In **Proc. 1992 ACM International Conference on Supercomputing (ICS'92)**, Washington, DC, pp. 397-408, July 1992.

Experiments with an Ocean Circulation Model on CEDAR

L. DeRose, K. Gallivan and E. Gallopoulos

Feb. 1992

CSRD Report No. 1200

Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801

Experiments with an Ocean Circulation Model on CEDAR

L. DeRose, K. Gallivan, E. Gallopoulos
Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
U.S.A.

Abstract

We present the design of the GFDL ocean circulation model as adapted for simulations of the Mediterranean basin for the Cedar multicluster architecture. The model simulates the basic aspects of large-scale, baroclinic ocean circulation, including treatment of irregular bottom topography. The data and computational mapping strategies and their effect on the design are discussed. The code was parameterized to offer several choices for data partitionings of the computational domain, for placement strategies for the data in the memory hierarchy, and for the number of clusters and processors used in the computational hierarchy of Cedar. The experiments and performance trends are discussed. Using four clusters and 32 processors the code demonstrates significant speedup compared to a single cluster and compared to a single processor.

1 Introduction

The numerical modeling of ocean circulation is a task of great importance, both in its own right, as well as a component of climate studies. Numerical techniques are based on approximating the three-dimensional conservation equations for momentum, mass, salinity and temperature in order to understand the evolution of these fields with time. Due to the interaction between the atmosphere and the oceans, developing adequate predictive skill for the sea surface temperature is an important step towards better climate studies. This fact has been recognized with the inclusion of computational ocean circulation as a Grand Challenge of the High Performance Computing and Communications Initiative Program [14]. Two major computational difficulties for conducting ocean circulation simulations are the necessary fine spatial resolution - orders of magnitude more detailed than what is used in computational simulations today to conduct meaningful forecasts, and the requirement to perform long term simulations for meaningful climatic studies. These requirements make numerical ocean circulation an activity of high computational complexity in both its time and space requirements. Therefore, it is not surprising that realistic global and detailed regional ocean circulation models (OCMs) were made feasible only recently.

The purpose of this paper is to discuss the parallelization of numerical ocean circulation for the Cedar architecture. Major goals in our investigation have been i) to explore Cedar's potential for grand challenge applications, ii) to show that parameterizing the coding for Cedar's hierarchical memory/computation paradigm so that it can be used for future studies in mapping this code to other emerging multiprocessor architectures. This work will also provide the framework for testing the ability of the hierarchical paradigm to sustain performance as resources and problem size scale and for understanding the behavior of the paradigm for a finite difference model with an important explicit component.

The ocean general circulation model used in this work is based on a basic model of the Geophysical Fluid Dynamics Laboratory (GFDL) [8], as adapted in the Istituto per lo Studio delle Metodologie Geofisiche Ambientali (IMGA-CNR) to the Mediterranean basin geometry [23]. This model, from now on referred to as GFDL-IMGA, simulates the basic aspects of large-scale, baroclinic ocean circulation, including treatment of irregular bottom topography. It is used in climate studies and also to study the development of mid–ocean eddies. Temperature, salinity and the prediction of currents are the main physical phenomena of interest. As most of the major OCM to date, this model is based on the primitive equations model designed by Bryan and Cox [4][8].

The structure of this paper is as follows: Section 2 contains remarks on this and previous work. Section 3 presents a brief description of the equations and the original code design. A complete mathematical formulation of the model can be found in [4][8][10]. An overview of the Cedar architecture, and the design and implementation of the code on Cedar are described in Section 4. Section 5 discusses experiments

and results. Finally our conclusions and remarks on future plans are presented in Section 6.

2 Remarks on previous work

Major reorganizations of the original model of Bryan and Cox were due to Cox [8], directed toward vector machines with long start-up times, and Semtner [24], for register-to-register vector architectures such as the Cray-1. See also [7] [25] [26].

Recent attention has focussed toward the exploitation of parallelism. We note for example [1][2] for the Cray 2, with major emphasis on the use of efficient solvers for the two-dimensional mass transport stream function; the award-winning work of [27], on a modified version of [24] using microtasking and vector processing, to achieve impressive performance, on a 4-CPU Cray X-MP. The replacement of the relaxation step for the stream function with a parallelized pre-conditioned conjugate gradient solution of a pressure equation as described in [28] led to an important improvement of the performance on the CM-2 Connection Machine.

It is also noted that that the Department of Energy's CHAMMP project [5] and the recent release of the Modular Ocean Model from GFDL [22] are expected to contribute to the effort to develop parallel ocean models [19].

Performance studies of the GFDL-IMGA model on the Alliant FX/8 were conducted in [11]. Preliminary results on the Cedar multiprocessor are described in [12].

3 Description of the model and original code design

The mathematical model uses the Navier-Stokes equations with three basic assumptions: Boussinesq approximation, in which density differences are neglected, except in the buoyancy term, hydrostatic assumption, where local acceleration and other terms of equal order are eliminated from the equation of vertical motion, and turbulent viscosity hypothesis, in which stresses exerted by scales of motion too small to be resolved by the grid are represented as enhanced molecular mixing. Temperature and salinity are calculated using conservation equations, and the equations are linked by a simplified equation of state.

3.1 Model equations

The equations are written in the spherical coordinate system (λ , ϕ , z), with λ denoting longitude and ϕ latitude, only that z denotes depth, defined as negative downward from the surface $z = 0$.

The model equations are:

$$\frac{\partial \vec{u}_h}{\partial t} + \vec{u} \cdot \nabla \vec{u}_h + \vec{f} \times \vec{u}_h = -\frac{1}{\rho_0}\vec{\nabla} p + A_h \nabla^2 \vec{u}_h + \left(A_v \vec{u}_{hz}\right)_z , \quad (1)$$

$$p_z = -\rho g , \quad (2)$$

$$\nabla \cdot \vec{u} = 0 , \quad (3)$$

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T = K_h \nabla^2 T + \left(K_v T_z\right)_z , \quad (4)$$

$$\frac{\partial S}{\partial t} + \vec{u} \cdot \nabla S = K_h \nabla^2 S + \left(K_v S_z\right)_z , \quad (5)$$

$$\rho = \rho(T, S, p), \quad (6)$$

where $\vec{u} = (u, v, w)$ is the velocity vector, $\vec{u}_h$ its horizontal components, p, ρ are pressure and density, $\vec{f} = 2\Omega \sin\phi \vec{k}$, T, S are the temperature and salinity tracers, and $A_{h,v}$, $K_{h,v}$ are the turbulent diffusion coefficients.

The boundary conditions for momentum and tracer fluxes at the ocean surface ($z = 0$) are:

$$\rho_0 A_v (u_{h,z} \;\; v_{h,z}) = \vec{\tau} , \quad (7)$$

$$K_v \frac{\partial}{\partial z}(T, S) = 0 , \quad (8)$$

$$w = 0, \quad (9)$$

where $\vec{\tau}$ is a seasonal wind stress. At the bottom $z = -H(\lambda, \phi)$, the vertical velocity w is set to be:

$$w = -\vec{u}_h \cdot \nabla H . \quad (10)$$

At the side wall boundaries, the normal and tangential horizontal velocities, and the horizontal fluxes of sensible temperature and salinity are set to zero.

To eliminate fast moving gravity waves and thus avoid the ensuing severe time step restrictions OCMs employ a rigid-lid approximation to the equations. Manipulating the equations under this approximation, the vertically averaged velocity components of momentum $(\overline{u}, \overline{v})$ can be written in terms of a two-dimensional stream function ψ:

$$\overline{u} = -\frac{1}{Ha}\frac{\partial \psi}{\partial \phi} , \quad (11)$$

$$\overline{v} = \frac{1}{Ha\cos\phi}\frac{\partial \psi}{\partial \lambda} . \quad (12)$$

Using the incompressibility condition Eq. 3, a Poisson equation is derived for the time derivative of the stream function ψ. Momentum is then expressed as the sum of the external mode $(\overline{u}, \overline{v})$ and internal mode. Since internal and external modes of momentum can be computed separately, the overall computation can be split into an explicit *baroclinic* phase, approximating three dimensional fields of tracers and deviations of horizontal velocity from the vertically averaged flow, and an implicit *barotropic* phase, requiring the solution of a Poisson equation for the transport stream function in the horizontal (λ, ϕ) direction.

3.2 Original code design

Derivative terms are approximated with second order finite differences in space and time on an Arakawa B staggered grid (u, v and T, S, ψ respectively defined at different points of the computational cell). Discretization is uniform in the horizontal direction and non-uniform in the vertical. Explicit leapfrog with occasional mixing with a forward Euler step is used for time stepping. Hence, data from three time steps are required during the computation of any grid point, because the leapfrog method uses data from the current and the previous time steps to predict values for the future time step. We refer to [8] for the main aspects of the implementation of the code on the Cyber 205 at the Geophysical Fluid Dynamics Laboratory, that was the basic model used in this work.

The explicit nature of the baroclinic phase allows the computation to be applied on sequences of two dimensional (λ, z) data slabs. This approach was favored in ocean modeling since it only required the storage of two dimensional data in fast memory [3]. Since the GFDL-IMGA model was based on the model of [8], computations in the baroclinic phase are performed over both ocean and land areas using masks to distinguish ocean from land cells.

The program expects as input initial and boundary conditions, along with topography and wind information. It then proceeds as follows. First, several initializations are performed, then the time step loop is entered. This consists of further initializations, the update of the internal modes of momentum, vorticity driving function, temperature, salinity, densities and performs a convective adjustment of water columns if needed. This is followed by the update of the mass transport stream function using the vorticity driving function. Wrap-up time step computations are performed, and, if desired, current results are printed. If that was the last time step, wrap-up computations for the entire program are performed, else the time step is repeated.

During the computation of prognostic variables in any grid cell, the leapfrog method reads and writes data from three time steps, which, consequently, should be present in memory. As no data from other time steps is required, a three-stage buffering method is used. The buffers contain three dimensional fields of temperature, salinity, and velocity components u and v, as well as two dimensional topography and wind stress data. Seven slabs are needed to compute second order approximations to the horizontal derivatives at any point of a given slab, namely three slabs from the current time step, three from the previous time step and one slab with the values currently being computed. Two additional buffers are needed for computing the mass transport stream function, but these are shared with the space used for the baroclinic phase.

4 Implementation on Cedar

4.1 Cedar system

The Cedar system was developed at the Center for Supercomputing Research and Development of the University of Illinois. Its main characteristic are the hierarchical organization of its computational capabilities and memory system. Cedar is a multicluster-based architecture with four clusters, where each cluster is a modified Alliant FX/8 machine with 8 computational elements (CEs). Three levels of parallelism

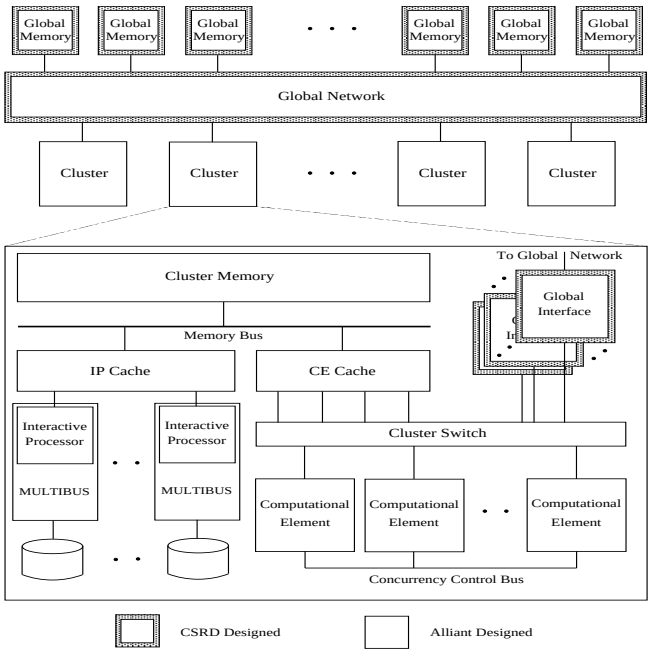

Figure 1: Cedar architecture

can be applied. First, vectorization can be used within a CE. Second, small grain parallelism can be exploited by using concurrency within the cluster. Finally, medium and large grain parallelism can be used across the clusters.

There are four levels of memory system hierarchy, namely registers for each CE, cache for each cluster, cluster memory and global memory. The cluster cache provides a maximum bandwidth of 47 MW/sec, for a total of 188 MW/sec for the four clusters. As expected, the cost of access increases at each level. The observed relative performance degradation of the memory hierarchy is a factor of 2, from cache to cluster memory, a factor of 6 from cache to global memory, and a factor of 2 from cache to global memory, when the prefetch system is used. Each CE has its own set of scalar and vector registers. The CEs in each cluster share the cache and cluster memory, while the global memory is shared by the CEs of all clusters. Each cluster has 64 Mbytes of cluster memory and 512 Kbytes of cache. The global memory size is 64 Mbytes (32 memory banks). Cedar utilizes

pipelining in the global memory system and allows multiple outstanding memory requests from each CE. Since each Alliant processor has a limit on the maximum number of outstanding read requests, *data prefetching* is used to to fully exploit the memory system pipeline. This lets processors start block moves from global memory and continue execution regardless of how many read requests are outstanding. Prefetching can be disabled via a compiler flag. The processors and the global memory are connected via the global interconnection network, that consists of two unidirectional packet-switched networks. These switching networks are 2-stage Omega networks built from 8×8 cross-bar switches [18][21]. The Cedar operating system, Xylem, extends the Alliant Concentrix operating system to include multitasking and virtual memory management of the Cedar memory hierarchy (see [13]). A Xylem process consists of one or more independently scheduled program segments, called clustertasks, that execute asynchronously across the Cedar system, as directed by Xylem system calls. Programs are mostly written in Cedar Fortran [17, 16], a language resembling Fortran 77, with vector constructs such as those proposed for the Fortran 90 standard and has extensions for memory allocation, concurrency control, multitasking and synchronization. It allows the specification of the location and visibility of the data (private, shared, cluster or global), reflecting the Xylem memory access and locality structure. Concurrent execution of loops within a single cluster, or across clusters are provided with DOALL and DOACROSS constructs. Three groups of synchronization routines are provided: DOACROSS loop synchronization; Zhu-Yew synchronization primitives (see [29]); and Cray-Style synchronization operations (see [9]).

4.2 Design of the parallel code

The work described in this paper will focus in the baroclinic phase, the primary motivation being the fast convergence of the SOR algorithm, a peculiarity of the domain and data sets of interest here. The Cedar multicluster code was designed to allow experiments with several parameterized partitionings of the computational domain. Section 5 presents performance results from experiments where the computational resources (e.g. number of clusters, number of processors per cluster), data partition and placement strategies vary. We next describe the data partitioning and placement strategies used in the Cedar code.

4.2.1 Data partitioning

We already mentioned that memory considerations led to the organization of data into (λ, z) slabs. This strategy was also beneficial for vector architectures with long start-up times since it allowed the generation of long vectors by reshaping two dimensional grid arrays in the (λ, z) direction to one dimensional vectors.

Several partitioning strategies of grid points of the computational domain to resources can be used. The models discussed in [19] demonstrate the importance of the partitioning strategy for the parallel implementation. For Cedar, we consider partitionings to be of two types, one primary, taking into account data partitioning across clusters, the other secondary, specifying the partitioning across vector processors in each cluster. Hence secondary partitionings were applied on slabs or their subdivisions. The implemented partitionings are depicted in Figures 2, 3, and 4 for the case of four clusters. The first two partitionings, referred to as *row* and *column* are one dimensional as they cut across a single (ϕ and λ respectively) grid dimension. The third partitioning is *two dimensional*, as it cuts across two dimensions of the grid.

We see that in all cases, each cluster is responsible for a set of two-dimensional slab pieces. The reasoning behind the partitioning strategies used in this paper was based on the following design principles:

1. The slab organization was to be preserved and partitionings for multiple cluster parallelism were to be built as extensions of the slabs.

2. The partitioning of data and parallelization of computation in each cluster would mostly rely on automatic restructuring performed by available preprocessors (KAP and VAST).

We thus used relatively mature automatic restructuring tools for each cluster, while the more difficult high-level partitioning is being done manually. We note that another goal of this project is to provide information to compiler designers about the effectiveness of several multi-cluster restructuring strategies.

The computational domain consists of a parallelepiped of $I \times J \times K$ computational cells, with I, J and K cells in the λ, ϕ and z directions respectively, at the centers and sides of which momentum and tracer variables are defined. For our experiments we set the parameter controlling the number of partitions to be equal to the number of clusters. A feature of the slab organization is that the restructuring techniques resemble those used for the original code when running on a single cluster. In most cases this means that each cluster executes concurrently in the z direction and in vector mode in the λ direction.

In **row partitioning**, (Fig. 2), the grid is cut in the ϕ direction, so that J slabs are partitioned in C blocks of contiguous slabs. Each block, containing $\lceil J/C \rceil + 2$ slabs, is assigned to one of the C clusters. The two extra slabs contain data for the South and North borders of the partition. This data is necessary for the the second-order approximation of the derivatives, when the first and last rows of the partition are being updated.

As mentioned earlier, each cluster needs seven full slabs in its work space. Hence this scheme will become less at-

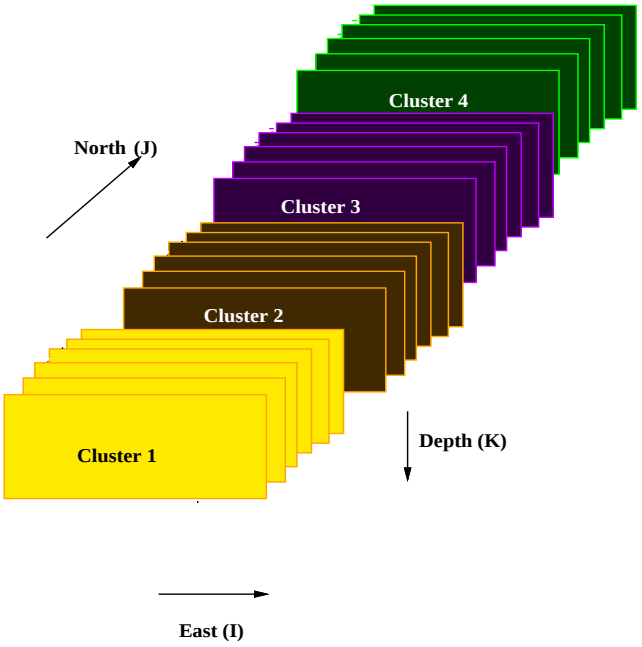

Figure 2: Representation of the grids with row partitioning.

tractive for models with high resolution, because work space cannot be distributed amongst the clusters. One solution to this problem is to use **column partitioning** (Fig. 3) for which the grid is cut in the λ direction. The I points are distributed among the clusters, while two extra columns are provided for the East and West borders. During each time step, the slab-by-slab computation of each subdivision of the basin is executed independently by each cluster. One advantage of this scheme over the row partitioning is that each of the C clusters will work with approximately $1/C$ of the work space necessary to be present in memory for the computation of each slab piece. Hence the work space is distributed between clusters, demanding less memory and obtaining more data locality.

The **two-dimensional partitioning**, (Fig. 4), is a combination of the two previous methods. In the four cluster case, the two horizontal dimensions are divided equally and each quadrant is assigned to a different cluster. Two-dimensional partitioning can be used when four or more clusters are available. This scheme is used as a solution for the problem of memory versus granularity which was mentioned earlier. Note that the vector lengths do not become as small as in column partitioning as the number of clusters increases, nor are the memory requirements as large as in row partitioning. This scheme is also flexible in that it can be extended to handle more arbitrary sections of data.

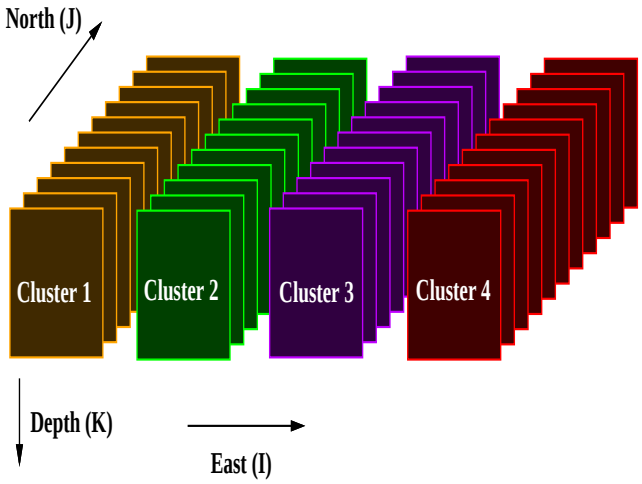

Figure 3: Representation of the grids with column partitioning.

4.2.2 Data placement

The multi-level memory hierarchy allows several data placement options. We distinguish two types of data, namely work space, which is two dimensional by virtue of the slab organization, and virtual disk space, containing the entire set of three dimensional fields. Four data placements are possible, referred to as "CC", "GC", "GG" or "CG", where the first letter denotes where the virtual disk is stored, the second letter denotes the location of the work space and "C" and "G" specify cluster and global memory respectively. Each of these data placement choices corresponds to a different view of Cedar. In particular, a "flat" shared memory multiprocessor paradigm, close to the Cray architecture, corresponds to the GG placement of virtual disk and work space in global memory and using multitasking or some form of microtasking. On the other extreme, the CC placement of disk and work space across cluster memories corresponds to a distributed memory paradigm, where each node is a cluster of processors, with global memory used as a communication buffer.

In our implementation, the CC version also uses global memory to store scalar quantities such as energy, temperature, salinity, etc, that are partially computed by each cluster. As expected, although explicit message passing is not used in the CC version, there is the need to communicate data through the slower global memory.

In between lies the hierarchical placement GC, which uses both cluster and global memory. As will be seen from our experiments, this is the that best exploits Cedar's memory system, using the fast local memory for frequently accessible work space and the global memory for the three dimensional virtual disk space. An analogy can be made between this hierarchical memory usage on Cedar and the use of the SSD

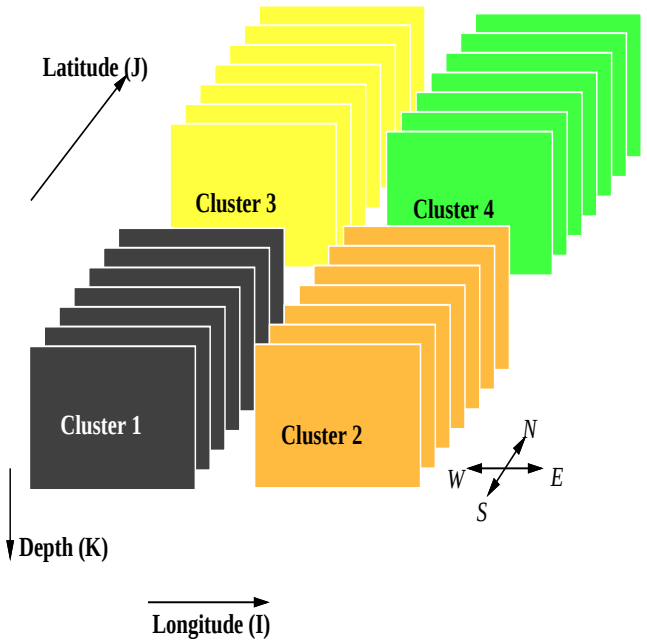

Figure 4: Representation of the grids with two-dimensional partitioning.

on the Cray Y/MP system.

To achieve good performance it is clearly desirable to maximize the ratio between floating point operations using cluster memory data, and global memory accesses. Note that the GC version during the baroclinic phase, for each point in the three-dimensional grid requires approximately 12 global memory accesses (8 reads and 4 writes) while almost 300 floating point operations are executed. Hence, the three-dimensional phase achieves a ratio of 25 floating point operations using data in cluster memory for each global memory access.

4.3 Remarks on the design of the parallel code

The Cedar multicluster program was parameterized to run with several clusters and partitioning schemes, as well as different number of horizontal grid points, vertical levels and islands. The partitionings of Section 4.2.1 were implemented by creating one Cedar Fortran task per partition and statically mapping it on a cluster. Within each cluster, loops were executed with dynamic self-scheduling. Data placement of work and virtual disk space data to global and cluster memory in the GG, CC and GC configurations was easily achieved using Cedar Fortran memory attributes.

A partial parallelization of the relaxation phase was achieved by writing the relaxation phase as a combination of one dimensional steps and observing that one of these steps is readily parallelizable while the other step is a recurrence which is solved with a fast lower bidiagonal solver [6][10]. Nevertheless, Krylov subspace type algorithms are likely to be more efficient for general OCM, as can be testified by the work in [2] as well as by the solvers used in the MOM code [22]. The fast convergence of the relaxation step caused the computation of the necessary matrix coefficients to consume an important amount of time relative to the solution step [10]. Hence the time-independent part of the matrix construction was moved outside the time step loop.

We next summarize the steps of the parallelization strategy.

1. The performance of the original code for several dimensions of the test data and several compilation options using the VAST restructuring compiler was obtained on an Alliant FX/8 (which is the basic cluster unit of Cedar). These experiments showed that the computational bottleneck was the baroclinic phase.

2. Data and control were partitioned to introduce parallelization. See [10] for further details. Corresponding to the flexibility of Cedar, the partitioning strategies of Section 4.2.1 were investigated.

3. Routine loop parallelization and vectorization within single tasks were handled by the KAP and VAST restructuring compilers.

4. The Poisson solver was restructured for greater parallelism.

5 Results

In this section we present the performance of the Cedar multicluster ocean simulation code and compare them with the original code optimized with the VAST restructurer and Alliant compiler. The data sets used in the experiment simulate the Mediterranean basin. They are denoted by $P_n L_k$ where $1/n$ is the horizontal resolution in degrees and k is the number of vertical levels. In particular:

$P_8 L_{16}$ This model uses grid spacing of 0.125° (approximately 13.87 km). The grid size is 334×118 in the horizontal direction, and 16 levels in the vertical direction. Nine islands are represented with this resolution and each time step simulates one hour.

$P_4 L_8$ This model uses grid spacing of 0.25° (approximately 27.75 km). The grid size is 167×57 in the horizontal direction and 8 levels in the vertical direction. Five islands are represented with this resolution and each time step simulated three hours.

Although memory limitations forced us to use 32 bit precision, the results were in good agreement with those obtained from 64 bit precision. All times reported for Cedar correspond to wall clock time obtained from the high-resolution

library routines `hrcget` and `hrcdelta` (see [20]), in single user mode. Results on the Alliant FX/8 are CPU times collected in single user mode, using the Alliant library routine `etime`. All these routines have 10 μsec accuracy. Timing results are in *seconds per timestep*; these were derived by running 12 (resp. 16) time steps and averaging the last 10 (resp. 14) for model P_8L_{16} (resp. P_4L_8). By ignoring the first two time steps we eliminated the effect of startup overhead in the performance, since this is expected to be minimal in long term simulations. Timings were obtained by executing several runs for each version of the code and each data set, while varying the number of clusters and CEs per cluster. Experiments were also run for the GG and GC versions of the code to simulate slowdown of global memory access time by explicitly disabling the prefetch unit. Indeed, unless stated otherwise, the prefetch unit will be on.

To account for the hierarchical nature of Cedar, in the following discussion we first introduce some notation for four different instances of speedup. Considering $T_v(C,p)$ the run time of version "v", using C clusters and p CEs per cluster, we define the *overall*, *multi-CE*, and *multicluster* speedups for C clusters and p CEs per cluster using version "v" respectively as:

$$S_v^o(C,p) = \frac{T_v(1,1)}{T_v(C,p)}\ , \tag{13}$$

$$S_v^{ce}(C,p) = \frac{T_v(C,1)}{T_v(C,p)}\ , \tag{14}$$

$$S_v^{cl}(C,p) = \frac{T_v(1,p)}{T_v(C,p)}\ . \tag{15}$$

and the *true* speedup for p CEs as

$$S_p = \frac{t_1'}{t_p'}\ . \tag{16}$$

where t_1' is the best execution time using one CE, and t_p' is the best execution time using p CEs.

Similarly, we define the overall, multi-CE, and multicluster efficiency ($E_v^o(C,p)$, $E_v^{ce}(C,p)$, $E_v^{cl}(C,p)$) as the correspondent speedup divided respectively by the total number of CEs, number of CEs in one cluster, and number of clusters.

We next present and discuss results for both models. Table 1 has the results for the original code compiled using VAST running model P_4L_8 and P_8L_{16} on the Alliant FX/8. We used scalar optimization, concurrency, vectorization, and associativity transformations as VAST optimization options.

For model P_4L_8, the results for the GC version with the prefetching unit turned on and off are in Tables 2 and 3 respectively. The results for the CC version are presented in Table 4, and the results for the GG version, with and without prefetch are presented in Tables 5 and 6. For model P_8L_{16}, Tables 7 and 8 have the results for the GC version with and without prefetch. The results for the CC version

CEs	P_4L_8	P_8L_{16}
1	10.7	92.1
2	6.0	52.1
4	3.7	32.2
8	2.6	23.3

Table 1: Original program, average runtime per time step.

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	12.9	6.7	6.8	4.9	4.9	3.8	4.1	3.7
2	6.7	3.5	3.6	2.6	2.6	2.0	2.2	2.0
4	3.7	1.9	2.0	1.5	1.5	1.2	1.3	1.1
8	2.2	1.2	1.2	0.9	1.0	0.8	0.9	0.8

Table 2: Average time in sec per timestep, model P_4L_8 - GC version with prefetch.

are presented in Table 9, and the results for the GG version, with and without prefetch are presented in Tables 10 and 11.

As expected, the use of global memory to store the three-dimensional data and of the cluster memory to store the work space best exploited Cedar's hierarchical memory, giving the best performance among the data placement strategies we tried. The best performance using the model P_4L_8 (see Table 2) averages 0.75 seconds per time step, ($S_{32} = 14.3$), while model P_8L_{16} (Table 7) averages 4.21 seconds per time step, with a true speedup S_{32} of 21.88, and a rate of approximately 50 MFLOPS (using single precision).

Since a large portion of the computations of the code admits a simple distributed memory partitioning of all of the data, an important comparison is between the GC version and the CC version. The results for the GC version were consistently better than the ones for the CC version, but always by less than 10%. One reason for the similarity between these results is the fact that the extra time that is spent accessing the global memory in the GC version is compensated by the communication between clusters, containing the information from the borders, that is necessary in the CC version. The most important reason though is the ratio between the number of floating point operations using cluster memory data

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	13.5	7.0	7.1	5.2	5.1	4.0	4.2	3.8
2	7.0	3.7	3.7	2.7	2.7	2.2	2.3	2.1
4	3.8	2.0	2.1	1.6	1.6	1.2	1.4	1.2
8	2.3	1.2	1.3	1.0	1.0	0.8	0.9	0.8

Table 3: Average time in sec per timestep, model P_4L_8 - GC version without prefetch.

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	13.0	6.7	6.8	5.0	5.0	3.9	4.2	3.8
2	6.7	3.5	3.6	2.7	2.7	2.1	2.3	2.1
4	3.7	2.0	2.0	1.5	1.5	1.2	1.4	1.2
8	2.3	1.2	1.3	1.0	1.0	0.8	1.0	0.8

Table 4: Average time in sec per timestep, model P_4L_8 - CC version.

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	16.9	8.8	8.9	6.5	6.8	5.4	5.8	5.0
2	9.0	4.7	4.8	3.5	3.6	2.9	3.2	2.7
4	4.9	2.6	2.7	1.9	2.0	1.6	1.8	1.5
8	2.7	1.5	1.6	1.1	1.2	1.0	1.1	0.9

Table 5: Average time in sec per timestep, model P_4L_8 - GG version with prefetch.

to the number of global memory accesses which is roughly 25. Therefore, since using the prefetch capability of Cedar to transfer data from global to cluster memory in blocks causes the cost of global memory access to approach that of cluster memory access, the transfer of the three-dimensional data into cluster memory is relatively inexpensive. In fact, the ratio of 25 is sufficient to obviate the need for the block fetching of data given the present Cedar relative memory costs. Comparing the results from Tables 7 and 8, we observe that in the GC version, the runs that used the prefetch unit were at most 5% faster than the runs that did not.

Clearly, such a result will change as the cost functions change. The data transfer between the work space and the virtual disk, (respectively cluster and global memory in the GC version), is done with the use of two subroutines, that are called at the beginning and end of each time step. During the baroclinic phase, the global memory is accessed only when these routines are called. Hence, we were able to artificially vary the global memory access time by calling these routines more than once whenever a read or a write was required. As an experiment, we ran the GC version artificially slowing down the global memory access time by factors ranging

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	28.3	14.6	14.6	10.4	10.1	8.1	8.0	7.7
2	14.5	7.5	7.6	5.4	5.3	4.3	4.3	4.1
4	7.6	4.0	4.1	2.9	2.9	2.3	2.4	2.2
8	4.1	2.2	2.3	1.7	1.7	1.3	1.5	1.3

Table 6: Average time in sec per timestep, model P_4L_8 - GG version without prefetch.

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	96.6	48.8	47.4	34.2	31.9	26.1	24.6	24.8
2	50.3	25.6	24.7	17.9	16.5	13.8	12.7	13.0
4	28.1	14.4	13.7	10.0	9.0	7.8	7.0	7.3
8	17.7	9.0	8.3	6.4	5.5	5.0	4.2	4.4

Table 7: Average time in sec per timestep, model P_8L_{16} - GC version with prefetch.

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	99.4	51.2	50.2	35.6	33.3	27.2	25.3	25.9
2	52.1	26.6	25.9	18.7	17.5	14.3	13.1	13.6
4	38.0	14.9	14.1	10.4	9.4	8.1	7.2	7.5
8	18.0	9.2	8.7	6.6	5.6	5.1	4.6	4.7

Table 8: Average time in sec per timestep, model P_8L_{16} - GC version without prefetch.

from 1 (no slowdown, *i.e.* the original) to 50. The slowdown ratios are given by t_k/t_1 where t_k is the run time with slowdown factor of k, and t_1 is the execution time without slowdown. The results are plotted in Figures 5 and 6 for the models P_4L_8 and P_8L_{16} respectively. The best results when no prefetch was used, were obtained with column partitioning for model P_8L_{16}, and with 2D partitioning for model P_4L_8; hence, the line for the 2D partitioning and the line for the column partitioning are duplicated in Figures 5 and 6 respectively, representing the runs with and without prefetch in both cases (lower and upper line respectively).

Using these plots one could estimate the program's run time for a slower global memory. In both figures one can easily observe that the slope of the line for the run without prefetch is larger than the others, thus these plots also show the increasing need for data prefetch as global memory becomes more remote. The moderate rate of degradation of performance is leading evidence that if the increasing remoteness of memory is caused by increasing the number of clusters then we would expect to maintain reasonable performance for the code over a significant increase in the number of processors. The increase in the number of clusters

CEs	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	95.7	48.8	48.3	34.3	33.2	26.2	25.2	25.2
2	50.4	25.7	25.1	18.1	17.4	13.9	13.2	13.2
4	28.1	14.5	13.9	10.2	9.6	7.8	7.4	7.4
8	17.9	9.4	8.9	6.6	6.0	5.1	4.6	4.7

Table 9: Average time in sec per timestep, model P_8L_{16} - CC version.

C E s	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	118.9	60.6	61.8	43.4	46.6	36.3	36.1	35.6
2	63.1	32.2	32.9	22.6	20.0	18.6	18.7	18.3
4	33.3	17.3	17.7	12.1	12.6	9.9	10.0	9.8
8	17.3	9.1	9.4	6.5	6.8	5.3	5.5	5.4

Table 10: Average time in sec per timestep, model P_8L_{16} - GG version with prefetch.

C E s	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
1	214.6	108.8	109.6	75.7	74.6	57.3	56.7	56.3
2	109.1	55.5	55.9	38.5	38.5	29.3	29.1	28.8
4	55.6	28.5	28.7	19.9	19.8	15.3	15.3	15.1
8	28.7	14.8	15.0	10.5	10.5	8.2	8.3	8.2

Table 11: Average time in sec per timestep, model P_8L_{16} - GG version without prefetch.

would, of course, require the problem size to grow but the operation-transfer ratio of 25 is maintained independent of problem size. All of these results use the prefetch unit as a block fetch mechanism. The performance could also be maintained as the remoteness of global memory increases by using a larger grain true prefetch, i.e., prefetching the data needed to compute slab $j+1$ to cluster memory while computing slab j. At present, Cedar Fortran does not explicitly support such a divided use of the CE's within a cluster but it is possible to implement manually.

Another result that shows the importance of the use of data prefetch is the difference in performance of the GG version with and without the use of the prefetch unit. Comparing the best results for the GG version with and without prefetch (Tables 5 and 6 for model P_4L_8, and Tables 10 and 11 for model P_8L_{16}), we notice that the performance dropped approximately 40% and 54% for models P_4L_8 and P_8L_{16} respectively, when prefetch was not used. These results show the importance of the use of data prefetch when there is a small ratio between the number of floating point operations and global memory accesses (in the GG version this ratio is less than one).

It is also instructive to compare the difference in performance between the GG and GC version with and without the use of prefetch. Basic memory system performance on Cedar [15] predicts potential improvement by using the prefetch unit to offset latency may bring the global memory access rate close to the cluster memory access rate. We can confirm this prediction by comparing the results from the GC and GG runs using model P_8L_{16} with 8 CEs. Table 12 shows the percentage of difference in execution time for P_8L_{16} with 8 CEs per cluster between GG and GC versions with and without prefetch used.

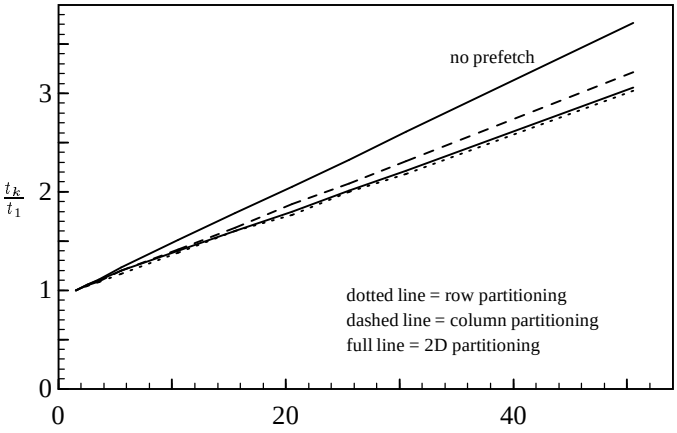

Figure 5: Slowdown for model P_4L_8, varying the global memory access time.

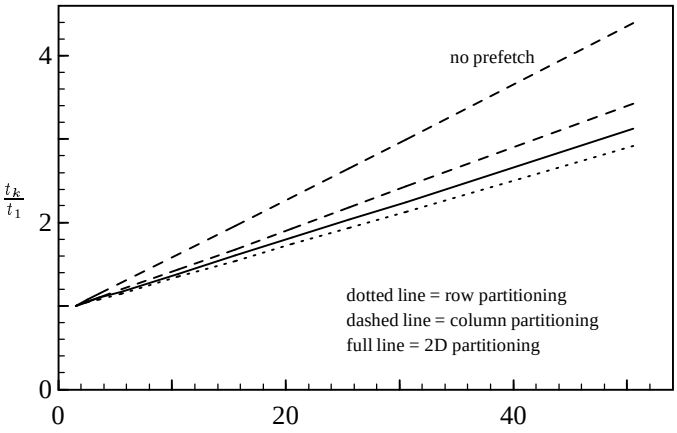

Figure 6: Slowdown for model P_8L_{16}, varying the global memory access time.

Some interesting trends arise from comparing Table 12, Table 7 and Table 10. The first is the obvious result that when no prefetching is used there is a significant difference between the versions. The fact that the difference is not as large as one might expect given the basic performance characteristics of the Cedar system, [15], indicates that in both versions there is a nonnegligible amount of cluster and/or register activity. Secondly, consider the data when prefetching is turned on. Note that while the row partitioning is the least affected by moving to GG from GC its percentage change is the most sensitive to the number of clusters used. This is also a direct consequence of the block lengths used for global memory accesses. The row partitioning suffers most, relatively speaking, from the increased contention of accessing global memory with more clusters since the size

pref.	Number of Clusters							
	1	2		3		4		
		row	col	row	col	row	col	2D
off	59.0	60.3	72.8	59.8	87.8	61.5	83.3	75.7
on	-2.0	0.1	13.4	1.7	24.2	6.2	30.6	20.4

Table 12: Percentage of difference in execution time from the GG to the GC versions, using model P_8L_{16} with 8 CEs per cluster.

of its block fetches are longer than the column and 2D partitionings and the resulting memory requests are denser in time. Thirdly, the performance of the GG version for all partitionings with prefetch is very close to the GC version with row partitioning and prefetch. This confirms the expected complementary nature of prefetching from global memory or data distribution into cluster memory. The interesting part of this particular observation is, however, that the variation in the relative performance of the GG and GC versions for different partitionings is a result of the variation of the GC performance not the GG version. It is the explanation of this variation that we address next.

The variation in performance of the different partitionings for the GC version and the CC version is related to vector length and cache usage. The row partitioning vector length is independent of the number of clusters (but the number of slabs processed per cluster varies) while the column partitioning vector length decreases as the number of clusters increases (but the number of slabs processed remains constant). Both partitionings process the same amount of data but the distribution into more fetches of smaller length makes the column partitioning more sensitive to overhead of vector/concurrent operation. Similarly, the 2D partitioning on 4 clusters falls in between the row and column partitionings due to its compromise vector length and number of slabs processed. As a result, one trend would be for the column and 2D partitionings to have worse performance than the row partitioning for small to moderate sized problems. On the other hand, the column and 2D partitionings work with a smaller amount of workspace data and for small to moderate sized problems there will be a cache usage effect. For the P_8L_{16} data set, the column partitioning requires work space that is only slightly larger than the cluster cache while the row partitioning is large enough to flush cache with a single sweep through the work space. In general the code has very little reuse of data and therefore whenever caching can be exploited it can have a significant effect. For the P_8L_{16} data set, these two contradictory effects end up with a net gain for the column and 2D partitionings (although this is not true for the P_4L_8 data set where the row partitioning will also exploit cache). The net effect is somewhat disappointing however and future work will attempt to quantify the expected performance advantage for the column partitioning more carefully to see if it can be enhanced. If it can, in fact, be enhanced via, for example, more careful code generation, then a two level blocking strategy may be useful. This is needed because the cache usage advantage of the column and 2D partitionings will disappear as the problem size and workspace requirements increase. A second level of partitioning of the subdomain given to a particular cluster into column or 2D partitions could then be used to improve performance. The effect of the cache can also be seen by considering the multicluster speedups $S^{cl}_{GC}(4,8)$ and $S^{cl}_{CC}(4,8)$ for the row and column partitionings. For the row partitionings we have, $S^{cl}_{GC}(4,8) = 3.5$ and $S^{cl}_{CC}(4,8) = 3.5$ which are certainly respectable and for the most part expected given the decoupled nature of the computations. The column partitionings yield the near-perfect and superlinear values $S^{cl}_{CC}(4,8) = 3.9$ and $S^{cl}_{GC}(4,8) = 4.2$. These results are due to the fact that as clusters are added the column partitioning reduces the amount of work space required per cluster and eventually benefits from the cache effect. Since data accessed in global memory is not cached in the cluster cache, the performance variation across partitions is reduced considerably.

While the multicluster speedups are reasonable, the multi-CE speedups confirm some expected problems with the use of the Alliant FX/8's as clusters. Specifically, the cluster memory system bandwidth is saturated quicker than global memory for the same sparsity of memory requests per cluster. The comparison of the multi-CE speedups for all versions and partitionings, is presented in Table 13 for model P_8L_{16} using 1 and 4 clusters with 8 CEs per cluster. The data clearly shows the effect of saturation of the cluster memory bandwidth. The values of $S^{ce}_v(1,8)$ for the GG version with and without prefetch show that control overhead and other parallel considerations are not a source of performance degradation for 8 CE's. Therefore, the GC and CC versions, which use cluster memory, have their performance degraded by the saturation of the cluster memory bandwidth. The values of $S^{ce}_v(4,8)$ show similar trends. The GG version without prefetch, as expected, has no trouble due to global memory contention. The use of prefetch degrades the speedup somewhat but the sparsity of the memory requests is such that a reasonable efficiency is maintained. The GC and CC version for similar sequences of memory requests in cluster memory[1] experience clear performance degradation. It can be concluded that the global memory system is more robust when stressed than the cluster memory systems.This is not unexpected since the transfer of data between cluster memory and cache is accomplished via a bus and the global memory system makes use of two unidirectional Omega networks.

[1]The stream of memory requests may be different when the arrays are in global memory from those seen when accessing cluster memory due to some idiosyncrasies of the code generator. It is much more conservative with common subexpression elimination when the data resides in global memory which can in some cases significantly alter the type of memory traffic generated. This is not a major problem for this code.

Version	$S_v^{ce}(1,8)$	$S_v^{ce}(4,8)$		
		row	col	2D
GG no prefetch	7.49	6.97	6.80	6.89
GG with prefetch	6.88	6.82	6.57	6.62
GC no prefetch	5.52	5.33	5.55	5.57
GC with prefetch	5.47	5.21	5.83	5.57
CC	5.34	5.09	5.44	5.39

Table 13: Multi-CE speedups for model P_8L_{16}.

6 Conclusions

We implemented a multicluster ocean circulation model on Cedar, and experimented with three partitioning schemes, using several data placement strategies, and mappings to the components of Cedar. The experiments simulated the Mediterranean Sea topography.

The best result using the model with the P_4L_8 data set requires 0.75 seconds per time step, while the original code restructured automatically on an Alliant FX/8 required 2.58 seconds per time step. The best result with the P_8L_{16} data set requires 4.21 seconds per time step, with a speedup of 21.88 for the 32 CEs and a multicluster speedup in excess of 3.5 for all of the versions of the code.

As expected the GC version of the code which placed the virtual disk data in global memory and work space in cluster memory, best exploited Cedar's control and memory structures. The experiments also showed that the use of the prefetch unit could mitigate the effect of the latency of the global memory system, bringing the times for the GG version close to those of the GC version. The complementary nature of the partitioning of the data into the cluster memories and the use of prefetch on all or part of the data from global memory is seen in the similar performance of the GC and CC versions.

The multicluster versions of the code were derived by a straightforward and perhaps eventually automatable large grain restructuring of the code. This technique allowed the clusters to exploit the mapping of the computations to concurrent and vector processing that was used in the original code but which would not scale beyond a moderate number of processors.

Future work on this code includes the more detailed quantitative characterization of the observed performance trends as well as the application of Cedar performance prediction techniques to evaluate alternate, and potentially more efficient, organizations of the computations within a cluster. These reorganizations of cluster work will center on the issue of exploitation of topography information in the reduction of unnecessary operations. These alterations in cluster processing strategies may also require further parameterization of the large grain partitioning used across clusters. Finally, the development and analysis of an efficient multicluster implementation of the relaxation phase of the code will be considered. For the test cases used for the results above, the relaxation phase performance was not a major factor.

Acknowledgments

This work was supported by the U.S. Department of Energy under Grant No. DOE DE-FG02-85ER25001, with additional support provided by the State of Illinois Department of Commerce and Community affairs, State Technology challenge Fund under grant No. SCCA 91-108, by the Conselho Nacional de Desenvolvimento Cientifico e Tecnologico (CNPq), Brazil, and by the National Science Foundation under Grant No. NSF CCR900000N for the use of the Cray Y-MP/48 at the National Center for Supercomputing Applications, Univ. of Illinois at Urbana-Champaign. We thank Antonio Navarra and Nadia Pinardi for their help and for providing the code.

References

[1] ANDRICH, P., DELECLUSE, P., LEVY, C., AND MADEC, G. A multitasked general circulation model of the ocean. In *Proceedings Fourth International Symposium, Cray Research* (1988), pp. 407–428.

[2] ANDRICH, P., AND MADEC, G. Performance evaluation for an ocean general circulation model: vectorization and multitasking. In *1988 International Conference on Supercomputing* (St. Malo, France, July 1988), ACM, pp. 295–302.

[3] BELL, J., AND PATTERSON JR., G. Data organization in large numerical computations. *The Journal of Supercomputing 1* (1987), 105–136.

[4] BRYAN, K. A numerical method for the study of the circulation of the world ocean. *Journal of Computational Physics 4* (1969), 347–376.

[5] Building an advanced climate model: Program plan for the CHAMMP climate modeling program. Tech. Rep. DOE/ER-0479T, U.S. Dept. of Energy, Washington, D.C., Dec. 1990.

[6] CHEN, S. C., KUCK, D. J., AND SAMEH, A. H. Practical parallel band triangular system solvers. *ACM Trans. on Mathematical Software 4*, 3 (Sept., 1978), 270–277.

[7] CHERVIN, R. M., AND SEMTNER JR., A. J. An ocean modelling system for supercomputer architectures of the 1990s. In *Proceedings of the NATO Advanced Research Workshop on Climate-Ocean Interaction* (1988), M. E. Schlesinger, Ed., Kluwer Academic Publishers., pp. 87–95.

[8] COX, M. D. A primitive equation, 3-dimensional model of the ocean. Tech. Rep. 1, Geophysical Fluid Dynamics Laboratory/NOAA, Princeton University, Princeton, NJ 08542, August 1984.

[9] CRAY RESEARCH INC. *Multitasking User Guide*, January 1985.

[10] DEROSE, L. Parallel ocean circulation modeling on Cedar. Master's thesis, Univ. of Illinois at Urbana-Champaign, Center for Supercomputing Res. & Dev., December 1991.

[11] DEROSE, L., GALLIVAN, K., AND GALLOPOULOS, E. Trace analysis of the GFDL ocean circulation model: A preliminary study. Tech. Rep. 863, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1989.

[12] DEROSE, L., GALLIVAN, K., GALLOPOULOS, E., AND NAVARRA, A. Parallel ocean circulation modeling on Cedar. In *Proc. Fifth SIAM Conf. Parallel Processing for Scientific Computing* (May 1991), D. C. Sorensen, Ed. To appear. Also CSRD TR-1124.

[13] EMRATH, P., ANDERSON, M., BARTON, R., AND MCGRATH, R. The Xylem Operating System. In *Proc. 1991 Int'l Conference on Parallel Processing* (St. Charles, IL, Aug. 1991), vol. I, pp. 67–70.

[14] Grand challenges: High performance computing and communications. Office of Science and Technology Policy, 1991. A report by the committee on Physical, Mathematical, and Engineering Sciences.

[15] GALLIVAN, K., JALBY, W., TURNER, S., VEIDENBAUM, A., AND WIJSHOFF, H. . Preliminary basic performance analysis of the Cedar multiprocessor memory systems. In *Proc. 1991 Int'l Conference on Parallel Processing* (Aug. 1991), vol. I, pp. 71–75.

[16] GUZZI, M. D., PADUA, D. A., HOEFLINGER, J. P., AND LAWRIE, D. H. Cedar Fortran and other vector and parallel Fortran dialects. *Journal of Supercomputing* (March 1990), 37–62.

[17] HOEFLINGER, J. Cedar Fortran programmer's handbook. Tech. Rep. 1157, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, October 1991.

[18] KONICEK, J., TILTON, T., VEIDENBAUM, A., ZHU, C., DAVIDSON, E., DOWNING, R., HANEY, M., SHARMA, M., YEW, P., FARMWALD, P., KUCK, D., LAVERY, D., LINDSEY, R., POINTER, D., ANDREWS, J., BECK, T., MURPHY, T., TURNER, S., AND WARTER, N. The organization of the Cedar system. In *Proc. 1991 Int'l Conference on Parallel Processing* (St. Charles, IL, Aug. 1991), vol. I, pp. 49–56.

[19] MALONE, R. C., CHERVIN, R., SMITH, R., AND DANNEVIK, W. P. Minisymposium: Computing climate change: Can we beat nature? In *Proc. Supercomputing'91*. IEEE, Albuquerque, New Mexico, Nov. 1991, pp. 677–679.

[20] MALONY, A. D. High resolution process timing user's manual. Tech. rep., Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, June 1987. CSRD Report No. 676.

[21] MCGRATH, R., AND EMRATH, P. Using memory in the Cedar system. Tech. Rep. 655, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1987.

[22] PACANOWSKI, R. C., DIXON, K., AND ROSATI, A. *GFDL MOM 1.0*. Geophysical Fluid Dynamics Laboratory/NOAA, December 1990.

[23] PINARDI, N., AND NAVARRA, A. A brief review of global Mediterranean wind-driven general circulation experiments. Tech. Rep. 132, IMGA-CNR, Modena Italy, 1988.

[24] SEMTNER JR., A. J. An oceanic general circulation model with botton topography. Tech. Rep. 9, UCLA Dept. of Metheorology, 1974.

[25] SEMTNER JR., A. J. Finite–diferrence formulation of a world ocean model. In *Proc. NATO Institute of Advanced Physical Oceanographic Numerical Modelling*. (1986), J. J. O'Brien, Ed., D. Reidel Publishing Co., pp. 187–202.

[26] SEMTNER JR., A. J. History and methodology of modelling the circulation of the world ocean. In *Proc. NATO Institute of Advanced Physical Oceanographic Numerical Modelling*. (1986), J. J. O'Brien, Ed., D. Reidel Publishing Co., pp. 23–32.

[27] SEMTNER JR., A. J., AND CHERVIN, R. M. A simulation of the global ocean circulation with resolved eddies. *Journal of Geophysical Research 93*, C12 (1988), 15502–15522.

[28] SMITH, R. D., DUKOWICZ, J. K., AND MALONE, R. C. Massively parallel global ocean modeling. Tech. Rep. LA-UR-91-2583, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, 1991.

[29] ZHU, C., AND YEW, P. A scheme to enforce data dependence on large multiprocessor systems. *IEEE Transactions on Software Engineering SE-13(6)* (June 1987), 726–739.