PARALLEL COMPUTING (IN PRESS)

DESIGN AND PERFORMANCE OF A SCALABLE PARALLEL COMMUNITY CLIMATE MODEL

JOHN DRAKE*, IAN FOSTER†, JOHN MICHALAKES†,
BRIAN TOONEN†, PATRICK WORLEY*

Abstract. We describe the design of a parallel global atmospheric circulation model, PCCM2. This parallel model is functionally equivalent to the National Center for Atmospheric Research's Community Climate Model, CCM2, but is structured to exploit distributed memory multicomputers. PCCM2 incorporates parallel spectral transform, semi-Lagrangian transport, and load balancing algorithms. We present detailed performance results on the IBM SP2 and Intel Paragon. These results provide insights into the scalability of the individual parallel algorithms and of the parallel model as a whole.

1. Introduction. Computer models of the atmospheric circulation are used both to predict tomorrow's weather and to study the mechanisms of global climate change. Over the last several years, we have studied the numerical methods, algorithms, and programming techniques required to implement these models on so-called massively parallel processing (MPP) computers: that is, computers with hundreds or thousands of processors. In the course of this study, we have developed new parallel algorithms for numerical methods used in atmospheric circulation models, and evaluated these and other parallel algorithms using both testbed codes and analytic performance models [8, 10, 11, 29, 32]. We have also incorporated some of the more promising algorithms into a production parallel climate model called PCCM2 [6]. The latter development has allowed us to validate performance results obtained using simpler testbed codes, and to investigate issues such as load balancing and parallel I/O. It has also made the computational capabilities of MPP computers available to scientists for global change studies, providing the opportunity for increases in spatial resolution, incorporation of improved process models, and longer simulations.

In this paper, we provide a comprehensive description of the design of PCCM2 and a detailed evaluation of its performance on two different parallel computers, the IBM SP2 and Intel Paragon. We also touch upon aspects of the experimental studies used to select the parallel algorithms used in PCCM2. However, these studies are not the primary focus of the paper.

PCCM2 is a scalable parallel implementation of the National Center for Atmospheric Research (NCAR)'s Community Climate Model version 2 (CCM2). CCM2 is a comprehensive, three-dimensional global atmospheric general circulation model for use in the analysis and prediction of global climate [15, 16]. It uses two different numerical methods to simulate the fluid dynamics of the atmosphere. The spherical harmonic (spectral) transform method [7, 22] is used for the horizontal discretization of vorticity, divergence, temperature, and surface pressure; this method features regular, static, global data dependencies. A semi-Lagrangian transport scheme [31] is used for highly accurate advection of water vapor and an arbitrary number of other scalar fields, such

[1] Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367.
[2] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

as sulfate aerosols and chemical constituents; this scheme features irregular, dynamic, mostly local data dependencies. Neither method is straightforward to implement efficiently on a parallel computer.

CCM2, like other weather and climate models, also performs numerous calculations to simulate a wide range of atmospheric processes relating to clouds and radiation absorption, moist convection, the planetary boundary layer, and the land surface. These processes share the common feature that they are coupled horizontally only through the dynamics. These processes, collectively termed "physics," have the important property of being independent for each vertical column of grid cells in the model. However, they also introduce significant spatial and temporal variations in computational load, and it proves useful to incorporate load balancing algorithms that compensate for these variations.

PCCM2 was the first spectral, semi-Lagrangian model developed for distributed-memory multicomputers [6]. However, we are not the only group to have examined parallel formulations of such models. At the European Center for Medium-range Weather Forecasts (ECMWF), Dent [4] and his colleagues have developed the Integrated Forecast System for both vector multiprocessors and distributed memory multiprocessors. Sela [25] has adapted the National Meteorological Center's spectral model for use on the CM5. Hammond et al. [17] have developed a data-parallel implementation of CCM2, and Hack et al. [16] have adapted a vector multiprocessor version of CCM2 to execute on small multicomputers. All these developments build on previous work on parallel algorithms for atmospheric circulation models, as described in the papers referenced above and in [1, 3, 14, 18, 19, 24].

Space does not permit a description of the primitive equations used to model the fluid dynamics of the atmosphere [30] or of CCM2. Instead, we proceed directly in Sections 2 and 3 to a description of the spectral transform method and the techniques used to parallelize it in PCCM2. In Sections 4 and 5, we describe the parallel semi-Lagrangian transport and load balancing algorithms used in PCCM2. In Sections 6 and 7, we present our performance results and our conclusions, respectively.

2. The Spectral Transform Algorithm. The spectral transform method is based on a dual representation of the scalar fields on a given vertical level in terms of a truncated series of spherical harmonic functions and the values on a rectangular grid whose axes, in a climate model, represent longitude and latitude. State variables in spectral space are represented by the coefficients of an expansion in terms of complex exponentials and associated Legendre functions,

$$\xi(\lambda,\mu) = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) e^{i\cdot m\lambda}, \tag{1}$$

where $P_n^m(\mu)$ is the (normalized) associated Legendre function, $i = \sqrt{-1}$, λ is the longitude, and $\mu = \sin\theta$, where θ is the latitude. The spectral coefficients are then determined by the equation

$$\xi_n^m = \int_{-1}^{1} \left[\frac{1}{2\pi} \int_0^{2\pi} \xi(\lambda,\mu) e^{-i\cdot m\lambda} d\lambda \right] P_n^m(\mu) d\mu \equiv \int_{-1}^{1} \xi^m(\mu) P_n^m(\mu) d\mu \tag{2}$$

since the spherical harmonics $P_n^m(\mu)e^{i\cdot m\lambda}$ form an orthonormal basis for square integrable functions on the sphere. In the truncated expansion, M is the highest Fourier mode and $N(m)$ is the highest degree of the associated Legendre function in the north-south representation. Since the physical quantities are real, ξ_n^{-m} is the complex conjugate of ξ_n^m and only spectral coefficients for nonnegative modes need to be calculated.

To evaluate the spectral coefficients numerically, a fast Fourier transform (FFT) is used to find $\xi^m(\mu)$ for any given μ. The Legendre transform (LT) is approximated using a Gaussian quadrature rule. Denoting the Gauss points in $[-1,1]$ by μ_j and the Gauss weights by w_j,

$$\xi_n^m = \sum_{j=1}^{J} \xi^m(\mu_j) P_n^m(\mu_j) w_j. \tag{3}$$

Here J is the number of Gauss points. (For simplicity, we will henceforth refer to (3) as the forward Legendre transform.) The point values are recovered from the spectral coefficients by computing

$$\xi^m(\mu) = \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) \tag{4}$$

for each m (which we will refer to as the inverse Legendre transform), followed by FFTs to calculate $\xi(\lambda,\mu)$.

The physical space grid is rectangular with I grid lines evenly spaced along the longitude axis and J grid lines along the latitude axis placed at the Gaussian quadrature points used in the forward LT. In this paper, we assume a triangular spectral truncation: $N(m) = M$ and the (m,n) indices of the spectral coefficients for a single vertical layer form a triangular grid. To allow exact, unaliased transforms of quadratic terms, we select I to be the minimum multiple of two such that $I \geq 3M+1$, and set $J = I/2$ [22]. The value of M can then be used to characterize the horizontal grid resolution, and we use the term "TM" to denote a particular discretization. Thus with $M = 42$, we choose $I = 128$ and $J = 64$ for T42 resolution.

In summary, the spectral transform as used in climate models proceeds in two stages. First, the FFT is used to integrate information along each west-east line of a latitude/longitude grid, transforming physical space data to Fourier space. This stage entails a data dependency in the west-east dimension. Second, a LT is applied to each wavenumber of the resulting latitude/wavenumber grid to integrate the results of the FFT in the north-south, or latitudinal, direction, transforming Fourier space data to spectral space [2]. This stage entails a data dependency in the north-south dimension. In the inverse transform, the order of these operations is reversed.

3. PCCM2 Parallel Algorithms. We now introduce the parallel algorithms used in PCCM2. This model is designed to execute on distributed memory multicomputers, in which processors, each with a local memory, work independently and exchange data via an interconnection network. Hence, our parallel algorithms must specify data distribution, computation distribution, and the communication required

to move data from where it is located to where it is required. An important characteristic of the parallel algorithms incorporated in PCCM2 is that they compute bit-for-bit identical results regardless of the number of processors applied to a problem. This uniformity is not straightforward to achieve, because of the lack of associativity in floating-point addition, but was obtained by modifying the sequential implementation of various summation operations to use the same tree-based summation algorithm as the parallel code.

We start our description of PCCM2 by examining the spectral transform, as this determines in large part the data distributions used in other program components.

3.1. Algorithmic Alternatives. The spectral transform is a composition of FFT and LT phases, and parallel FFT and LT algorithms are well understood (e.g., see [12, 23, 26, 27]). Nevertheless, the design of parallel spectral transform algorithms is a nontrivial problem, both because the matrices involved are typically much smaller than in other situations (e.g., 64–1024 in each dimension, rather than tens of thousands) and because the FFT and LT phases interact in interesting ways on certain architectures.

Spectral models such as CCM2 operate on data structures representing physical, Fourier, and spectral space variables. Each of these data structures can be thought of as a three-dimensional grid, with the vertical dimension being the third coordinate in each case. The other two coordinates are, in physical space, latitude and longitude; in Fourier space, Fourier wavenumber and latitude; and in spectral space, wavenumber and degree of associated Legendre polynomial (n). For maximum scalability, each of these three spaces must be decomposed over available processors. In principle, data structures can be decomposed in all three dimensions. However, we restrict our attention to two dimensional decompositions, as these provide adequate parallelism on hundreds or thousands of processors and simplify code development.

A variety of different decompositions, and hence parallel algorithms, are possible in spectral atmospheric models [10, 11]. Physical space is best partitioned by latitude and longitude, as a decomposition in the vertical dimension requires considerable communication in the physics component. However, Fourier and spectral space can be partitioned in several different ways. A latitude/wavenumber decomposition of Fourier space requires communication within the FFT [8, 29]. Alternatively, a latitude/vertical decomposition of Fourier space allows FFTs to proceed without communication, if a transpose operation is first used to transform physical space from a latitude/longitude to a latitude/vertical decomposition. Similar alternatives exist for spectral space, with one decomposition requiring communication within the LT, and another avoiding this communication by first performing a transpose of the spectral coefficients. Another possibility, described below, is to decompose spectral space in just one dimension, replicating it in the other. This reduces communication costs at the expense of a modest increase in storage requirements.

The different FFT and LT algorithms just described can be combined in different ways, leading to a large number of possible parallel algorithms. Performance is also influenced by the number of processors allocated to each decomposed dimension (that is, the aspect ratio of the abstract processor mesh) and the protocols used to imple-

ment the communication algorithms. We have obtained a detailed understanding of the relative performance of these different algorithms by incorporating them in a sophisticated testbed code called PSTSWM [10, 11, 33]. Extensive studies with this code have allowed us to identify optimal algorithm choices, processor mesh aspect ratios, and communication protocols for different problem sizes, computers, and numbers of processors. As much as possible, these optimal choices are incorporated in PCCM2.

Scaling arguments show that for large problem sizes and large numbers of processors, transpose algorithms are the most efficient, as they communicate the least data. (These algorithms are used in the ECMWF's Integrated Forecast System model, which is designed to operate at T213 resolution using 31 vertical levels [4].) However, both experimental and analytic studies indicate that for the smaller problem sizes considered in climate modeling, nontranspose algorithms are often competitive, particularly for the LT.

3.2. PCCM2 Data Decompositions. PCCM2 uses a latitude/longitude decomposition of physical space, a latitude/wavenumber decomposition of Fourier space, and a one-dimensional decomposition of spectral space. These choices are influenced by both performance and software engineering concerns. Performance considerations determine the spectral space decomposition: the one-dimensional decomposition permits the use of a nontranspose algorithm that is known to be more efficient than a transpose-based LT for many problem sizes of interest. Software engineering concerns determine the choice of Fourier space decomposition. While a latitude/vertical decomposition would allow the use of more efficient algorithms in some situations, it would significantly complicate the conversion of CCM2 into a distributed-memory parallel code, and is subject to significant load balance problems when the number of vertical layers in the model is small compared to the number of processors used to decompose the vertical dimension. Hence, we use a latitude/wavenumber decomposition, and support both a parallel FFT that requires communication within the FFT, and a double-transpose FFT that avoids communication within the FFT at the cost of two transpose operations.

We now provide a more detailed description of the PCCM2 domain decompositions. We view the multicomputer as a logical $P \times Q$ two-dimensional processor grid. In physical space, the latitudinal dimension is partitioned into $2Q$ intervals, each containing $J/2Q$ consecutive latitudes (Gauss points) along the latitude axis. Each processor row is assigned two of these intervals, one from the northern hemisphere, and the reflected latitudes in the southern hemisphere. This assignment allows symmetry of the associated Legendre polynomials to be exploited in the LT. The assignment also restricts Q, the number of processor rows, to be no larger than $J/2$. The longitudinal dimension is partitioned into P equal intervals, with each interval being assigned to a different processor column. The resulting "block" decomposition of the physical domain is illustrated in Figure 1 for a small example.

The latitude-wavenumber decomposition used for Fourier space in PCCM2 when using the double transpose parallel FFT is illustrated in Figure 1. (A different decomposition in used in the other parallel FFT [11].) The latitude dimension is partitioned as in the physical domain, while the wavenumber dimension is partitioned into P sets

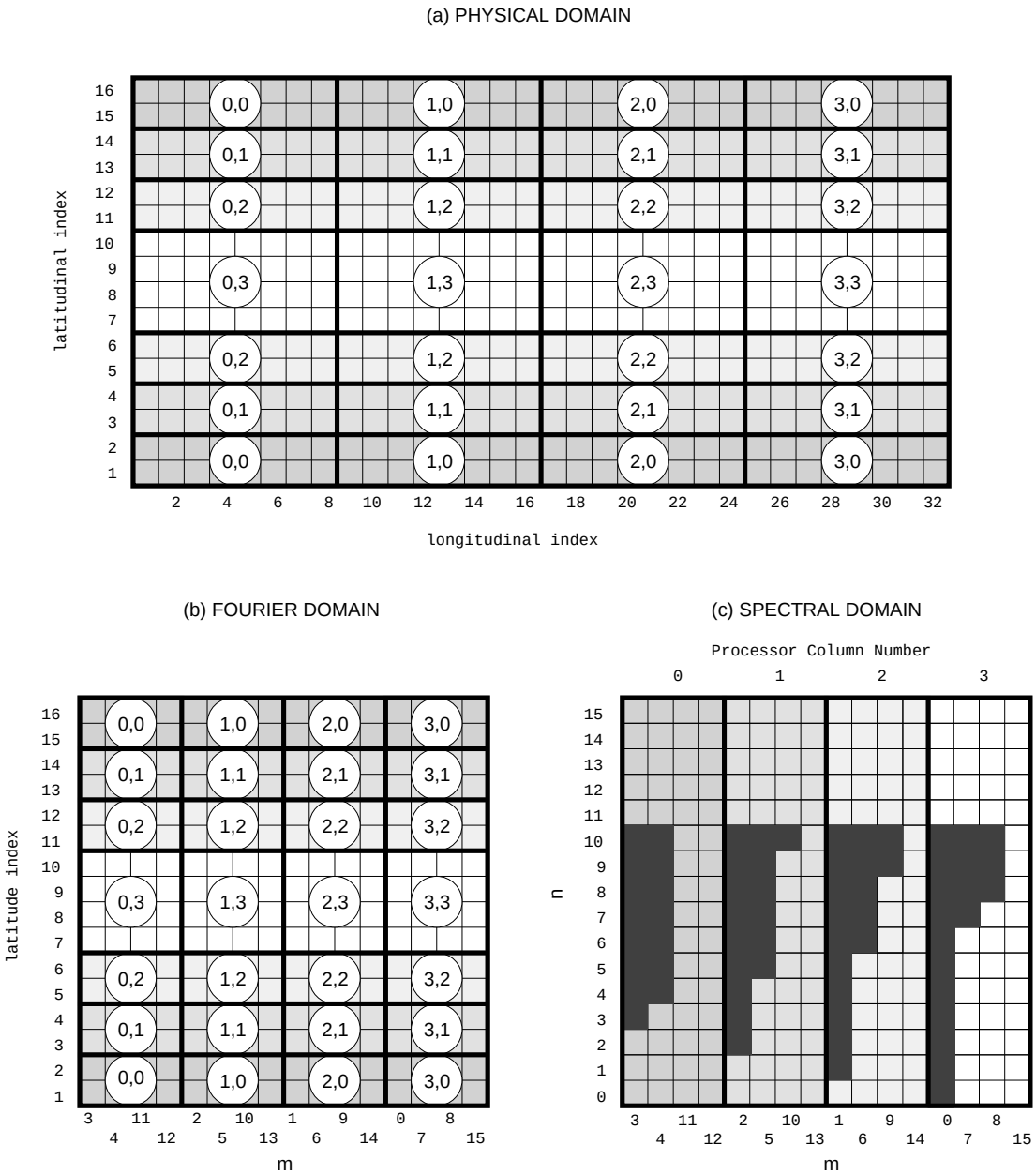

FIG. 1. *The decomposition of the (a) physical, (b) spectral, and (c) Fourier domains over a* 4×4 *grid of processors. For figures (a) and (b), each small cell represents a data item. The thicker lines show the boundaries between processors. The circles contain the processor coordinates. The shaded cells in figure (c) represent the spectral coefficients to be included in the spectral transform, and shows how these are decomposed over processor columns*

of wavenumbers, with each set being assigned to a different processor column. As illustrated in Figure 1, the ordering for the Fourier coefficients differs from the ordering of the longitude dimension in physical space. As described below, this serves to balance the distribution of spectral coefficients.

PCCM2 partitions spectral space in one dimension only (wavenumber), replicating it in the other. The wavenumber dimension is distributed as in Fourier space: wavenumbers are reordered, partitioned into consecutive blocks, and assigned to processor columns. The spectral coefficients associated with a given wavenumber are replicated across all processors in the processor column to which that wavenumber was assigned. This replication reduces communication costs. Again, see Figure 1 for an example decomposition.

Note that in a triangular truncation, the number of spectral coefficients associated with a given Fourier wavenumber decreases as the wavenumber increases. Without the reordering of the wavenumbers generated by the parallel FFT, a noticeable load imbalance would occur, as those processor columns associated with larger wavenumbers would have very few spectral coefficients. The particular ordering used, which was first proposed by Barros and Kauranne [1], minimizes this imbalance.

3.3. Parallel Fast Fourier Transform. The data decompositions used in PCCM2 allow "physics" computations to proceed without communication within each vertical column. However, communication is required for the FFT and LT. As noted above, communication for the FFT can be achieved in two different ways. One approach is to use a conventional parallel FFT; we do not describe this here as the basic techniques are described in [11].

An alternative approach that is more efficient in many cases uses matrix transpose operations to transform the physical space data passed to the FFT from a latitude/longitude decomposition to a latitude/vertical+field decomposition, partitioning the fields being transformed as well as the vertical layers. A second transpose is then used to remap the Fourier coefficients produced by the FFT from a latitude/vertical+field decomposition to the latitude/wavenumber decomposition employed in Fourier space. This alternative approach avoids the need for communication during the FFT itself, at the cost of two transpose operations, but tends to avoid load balance problems that can arise in a latitude/vertical decomposition.

In both cases, FFTs from different latitudes are grouped to reduce the number of communications. Hence, in the conventional parallel FFT each node sends $\log Q$ messages containing a total of $D \log Q$ data, where D is the amount of data to be transformed, while in the double-transpose approach each node sends about $2Q$ messages containing a total of about $2D$ data. Clearly the choice of algorithm depends on the values of D and Q and on the communication parameters of a particular computer.

3.4. Parallel Legendre Transform. Communication is also required for the Legendre transform used to move between Fourier and spectral space. In PCCM2, this operation is achieved using a parallel vector sum algorithm; other algorithms are slightly more efficient in some cases [11, 32], but are harder to integrate into PCCM2.

The forward and inverse Legendre transforms are

$$\xi_n^m = \sum_{j=1}^{J} \xi^m(\mu_j) P_n^m(\mu_j) w_j$$

and

$$\xi^m(\mu_j) = \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu_j)$$

respectively. For the forward Legendre transform, each ξ_n^m depends only on data associated with the same wavenumber m, and so depends only on data assigned to a single processor column. Each processor in that column can calculate independently its contribution to ξ_n^m, using data associated with the latitudes assigned to that processor. To finish the calculation, these P contributions need to be summed, and the result needs to be broadcast to all P processors, since spectral coefficients are duplicated within the processor column. To minimize communication costs, local contributions to all spectral coefficients are calculated first. A distributed vector sum algorithm, which adds the local contributions from each of the P processors, is then used to accumulate the spectral coefficients and broadcast the results within each processor column. This sum can be computed in a number of ways; we support both a variant of the recursive halving algorithm [28], and a ring algorithm [11]. For the inverse transform, calculation of $\xi^m(\mu_j)$ requires only spectral coefficients associated with wavenumber m, all of which are local to every processor in the corresponding processor column. Thus, no interprocessor communication is required in the inverse transform.

This parallel LT algorithm proves to be efficient for moderate numbers of processors. It also leaves the vertical dimension undecomposed, which avoids the need for communication in the vertical coupling in spectral space that is required by the semi-implicit timestepping algorithm. Finally, it is easily integrated into CCM2. (Other methods, such as the interleaved ring and transpose algorithms, require substantial restructuring [11, 32].) A disadvantage is that all computations within the spectral domain must be calculated redundantly within each processor column. However, since CCM2 performs relatively little work in the spectral domain, the redundant work has not proved to be significant. A second disadvantage is that additional storage is required to hold the duplicated spectral coefficients. This effect is rendered less significant by the facts that spectral space variables are small and distributed memory multicomputers typically have a lot of memory. In Section 6.3, we present data that enable us to evaluate the impact of this structure on scalability.

4. Semi-Lagrangian Transport. The semi-Lagrangian transport (SLT) method used in CCM2 updates the value of the moisture field at a grid point (the arrival point, A) by first establishing a trajectory through which the particle arriving at A has moved during the current timestep. From this trajectory the departure point, D, is calculated, and the moisture field is interpolated at D using shape preserving interpolation [31].

PCCM2 uses a simple strategy to parallelize the SLT. Since the SLT occurs entirely in physical space, the associated calculations can be decomposed by using the latitude/longitude decomposition used to parallelize the physics and spectral transform. Communication is then required whenever D and A are on different processors. In order to minimize the number of messages generated, we extend local arrays on each processor with "shadow" regions corresponding to grid points assigned to neighboring processors. Communication is performed to update these shadow regions prior to each timestep, after which calculations can proceed independently on each processor.

This SLT implementation strategy works well everywhere except in polar regions, where CCM2's square latitude/longitude grid gives a small west/east grid spacing. Unfortunately, this small spacing can place departure points far away (in terms of grid points) from arrival points. In extreme cases, velocity vectors that cross the pole can result in departure points that are $I/2$ grid points distant. This situation is not handled well by the shadow region approach.

This problem can be addressed in a number of ways. One partial solution, incorporated into PCCM2, is to make the size of the overlap region a dynamically varying quantity. At each step and in each latitude and vertical level, the size of the overlap needed is determined from the departure points at the previous time step. Another approach, motivated by other considerations but with advantages for the parallel SLT, is to modify CCM2 (and PCCM2) to use a "reduced grid" with fewer grid points per latitude in polar regions and with polar points on a single processor. This approach has proved successful in the IFS model [4], and is currently being investigated for use in CCM2 and PCCM2.

5. Load Balancing. As noted in the introduction, load imbalances can arise in parallel climate models as a result of spatial and temporal variations in the computation requirements of physics routines [20, 21]. For example, CCM2 performs radiation computations only for grid points that are in sunlight. This situation can be corrected by employing load-balancing algorithms that, either statically or dynamically, map computation to processors in different ways.

A flexible load-balancing library has been developed as part of the PCCM2 effort, suitable for use in PCCM2 and other similar climate models. This library has been incorporated into PCCM2 and used to experiment with alternative load-balancing strategies [9]. One simple strategy swaps every second grid point with its partner 180 degrees apart in longitude during radiation calculations. This does a good job of balancing the diurnal cycle: because CCM2 pairs N/S latitudes, it ensures that each processor always has approximately equal numbers of day and night points. However, it generates a large volume of communication. Another strategy uses a dynamic data-distribution strategy that performs less communication but requires periodic reorganizations of model state data. Currently, the former scheme gives better performance in most cases, and succeeds in eliminating about 75 per cent of the inefficiency attributable to load imbalances when PCCM2 runs on 512 processors [9]. This scheme is incorporated in the production PCCM2, and is used in the performance studies reported in the next section. Figure 2 illustrates the impacts of load balancing.

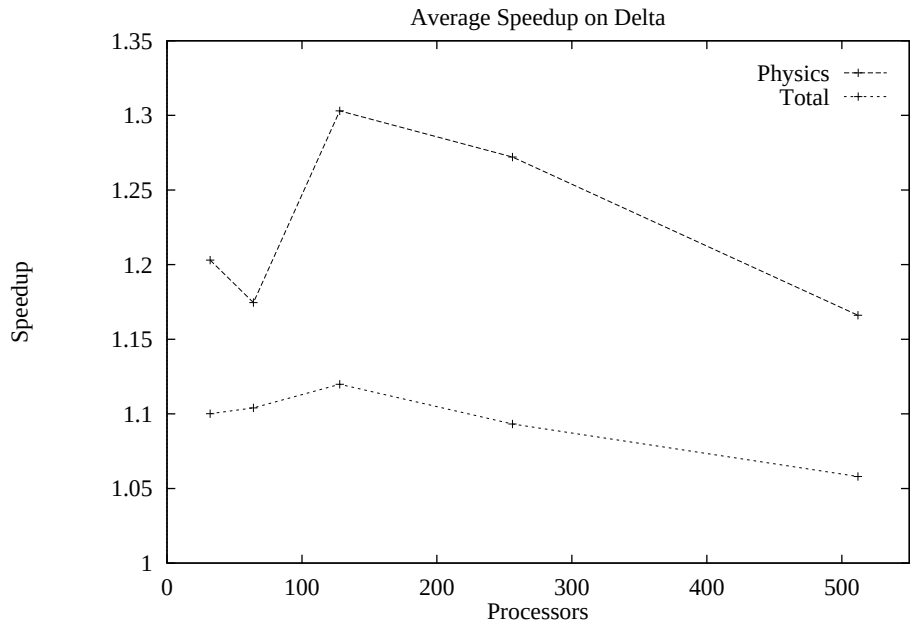

FIG. 2. *Factor of improvement from load balancing on the Intel DELTA, as a function of number of processors. Improvements are relative to a version of PCCM2 that does not incorporate load balancing, and are given for all of PCCM2 ("Total") and for just the physics component.*

6. Performance Studies. We performed a wide variety of experiments to measure and characterize PCCM2 performance. As noted above, performance of spectral and semi-Lagrangian transport codes depends significantly on the parallel algorithms, domain decomposition, and communication protocols used. The results presented here use the double transpose parallel FFT, distributed LT, and load balancing algorithms described in preceding sections. A square or almost square processor mesh was used in most cases, as experiments indicated that these generally gave the best performance. Communication protocols are for the most part those identified as efficient in studies with PSTSWM [10, 11, 33].

6.1. Computers. Table 1 describes the two parallel computer systems on which experiments were performed: the Intel Paragon XP/S MP 150 and the IBM SP2. These systems have similar architectures and programming models, but vary considerably in their communication and computational capabilities. Our values for message startup time (t_s) and per-byte transfer time (t_b) are based on the minimum observed times for swapping data between two nodes using our communication routines, which incorporate extra subroutine call overhead and communication logic relative to low-level libraries. Hence, they represent achievable, although not necessarily typical, values. The linear (t_s, t_b) parameterization of communication costs is a good fit to observed data for the Paragon but not for the SP2. The MBytes/second measure is bidirectional bandwidth: approximately twice $1/t_b$. The computational rate (Mflop/sec) is the maximum ob-

TABLE 1

Parallel Computers Used in Empirical Studies, Characterized by Operating System Version, Microprocessor, Interconnection Network, Maximum Machine Size Used in Experiments (N), Message Startup Cost (t_s), Per-Byte Transfer Cost (t_b), and Achieved Per-Processor Mflop/Sec at Single and Double Precision

Name	OS	Processor	Network	N
Paragon	SUNMOS 1.6.5	i860XP	16×64 mesh	1024
SP2	AIX + MPL	Power 2	multistage crossbar	128

Name	t_s (μsec)	t_b (μsec)	MB/sec (swap)	MFlop/sec	
				Single	Double
Paragon	72	0.007	282	11.60	8.5
SP2	70	0.044	45	44.86	53.8

served by running the PSTSWM testbed code [33] on a single processor for a variety of problem resolutions, and so is an achieved peak rate rather than a theoretical peak rate.

N in Table 1 and the X axis for all figures refer to the number of nodes, not the number of processors. This distinction is important because the Paragon used in these studies has three processors per node. In our experiments, one processor per node was always used as a dedicated message coprocessor, a second processor was used for computation, and the third processor was idle. (Some modifications to the source code are required in order to exploit the additional compute processor on the Paragon. These modifications are currently ongoing.) The Paragon measurements used the SUNMOS operating system developed at Sandia National Laboratories and the University of New Mexico, which currently provides better communication performance than the standard Intel OSF operating system. Interprocessor communication on the Paragon was performed by using the SUNMOS native communication commands, and on the SP2 by using the MPL communication library.

Table 1 gives single processor performance data at both single-precision (32-bit floating point values) and double precision (64-bit) arithmetic. Double precision is significantly slower on the 32-bit i860 processor and faster on the 64-bit Power 2 processor. Below, we give multiprocessor performance results for both single and double precision. We shall see that the SP2 generally performs better at single precision on multiple processors, no doubt because of reduced communication volume when sending 32-bit values.

6.2. Performance Results. Figures 3–5 and Table 2 present PCCM2 performance at different resolutions (T42 and T170, both with 18 vertical levels), on different numbers of nodes, and at both single and double precision. We show both execution times, expressed as elapsed seconds per model day, and sustained computational rate, expressed both as Gflop/sec and as Mflop/sec/node. Only a small number of data points are available at T170 resolution because of memory limitations. In all cases, our measurements ignore initialization time, as this is not relevant to model performance for long simulations. The computational rates are based on CCM2 operation counts

TABLE 2
Elapsed time per model day and computational rate at T170 resolution on the Paragon and SP2 for single and double precision

Name	Nodes	Time/model day (sec)	Computational Rate	
			Gflop/sec	Mflop/sec/node
Double Precision				
Paragon	512	1510	1.71	3.3
	1024	814	3.18	3.1
SP2	128	1092	2.27	18.5
Single Precision				
Paragon	1024	525.6	4.93	4.8
SP2	64	1606	1.61	25.2
	128	1077	2.40	18.8

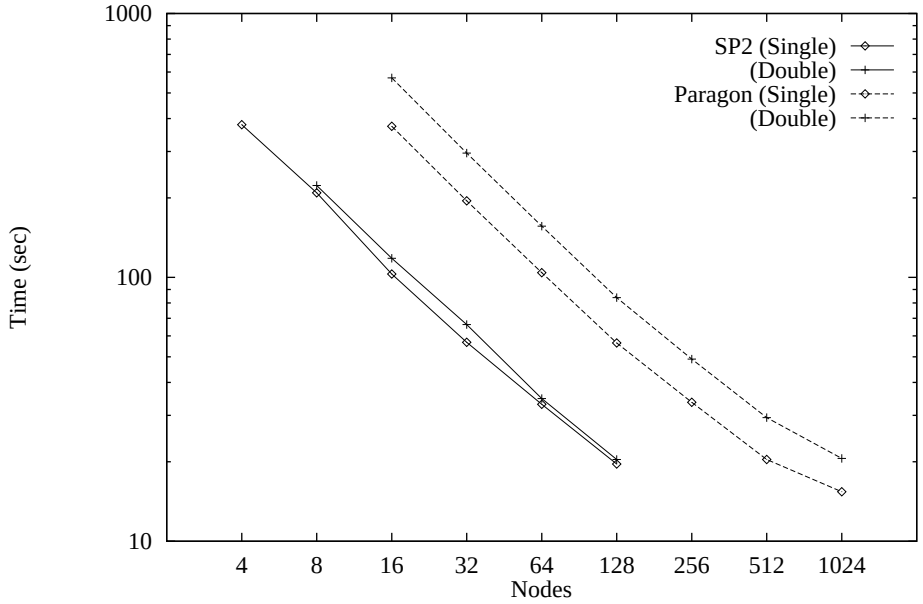

FIG. 3. *Elapsed time per model day as a function of node count at T42 resolution on the Paragon and SP2, for single and double precision.*

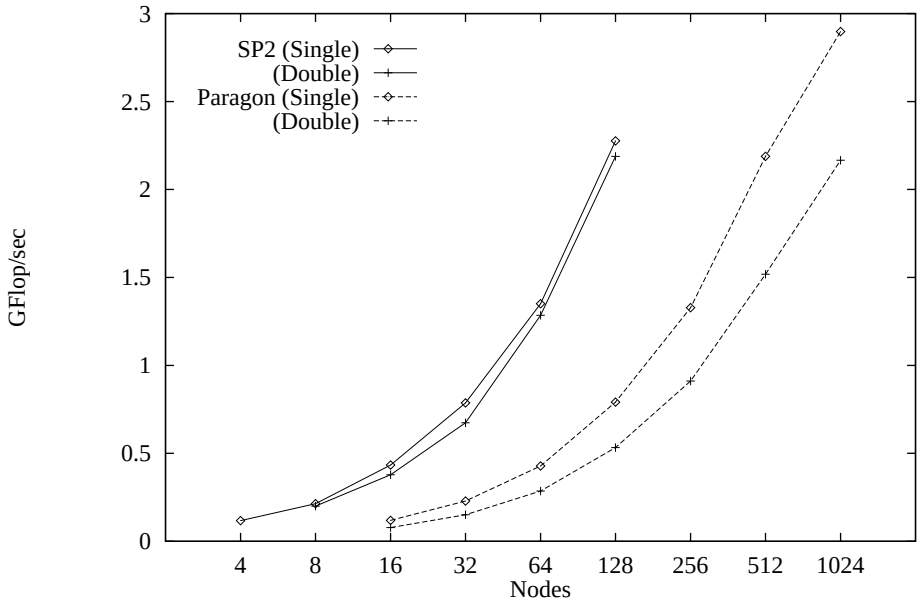

FIG. 4. *Computational rate in Gflop/sec as a function of node count at T42 resolution on the Paragon and SP2, for single and double precision.*

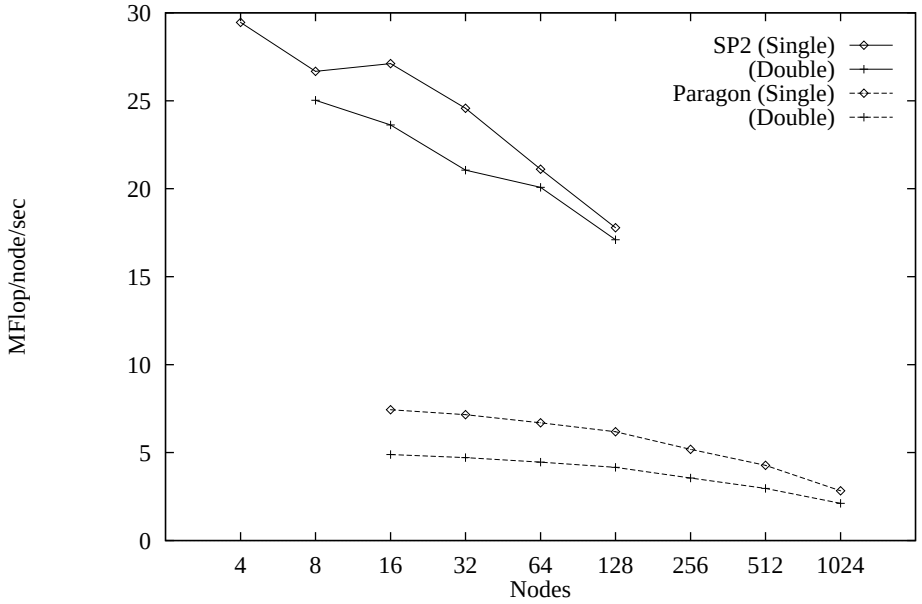

FIG. 5. *Computational rate in Mflop/sec/node as a function of node count at T42 resolution on the Paragon and SP2, for single and double precision.*

obtained using the hardware performance monitor on a Cray C90.

Note that both the single (32 bit) and double (64 bit) precision results are of interest. For large problem resolutions, the increased accuracy of double precision may be needed, and hence the double precision results are more meaningful. For smaller problem sizes, single precision is generally considered to be sufficient, and hence in this case the single precision data are more significant.

These results permit several general conclusions regarding the performance of PCCM2 on the Paragon and SP2. At T42 resolution, the model scales well on the SP2 up to 128 nodes and on the Paragon up to 512 nodes. The 2.2 Gflop/sec rate (double precision) achieved on the 128-node IBM SP2 and the 1024-node Paragon compares well with that achieved on a Cray Y-MP/8 (1.2 Gflop/sec) but is less than that achieved on a Cray C90/16 (about 6 Gflop/sec). The paucity of data at T170 prevents a complete analysis, but available data indicate improved scalability on the Paragon. At T170 (double precision) the Paragon computational rate increases to 3.2 Gflop/sec on 1024 nodes while the 128 node SP2 computational rate remains unchanged. This compares with 5.3 Gflop/sec on the C90/16.

For T42, PCCM2 in single precision executes about 3.0 times faster on the SP2 than on the Paragon with the same number of nodes. In double precision, the SP2 is about 4.5 times faster than the Paragon. As Figure 5 makes clear, these results are primarily due to the SP2's faster processors. Paragon performance can be improved somewhat by using the second compute processor. Preliminary experiments indicate that this may improve single-node performance by around 30 percent, which on large numbers of processors might represent a 10 to 20 percent improvement in overall performance.

We see that on both machines, execution rates are consistently higher at single precision than at double precision. However, on the SP2 the difference is small on larger numbers of nodes. These results can be explained as follows. On the Paragon, the use of double precision rather than single precision arithmetic increases both computation and communication time. On the SP2, double precision floating point computation is faster than single precision since the hardware does all computations in double and then truncates to a single result. (Preliminary experiments suggest that PCCM2 single precision performance can be improved by about 10 percent by using a compiler flag that prevents this truncation for intermediate results; however, the effect of this optimization on reproducibility has not yet been determined.) So for the SP2, double precision reduces computation costs (as the processor is faster at double precision) but increases communication costs (as the data volume increases).

The per-node Mflop/sec data presented in Figure 5 show that neither the i860 nor Power 2 microprocessor achieves outstanding performance. (Their peak performance for double precision computation is 75 and 256 Mflop/sec, respectively.) The percent of peak performance achieved on a single processor depends on the way the program utilizes the cache for memory access and on the ability of the compiler to optimize memory access and computation. The PCCM2 code is structured so that vector operations can be easily recognized, but efficient execution for cache based machines can require different constructs. The development of program structuring techniques which

are able to better exploit microprocessor-based MPPs must be an urgent topic of future research. The per-node Mflop/sec data are also influenced by communication costs and load imbalances, both of which tend to increase relative to computation costs as the number of nodes increases.

6.3. Additional Results. Table 3 shows how much time is spent in different PCCM2 components, while Table 4 shows PCCM2 average message sizes. In each case, data is given for a range of problem sizes and node counts. For brevity, only results from the SP2 or the Paragon are given in each table, as indicated in the captions. Qualitative results for one platform are indicative of results for the other.

The time breakdown information (Table 3) provides insights into the scalability of different parts of the model. The table indicates how much time was spent in different parts of the model, expressed both as absolute time per time step and as a percentage of total time. These data represent the average across time steps for each of a number of regions. The table also indicates the aspect ratio chosen in each case: a term $P \times Q$ indicates that the latitude dimension is decomposed over P nodes and the longitude dimension is decomposed over Q nodes, or, equivalently, that P nodes are allocated to each longitude and Q nodes to each latitude. The extreme aspect ratios chosen for some of the T170 runs are due to memory limitations. An error in PCCM2 (since corrected) meant that certain aspect ratios had exaggerated memory requirements.

Indentation in Table 3 indicates subregions within a larger region. Hence, region *Main* represents the entire model, and *Scan1*, *SLT*, *Spectral*, and *Scan2* represent disjoint subsets of the entire model.

Region *Scan1* includes physics, the forward FFTs, and part of the transform calculations into spectral space (the remainder are in region "Spectral"). Subregion *Physics* includes all model physics plus the calculation of the non-linear terms in dynamics tendencies. Region *SLT* represents the semi-Lagrangian transport. Subregion *Initialization* corresponds to the code that sets up the overlap regions.

Region *Spectral* represents various computations performed in spectral space: Gaussian quadrature, completion of the semi-implicit time step, and horizontal diffusion calculations. Finally, region *Scan2* represents the inverse transform from spectral space to grid space, and the final computation of global integrals of mass and moisture integrals for the time step.

An examination of how the distribution of time between different regions changes with the number of nodes (N) provides insights into the scalability of different parallel algorithms. The proportion of time spent in *SLT* increases significantly with Q, indicating poor scalability. This is due to high communication requirements, as will be seen below. *Spectral* also increases when the number of nodes in the N/S direction (P) increases or when the model size increases, again because of increased communication. The proportion of time spent in *Physics* decreases with N, and the FFT regions increase only slightly, because their communication costs are relatively low. Notice that a smaller proportion of total time is spent in *Physics* at double precision than at single precision; this is because of the SP2's better double precision compute performance.

Further insights in the scaling behavior of PCCM2 is provided by Table 4, which

TABLE 3

Total Time, and Percentage of Time, Spent in Different Parts of PCCM2 on the SP2, for Different Problem Sizes and Node Counts, and for both Single and Double Precision. Times are in Msec, and are for a Single Time Step

Region	T42			T170	
	N=32	N=64	N=128	N=64	N=128
Single Precision	8x4	8x8	16x8	32x2	16x8
Main (Time in msec)	788.4	459.0	272.4	5575.8	3740.6
Scan1	402.8	214.8	118.5	2206.9	1167.3
Physics	361.9	187.6	100.8	1920.3	950.3
Forward FFT	24.2	16.6	9.1	155.4	111.4
SLT	261.0	175.4	101.7	1337.2	1323.3
Initialization	150.3	107.2	66.5	302.2	740.7
Spectral	44.3	24.3	27.1	1186.7	738.4
Scan2	79.5	44.1	25.0	827.6	501.7
Backward FFT	28.2	18.8	10.6	183.2	134.6
Main (Percent of total)	100.0	100.0	100.0	100.0	100.0
Scan1	51.1	46.8	43.5	39.6	31.2
Physics	45.9	40.9	37.0	34.4	25.4
Forward FFT	3.1	3.6	3.3	2.8	3.0
SLT	33.1	38.2	37.3	24.0	35.4
Initialization	19.1	23.4	24.4	5.4	19.8
Spectral	5.6	5.3	9.9	21.3	19.7
Scan2	10.1	9.6	9.2	14.8	13.4
Backward FFT	3.6	4.1	3.9	3.3	3.6
Double Precision	8x4	8x8	16x8	N/A	64x2
Main (Time in msec)	920.1	482.5	283.3		4033.4
Scan1	469.0	221.1	119.6		1085.6
Physics	412.3	183.5	98.0		855.4
Forward FFT	35.1	23.7	12.9		112.2
SLT	307.1	178.5	103.6		1277.3
Initialization	178.8	110.4	68.7		741.3
Spectral	53.2	27.0	31.4		1139.3
Scan2	90.1	55.6	28.4		511.2
Backward FFT	42.6	27.8	14.5		129.9
Main (Percent of total)	100.0	100.0	100.0		100.0
Scan1	51.0	45.8	42.2		26.9
Physics	44.8	38.0	34.6		21.2
Forward FFT	3.8	4.9	4.6		2.8
SLT	33.4	37.0	36.6		31.7
Initialization	19.4	22.9	24.2		18.4
Spectral	5.8	5.6	11.1		28.2
Scan2	9.8	11.5	10.0		12.7
Backward FFT	4.6	5.8	5.1		3.2

TABLE 4

Average Message Size in Bytes, and Average Number of Messages, for Different Node Counts at T42 Resolution (Single Precision) on the Paragon

Region	Nodes					
	32 (8x4)		64 (8x8)		128 (16x8)	
Main	6295	113	2755	150	1762	161
Scan1	9486	23	3445	37	1779	36
Physics	7200	16	3600	16	3600	8
SLT	3106	30	2044	33	1717	34
Init	4137	22	2788	24	2480	23
Spectral	5856	31	2928	31	2370	42
Scan2	7494	28	2603	48	1264	49

Region	Nodes					
	256 (16x16)		512 (32x16)		1024 (32x32)	
Main	764	216	814	236	428	345
Scan1	558	61	281	61	83	111
Physics	1800	8	1800	4	900	4
SLT	1221	37	2037	50	1622	59
Init	1788	25	2781	36	2168	44
Spectral	1421	35	1332	41	906	36
Scan2	417	81	206	82	65	138

gives the average message size and number of messages per time step, as a function of the number of nodes and problem size. (Note that the forward and backward FFT are included in *Scan1* and *Scan2*, respectively.) These data are for the Paragon; on the SP2 the number of messages increases by 50 percent, because a two-message protocol is used that requires that each message exchange be preceded by a synchronization message from one of the two nodes involved in the exchange to the other. The use of this two-message protocol is found to improve performance significantly on the SP2.

Table 4 explains some of the trends identified in Table 3, and provides additional information about what happens as the number of nodes increases beyond 128. We see that average message size drops rapidly in the FFTs, roughly halving each time the number of nodes is doubled. A similar trend is visible in Physics; this corresponds to the communication performed for load balancing. In contrast, message sizes in the SLT stay fairly constant, and eventually completely dominate the total communication volume. Examining message counts, we see that the number of messages performed in the FFTs increases with the number of nodes, and becomes a significant contributor to communication costs on large number of nodes. This suggests that there may be advantages to using either a distributed FFT or a $\mathcal{O}(\log P)$ transpose FFTs [11] when solving small problems on large number of nodes.

7. Summary. PCCM2 is a production climate model designed to execute on massively parallel computer systems. It is functionally equivalent to the shared-memory

parallel CCM2 model on which it is based, has successfully completed ten year simulations, and is ready for use in scientific studies.

PCCM2 has been developed as part of a research project investigating parallel algorithms for atmospheric circulation models. The particular algorithms incorporated in PCCM2 represent pragmatic tradeoffs between choices known from our research efforts to be optimal in different environments, and the software engineering constraints involved in adapting an existing vector code to obtain a robust, portable model. Performance studies indicate that the resulting code performs well on a range of different parallel computers, problem sizes, and number of nodes. Overall performance is impressive, but would be significantly better if uniprocessor performance could be improved. Parallel scalability on large numbers of nodes is limited by the high communication requirements of the parallel semi-Lagrangian transport algorithm.

Current research and development efforts are working both to improve PCCM2 and to apply it to climate modeling problems. In two separate efforts, PCCM2 is being coupled with ocean models to obtain an earth system model suitable for long duration climate simulations. Work is also proceeding to develop a fully semi-Lagrangian, reduced grid version. Another area of investigation is parallel I/O. Both the Paragon and SP2 versions of PCCM2 use parallel I/O techniques to achieve sufficient I/O bandwidth to the parallel file systems. Asynchronous operations are used to allow the overlapping of model output with computation. In future work, we expect to generalize these techniques to use a common interface on both machines.

Finally, we note that in our work to date, we have focused on the development of explicitly parallel codes in which parallel algorithms are specified in terms of low-level message-passing operations. We choose this approach both because it offers portability and performance on a range of parallel computers, and because it provides us the flexibility required to investigate a wide range of algorithmic alternatives. Ultimately, we expect that the more efficient algorithms identified in this project may be specified by using more convenient notations.

Acknowledgments. This work was supported by the CHAMMP program of the Office of Health and Environmental Research, Environmental Sciences Division, of the U.S. Department of Energy under Contract W-31-109-Eng-38 with the University of Chicago and under Contract DE-AC05-84OR21400 with Lockheed-Martin Energy Systems. We are grateful to Ray Flanery, Jace Mogill, Ravi Nanjundiah, Dave Semeraro, Tim Sheehan, and David Walker for their work on various aspects of the parallel model. We also acknowledge our colleagues at NCAR for sharing codes and results throughout this project. We are grateful to Wu-Sun Cheng and David Soll for their help with performance tuning on the IBM SP2.

This research used the Intel Paragon multiprocessor system of the Oak Ridge National Laboratory Center for Computational Sciences (CCS), funded by the Department of Energy's Mathematical, Information, and Computational Sciences (MICS) Division of the Office of Computational and Technology Research; the Intel Paragon multiprocessor system at Sandia National Laboratories; the Intel Touchstone DELTA System operated by Caltech on behalf of the Concurrent Supercomputing Consortium; the IBM

SP2 at Argonne National Laboratory; and the IBM SP2 of the NAS Program at the NASA Ames Research Center.

REFERENCES

[1] S. Barros and T. Kauranne, On the parallelization of global spectral Eulerian shallow-water models, in *Parallel Supercomputing in Atmospheric Science: Proceedings of the Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology*, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 36–43.

[2] W. Bourke, An efficient, one-level, primitive-equation spectral model, *Mon. Wea. Rev.*, 102 (1972), pp. 687–701.

[3] D. Dent, The ECMWF model on the Cray Y-MP8, in *The Dawn of Massively Parallel Processing in Meteorology*, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 1990.

[4] D. Dent, The IFS Model: A Parallel Production Weather Code, submitted to *Parallel Computing*, 1995.

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

[6] J. B. Drake, R. E. Flanery, I. T. Foster, J. J. Hack, J. G. Michalakes, R. L. Stevens, D. W. Walker, D. L. Williamson, and P. H. Worley, The message-passing version of the parallel community climate model, in *Parallel Supercomputing in Atmospheric Science*, World Scientific Publishing, Singapore, 1993, pp. 500–513.

[7] E. Eliasen, B. Machenhauer, and E. Rasmussen, On a numerical method for integration of the hydrodynamical equations with a spectral representation of the horizontal fields, Report No. 2, Institut for Teoretisk Meteorologi, Kobenhavns Universitet, Denmark, 1970.

[8] I. T. Foster, W. Gropp, and R. Stevens, The parallel scalability of the spectral transform method, *Mon. Wea. Rev.*, 120 (1992), pp. 835–850.

[9] I. T. Foster and B. Toonen, Load balancing in climate models, *Proc. 1994 Scalable High-Performance Computing Conf.*, IEEE Computer Society Press, Los Alamitos, CA, 1994.

[10] I. T. Foster and P. H. Worley, Parallelizing the spectral transform method: A comparison of alternative parallel algorithms, in *Parallel Processing for Scientific Computing*, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 100–107.

[11] I. T. Foster and P. H. Worley, Parallel algorithms for the spectral transform method, submitted for publication. Also available as Tech. Report ORNL/TM–12507, Oak Ridge National Laboratory, Oak Ridge, TN, May 1994.

[12] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[13] H. Franke, P. Hochschild, P. Pattnaik, J.-P. Prost, and M. Snir, MPI-F: Current status and future directions, *Proc. Scalable Parallel Libraries Conference*, Mississippi State, October, IEEE Computer Society Press, 1994.

[14] U. Gärtel, W. Joppich, and A. Schüller, Parallelizing the ECMWF's weather forecast program: The 2D case, *Parallel Computing*, 19 (1993), pp. 1413–1426.

[15] J. J. Hack, B. A. Boville, B. P. Briegleb, J. T. Kiehl, P. J. Rasch, and D. L. Williamson, Description of the NCAR Community Climate Model (CCM2), NCAR Technical Note TN-382+STR, National Center for Atmospheric Research, Boulder, Colo., 1992.

[16] J. J. Hack, J. M. Rosinski, D. L. Williamson, B. A. Boville, and J. E. Truesdale, Computational design of the NCAR Community Climate Model, to appear in this issue of *Parallel Computing*, 1995.

[17] S. Hammond, R. Loft, J. Dennis, and R. Sato, Implementation and performance issues of a massively parallel atmospheric model, *Parallel Computing*, this issue.

[18] T. Kauranne and S. Barros, Scalability estimates of parallel spectral atmospheric models, in *Par-*

allel Supercomputing in Atmospheric Science, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 312–328.

[19] R. D. Loft and R. K. Sato, Implementation of the NCAR CCM2 on the Connection Machine, in *Parallel Supercomputing in Atmospheric Science*, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 371–393.

[20] J. Michalakes, Analysis of workload and load balancing issues in NCAR Community Climate Model, Technical Report ANL/MCS-TM-144, Argonne National Laboratory, Argonne, Ill., 1991.

[21] J. Michalakes and R. Nanjundiah, Computational load in model physics of the parallel NCAR Community Climate Model, Technical Report ANL/MCS-TM-186, Argonne National Laboratory, Argonne, Ill., 1994.

[22] S. A. Orszag, Transform method for calculation of vector-coupled sums: Application to the spectral form of the vorticity equation, *J. Atmos. Sci.*, 27, 890–895, 1970.

[23] M. Pease, An adaptation of the fast Fourier transform for parallel processing, *J. Assoc. Comput. Mach.*, 15 (1968), pp. 252–264.

[24] R. B. Pelz and W. F. Stern, A balanced parallel algorithm for spectral global climate models, in *Parallel Processing for Scientific Computing*, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 126–128.

[25] J.G. Sela, Weather forecasting on parallel architectures, to appear in this issue of *Parallel Computing*, 1995.

[26] P. N. Swarztrauber, Multiprocessor FFTs, *Parallel Computing*, 5 (1987), pp. 197–210.

[27] P. N. Swarztrauber, W. L. Briggs, R. A. Sweet, V. E. Henson, and J. Otto, Bluestein's FFT for arbitrary n on the hypercube, *Parallel Computing*, 17 (1991), pp. 607–618.

[28] R. A. van de Geijn, On Global Combine Operations, LAPACK Working Note 29, CS-91-129, April 1991, Computer Science Department, University of Tennessee.

[29] D. W. Walker, P. H. Worley, and J. B. Drake, Parallelizing the spectral transform method. Part II, *Concurrency: Practice and Experience*, 4 (1992), pp. 509–531.

[30] W. Washington and C. Parkinson, *An Introduction to Three-Dimensional Climate Modeling*, University Science Books, Mill Valley, Calif., 1986.

[31] D. L. Williamson and P. J. Rasch, Two-dimensional semi-Lagrangian transport with shape-preserving interpolation, *Mon. Wea. Rev.*, 117, 102–129, 1989.

[32] P. H. Worley and J. B. Drake, Parallelizing the spectral transform method, *Concurrency: Practice and Experience*, 4 (1992), pp. 269–291.

[33] P. H. Worley and I. Foster, 1994. Parallel spectral transform shallow water model: A runtime-tunable parallel benchmark code, in *Proc. Scalable High Performance Computing Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., pp. 207–214.