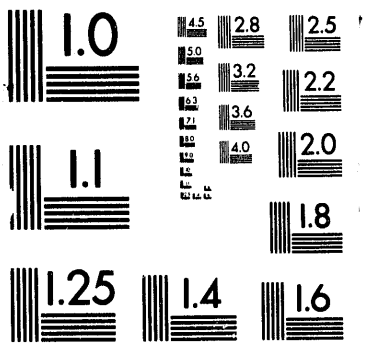
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

1 of 1

Conf-9306285--1

LA-UR- 93-4224

Title: Global Atmospheric And Ocean Modeling On The Connection Machine

Author(s): Susan R. Atlas
Thinking Machines Corpo ation

Submitted to: F.X. Le Dimet, ed., Proceedings of the NATO ASI Workshop in High Performance Computing in the Geosciences

GLOBAL ATMOSPHERIC AND OCEAN MODELING ON THE CONNECTION MACHINE

SUSAN R. ATLAS*
Thinking Machines Corporation
245 First St.
Cambridge, MA 02142 USA

ABSTRACT. This paper describes the high-level architecture of two parallel global climate models: an atmospheric model based on the Geophysical Fluid Dynamics Laboratory (GFDL) SKYHI model, and an ocean model descended from the Bryan-Cox-Semtner ocean general circulation model. These parallel models are being developed as part of a long-term research collaboration between Los Alamos National Laboratory (LANL) and the GFDL. The goal of this collaboration is to develop parallel global climate models which are modular in structure, portable across a wide variety of machine architectures and programming paradigms, and provide an appropriate starting point for a fully coupled model. Several design considerations have emerged as central to achieving these goals. These include the expression of the models in terms of mathematical primitives such as stencil operators, to facilitate performance optimization on different computational platforms; the isolation of communication from computation to allow flexible implementation of a single code under message-passing or data parallel programming paradigms; and judicious memory management to achieve modularity without memory explosion costs.

1. Introduction

As an enterprise of inherently international scope, the long-term, high-resolution simulation of the Earth's global climate presents unique opportunities for large-scale collaboration and research. The prospect of imminent availability of raw processing power in the 100 GFlops range, combined with rapid and continuing refinements in the models themselves, led to the initiation of the U. S. Department of Energy CHAMMP (Computer Hardware, Advanced Mathematics, and Model Physics) Climate Modeling Program in 1990 [1]. The mission of this program is to develop new algorithms and numerical techniques to take advantage of the increasing computational power provided by a new generation of (parallel) computer architectures, and thus to facilitate the development of integrated models that incorporate the effects of such complex processes as atmospheric photochemistry, land vegetation, and atmosphere/ocean/sea-ice interactions.

---

* The work described in this paper is the result of a collaboration between scientists at Los Alamos National Laboratory (LANL), the Geophysical Fluid Dynamics Laboratory (GFDL/NOAA, Princeton University), Thinking Machines Corporation (TMC), and Cray Research, Inc. (CRI), under the auspices of the U. S. Department of Energy CHAMMP (Computer Hardware, Advanced Mathematics, and Model Physics) Climate Modeling Program, directed by R. C. Malone (LANL). The collaboration consists of J. K. Dukowicz, P. W. Jones, S. Kortas, R. C. Malone, and R. D. Smith (LANL); C. H. Goldberg and R. Hemler (GFDL); S. R. Atlas and K.-S. Cho (TMC); and C. L. Kerr (CRI).

For the past several years, the Climate Modeling Group at Los Alamos National Laboratory (LANL) has been engaged in a research collaboration with the Geophysical Fluid Dynamics Laboratory of the National Oceanic and Atmospheric Administration (GFDL/NOAA), under the auspices of CHAMMP, to develop massively parallel versions of two of the GFDL's ocean and atmospheric modeling codes. The initial target for the parallel versions of these codes has been the Connection Machine installed at LANL's Advanced Computing Laboratory (ACL)– initially a 64k CM-200, which was recently upgraded to a 1024 node CM-5. Since the CM-5 can be programmed to run in either message passing or data parallel mode [2], it has been possible to address the issue of portability concurrently with the design of the parallel models themselves. The eventual goal is to produce full-scale models which can run on virtually any platform, ranging from a serial, standalone workstation, to a multitasked Cray YMP/8, to a loosely-coupled cluster of workstations running PVM, to a massively parallel machine such as the CM-5 running High Performance Fortran (HPF)– *all without significant sacrifice of performance on any individual platform.* This is particularly important for implementations running on the largest parallel machines, such as the CM-5, since these will be used to perform very long, high-resolution experiments, probing the limits of climate predictability on centuries-long timescales.

Another driving force in this work, particularly with regard to the atmospheric model, has been the development of modular codes, to facilitate the interchange of individual components such as radiation packages or dynamics solvers within the models. In this way, different implementations (grid vs. spectral; alternate physical parametrizations) can be compared in a controlled fashion, or provided to researchers as standalone modules.

The focus of this paper will be the high-level architecture of the ocean and atmosphere models and their implementation on the CM-5. We begin with a history and brief description of the atmospheric and ocean models in Section 2, followed by an overview of the CM-5 architecture in Section 3. The design of the parallel models is described in Section 4, with examples to illustrate the accomodation of various communication, memory management, and programming paradigms. We conclude with a discussion of future directions in Section 5.

2. Two Models

2.1 SKYHI

SKYHI is a general circulation model of the atmosphere, developed at GFDL [3]-[6]. It is a global hydrostatic model with 40 levels extending to 80 kilometers, utilizing a rectangular latitude/longitude grid in the horizontal, and a sigma coordinate system in the vertical. Fourier filtering is used to eliminate high-frequency features near the poles. The model integrates the primitive equations for the prognostic variables, including wind velocity, temperature, water vapor mixing ratio, and (optional) user-selected tracers. Radiative heating, an input to the primitive equations, is re-calculated every fixed number of timesteps, as specified by the user. Following each time step, diagnostic variables are calculated and optionally written out to storage.

The process of converting SKYHI to parallel form began in 1990, with the development of an array-syntax version of the original 40,000 line Fortran 77 code. This new version of SKYHI consisted of 20,000 lines of Fortran 90 which could be run on GFDL's Cray YMP/8 following the application of a pre-processor [7] that converted the Fortran 90 back into Fortran 77 for the Cray compiler. At that point, the code was still structured in terms of array "chunks", in which prognostic variables were integrated in successive grid slices, each three latitudes wide. This was

necessary because of the limited amount of main memory on the Cray. In November, 1992, work began at LANL and GFDL to further evolve SKYHI into a true parallel model. An "in-core" model was developed, full conversion to array syntax was completed, and HPF-style layout directives were added to specify domain decomposition across processors. The new, parallel version of SKYHI is presently undergoing performance optimization and verification on the CM-5 at LANL and on the YMP/8 at GFDL. The first numerical experiments using SKYHI on the CM-5 will begin in early 1994.

2.2 PARALLEL OCEAN PROGRAM (POP)

POP is a global ocean model presently running in production on the CM-5 at LANL. It was originally developed as a CM-2 port [8] of the Semtner-Chervin ocean model [9,10]. The Semtner-Chervin model is a descendant of the Bryan-Cox model [11], variants of which form the basis for many ocean modeling programs in use today, including the MOM model from GFDL. As detailed in reference [8], the port of the Semtner-Chervin model to the CM-2 required a significant re-write, since, as in SKYHI, the data structures used in the Cray implementation were ill-suited to the efficient use of a massively parallel architecture. The authors of the port (Smith et al.) also found that the performance of POP on the CM-2 was severely limited by the barotropic solver implementation in the original code. They subsequently developed a reformulation of the solver in terms of surface-pressure (rather than the streamfunction) which not only parallelized better on the CM-2 but also made it possible to include an arbitrary number of islands in the model without excessive computational cost [12,13]. For further details, see [8].

Many of the programming features which promote modularity and portability, such as isolating communication within stencil routines (see Section 4.3) were first developed in POP. The model now runs on a variety of architectures, including a workstation cluster running PVM, Cray vector computers, and several massively parallel supercomputers, including the CM-5. As part of the LANL/GFDL collaboration, the capabilities of the model are presently being expanded to include the complete range of diagnostics and user interface available in MOM.

3. CM-5 Architecture

The CM-5 system [2] consists of a set of 16 to 16,384 processing nodes (PNs) with associated memory. Each PN contains a SPARC microprocessor, 32 to 128 MB of memory, four vector execution/memory interface units, and a network interface. Peak processing power is 32 MFlops per vector unit, for a peak of 128 MFlops/PN. The processing nodes are coordinated by a control processor (CP), essentially a SPARC microprocessor. The CM-5 can be divided (by the system administrator) into groups of processors called partitions. Each partition is under the control of its own CP, which runs an extended form of Unix known as CMOST. The CP is responsible for coordinating and timesharing the processes in its partition. The PNs and CPs are connected to two scalable networks, the Data Network and the Control Network, both of which are based on a hyper-tree topology. The Data Network supports point-to-point communication between processors, for example SENDs and RECEIVEs in MIMD-style message passing. The structure of the Data Network avoids bandwidth contention between partitions, and the design guarantees a minimum network bandwidth of 5 MB/sec per node, regardless of destination. The Control Network supports communication patterns that involve all processors in a single operation; for

example, broadcasts, synchronization, and error signaling. The Control Network also provides hardware support for cooperative arithmetic operations such as parallel prefix and reduction.

Parallel I/O to the CM-5 is provided by the Scalable Disk Array (SDA), a high performance, scalable, RAID 3 disk storage system. Like the PNs, the SDA disk storage nodes are connected to the CM-5 Data Network through a network interface. Each SDA module consists of three disk storage nodes, and provides 25 Gbytes of storage and 33 MB/s real application bandwidth. Storage capacity and bandwidth scale linearly with the number of modules. Data can be transferred to and from the SDA *in parallel* using standard Fortran READs and WRITEs, and SDA files are both UNIX compatible and NFS mountable.

4. Code Design

4.1 'GENERIC' PARALLEL COMPUTATION

At the simplest level, a parallel computer can be viewed as a collection of processing nodes, or PN's, with associated memory, coupled by a high-performance network. For purposes of high-level code design, the actual PN architecture and network topology are largely irrelevant. The fundamental question which determines the structure of a given applications program is the *domain decomposition*: the distribution of the data across the processors which are to operate upon it. Once this crucial decision has been made, it is possible to implement any given algorithm, across a wide variety of machines, using a single high-level program.

There is a fundamental reality of parallel computing which makes this statement true. This is the fact that all parallel machines in existence today, including the CM-5, are communcation- rather than computation-limited. Thus, there is no case in which it is desirable to structure the decomposition of a problem to avoid computation by shuffling data within the machine.

Once the domain decomposition is fixed, it is the responsibility of the programmer to ensure that the results of the computation that each PN has performed on its data are available to any other processors that might require it during the next round of computation. The way in which this is done is architecture-dependent, and is determined by the degree of control that the user wishes to retain over communication. This choice, in turn, is guided by the structure of the memory hierarchy on the machine, and the flexibility of the programming languages and compilers which manage that memory within an application.

As a concrete example, consider a distributed-memory parallel computer such as the CM-5. There are two programming styles that can be used to program the CM-5: data parallel and message passing. In data parallel programming, Fortran 90 array syntax [14] is used to expose the inherent parallelism of a data set. For example, in both SKYHI and POP, it forces the programmer to deal with the full latitude/longitude/depth grid as a single data structure, rather than as individual longitude/depth slabs (as in the original Cray Fortran 77 versions). However, Fortran 90 syntax says nothing about the manner in which data is to be laid out across processors on an actual machine. On the CM-5, this *domain decomposition* is accomplished through the addition of HPF-style directives [15], which give the user considerable control over the precise arrangement of data on each PN. For example, consider an array of the form A(i,j,k), where (i,j) labels a point on the latitude/longitude grid, and k is the vertical coordinate. This is a common construct in SKYHI. Using CM Fortran LAYOUT directives (DISTRIBUTE directives in HPF) one can specify that the (i,j) axes be spread across all the processors on the machine, and that the k axis be located entirely in-processor. This is written as:

```
      real, array(imax,jmax,kmax) ::  A     ! F90 array syntax
cmf$ layout A(:news, :news, :serial)        ! domain decomposition
```

When this array is used in calculations involving other arrays of the same size and LAYOUT (such arrays are said to be "conformable"), operations involving variables that correspond to the same grid point but different vertical index values (common in the radiation package, for example) will involve no communication between processors. Note that there is no need for the programmer to have any *a priori* knowledge about the size of the machine on which the code is eventually to run; the actual data distribution occurs at run time, and information on the location of each grid point is stored in the *geometry* associated with the array. During compilation, the LAYOUT information is used to insert low-level communication routines into the object code wherever the compiler detects the possibility of off-processor data motion. Depending on the fine-grainedness of the control exercised by the programmer, there may be instances where the compiler inserts communication calls "just to be safe", but in fact, information is really only being moved within the memory of the processor itself. One way of avoiding this unnecessary "communication" is to assert full control over the data distribution, and insert inter-processor communication calls from within the high-level program itself. This can can be done using a message-passing programming model. Another way is to use detailed memory management, a form of local array equivalencing in CM Fortran. In general, data parallel codes tend to be faster to write, easier to maintain, and more straightforward to understand than message-passing code (the correspondence between code and equations is more transparent), but to achieve performance comparable to a tailored message-passing code, careful attention must be paid to memory management and the minimization of unnecessary communication.

4.2 PROGRAMMING MODELS

As mentioned in the Introduction, an important part of this work was the design of a high-level structure for the two models which could provide a unified framework to support a range of programming styles. Since the CM-5 supports both message passing and data parallel programming paradigms, the availability of such a framework incidentally provides the interesting opportunity to make a direct comparison between two different implementations of the same code on the same machine (this work is in progress for POP). While such a comparison will undoubtedly provide useful information, it is important to emphasize that the main reason for undertaking the development of a unified framework is to allow a single version of each model to be run on a wide variety of platforms, under a range of compilers.

An important early decision in the development of this framework was the adoption of Fortran 90 as the "maintenance language" for both models. This language has recently become an international standard, intended as the eventual replacement for Fortran 77. Until Fortran 90 compilers become widely available on serial workstations and vector supercomputers, however, a pre-compiler can be used to transform Fortran 90 syntax into Fortran 77.

Another design decision was the adoption of HPF syntax to describe the distribution of data across processing nodes. Since HPF is not yet a standard, a mechanism for supporting vendor-specific versions of these directives had to be developed. We have utilized the simple expedient of a "definitions file" #included in each subroutine; this file contains vendor-specific #ifdefs which are

turned on or off for appropriate pre-processing under cpp (C pre-processor) in an architecture-specific makefile. Here is the generic form of the CM-5 layout example from Section 4.1:

File define.h:

```
#ifdef distributed_memory
   #define space_              ! blank space requires special handling
   #define hpf_distribute cmf$ space_ layout
   #define block_  :news
   #define star_   :serial
#endif
```

SKYHI subroutine:

```
   subroutine example
#include "define.h"
   real, array(imax,jmax,kmax) :: A       ! F90 array syntax
#ifdef distributed_memory
   hpf_distribute A(block_, block_ ,star_) ! HPF domain decomposition
#endif
   return
   end
```

The actual "multi-purpose programming structure" [16] for POP and SKYHI was developed by R. C. Malone and C. L. Kerr at GFDL during the summer of 1991 [16,17]. It utilizes self-similar domain decomposition, coupled with encapsulated communication. A summary of the approach is as follows. (Note: for simplicity, we shall ignore the vertical coordinate in this discussion; as described in Section 4.1, in atmospheric and ocean modeling codes this coordinate is naturally "serial", i.e. in-processor. The domain decomposition problem is thus reduced to a consideration of the distribution of data with respect to latitude/longitude coordinates). Consider a grid *NX* x *NY* in extent. Define a finer level of granularity, a *subgrid* of the larger grid, dimensioned *nx* x *ny*, and let *nx = NX / xparts; ny = NY / yparts.* Let the code be structured so that all mathematical operations involving communication (in POP and SKYHI these are either stencil operations corresponding to differential operators, or global sums to compute diagnostics) are encapsulated into subroutines that are dimensioned at the size of the subgrid. Use makefiles to ensure that the appropriate (depending on the programming model) versions of each communication-based subroutine are linked in to the rest of the program (including a manual domain-decomposition subroutine invoked at the start of the program in the case of message-passing). Then, as detailed in [17], it is possible, with this high-level structure, to support both data parallel and message-passing implementations on distributed memory machines, as well as "worksharing" [18] on distributed or shared memory architectures. The implementation is determined by the combination of communications routines and the values selected for *xparts* and *yparts*. For example, consider the case of a distributed-memory parallel machine such as the CM-5. When *xparts*=*yparts*=1, the "subgrid" extends over the entire global domain; the code is an ordinary data

parallel program, and the corresponding communications-based subroutines are expressed in terms of Fortran 90 constructs such as CSHIFTs. The actual communication is handled implicitly by the CM-5 run-time system. When *xparts* and *yparts* are > 1, the subgrids are local to the processing nodes, and the communications-based subroutines employ MIMD-style SENDs and RECEIVEs. In this latter case, the availability of Fortran 90-to-Fortran 77 preprocessors makes it possible to support message-passing architectures other than the CM-5 (eg., the Intel Paragon) for which node-level Fortran 90 compilers are not yet available.

4.3 MATHEMATICAL PRIMITIVES

In optimizing the data parallel version of POP on the CM-5, immediately following its migration from the CM-200 (late 1992), it was determined that stencil operations on the CM-5 were occupying a fairly large percentage of the total execution time. The two main stencils involved were a nine point stencil in the barotropic solver, and a five point stencil in the baroclinic solver. At very large subgrid ratios (~ 2048) [19], these operators were responsible for over 33% of the run time; at a moderate subgrid size of 128 (a size more typical of anticipated production runs of POP at LANL), this figure was roughly 72%. A detailed analysis of these results [20] showed that by optimizing communication at the level of the run-time-system (RTS), performance could be improved significantly. This was accomplished by adding generalized support within the RTS to cache geometry information, making it possible for the CM-5 to perform the arithmetic component of the stencil operations with a minimal amount of communications overhead.

In addition, since POP had been written in such a way as to isolate the communications component of the code (i.e. the stencils), it seemed natural to try to optimize their performance directly, at the assembly-language level. A summary of results obtained using these hand-coded stencils (written by E. D. Dahl of Thinking Machines Corporation) illustrates the improvement that was obtained, particularly at small subgrids (Table 1):

Table 1. Five point stencil performance on the CM-5.
"Stencil 1" denotes stencils coded using data parallel Fortran/RTS without geometry caching. "Stencil 2" denotes hand-coded assembly language stencils. All stencil times in seconds, double precision calculations, 20 vertical levels.

Grid	Partition size (PNs)	Subgrid ratio	Stencil 1	Stencil 2
256 x 128	1024	8	3.172	0.605
128 x 64	128	16	3.165	0.681
256 x 128	512	16	3.220	0.700
256 x 128	256	32	3.278	0.776
512 x 256	1024	32	3.367	0.801
256 x 128	128	64	3.453	0.993
512 x 256	512	64	3.485	1.014
512 x 256	256	128	4.085	1.231
1024 x 512	1024	128	3.962	1.237
512 x 256	128	256	5.425	1.731
1024 x 512	512	256	5.250	1.827

Note that asymptotically, timings are determined entirely by the subgrid ratio; i.e., two different grids run on two different machines will execute in roughly the same time if the subgrids for the two problems are the same. At small subgrid sizes, the stencil time is dominated by the start-up

time (latency) for off-node communication; as the subgrid increases, the startup time is amortized over the time required to actually move the data. (N.B. The version of the RTS which caches geometry information yields similar results).

In addition to illustrating the isolation of communication from computation (Section 4.1) in a parallel climate code, the use of stencils in POP serves to demonstrate the value of defining high-level mathematical primitives in scientific applications [21]. In this context, "high-level primitives" refer to mathematical operators with associated physical significance– something beyond the level of a simple Ax + B operation, for example. Programming in terms of such primitives makes sense not only because it produces clearer code, but also because it provides a natural bridge between applications scientists and software engineers. Language and compiler development for parallel machines presents significant challenges, and identification of the "natural" primitives arising in real scientific applications provides valuable feedback which can influence the future directions of compiler design.

4.4 MEMORY MANAGEMENT

As indicated in the Introduction, it is difficult to separate issues of memory management from communication when dealing with parallel machines. The relative magnitudes of memory access times within the memory hierarchy of a given machine lead to architecture-dependent compiler tradeoffs, typically within the context of communication-related constructs. These hidden tradeoffs are inevitable, and they represent the next major challenge in proceeding with the development of efficient, unified, parallel versions of the two global climate models described in this paper. In Section 4.2 it was shown that encapsulated communication provided a solution for dealing with the very different communication protocols of the data-parallel and message-passing programming paradigms. It also made it possible to focus on performance issues within a small, carefully-circumscribed domain. However, it now seems clear that the unifying framework of Section 4.1 will need to be extended to address the more complex issues associated with efficient memory management across a range of computational platforms.

As a simple example, consider the following tradeoff between the new "serial optimizations" in Version 2.0 of the (data parallel) CM Fortran compiler, and the memory explosion which can occur when the additional arrays required to implement these optimizations are introduced. In the new version of the compiler, array section assignments along serial dimensions are recognized as being in-processor, and generate local memory references instead of RTS communication calls. This makes it possible to write code that allows the compiler to vectorize along both parallel (subgrid) and serial axes at the same time, leading to significant performance enhancements. The tradeoff is memory, as illustrated in the following hypothetical fragment from an atmospheric model radiation package:

Version 1 (2D arrays, no serial optimization):

```
      real, array(imax, jmax) :: A, B      ! latitude/longitude grid
cmf$ layout A(:news, :news), B(:news, :news)
      do k=2,kmax                          ! unrolled DO loop over serial axis
        call physics_1(A,k-1)
        call physics_2(B,k)
        B= B+A
```

```
    enddo

Version 2 (3D arrays, serial vectorization):

real, array(imax, jmax, kmax) :: C, D
cmf$ layout C(:news, :news, :serial), D(:news, :news, :serial)
call physics_1_new(C)
call physics_2_new(D)
D(:,:,2:kmax)= D(:,:,2:kmax)+ C(:, :, 1:kmax-1)
```

Version 2 is able to take full advantage of the serial optimizations in the compiler (reflected in the use of array syntax for the third (serial) coordinate), but the resulting performance enhancement comes at the cost of factor of 2 x kmax in storage. Constructs of this form are very common in SKYHI, and if the full range of serial optimizations is implemented, the result is an explosion in memory usage. This limits the resolution of the models which can be run, affecting both the quality of the numerical experiments which can be performed, as well as the performance, since, on the CM-5, the larger the subgrid, the better the performance of the vector units.

In the early years of parallel programming, much of the discussion relating to different machine architectures tended to center on questions of syntax and the standardization of message-passing protocols. While these issues have not entirely been laid to rest, resolving dilemmas such as the one presented in the previous paragraph will require significant effort in another arena, managing the complex layers of memory, and modes of access to these layers, in the new generation of machines. Concepts such as worksharing, array aliasing, and detailed layouts represent serious attempts by compiler designers to grapple with these issues. For example, a recent extension of data parallel programming, the *global/local* model, allows programmers to take advantage of the data parallel programing style when it suits them, with the additional freedom to drop down to a local (MIMD) level in the middle of their global program. This global/local cycle can be repeated as many times as desired. While it is possible to think of this programming style in terms of an interplay between single and multiple threads of control, it can also be viewed as an extension of the programmer's freedom to manage memory at a fine-grained level. Research is presently underway to investigate the application of new approaches such as these to the memory explosion/optimization tradeoff in SKYHI.

5. Conclusion

It is hoped that the examples and discussion presented in this paper have served to illustrate some of the interesting and complex questions associated with the design of large-scale atmospheric and ocean models for parallel architectures such as the CM-5. At the heart of these questions lies the fundamental issue of communication, whether at the message-passing/run-time level, or the more subtle level implicit in the access to the memory hierarchy of a given machine. Future work will involve trying to find a unified way to manage memory effectively across a range of computer architectures. This will require negotiating the idiosyncracies of hardware and compiler design at a much deeper level than was required for the development of the stencil interface described here. While computational design issues such as these are by no means unique to the field of global climate modeling, the numerical, scientific, and practical challenges presented by the development

of global climate models has proved a particularly exciting and productive crucible for our exploration of new conceptual frameworks in parallel program design.

Acknowledgements

I would like to thank Bob Malone and Stephen Boyd for careful readings of this manuscript, and for helpful comments. I would also like to thank the members of the GFDL/LANL collaboration for providing a stimulating and enjoyable atmosphere in which to pursue this work.

References

[1] *Building an Advanced Climate Model,* Program Plan for the CHAMMP Climate Modeling Program, U. S. Department of Energy, Washington, DC 20585 (October, 1990).

[2] *Connection Machine CM-5 Technical Summary*, Thinking Machines Corporation, Cambridge, MA (November 1992).

[3] J. L. Holloway, Jr., and S. Manabe, *Simulation of climate by a global general circulation model: I. Hydrologic cycle and heat balance,* Mon. Wea. Rev. **99**, 335 (1971).

[4] Y. Kurihara and J. L. Holloway, Jr., *Numerical Integration of a nine-level global primitive equations model formulated by the box method,* Mon. Wea. Rev. **95**, 509 (1967).

[5] S. B. Fels, J. D. Mahlman, M. D. Schwarzkopf, and R. W. Sinclair, *Stratospheric sensitivity to perturbations in ozone and carbon dioxide: radiative and dynamical response,* J. Atmos. Sci. **37**, 2265 (1980).

[6] *SKYHI,* in: *Redesign of Research Model Code Sol 52-SAAA-3-00052*, U. S. Department of Commerce/NOAA, Washington D.C. (January, 1993).

[7] C. H. Goldberg and C. L. Kerr, *Development of the Geophysical Fluid Dynamics Laboratory's Climate Models for Scalable High Performance Computer Systems,* Proposal to the Compter Hardware, Advanced Mathematics, and Model Physics Project (January, 1993).

[8] R. D. Smith, J. K. Dukowicz, and R. C. Malone, *Parallel ocean general circulation modeling*, Physica D **60**, 38 (1992).

[9] A. Semtner Jr., *Finite-difference forumulation of a world ocean model,* in: Advanced Physical Oceanographic Numerical Modeling, ed. J. J. O'Brien, (Riedel, Dordrecht, 1986) p. 187.

[10] R. M. Chervin and A. J. Semtner Jr., *An ocean modeling system for supercomputer architectures of the 1990s,* in: Proc. of the NATO Advanced Research Workshop on Climate-Ocean Interaction, ed. M. Schlesinger (Kluwer, Dordrecht, 1988) p. 87.

[11] K. Bryan, *A numerical method for the study of the circulation of the world ocean,* J. Comput. Phys. **4**, 347 (1969).

[12] J. K. Dukowicz, R. D. Smith, and R. C. Malone, *A reformulation and implementation of the Bryan-Cox-Semtner ocean model on the Connection Machine,* J. Atmos. Ocean. Tech. **10**, 195 (1993).

[13] J. K. Dukowicz and R. D. Smith, *Implicit free-surface method for the Bryan-Cox-Semtner ocean model,* J. Geophys. Res., accepted.

[14] M. Metcalf and J. Reid, *Fortran 90 Explained,* (Oxford University Press, New York, 1992).

[15] *High Performance Fortran Language Specification, Version 1.0,* Scientific Programming **2**, no. 1 (June, 1993).

[16] R. C. Malone, personal communication (July 1991).

[17] J. Schwarzmeier and C. Kerr, *Issues associated with domain decomposition for climate models,* unpublished (April 1993).

[18] "Worksharing" refers to an F77-based programming style in which domain decomposition is effected through array scoping attributes ("private", "shared").

[19] The subgrid ratio of a data parallel array executing on a partition of the CM-5 is defined as the product of the extents of all NEWS-type axes (axes which are not in-processor, but which are spread across the machine), divided by the number of vector units in the partition (recall that there are four vector units per PN).

[20] S. R. Atlas, *Performance analysis of the LANL OCEAN code*, unpublished (January 1993).

[21] B. M. Boghosian, *Computational Physics on the Connection Machine,* Computers in Physics **4**, 14 (1990).

DATE
FILMED
2/9/94
END