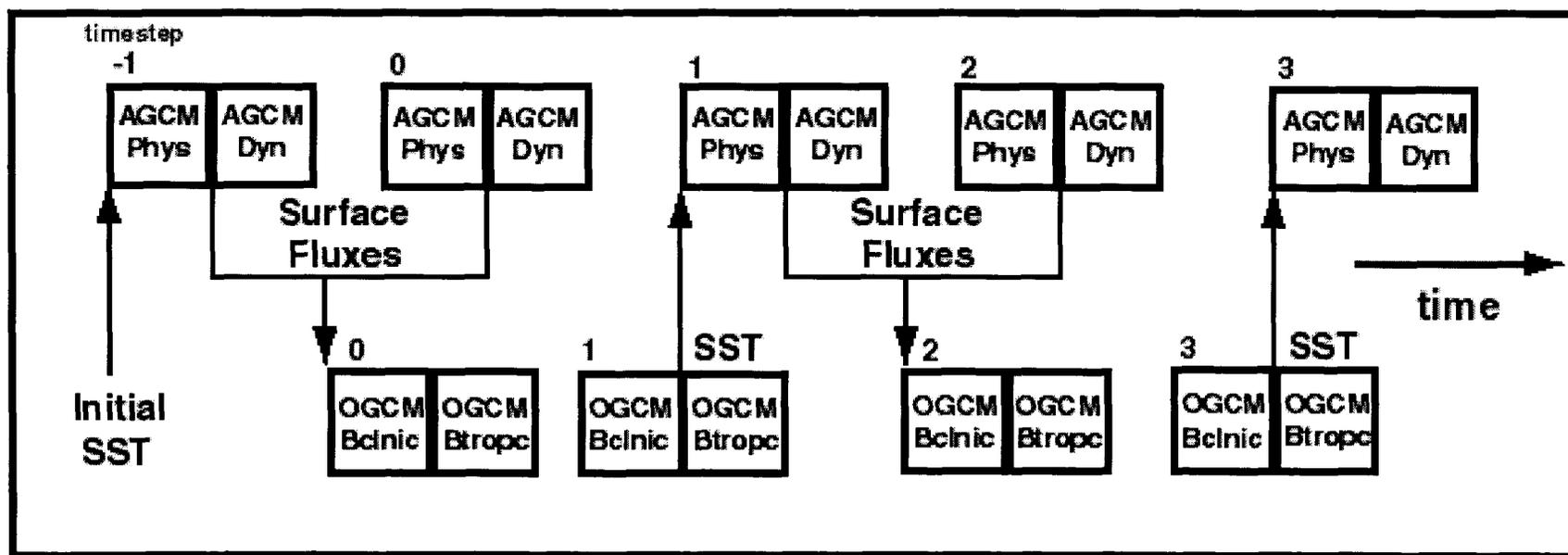
**Earth System Model designed for concurrent execution of models**



# Methodology for concurrent execution of ESM components



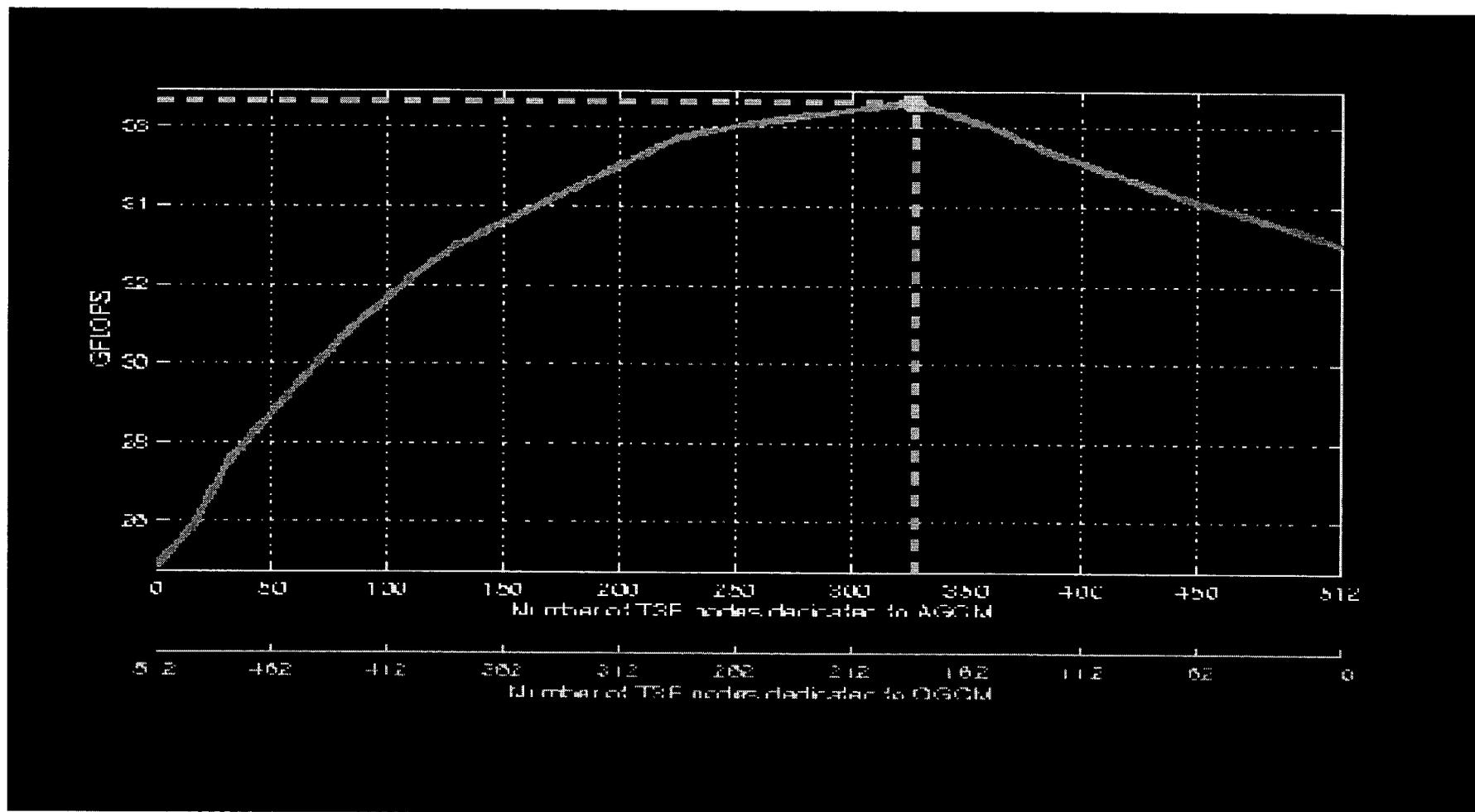
## AGCM-OGCM Coupling



Atmospheric surface fluxes that drive the ocean are produced by the model component known as AGCM/Physics; sea surface temperatures (SSTs) that drive the atmosphere are produced by the ocean model component known as OGCM/Baroclinic.



# AGCM and OGCM Coupled Performance



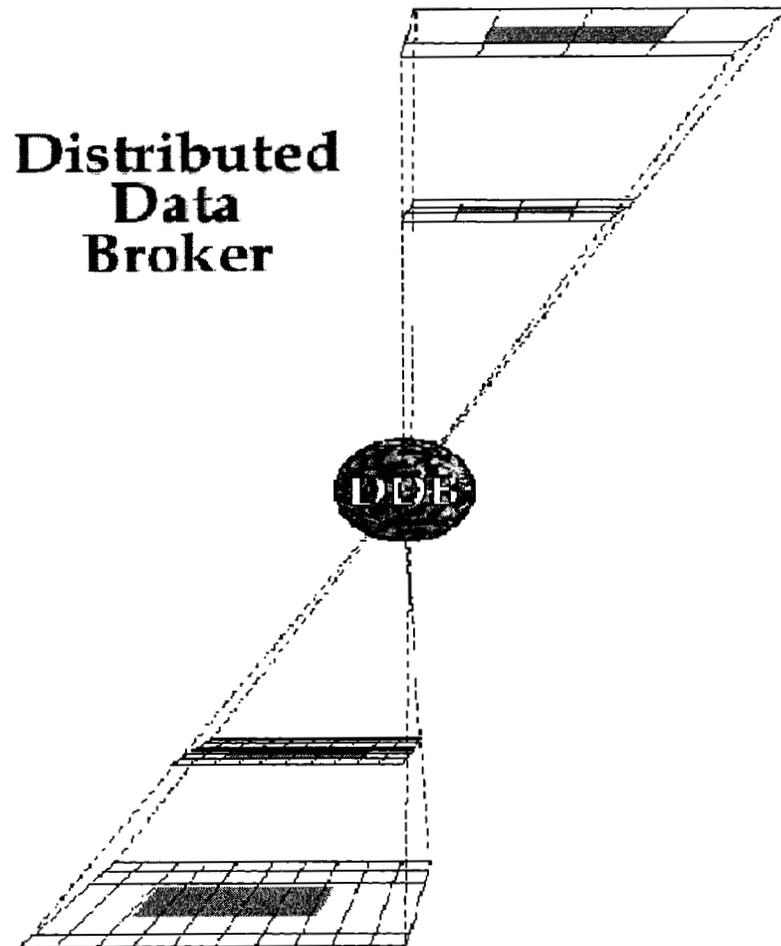


## Coupling between ESM components

- Data transfers between ESM components on different processors (example: SSTs from OGCM to AGCM) are handled by Data Brokers
- Other data transfers handled internally
  - For example, atmospheric circulation and chemistry are tightly coupled on the same processors
- UCLA code uses a Distributed Data Broker (DDB) to avoid performance bottlenecks and memory limitations that can occur with a Centralized Data Broker (CDB) which assembles the entire data field from a producer on a single processor before sending it to consumers



# Distributed Data Broker (DDB)



## Function

Software tool to handle distributed data exchanges between the ESM components

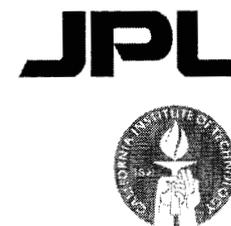
## Major Tasks

- Gather information from each model component (may have been decomposed into many subdomains running on different processors)
- Convert data resolutions, units..., etc. and redistribute them to the needed model components
- Keep track of coupling sequence, such as how often AGCM needs sea surface temperature and which processors have that information



# Distributed Data Broker (DDB) Components

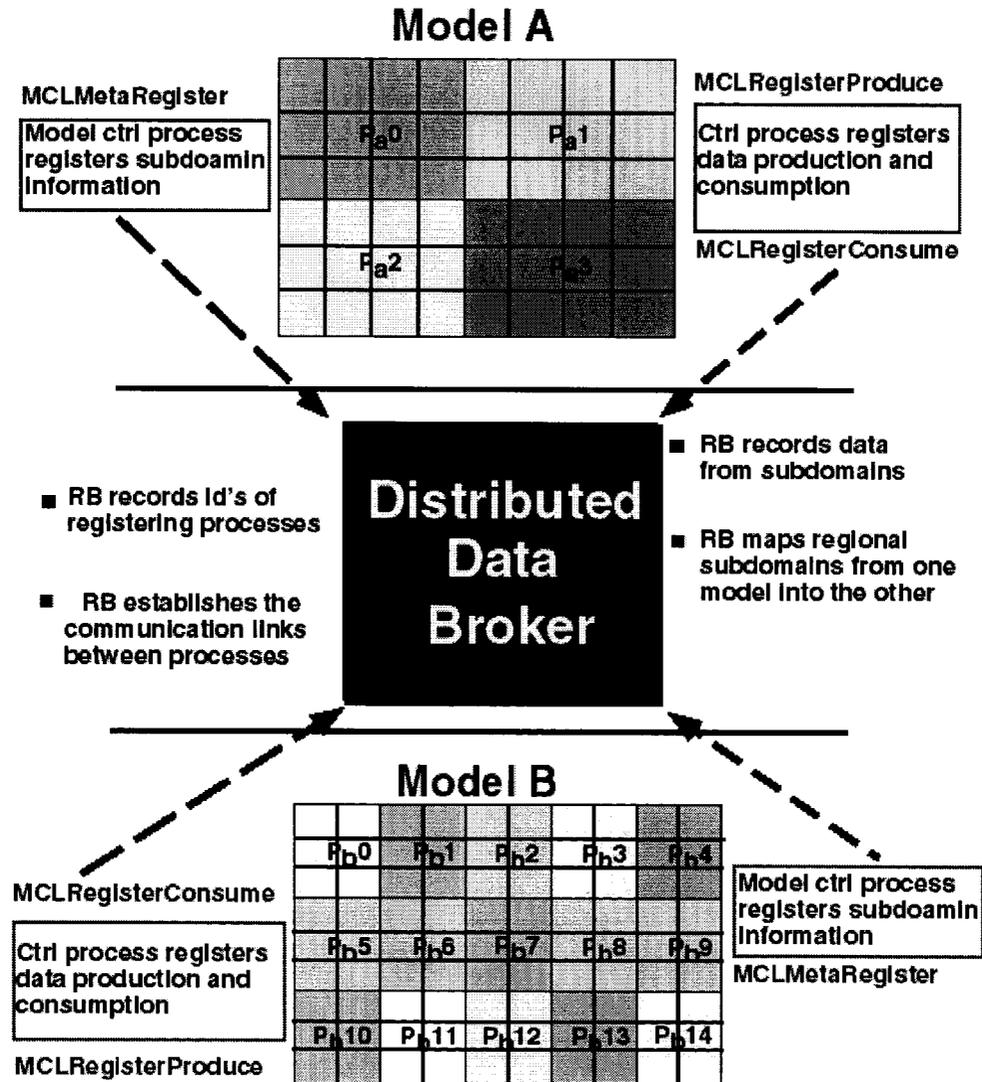
---



- **Model Communications Library**  
Callable routines for registering and exchanging information between model components.
- **Communications Library**  
General communication routines to manage the data exchanges based on standard communication toolkits  
PVM (current), MPI (through a PVM wrapper coming soon)
- **Data Translation Library**  
Routines for data regridding.

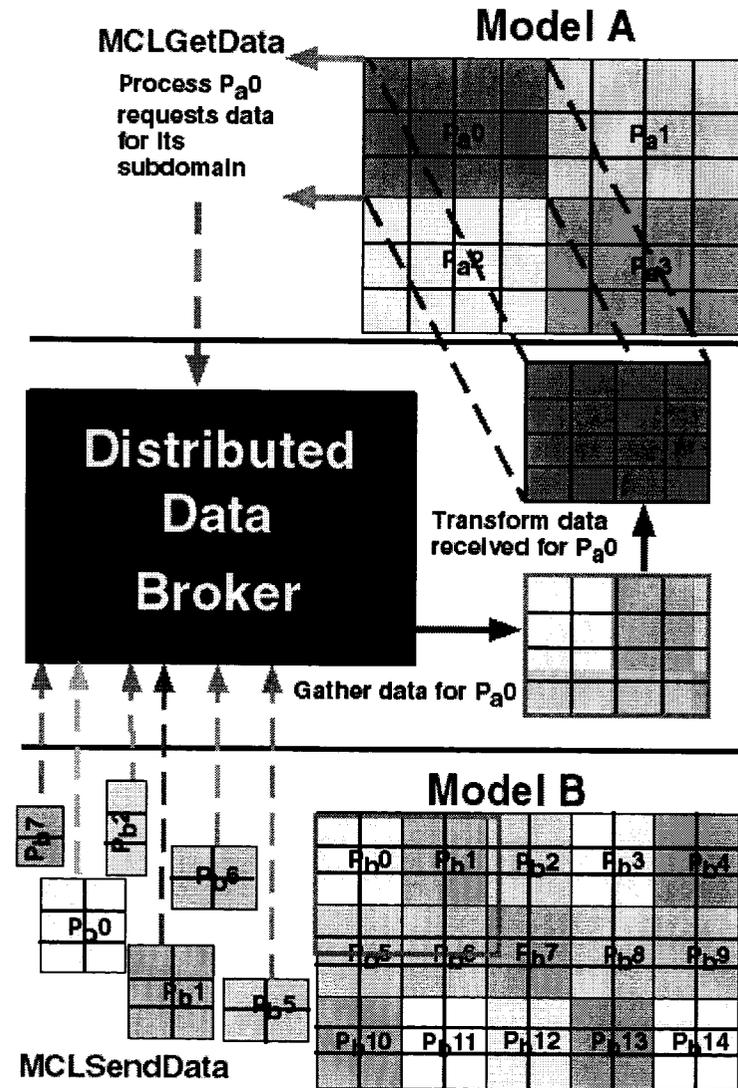


# MCL Registration





# MCL Send and Receive Data





# Status

- We intent to turn the UCLA coupled climate code into a CCA componentized application during the current fiscal year (Sept. 2003)
  - The code will become three CCA components: an atmosphere model, an ocean model, and a distributed data broker
  - Now modernizing the UCLA code
    - Changing the DDB from PVM to MPI
    - Using Fortran90 dynamic memory in place of Cray/Sun/SGI extensions in AGCM
  - Componentizing will start in a month or so



## Conclusions

- Knowledge of ongoing work within the CCA Forum (including our own) leads us to believe that the problems with learning the CCA methodology have been solved
- Most of the issues with using Fortran 90 have been resolved since Sept. 2002
- The rest will be resolved, most likely in the next 3-6 months
- Once this is done, the CCA model will be a promising method for building large parallel applications
- An example CCA climate application will be a powerful motivator for others in climate to investigate CCA
- A CCA DDB component may be very useful in non-climate applications



# Thank You for Your Attention

---

Any Questions?

Either now  
or later via email:

[Daniel.S.Katz@jpl.nasa.gov](mailto:Daniel.S.Katz@jpl.nasa.gov)

<http://pat.jpl.nasa.gov/public/dsk/>