DESIGN AND IMPLEMENTATION OF COMPONENTS IN THE EARTH SYSTEM MODELING FRAMEWORK

Nancy Collins[1]
Gerhard Theurich[2]
Cecelia DeLuca[1]
Max Suarez[3]
Atanas Trayanov[3]
V. Balaji[4]
Peggy Li[5]
Weiyu Yang[6]
Chris Hill[7]
Arlindo da Silva[3]

Abstract

The Earth System Modeling Framework is a component-based architecture for developing and assembling climate and related models. A virtual machine underlies the component-level constructs in ESMF, providing both a foundation for performance portability and mechanisms for resource allocation and component sequencing.

Key words: framework, high performance computing, climate modeling

The International Journal of High Performance Computing Applications,
Volume 19, No. 3, Summer 2005, pp. 341–350
DOI: 10.1177/1094342005056120
© 2005 Sage Publications

1 Introduction

Component-based design is a natural fit for climate modeling. At its simplest, a software component is a code that has a standard calling interface and behavior and a coherent function (see, for example, http://www.corba.org and http://www.cca-forum.org). Components are ideally suited for the representation of a system comprised of a set of substantial, distinct and interacting domains, such as atmosphere, land, sea ice, and ocean. Further, since Earth system domains are often studied and modeled as collections of processes (radiation and chemistry in an atmosphere, for example), it is convenient to model climate applications as a hierarchy of nested components.

Component-based software is also well suited for the manner in which climate models are developed and used. The multiple domains and processes in a model are usually developed as separate codes by specialists. The creation of a viable climate application requires the integration, testing, and tuning of the pieces, a scientifically and technically formidable task. When each piece is represented as a component with a standard interface and behavior, that integration, at least at the technical level, is more straightforward. Similarly, standard interfaces help to foster interoperability of components, and the use of components in different contexts. This is a primary concern for modelers, since they are motivated to explore and maintain alternative versions of algorithms (such as dynamical cores), whole physical domains (such as oceans), parametrizations (such as convection schemes), and configurations (such as “standalone” versions of physical domains).

The Earth System Modeling Framework (ESMF; Hill et al., 2004; see http://www.esmf.ucar.edu) project is a large, multi-agency collaboration whose goal is to develop common modeling infrastructure and deploy it in climate, weather, and data assimilation applications. It is currently in its third year of development. The climate applications that are evaluating and adopting ESMF include the Community Climate System Model (CCSM; Boville and

[1]NATIONAL CENTER FOR ATMOSPHERIC RESEARCH (NANCY@UCAR.EDU)

[2]SILICON GRAPHICS INCORPORATED

[3]GODDARD SPACE FLIGHT CENTER

[4]GEOPHYSICAL FLUID DYNAMICS LABORATORY

[5]JET PROPULSION LABORATORY

[6]NATIONAL CENTERS FOR ENVIRONMENTAL PREDICTION

[7]MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Gent, 1998; see http://www.ccsm.ucar.edu), the National Oceanic and Atmospheric Administration (NOAA) Geophysical Fluid Dynamics Laboratory (GFDL) models (see http://www.gfdl.noaa.gov/~fms), the new NASA Goddard Earth Observing System (GEOS-5) model, and the MITgcm (see http://www.mitgcm.org). The Strategic Plan for the U.S. Climate Change Science Program (CCSP), a report by the CCSP and the Subcommittee on Global Change Research (July 2003), recognizes ESMF as the realization of common modeling infrastructure for the climate community.

The ESMF software consists of an infrastructure of utilities and data structures for building model components, and a superstructure for combining them into applications. In this paper we concern ourselves with the superstructure, and how ESMF components and related constructs are designed and implemented.

2 The ESMF Superstructure

The ESMF superstructure defines an architecture for assembling Earth system applications from modeling components. A component may be defined in terms of the physical domain or process that it represents, such as an atmosphere or sea ice model. It may also be defined in terms of a computational or scientific function, such as an I/O or diagnostic system. Climate models often require that such components be coupled together to create an application. By coupling we mean the data transformations and, on parallel computing systems, data transfers, that are necessary to allow data from one component to be utilized by another. ESMF offers a suite of regridding methods and other tools to simplify the organization and execution of intercomponent data exchanges.

All ESMF methods can be called from Fortran 90, and some, including those at the component level, can be called from C++. The framework is implemented in a combination of these languages. Since most climate modelers work in Fortran, in this paper we focus on Fortran bindings. ESMF currently runs only in single program multiple datastream (SPMD) mode, although execution of components can be either concurrent or sequential. It is supported on IBM, SGI, Compaq, Mac OS X, Cray X1, and many Linux platforms.

The ESMF is implemented using an object-oriented design strategy. Superstructure and infrastructure are organized as sets of classes, which are data structures with associated methods. The main classes in the superstructure are component, state, and application driver.

2.1 COMPONENT

An ESMF component has two parts: one that is supplied by the ESMF and one that is supplied by the user. The part that is supplied by the framework is in the form of a predefined Fortran derived type. ESMF provides two kinds of components: a gridded component (`GridComp`) and a coupler component (`CplComp`). A gridded component represents a physical domain in which data are associated with one or more grids; for example, a dynamical core. A coupler component arranges and executes data transformations and transfers between one or more gridded components. All gridded components and coupler components possess initialize, run, and finalize methods with standard interfaces. These methods can be multiphase. Components also have create and destroy methods.

The second part of an ESMF component is the user code that will perform the computational work. The user code must be split up so that it too contains clear initialize, run, and finalize subroutines that match standard interfaces. The arguments must be ESMF data structures, although these data structures need not be used in the body of the code. Users set entry points within their code so that their initialize, run, and finalize subroutines are callable by the framework. In practice, setting entry points means that a special method called set services[1] is written by the user for every component. Within set services there are calls to ESMF set entry point methods that associate the name of a user's Fortran subroutine with the corresponding standard component operation. For example, a user might create an ESMF gridded component representing an ocean model known as the Parallel Ocean Program (POP) by making the following calls from a driver or parent component:

```
type(ESMF_GridComp) :: pop
pop = ESMF_GridCompCreate("POP Ocean", … , rc)
```

If the Fortran subroutine names of the user's initialize, run, and finalize methods were `popInit`, `popRun`, and `popFinal`, respectively, the set services method would contain the following fragment:

```
call ESMF_GridCompSetEntryPoint(pop,
ESMF_SETINIT, popInit, … ,rc)
call ESMF_GridCompSetEntryPoint(pop,
ESMF_SETRUN, popRun, … ,rc)
call ESMF_GridCompSetEntryPoint(pop,
ESMF_SETFINAL, popFinal, … ,rc)
```

These calls link the two pieces of the component: the gridded component derived type provided by the framework and the model methods provided by the user. The result is that the POP model can be dispatched by a driver or by a parent component in a very generic way. The create and destroy operations for components are not linked to user code; they act only on the component derived type.

The set services method (called `POPSetServices` here) and initialize, run, finalize methods are invoked from the driver or parent component. They follow the `ESMF_GridCompCreate()` call shown previously:

```
call ESMF_GridCompSetServices
                     (pop, POPSetServices, rc)

call ESMF_GridCompInitialize(pop, … ,rc=rc)
call ESMF_GridCompRun(pop, … ,rc=rc)
call ESMF_GridCompFinalize(pop, … ,rc=rc)
```

2.2 STATE

ESMF components exchange information with other components only through states. A `State` is a Fortran derived type that can contain other ESMF types representing fields, bundles of fields on the same grid, and arrays. It can also contain other states. A gridded component is associated with two states: an import state and an export state. Its import state holds the data that it receives from other gridded components. Its export state contains data that it can make available to other gridded components.

2.3 APPLICATION DRIVER

The application driver `(AppDriver)` is a small, generic driver program that contains the "main" routine for an ESMF application. It can be thought of as the "cap" for a hierarchical application.

Infrastructure classes closely associated with the ESMF superstructure include the `Clock` class and the virtual machine (`VM`) class. An ESMF clock contains information about the start time, stop time, and time-step of a component, based on a variety of supported calendars. The time-step may be changed during component execution. The clock may be associated with multiple alarms that "ring" for periodic or unique events. The VM controls computational resource allocation and is described in detail in Section The ESMF Virtual Machine and Component Parallelism.

A very simple ESMF coupled application might involve an AppDriver, a parent gridded component, two or more child gridded components that require an intercomponent data exchange, and a coupler component. Calls cascade so that when, for example, the initialize routine of a parent component is called, it in turn calls the initialize routines of all its children. Figure 1 illustrates the structure of such an application.

3 Application Example

The newly developed NASA GEOS-5 atmospheric general circulation model (AGCM) provides a more comprehensive example of how ESMF is being used to structure climate and related models. GEOS-5 is the atmospheric component of a variety of applications at the NASA Global Modeling and Assimilation Office (GMAO). It will be used for research in satellite data utilization; as part of the GMAO atmospheric data assimilation system; for weather, subseasonal, and seasonal to interannual forecasting; for atmospheric chemistry studies; carbon cycle research; and research on ocean–atmosphere and atmosphere–land interactions. In 2006, GEOS-5 is planned for use in a satellite-era reanalysis in support of the CCSP.

Figure 2 shows GEOS-5 science components connected for a standalone configuration of the coupled atmosphere–land system. Each box in the GEOS-5 diagram is an ESMF gridded component. GEOS-5 adopts the hierarchical topology that is natural under ESMF. The main computational modules are located in the leaves of the hierarchy (finite volume dynamical core, catchment, infrared, etc.). These are connected through composite components (radiation, dynamics, physics, etc.), which implement higher levels of integration. A single composite component (AGCM) integrates the entire standalone GCM. At the same level is an I/O component, which handles the diagnostic history interface. The whole system is capped by a GEOS-5 gridded component and an application driver.

One of the advantages of the hierarchical structure is that branches of the tree can be easily pruned and capped to form more limited standalone applications. For example, the GEOS-5 single column model is constructed by simply connecting physics to a new application "cap" and discarding the dynamics branch, but retaining history diagnostics. Another advantage of the architecture is that standard interfaces at each level make it technically straightforward to swap in new components; for example, a new radiation module.

4 Component Implementation

The connection of specific subroutines to the initialize, run, and finalize entry points is implemented using a virtual function table. The function table code is implemented in C++ but is callable from standard Fortran 90. For each entry this table holds the name of the entry, an `enum` that defines the types of arguments to be passed to the subroutine, an array of argument pointers declared as type `(void *)`, and a pointer to the subroutine to be called.

We can store a pointer to the subroutine to be called because Fortran 90 allows subroutines or function names to be passed as arguments. While the Fortran 90 language does not mandate how subroutines are passed when calling into C or C++ code, all compilers we have run on pass the subroutine as a pointer to the start of the execut-

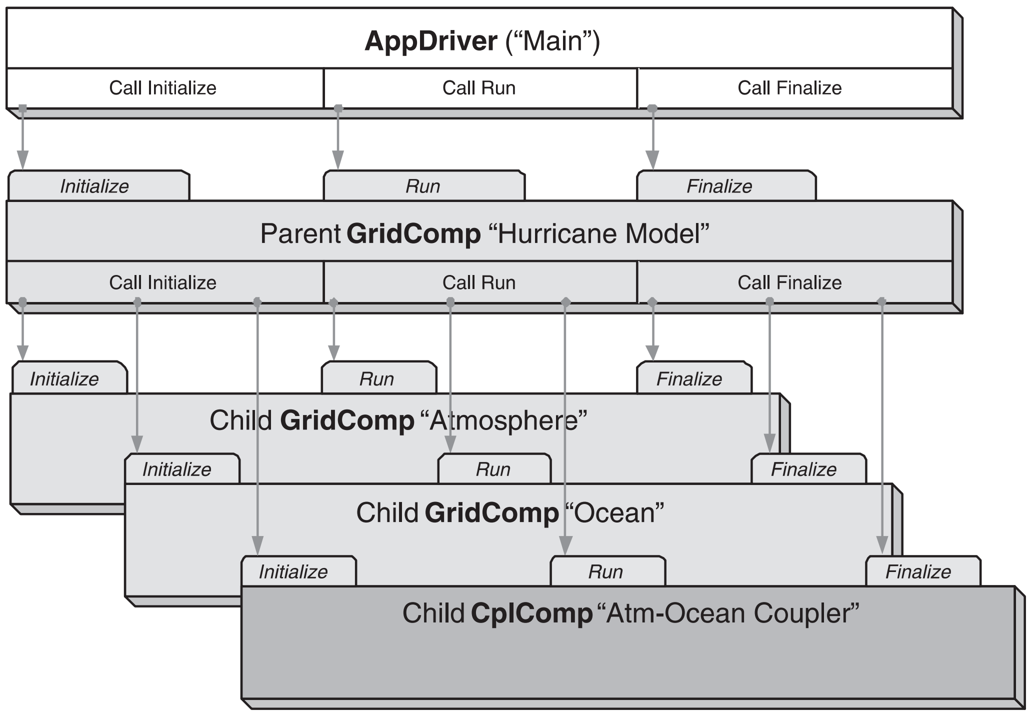

Fig. 1 Application example showing two ESMF gridded components (atmosphere and ocean) and a coupler component. These are the child components of a gridded component that simulates hurricanes. The entire assemblage is called by an application driver.

able code. This is stored in the virtual function table as `void *func`, and dispatched as `(*func)(arg1, arg2, ...)`. The framework uses this feature to populate the virtual function table and to dispatch subroutines from the table. Connections are defined at run-time and not compile time. In the future this will allow components to be dynamically replaced based on the evolution of a simulation, for example.

When a component method such as `ESMF_GridCompRun()` is called, control is transferred to the framework before user code is executed. The framework determines which execution threads should be active in the component, may optionally validate arguments, may optionally supply default values for any missing arguments, and then calls through the virtual function table to execute the code for the requested method. This isolates the components from any language dependence; components written in Fortran 90, C, or C++ can call subcomponents which are implemented in Fortran 90, C, or C++ without change.

A pointer to the virtual function table is stored as an entry in standard ESMF component derived types as an opaque 4- or 8-byte integer. All subroutines involving the function table simply require that the Fortran 90 interface has the component type as one of the arguments. The opaque entry is then passed to a C++ subroutine which uses it as the `this` pointer in the virtual function table class method.

We find that ESMF components work well with the `module` concept in Fortran 90, especially for controlling the scope of symbols. Module data and methods can be private to the module. The parent component includes the child code with the Fortran `use` statement. Only the set services subroutine name must be a public symbol, so it can be referenced by the parent component. All other module methods, even those which implement initialize, run, and finalize, can be private. Applications are linked into a single executable, which simplifies the interaction with the various batch systems common on high-end supercomputers, and requires no run-time dynamic loading of user code.

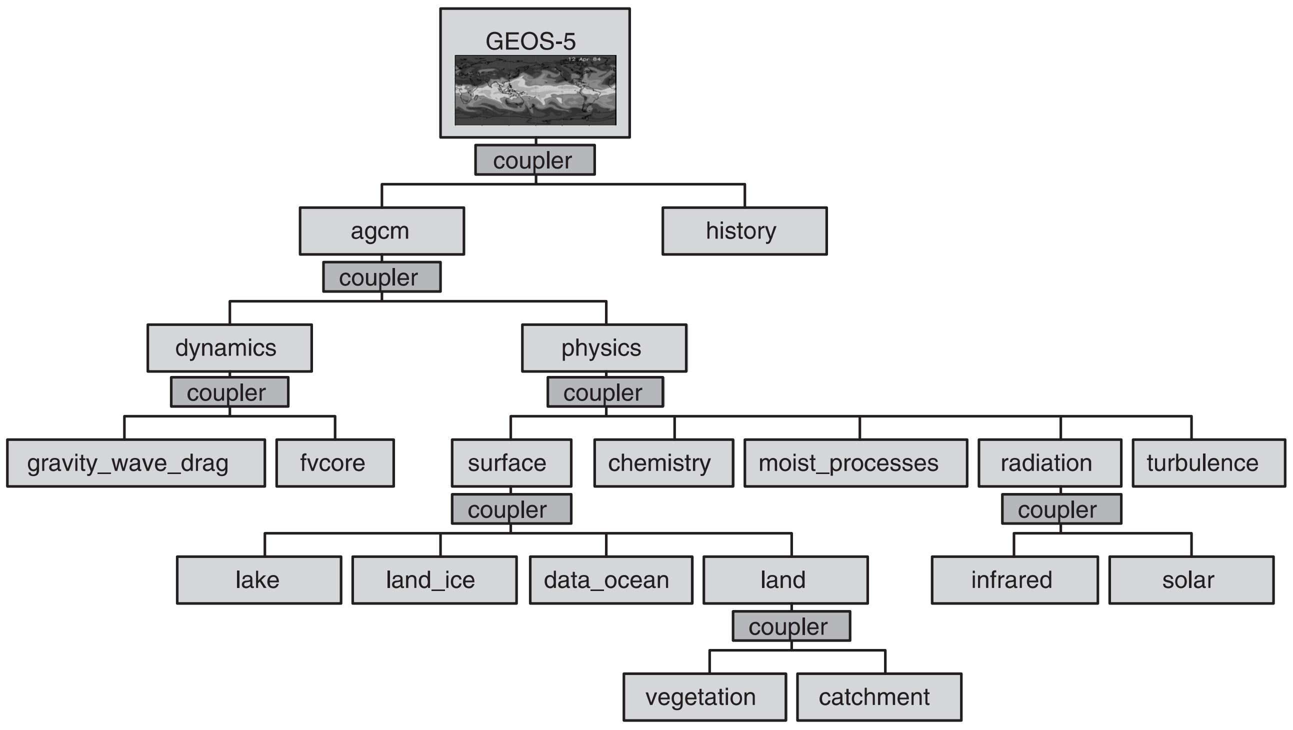

Fig. 2 ESMF-based architecture of the GEOS-5 atmospheric general circulation model. Each box is an ESMF component. Component and coupling interfaces are standardized to facilitate exchanges and extensions. The operations in each component or coupling transformation can be easily customized.

5 The ESMF Virtual Machine and Component Parallelism

Components are the building blocks of any ESMF application. We extended the component concept to include parallelism by making ESMF components the very units of parallel execution. In ESMF, components offer both data and task parallelism, and provide powerful concepts that aid the computational scientist to write highly efficient and scalable code.

The management of the available computational resources maps quite naturally onto the component hierarchy. When a component is created it obtains a set of resources from its parent component. In turn, when the component creates children of its own, it divides up its resources and provides them to its children. Critical in the design of the parallel aspects of components was to guarantee that components remain self-contained and could be written without detailed knowledge of the parent component code. This has been achieved by running each component in a separate parallel execution context.[2]

In order to provide the desired transparent interface to the parallel execution environment, ESMF components utilize the ESMF VM class. VMs are not new; the idea of a generic machine representation dates back to the 1960s and has appeared in a wide diversity of software packages and languages, including early IBM systems (Adair et al., 1966), Java (see http://java.sun.com), and, in the high performance computing realm, Parallel Virtual Machine (PVM; Geist et al., 1994).

The ESMF VM's interaction with the rest of the framework can be divided into two parts, as follows.

1. The execution engine provides separate parallel execution contexts or VMs for every component.
2. The communication interface offers efficient communications among elements of a VM.

The VM abstracts away many details of the underlying execution environment. Each VM can be viewed as a generic representation of hardware and system software resources available to each individual ESMF component. There is exactly one VM object associated with each component in an ESMF application. The VM handles resource management tasks and provides a topological description of the underlying configuration of compute resources accessible to the component. The communications interface provided by the VM utilizes this information to offer the best possible performance.

5.1 USING THE VIRTUAL MACHINE

At the beginning of every ESMF application, before making any framework calls, the user must make a call to an `ESMF_Initialize` method; likewise, at the end, the user must call `ESMF_Finalize`. These calls set up and shut down the whole framework, including the VMs. By default, ESMF is based on MPI-1 (see http://www.mpi-forum.org) and, as such, the main program is launched like any other SPMD message passing interface (MPI) application. ESMF does not directly interact with batch or queue management systems, and obtains its computational resources through the system-dependent MPI startup facility, such as mpirun, prun, poe, etc.

During the initialization phase, a default global VM is created. The user code can gain access to this VM by either providing an optional VM argument to `ESMF_Initialize()` or by calling `ESMF_VMGetGlobal()` anywhere in the ESMF application. The default global VM is equivalent to `MPI_COMM_WORLD` and provides the parallel execution context for the main program. Although there is no component associated with the main program, the default global VM can be seen as the parent of all VMs of the ESMF application.

The basic elements of a VM are persistent execution threads (PETs). PETs are equivalent to POSIX threads and have a lifetime of at least that of the associated component. By their nature, PETs provide an elementary OS unit of execution, associated with the virtual address space of the POSIX process in which the PET is running. Furthermore, each VM contains a mapping of its PETs onto the set of unique hardware processing elements (PEs) that were discovered during the framework initialization phase.

The PEs in a VM correspond to the smallest processing units that are recognized by the operating system. The VM layer keeps internal information about the topology of the PEs by noting whether they are part of a single CPU, lie within a single system image, or are connected via an interconnection fabric. All VMs in an ESMF application share this hardware map.

ESMF-level threading and user-level threading are two special features of the ESMF's VM implementation. ESMF-level threading provides coarse-grained parallelism by allowing multiple PETs to run within the same POSIX process. A component must be completely thread-safe to utilize this type of threading but can expect major reduction in its communication cost between threaded PETs. The VM communication API is completely transparent with respect to ESMF-level threading. User-level threading, on the other hand, does not require the entire component to be thread-safe. A typical user-level threading approach would employ OpenMP for fine-grained loop-parallelization. The VM assists user-level threading by controlling resource management, and allowing single PETs to be associated with multiple PEs. When during the execution of a component a region is encountered that profits from user-level threading, the user code can inquire how many PEs are associated with a particular PET and spawn this number of temporary threads without risking oversubscription of the available PEs. The decision to use ESMF-level and/or user-level threading is made on a per component basis, allowing a component writer to choose the best approach for the particular problem.

5.2 COMPONENT–VIRTUAL MACHINE INTERACTIONS

There are four distinct phases in the life cycle of an ESMF component during which it interacts with the execution engine of the VM.

1. Component Creation. When a parent component or driver creates a child component, by either calling `ESMF_GridCompCreate()` or `ESMF_CplCompCreate()`, it needs to provide a reference to its own VM. It is in the component creation call that a parent indicates which of its compute resources it wants to give to its child. This is done by adding the optional parameter `petList` to the arguments of the creation call. A `petList` is a list of parent PETs that are to be given to the child. The default, when no `petList` argument is provided, is to give all parent PETs to the child.
2. Component SetServices. As described in Section 2, The ESMF Superstructure, for each component the user writes a set services routine containing calls to set entry point methods. These link the initialize, run, and finalize subroutines of the user's code with the associated ESMF component derived type. In addition to setting entry points, during the child's set services method, the child component can also set certain properties of its VM. For example, when the call `ESMF_GridCompSetVMMaxPEs(pop, rc=rc)` is executed within set services it will associate as many PEs as possible for each child component PET. It is important

to note that the set entry point code is executed within the context of the parent VM, not the child VM (which does not yet exist). This means that users should not add additional operations such as data allocations to the SetServices method because they will not be associated with the correct VM. On return from the set services call the framework starts up the child VM and puts it on hold.

3. Call of standard component method. The child VM becomes active and executes the initialize, run, or finalize component method. When the end of the routine is reached the VM is placed back on hold, waiting for the next invocation, and control is handed back to the parent VM.
4. Component Destroy. The child VM is shut down and all resources are released.

5.3 CONCURRENT AND SEQUENTIAL EXECUTION OF COMPONENTS

By default, ESMF components run sequentially in the order in which they are called by their parent component or driver. ESMF also provides a very easy to use concurrent component model. The approach is an extension of the non-blocking concept well known from MPI's communication interface. In this model the run sequence (item 3 in the previous list) is split into two separate phases:

(a) Non-blocking call of registered component method. The child VM becomes active and executes the registered component routine within the child VM. When the end of the registered routine is reached the VM is placed back on hold, waiting for the next invocation. The parent VM does not wait until the child method has reached the end but continues execution in its own context immediately.

(b) Parent component calls wait. In order to synchronize its children, the parent component uses `ESMF_GridCompWait()` or `ESMF_CplComp-Wait()` to wait for the completion of a previously invoked child component method. The wait call will block all parent PETs until the child component's method has reached its end and the child VM has been placed on hold.

Within ESMF's concurrent component model, it is the parent's prerogative to decide if and which child components it will run concurrently. The child component code is unaffected by this. As with many other aspects, ESMF does not ensure that it is semantically correct to run certain components concurrently. It is up to the application writer to decide what makes sense and what does not. The following segment shows how phase 3 of the parent code must be split to run two Gridded Components, `gcomp1` and `gcomp2`, concurrently:

```
call ESMF_GridCompRun(gcomp1, … ,
          blockingFlag=ESMF_NONBLOCKING, rc=rc)
call ESMF_GridCompRun(gcomp2,...,
          blockingFlag=ESMF_NONBLOCKING, rc=rc)

call ESMF_GridCompWait(gcomp1, rc=rc)
call ESMF_GridCompWait(gcomp2, rc=rc)
```

6 Performance Overhead

At its relatively early stage of development, ESMF has just begun to perform exhaustive performance tests and optimizations for low-level communications, regridding methods, halo operations, and other parts of the framework. However, it has been a high priority from the start of the ESMF project to ensure that the basic component architecture of the framework is not associated with substantial performance overhead. In this section we describe results of initial performance evaluations. Table 1 shows the performance overhead associated with calling component methods through a virtual function table. Tests were run on the Compaq SC45 at NASA Goddard Space Flight Center. The "w/o ESMF threads" columns show the overhead for the case where ESMF threading is not enabled (although the user may choose to implement threading using OpenMP on their own). Timings are shown with and without a barrier synchronization upon completion of the component call.[3]

We have also examined the impact of ESMF on the performance of a real code, albeit one that, unlike climate models, emphasizes efficiency of initialization as well as efficiency of run-time. The National Centers for Environmental Prediction (NCEP) Spectral Statistical Interpolation (SSI) package (Parrish and Derber, 1992) is a three-dimensional variational analysis of observations used by the National Weather Service in their data assimilation system to initialize their global atmospheric model. The physical domain of SSI is the global atmosphere from the surface to the stratopause. Instead of physics parametrizations, the SSI is comprised of forward model elements and their adjoints. Some major forward model elements are radiative transfer algorithms for each satellite instrument, convective and large-scale precipitation, spectral transform, grid interpolation, the balance equation, and the divergence tendency equation. The analysis minimizes a combination of fits to observations, fits to model background, and a set of dynamical constraints. The minimization is performed in spectral space. Computation of forward models and their adjoints are required at every iteration, and both initialization

Table 1
Overhead associated with component-level calls

	Overhead (microseconds)			
Number of PETs	w/o ESMF threads w barrier	w ESMF threads w barrier	w/o ESMF threads w/o barrier	w ESMF threads w/o barrier
1	55	55	54	55
8	65	475	46	425
80	120	675	55	450

and run routines are executed at every cycle. One cycle of initialize and run was performed for this experiment.

The version of the SSI code tested with ESMF used only the superstructure classes. It was not coupled with any other ESMF components nor did it use any part of the ESMF infrastructure layer.

In addition to the general component overhead, the SSI code with ESMF also had algorithmic overhead, due to the lack of ESMF support for spectral fields at the time of the experiment. The result was that there were three extra data conversions in the ESMF code. The performance analysis was run on an IBM Power 3 cluster at the National Center for Atmospheric Research (NCAR).

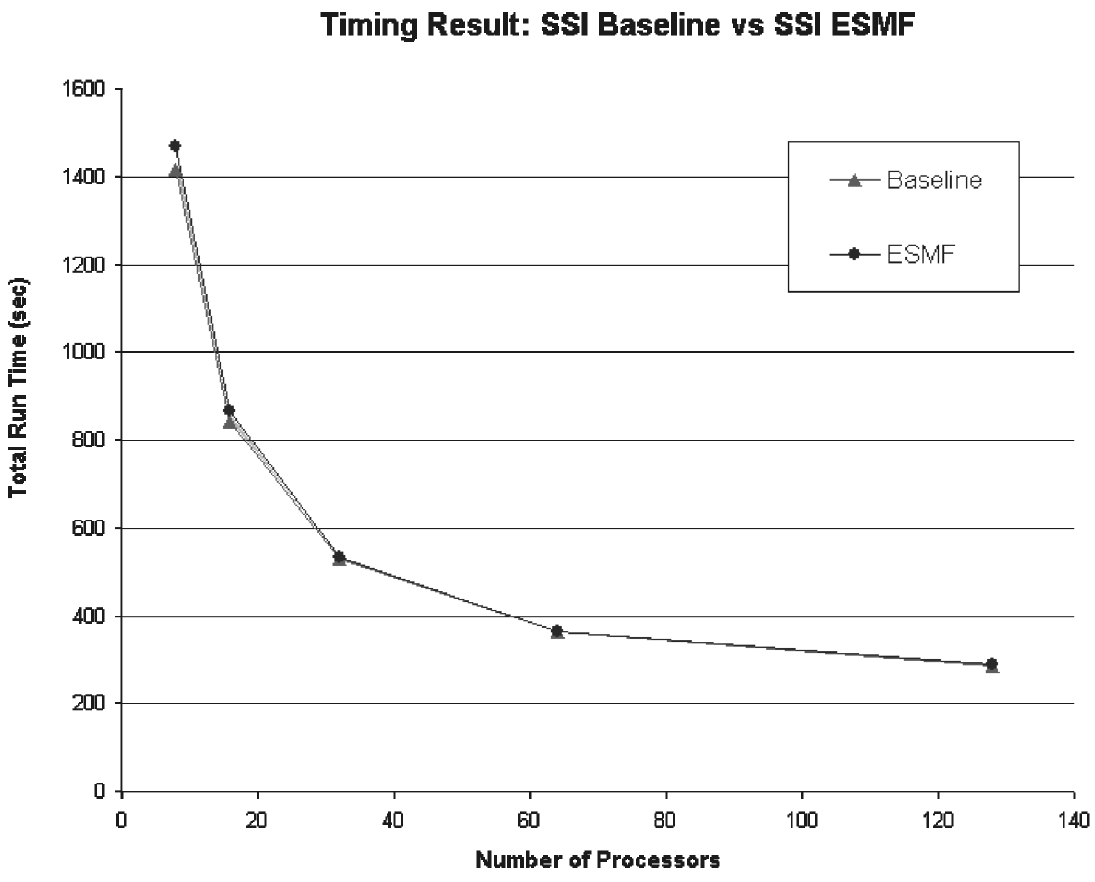

Fig. 3 ESMF overhead in the NCEP SSI.

Figure 3 shows the total ESMF overhead including the extra data conversion from the spectral fields into Gaussian grids and vice versa. The overhead was about 4 seconds (slightly increasing with increasing number of processors), or 0.26% for eight processors to 1.82% for 128 processors. For the SSI application, this timing difference is insignificant.

7 Conclusions

The ESMF hierarchical, component-based architecture is natural for the construction of climate and related applications. ESMF components are linked to a powerful VM construct which offers integrated parallelization and a generic representation of the high performance computing hardware and software environment. Initial performance tests indicate that the architecture will not impose a prohibitive performance overhead on applications. The ESMF project will continue to evolve and optimize its software in close collaboration with the Earth system modeling community.

ACKNOWLEDGMENTS

The ESMF Joint Specification Team defined the framework architecture. Thanks to the many members of this multi-agency team for their thoughtful advice, patience, and persistence. The NASA Earth Science Technology Office (under Cooperative Agreement CAN-00OES-01), the National Science Foundation, and the Department of Defense High Performance Computing Modernization Program support the research and development of ESMF.

AUTHOR BIOGRAPHIES

Nancy Collins is one of the lead architects of the Earth System Modeling Framework. She designed and implemented the component layer in ESMF, along with many other aspects of the framework. Previously she worked at IBM Research in Yorktown Heights, NY, in the Computer Science Department, where she specialized in the development of software for scientific visualization on parallel systems. She holds a Bachelor of Science in physics and chemistry from Whitworth College and a Masters of Science in computer science from the University of Colorado, Boulder.

Gerhard Theurich is an employee of Silicon Graphics Incorporated (Professional Services) and is currently part of the ESMF development team. His contributions to date focus on the low-level communications and execution layer of the ESMF. Before joining the ESMF development team, he was part of the applications team at the NASA Center for Computational Sciences (NCCS) at the Goddard Space Flight Center. His current interests include performance portability of parallel applications and multiparadigm parallel computing. He received a degree in Diplom Physik from the University of Erlangen, Germany, a Ph.D. in computational material science from the University of California, Santa Barbara, and worked as a postdoctoral associate at the University of Pennsylvania. During this time he developed and contributed to several electronic structure and quantum chemistry codes.

Cecelia DeLuca is currently the manager of the Earth System Modeling Framework implementation team at the NCAR. In this position, she is responsible for translating the needs of a broad scientific community and innovations in computer science into a robust infrastructure for climate and weather simulation. Previously she worked as a lead developer of a framework for real time signal processing at Massachusetts Institute of Technology Lincoln Laboratory. She is experienced with architecting large geophysical codes, assessing and selecting computational advances for use in production models, and using a systematic software process, including requirements collection. Ms. DeLuca holds a Masters of Science in engineering from Boston University and a Masters of Science in meteorology from Massachusetts Institute of Technology.

Max Suarez is a Meteorologist at Goddard Space Flight Center. He received a B.S. and M.E. in engineering science from the University of Florida, and an M.A. and Ph.D. in geophysical fluid dynamics from Princeton University. His interests include large-scale atmosphere/ ocean interactions, climate modeling, numerical methods, parametrization of subgrid-scale processes in atmospheric models, maintenance of the atmospheric general circulation, and climate sensitivity.

Atanas Trayanov is senior computational analyst at the Global Modeling and Assimilation Office, NASA Goddard Space Flight Center. He received an B.S and M.S. in physics in 1979 from Sofia University, Sofia, Bulgaria, and a Ph.D. in physics in 1988 from the Bulgarian Academy of Sciences, Sofia, Bulgaria. His interests include climate modeling, high performance computing, and parallel algorithms.

V. Balaji is the Head of the Modeling Systems Group at the NOAA Geophysical Fluid Dynamics Laboratory. He received a M.Sc. in physics from the Indian Institute of Technology and a Ph.D. in physics from Ohio State University. His interests include climate simulation, high performance computational systems and algorithms, and collaborative, standardized approaches to modeling.

Peggy Li received a B.S. and an M.S. in electrical engineering from the National Taiwan University, Taiwan, and a Ph.D. in computer science from the California Institute of Technology in 1986. She was the technical group supervisor for the Parallel and Distributed Application Technology Group (formally Automation and Scheduling Technology) in the Applications Development Section from 1996 to 1999. She has led several R&D tasks in the parallel and distributed computing area since she joined the Jet Propulsion Laboratory in 1989, including the development of a distributed discrete event simulation framework, a remote interactive visualization and analysis (RIVA) system for planetary data visualization. She is currently leading a parallel volume rendering project (ParVox) and a Fault Detection, Isolation, and Recovery (FDIR) Server task for TMOD Technology. Her research interests include scientific visualization, parallel rendering, and distributed computing.

Weiyu Yang is a research scientist in the NOAA NCEP. He received a B.S. and M.S. in atmospheric science from Peking University, China, and a Ph.D. in atmospheric science from the Institute of Atmospheric Physics, Chinese Academy of Sciences in 1989. He is the lead developer of the ESMF project at NCEP, developing the ESMF versions of the NCEP operational data assimilation systems, the global forecast model, and the ensemble forecast system. He designed the special parallel structure of the current NCEP operational data analysis system to obtain the best performance to fit the restrictive operational time requirement. Previously he worked at the Supercomputer Computational Research Institute, Florida State University, where he specialized in the development of the adjoint models of the NASA GEOS1 model and the NASA SISL model and researched on related data assimilation issues.

Chris Hill is a researcher in the Department of Earth, Atmospheric and Planetary Sciences at the Massachusetts Institute of Technology. His research is directed at applying advanced computing technology to the study of planetary scale circulation. He is particularly interested in how cutting edge technologies become effective research tools and how algorithms, hardware and software combine to make this possible and profitable. He received a B.S. in physics from Imperial College, London.

Arlindo da Silva is a Meteorologist in the Global Modeling and Assimilation Office at NASA Goddard Space Flight Center. He received a B.S. and M.S. in physics from Catholic University of Rio de Janeiro, Brazil and a Ph.D. in meteorology from the Massachusetts Institute of Technology. His current research interests include techniques for global atmospheric data assimilation, physical-space analysis systems, error covariance modeling, bias estimation and correction, quality control, land-surface, precipitation and aerosol data assimilation, and efficient methods for assimilation of remotely sensed data. Other research interests not in the area of data assimilation include aerosol forcing of climate, hydrological cycle of the subtropics, estimation of fluxes of heat, momentum and fresh water over the global oceans for observational studies and forcing ocean models.

NOTES

1 We took the set services terminology (along with many good design ideas) from the Common Component Architecture project.

2 ESMF does allow exceptions to this rule; a very fine grained component may be run in its parent's context (e.g. using the same VM). The child component in this case faces restrictions on its options for concurrency and resource use.

3 Times shown are reduced further in the special case in which a child component shares its parent's VM.

References

Adair, R. J., Bayles, R. U., Comeau, L. W., and Creasy, R. J. 1966. A virtual machine system for the 360/40. IBM Cambridge Scientific Center Report 320-2007, Cambridge, MA.

Boville, B. A. and Gent, P. R. 1998. The NCAR Climate System Model, Version One. *Journal of Climate* 11:1327–1341.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, W. S. 1994. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Network Parallel Computing*, MIT Press, Cambridge, MA.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., and da Silva, A. 2004. The architecture of the Earth System Modeling Framework. *Computing in Science and Engineering* 6(1): 18–28.

Parrish, D. F. and Derber, J. C. 1992. The National Meteorological Center's Spectral Statistical Interpolation Analysis System. *Monthly Weather Review* 120:1747–1763.