ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-171

A Toolkit for Building Earth System Models

by

Ian Foster

Mathematics and Computer Science Division

Technical Memorandum No. 171

March 1993

This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

A Toolkit for Building Earth System Models

Ian Foster

Abstract

An *earth system model* is a computer code designed to simulate the interrelated processes that determine the earth's weather and climate, such as atmospheric circulation, atmospheric physics, atmospheric chemistry, oceanic circulation, and biosphere. I propose a toolkit which would support a modular approach to the implementation of such models.

1 Motivation

An *earth system model* is a computer code designed to simulate the interrelated processes that determine the earth's weather and climate, such as atmospheric circulation, atmospheric physics, atmospheric chemistry, oceanic circulation, and biosphere. A scientist might use a diagram similar to Figure 1 to explain an earth system model. In this figure, boxes represent processes and arrows represent linkages between processes. This description is easy to follow. It hides unnecessary detail and makes the interfaces between components clear. These desirable characteristics, which have obvious value to the scientist, are also of value to the software engineer. In fact, they constitute the central attributes of modular or *object-oriented* design. Unfortunately, this natural modularity is normally lost when an earth system model is implemented as a computer program. The result is that it is difficult both to implement these models and to adapt them to changing requirements.

In this document, we propose an *object-oriented approach* to the implementation of earth system models and a *toolkit* that supports the use of this approach on sequential and parallel computers. These have the following features:

1. Process models written in sequential languages such as FORTRAN and C, or in parallel languages such as FORTRAN+NX, FORTRAN+PICL, C+p4, High Performance FORTRAN, and PCN, can be combined with little modification.

2. Code that performs generic functions such as I/O or the conversion of data between different grid systems can be encapsulated in reusable modules.

3. On a parallel computer, the mapping of computation to processors can be modified within a component without changing other parts of a program.

4. Programs produced using the toolkit are portable across many parallel and sequential computers.

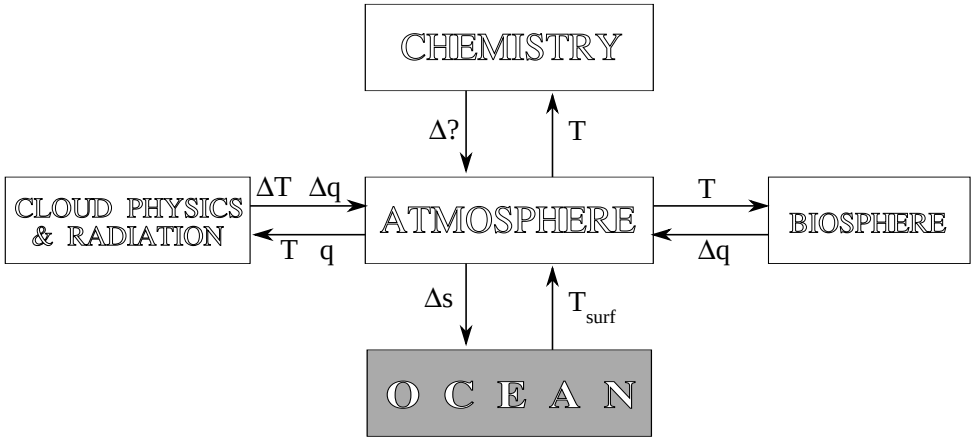

Figure 1: Simplified Schematic Description of Earth System Model

The first feature allows existing atmosphere, ocean models, etc., to be integrated, even if written in different programming languages. The second feature facilitates reuse and exchange of model components. The third feature simplifies the development of load balancing strategies.

2 Programming Concepts

Our modular or *object-oriented* approach to model development is based on four ideas: processes, ports, channels, and mappings. The *process* is the building block from with models are constructed. A process's interface with its environment is defined in terms of *ports*. An application is constructed by creating processes and connecting ports in these processes with *channels*. The programmer can also specify a *mapping* of processes to physical processors.

2.1 Processes

A process encapsulates data structures and the code that operates on those data structures. The code can be written in any programming language supported by the toolkit. If this is a parallel language, then the process may create subprocesses that perform internal communication. This internal communication is only visible within the process and cannot interfere with other communications.

An implementation of an earth system model will define one process for each box in a specification such as Figure 1, plus additional processes for I/O and data conversion functions.

2.2 Ports

A process's interface to its environment is defined in terms of ports. These come in two flavors: out-ports and in-ports; a process can send data on an out-port and receive information on an in-port. A port is similar in some respects to a I/O unit or file in

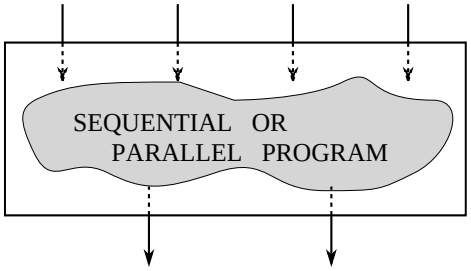

Figure 2: A Process and its Interface

Fortran or C; send and receive analogous to write and read operations. If a process is a parallel program, then its interface is likely to consist of an *array* of ports, with one port for each subprocess.

A process and its ports together define a reusable module. Figure 2 shows the programmer's view of a process. Four in-ports and two out-ports (represented as arrows) define the interface, while internal implementation details are hidden.

2.3 Channels

The programmer constructs a model by plugging together processes and creating channels to specify how these processes are to interact. A channel is a one-to-one communication link that connects an out-port and an in-port. It can be thought of as a first-in, first-out message queue, with send operations on the out-port adding messages to the queue, and receive operations on the in-port removing messages from the queue. A receive operation blocks if the queue is empty and resumes execution if and when a send operation adds to the queue.

The use of channels to connect processes is illustrated in Figure 3. At the top of the figure, an atmosphere model process, an ocean model process, and an interpolator process are shown, together with the ports that define their interfaces. At the bottom of the figure, the same processes are shown coupled together. As the interfaces between the processes consist of multiple channels, we can infer that the processes are probably parallel programs.

Because model components are represented as distinct processes and all inter-model interaction is constrained to occur via channels, it is easy to substitute components. For example, the ocean model in Figure 3 can be replaced with a program that reads sea surface temperature from a file: no changes to the interpolator or the atmosphere model are required. Similarly, an alternative interpolation algorithm can be substituted without changing the other two components.

2.4 Mapping

The programmer can provide a third piece of information when plugging processes together: where processes are to execute in a parallel computer or network. As a process can itself be a parallel program, the programmer can allocate a set of processors to a

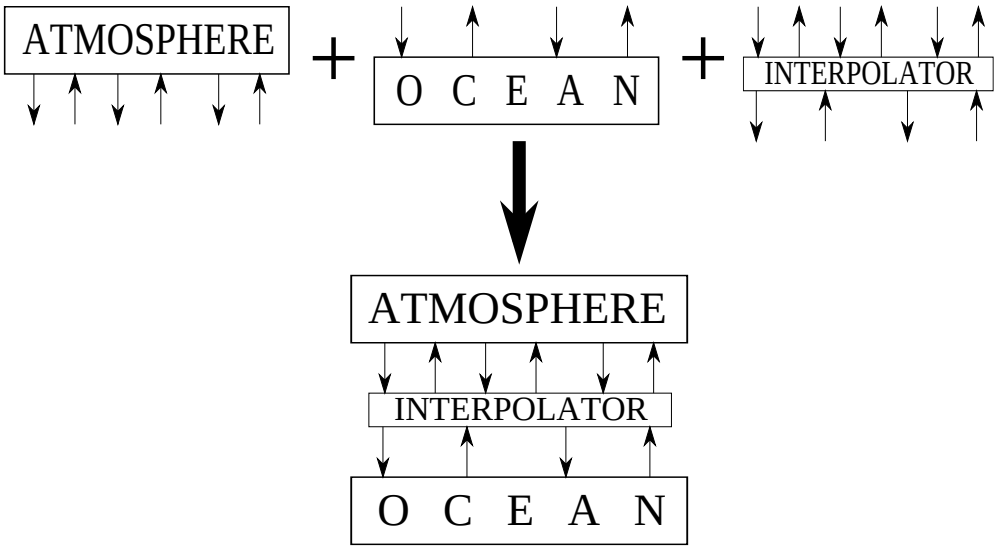

Figure 3: Process Coupling

particular process. This mapping information can have a profound impact on the performance of the program, but does not effect the result computed. Hence, mapping can be changed without changing other parts of a program. For example, Figure 4 shows a program comprising two processes (each a parallel program in its own right) mapped to a parallel computer, two parallel computers, and a workstation and a parallel computer.

3 Implementation

While the concepts outlined in Section 2 can be employed in any programming system, they are easier to use if supported by appropriate tools. In this section, we describe a toolkit that provides direct support for the concepts. This toolkit comprises the following components:

1. Compatability libraries that allow programs developed using standard sequential and parallel programming systems to be encapsulated as processes. Initially, we plan to target message-passing libraries such as NX, PICL, and p4. Support for High Performance FORTRAN (HPF) is to be added as HPF compilers become available.

2. A language for plugging together processes to form programs. Initially, we plan to use FORTRAN M for this purpose, as this provides syntax for representing processes, ports, channels, and mapping.

3.1 FORTRAN M

FORTRAN M is a small set of extensions to `Fortran` 77 that provides syntax for representing the programming concepts outlined in Section 2. Figure 5 shows a illustrative code fragment. The process `atmosphere` implements a simple atmosphere model that obtains

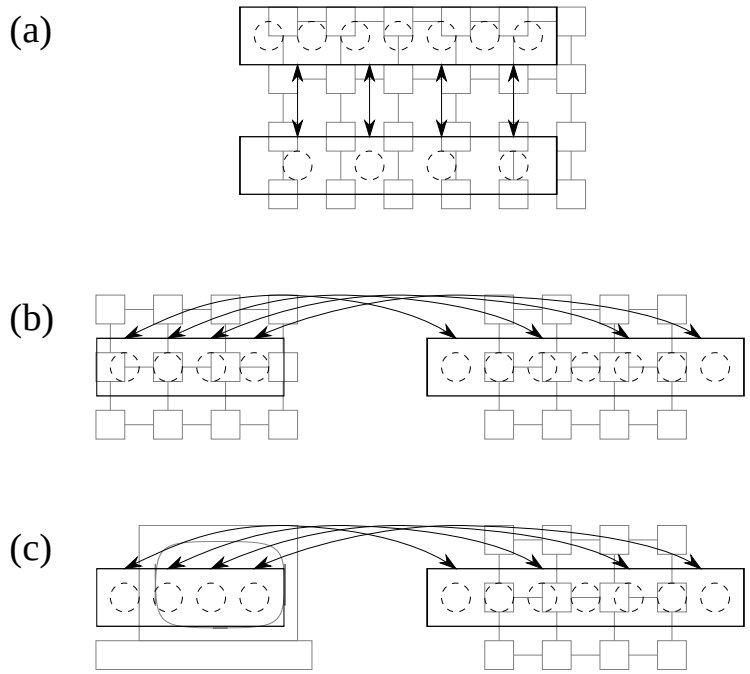

Figure 4: The same code can execute on (a) one parallel computer, (b) two parallel computers, and (c) workstation and parallel computer.

sea-surface temperature data on in-port `sst_i` and sends momentum data on out-port `uv_o`. The process `coupled_model` couples this process with an ocean model. A complete description of FORTRAN M is provided in a separate document [1].

3.2 Compatability Libraries

A compatability library allows a program written in a *foreign* parallel programming language such as FORTRAN+NX, FORTRAN+PICL, C+p4, PCN, or HPF to be invoked as a process in a FORTRAN M program. On distributed-memory parallel computers, programs written in these languages create one or more processes per physical processor; these processes communicate by calling low-level message-passing routines provided by the operating system.

A compatability library consists of two components: concurrency and interface. The *concurrency* component implements foreign language processes using the mechanisms used to implement FORTRAN M processes and organizes communication so that messages generated by the foreign computation are distinguishable from those generated by other parts of a computation. The *interface* component provides a mechanism by which a FORTRAN M program can invoke a foreign program, passing ports as arguments. It also provides the foreign language with routines that it can call to send and receive messages on these ports.

Hence, a compatability library comprises a component that is invisible to the programmer (a run-time library) and a component that is visible to the programmer (routines for sending and receiving messages on ports). In general, the modifications to a program required for it to execute in a coupled model consist primarily of the addition of the communication calls needed to send and receive interface data.

```
      process atmosphere(sst_i,uv_o)
      parameter(NLAT=128, NLON=256, TMAX=100)
C     The ports sst_i and uv_o are the external interface.
      inport (real x(NLAT,NLON)) sst_i
      outport (real x(NLAT,NLON), real y(NLAT,NLON)) uv_o
C     Process common variables.
      process common /atmo/ sst, u, v
      real sst(NLAT,NLON), u(NLAT,NLON), v(NLAT,NLON)
C     Repeat TMAX times: recv SST, update U & V, send U & V.
      do 10 i=1,TMAX
        send(uv_o) u,v
        receive(sst_i) sst
        call atm_compute
10    continue
C     Signal end of communication.
      endchannel(uv_o)
      end

      process coupled_model
      parameter(NLAT=128, NLON=256)
C     Local port variables.
      inport (real x(NLAT,NLON)) ssti
      outport (real x(NLAT,NLON)) ssto
      inport (real x(NLAT,NLON), real y(NLAT,NLON)) uvi
      outport (real x(NLAT,NLON), real y(NLAT,NLON)) uvo
C     Create channels and define ports.
      channel(out=ssto,in=ssti)
      channel(out=uvo,in=uvo)
C     Call two models with ports as arguments.
      processes
        call atmosphere(ssti,uvo)
        call ocean(uvi,ssto)
      endprocesses
      end
```

Figure 5: FORTRAN M Programming Example

3.3 Performance Issues

The use of the toolkit can introduce overhead that would not be incurred if a code were implemented as a monolithic program. Primary sources of overhead are additional copying due to the use of channels for data transfer between modules and process switching when multiple modules execute on the same processor. Although these costs must clearly be carefully evaluated, they must be weighed against the benefits of modular design, ease of modification, ease of reuse, and portability. We are currently conducting experiments to quantify these costs.

4 Status

A prototype FORTRAN M compiler for uniprocessors and shared-memory multiprocessors is to be available from Argonne National Laboratory in November 1992 and a compiler for distributed-memory parallel computers is scheduled for release soon after. These compilers use source-to-source transformations to implement the FORTRAN M extensions and operating system facilities to implement concurrent threads and message-passing.

The success of this toolkit depends to a large extent on the ease with which compatability libraries can be developed. As a first experiment, we propose to develop a compatability library for applications developed using Intel FORTRAN+NX. This work is to begin in early 1993.

References

[1] Foster, I., and Chandy, K. M., Fortran M: A language for modular parallel programming, Preprint, Argonne National Laboratory, Argonne, IL 60439, 1992.