FORTRAN M AS A LANGUAGE FOR BUILDING EARTH SYSTEM MODELS*

Ian Foster

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

1. Introduction

FORTRAN M is a small set of extensions to FORTRAN 77 that supports a modular or object-oriented approach to the development of parallel programs (Foster and Chandy, 1992). In this paper, I discuss the use of FORTRAN M as a tool for building earth system models on massively parallel computers. I hypothesize that the use of FORTRAN M has software engineering advantages and outline experiments that we are conducting to investigate this hypothesis.

2. Earth System Models

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. On massively parallel computers, extensive reengineering can be required to combine component models or to experiment with alternative mappings of computation to processors. The result is that it is difficult both to implement these models and

*This research was supported by the Atmospheric and Climate Research Division of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

to adapt them to changing requirements.

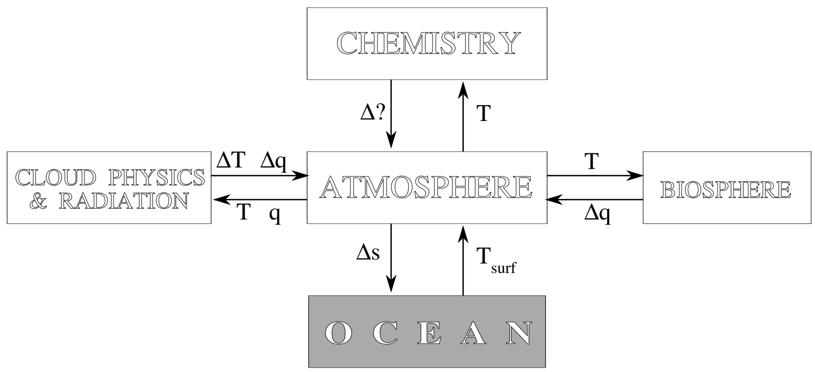

Figure 1: Simplified schematic description of an earth system model. Boxes represent component models and arrows represent data transfers between components.

3. Fortran M

Fortran M is a small set of extensions to Fortran 77 that supports a modular or object-oriented approach to the design of message-passing parallel programs. The following features of this language appear useful when developing earth system models.

Modularity. Programs are constructed by using explicitly-declared communication channels to plug together program modules called processes. A process can encapsulate common data (called `PROCESS COMMON` to emphasize that it is local to the process), subprocesses, and internal communication. Processes do not share data and can communicate only by sending and receiving data on channels. Access to channels is provided via ports that are passed to processes as arguments. Hence, a Fortran M implementation of an earth system model can preserve the modularity that is inherent in its scientific specification. Each component can be encapsulated as a module (Figure 2) and alternative implementations can be substituted without modifications to other program components.

Safety. Operations on channels are restricted so as to guarantee deterministic execution, even when programs execute on many processors. Channels are typed, so a compiler can check for correct usage. Hence, the risk of incorrect (and particularly time-dependent) errors is reduced.

Architecture Independence. The mapping of processes to processors can be specified with respect to a virtual computer with size and shape different

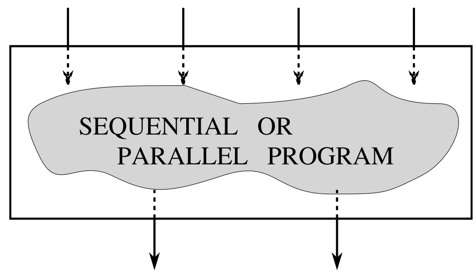

Figure 2: A process and its interface. In this example, four in-ports and two out-ports (represented as arrows) define the interface, while internal implementation details are hidden. These internal details can include subprocesses and internal communications.

from that of the target computer. Mapping is specified by annotations that influence performance but not correctness. Hence, the programmer need not change component programs in order to experiment with alternative configurations of processors and computers.

4. Building Models

The Fortran M implementation of a simplified ocean/atmosphere model is presented for illustrative purposes. In this model, two components execute concurrently and exchange information periodically: An ocean model provides an atmosphere model with an array of sea surface temperatures (SST), and an atmosphere model provides an ocean model with two arrays containing components of horizontal momentum, U and V. The two models are assumed to utilize the same grid system, numerical units, and time steps; hence, data can be transferred between them without additional computation. Normally, two additional components would be required to convert data between the representations used by the two models.

4.1 *Sequential Models*

The Fortran M program implements both models as processes, and defines an interface that allows for the exchange of SST, U, and V values. Initially, the two models are assumed to be sequential programs; hence, the interface can consist of one channel in each direction. For example, the atmospheric model interface consists of two ports, `sst_i` and `uv_o`, and is defined as follows. Notice the type declarations for the ports: The in-port `sst_i` can be used to receive arrays of real values representing sea surface temperatures, while the out-port `uv_o` can be used to send two such arrays representing U and V values.

```
process atmosphere(sst_i,uv_o)
parameter(NLAT=128,NLON=256)
inport  (real x(NLAT,NLON)) sst_i
outport (real x(NLAT,NLON), real y(NLAT,NLON)) uv_o
```

The following code shows the atmosphere model process in its entirety. It uses `SEND` and `RECEIVE` statements to repeatedly send U and V data on the port `uv_o`, receive SST data from the port `sst_i`, and call the atmosphere model proper. A total of `TMAX` messages are sent and received. Note the use of process common to hold the `sst`, `u`, and `v` arrays.

```
      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 /state/ sst, u, v
      real sst(NLAT,NLON), u(NLAT,NLON), v(NLAT,NLON)
      call atm_init
C     Repeat TMAX times: send U & V, recv SST, update U & V.
      do 10 i=1,TMAX
        send(uv_o) u,v
        receive(sst_i) sst
        call atm_compute
10    continue
      end
```

The ocean model might be as follows. This repeatedly sends SST data on the out-port `sst_o` and receives U and V data on the in-port `uv_i`.

```
      process ocean(uv_i,sst_o)
      parameter(NLAT=128, NLON=256, TMAX=100)
C     The ports uv_i and sst_o are the external interface.
      inport (real x(NLAT,NLON), real y(NLAT,NLON)) uv_i
      outport (real x(NLAT,NLON)) sst_o
C     Process common variables.
      process common /state/ sst, u, v
      real sst(NLAT,NLON), u(NLAT,NLON), v(NLAT,NLON)
      call ocn_init
```

```
C     Repeat TMAX times: send SST, recv U & V, compute SST.
      do 10 i=1,TMAX
        send(sst_o) sst
        receive(uv_i) u,v
        call ocn_compute
10    continue
      end
```

The implementation of the ocean/atmosphere model is completed by a main program that invokes the two processes in a parallel block (delineated by `PROCESSES` and `ENDPROCESSES` statements) and creates two channels, one for communicating SST values and the other for communicating U and V values. This structure is illustrated in Figure 3 and is created by the following program. The program creates two channels, spawns the `atmosphere` and `ocean` processes, and waits until these processes terminate.

```
      program 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=uvi)
C     Call two models with ports as arguments.
      processes
        call atmosphere(ssti,uvo)
        call ocean(uvi,ssto)
      endprocesses
      end
```

The values of the four port variables declared in this code fragment are initially undefined. The `CHANNEL` statements each create a channel and define their two port variable arguments to be references to this channel. These port variables are passed as arguments to the concurrently executing `atmosphere` and `ocean` processes, establishing the connections shown in Figure 3.

We now have a complete parallel program which can be executed on a sequential or parallel computer. This program can be executed on one processor or two without any change to its component modules: these different

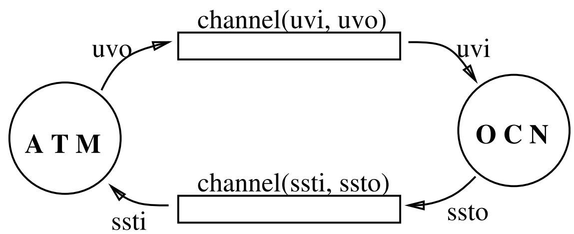

Figure 3: Ocean/atmosphere model. Two concurrently executing processes are connected by two single-producer, single-consumer channels.

behaviors are specified by annotations to the process calls in the main program. The execution order of the concurrently executing `atmosphere` and `ocean` processes is determined only by availability of messages on channels. The computed result does not depend on the order in which the processes execute. That is, the program is deterministic.

4.2 *Parallel Models*

The example in the preceding section shows how Fortran M is used to combine sequential models. Similar techniques can be used to combine parallel models. A parallel model implemented in Fortran M creates many processes (typically one per processor); these execute in parallel and exchange data by channels that are local to the parallel model. When defining an interface between two such models, it is not in general sufficient to utilize a single channel as this is likely to constitute a bottleneck. Instead, we define a parallel interface consisting of an array of channels. For example, assume that we have parallel versions of our ocean and atmosphere models, each designed to execute on NP×NP processors. We decompose two channels used to connect the sequential programs to obtain two arrays of NP×NP channels. While a channel in the sequential program was used to communicate arrays of size NLAT×NLON, each channel in the parallel program is used to communicate arrays of size (NLAT/NP)×(NLON/NP). The code used to combine the two parallel models is as follows.

```
program model
parameter(NLAT=128,NLON=256,NP=16,PLON=NLON/NP,PLAT=NLAT/NP)
inport (real x(PLAT,PLON)) SstI(NP,NP)
outport (real x(PLAT,PLON)) SstO(NP,NP)
inport (real x(PLAT,PLON), real y(PLAT,PLON)) UvI(NP,NP)
outport (real x(PLAT,PLON), real y(PLAT,PLON)) UvO(NP,NP)
```

```
      ...
C     Create NP×NP channels.
      do 10 i=1,NP
        do 11 j=1,NP
          channel(out=SstO(i,j),in=SstI(i,j))
          channel(out=UvO(i,j),in=UvI(i,j))
11      continue
10    continue
      ...
C     Pass port arrays to parallel models.
      processes
        call par_atmosphere(SstI,UvO)
        call par_ocean(SstO,UvI)
      endprocesses
      end
```

5. Discussion

As the simple examples presented in this paper show, the use of Fortran M allows earth system models to be developed in a modular fashion. I hypothesize that this modularity can reduce software engineering costs in at least four ways. First, existing programs can be integrated into earth system models without the need for extensive reengineering. Second, alternative implementations of components can be substituted without changes to other parts of a system. Third, reusable modules can be defined that implement such commonly-used functions as interpolation and parallel I/O. Fourth, models can be adapted to run on different computers or to use different processor configurations, without changes to the component programs.

These hypothetical benefits must be confirmed by empirical investigations in larger systems. These investigations need to focus on issues of compatibility, performance, and modularity. I discuss these issues briefly here and outline how we are addressing them.

Compatibility. Development of an earth system model would be easy if all parallel codes that were to be integrated into an earth system model were written in Fortran M. However, in practice we must be able to deal with codes developed with other technologies. In the short term, these will be primarily message-passing libraries such as p4/PARMACS, PICL, Express, and PVM. In the medium term, High Performance Fortran is also likely to be important. As it will not in general be feasible to rewrite message-passing or HPF programs in FORTRAN M, we plan to develop *compatability libraries* that allow message-passing and HPF programs to be integrated into FORTRAN M programs in a seamless fashion.

Performance. A program developed in the modular fashion advocated in this paper may incur overheads that would not be incurred if it were implemented as a monolithic program. Primary sources of overhead are additional copying due to the use of channels for data transfer between processes and process switching when multiple processes execute on the same processor. These costs must be carefully evaluated and weighed against the benefits of modular design, ease of modification, ease of reuse, and portability. We plan to conduct experiments to quantify these costs.

Modularity. The modular programming style encouraged by FORTRAN M is expected to allow the development of generic, reusable modules for such commonly used functions as interpolation between grid systems and parallel I/O. However, it is possible that the complexities of different grid systems, data representations, etc., will frustrate such attempts. In order to investigate this issue, we are developing prototype parallel versions of generic data transfer modules developed at GFDL by Ron Pacanowski and his colleagues.

We have recently completed implementation of a prototype FORTRAN M compiler (send electronic mail to `fortran-m@mcs.anl.gov` for details) which we plan to use for these investigations.

Acknowledgments

FORTRAN M is joint work with Mani Chandy of the California Institute of Technology and is supported in part by the National Science Foundation's Center for Research in Parallel Computation, under Contract CCR-8809615. Robert Olson has made major contributions to the development of the compiler. I am grateful to Ron Pacanowski of GFDL for discussions on modular construction of climate models.

References

I. Foster and K. M. Chandy, 1992: FORTRAN M: A Language for Modular Parallel Programming, Preprint MCS-P327-0992, Argonne National Laboratory, Argonne, Ill.