ELSEVIER

Parallel Computing 21 (1995) 1593-1619

PARALLEL
COMPUTING

Implementation and performance issues of a massively parallel atmospheric model

Steven W. Hammond [*], Richard D. Loft, John M. Dennis, Richard K. Sato

Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307, USA.

Received 7 April 1995; revised 22 June 1995

Abstract

We present implementation and performance issues of a data parallel version of the National Center for Atmospheric Research (NCAR) Community Climate Model (CCM2). We describe automatic conversion tools used to aid in converting a production code written for a traditional vector architecture to data parallel code suitable for the Thinking Machines Corporation CM-5. Also, we describe the 3-D transposition method used to parallelize the spherical harmonic transforms in CCM2. This method employs dynamic data mapping techniques to improve data locality and parallel efficiency of these computations. We present performance data for the 3-D transposition method on the CM-5 for machine size up to 512 processors. We conclude that the parallel performance of the 3-D transposition method is adversely affected on the CM-5 by short vector lengths and array padding. We also find that the CM-5 spherical harmonic transforms spend about 70% of their execution time in communication. We detail a transposition-based data parallel implementation of the semi-Lagrangian Transport (SLT) algorithm used in CCM2. We analyze two approaches to parallelizing the SLT, called the departure point and arrival point based methods. We develop a performance model for choosing between these methods. We present SLT performance data which shows that the localized horizontal interpolation in the SLT takes 70% of the time, while the data remapping itself only require approximately 16%. We discuss the importance of scalable I/O to CCM2, and present the I/O rates measured on the CM-5. We compare the performance of the data parallel version of CCM2 on a 32-processor CM-5 with the optimized vector code running on a single processor Cray Y-MP. We show that the CM-5 code is 75% faster. We also give the overall performance of CCM2 running at higher resolutions on different numbers of CM-5 processors. We conclude by discussing the significance of these results and their implications for data parallel climate models.

[*] Corresponding author. Email: hammond@ncar.ucar.edu

Keywords: Atmospheric general circulation modeling; Climate modeling; Data parallelism; Spectral transform; Semi-Lagrangian transport

1. Introduction

In recent years the numerical simulation of climate has emerged as one of the "grand challenge" problems in high performance computing [17]. Simultaneously, the explosive development of massively parallel processor (MPP) systems in the last decade has been widely viewed as a mechanism by which a new level of performance for grand challenge applications could be obtained. In order to take advantage of MPP technology in the climate modeling arena, the U.S. Department of Energy (DOE) initiated the Computer Hardware, Applied Mathematics and Model Physics (CHAMMP) program in 1990 [2]. The ultimate goal of CHAMMP was to produce an advanced climate model capable of a teraflop.

The initial goal of the model development component of the CHAMMP initiative was to port existing state-of-the-science climate models to MPP platforms. Under this component NCAR has participated in a joint project with Argonne National Laboratory (ANL) and Oak Ridge National Laboratory (ORNL) to port CCM2. Our group at NCAR developed a data parallel version of CCM2. The related effort at ANL and ORNL simultaneously focused on developing an efficient message passing parallel version of the model [6].

In this paper we present the performance results from the data parallel implementation of CCM2 for the CM-5. We begin with a overview of the CCM2 model in Section 2, followed by overall performance results in Section 3. At the outset, we had two goals in developing the data parallel version. The first was to develop a framework which would allow this data parallel model to track future developments of the base (Cray) code. This was achieved by developing tools to automate much of the code translation described in Section 4. Secondly, we wanted to develop an efficient and scalable data parallel version of the model for MPP systems. This was realized by developing data parallel implementations of the 3-D transposition algorithm which rearranges model data to achieve data locality for the spectral transform (Section 5) and semi-Lagrangian Transport (SLT) algorithms (Section 6). Section 7 details our experiences with the difficult problem of efficient and scalable I/O on MPP systems. Finally, we advance some conclusions about what was learned from this effort in Section 8.

2. Description of CCM2

CCM2 is an atmospheric general circulation model that has been developed at NCAR and provided to atmospheric scientists for over a decade [1,8,9]. Readers interested in an introductory discussion of the scientific issues of climate modeling are referred to [19]. The Cray version of CCM2 consists of approximately 40,000 lines of Fortran 77 organized as 232 subroutine and include files which have been optimized for vector processor systems. The vertical and temporal aspects of the model are represented by finite-difference approximations. The spherical harmonic transform (spectral transform) method is employed to compute the dry dynamics of CCM2 [8,19]. It consists of computing the spherical harmonic function coefficient representation of the atmospheric state variables through a series of highly non-local operations. The set of spectral coefficients is typically truncated in some fashion to avoid aliasing. Horizontal derivatives and linear terms involving these variables are calculated in spectral space and are combined to form the dynamical right hand sides in spectral coefficient space. This operation is completely local in spectral coefficient space. The data are then transformed back into grid space where they are used to update the model variables.

For accuracy reasons, the spectral transform calculations are performed on a polar grid which is irregularly spaced in latitude, called a Gaussian polar grid. The calculation of non linear terms in the equations of motion are carried out on this grid, as are the physical parameterizations of CCM2. These "physics" computations involve only the vertical column above each grid point and are thus numerically independent of each other in the horizontal direction. Finally, trace gases, including water vapor, are transported by the wind fields using a shape preserving SLT scheme [15,20]. This transport involves indirect addressing on the Gaussian polar grid. The SLT is discussed in more detail in Section 6.

For spectral climate models such as CCM2 it is canonical to denote the resolution by the truncation wave number and the number of vertical layers in the model discretization. For example, a spectral atmospheric model that uses a 128 longitude by 64 latitude grid and 18 vertical levels is called a T42L18 model. The "T" indicates a triangular truncation of the spherical harmonic coefficients, 42 indicates the maximum longitudinal wavenumber used in the model, and the "L" denotes the number of vertical levels. At present T42L18 is the production resolution of CCM2. Table 1 shows the grid size, nominal grid point spacing, and model time step for three CCM2 resolutions.

3. Overall performance

In this section we discuss the aggregate performance of the data parallel CCM2 model at different resolutions and on different numbers of CM-5 processors. These

Table 1
Typical CCM2 resolutions, grid spacings, and time steps

Model resolution	Horizontal grid size	Nominal grid spacing	Time step
T42L18	64×128	2.8°	20.0 min.
T85L18	128×256	1.4°	10.0 min.
T170L18	256×512	0.7°	5.0 min.

results are compared to the current computational platform at NCAR, the Cray Y-MP. It should be noted that the CCM2 models runs quite efficiently on shared memory multiprocessors such as the Cray Y-MP. For reference, the single processor CCM2 performance on the Cray Y-MP is 155 Mflops. The code also multitasks efficiently, attaining just over 1.2 Gflops on a Cray Y-MP/8.

We have chosen to report the performance of the total model in terms of seconds per simulated model day because we believe it is a more reliable measure of relative performance between computers than MFlops since operation counts can differ widely between different computer systems and implementations. All computations discussed here are performed in 64-bit precision. We measure the time per simulated model day for low resolutions by subtracting the time for a one model day run from a 21 model day run and dividing the difference by 20. For the higher resolution runs we measure the performance by subtracting the time for a five model day run from a ten model day run and dividing the difference by five. The model execution time has been divided into categories including FFT, DYNAMICS, semi-Lagrange Transport (SLT), adjustment physics (APHYS) and tendency physics (TPHYS).

In Fig. 1 we show time for one day of simulation (72 times steps at model resolution T42L18) on one processor of a Cray Y-MP and five different sized CM-5s. The total time is shown at the top of each bar graph. The time spent in each category is shown next to the name. The time required for data remapping communications, if any, is shown in parentheses next to the category name. The time on the CM-5 is measured using the "elapsed" time provided by the CM timing utilities. This is a good approximation to wall clock time on a dedicated machine.

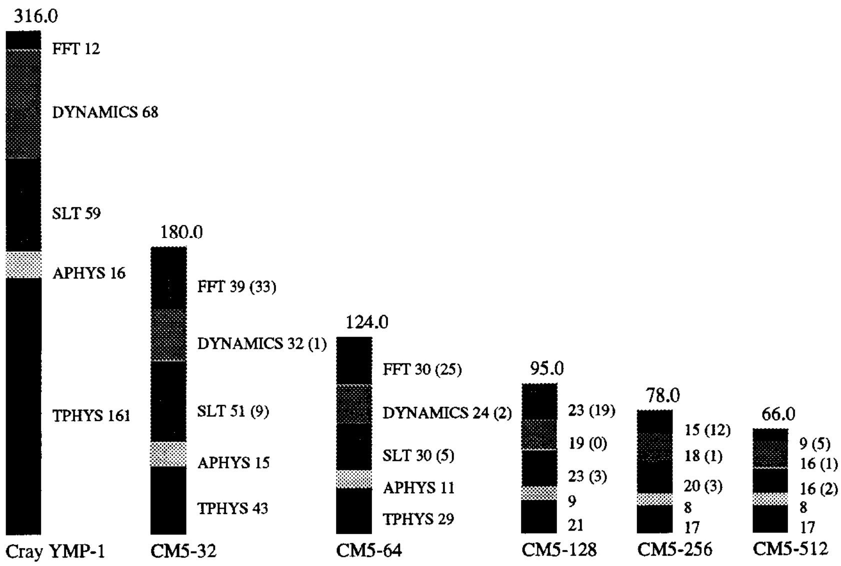

Fig. 1. Performance comparison of serial and data parallel CCM2 at T42L18 resolution.

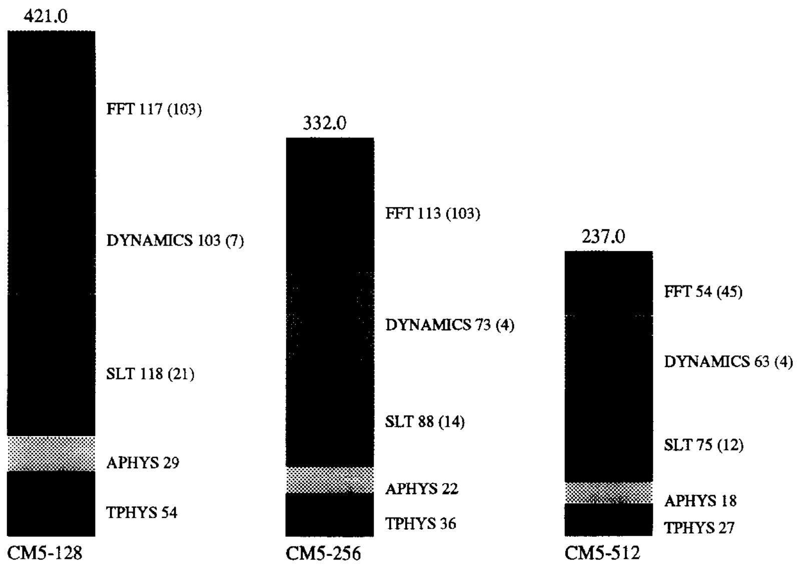

Fig. 2. Times for 1 model day at T85L18 resolution on the CM-5.

One model day takes 316 seconds on one processor of a Cray Y-MP and 180 seconds on the 32-processor CM-5. The most noticeable gain in performance in the CM-5 version is in the tendency physics (TPHYS) which is computationally intensive and does not require communication. Fig. 1 shows that for model resolution T42 the code does not scale particularly well going from 32 processors to 512 processors on the CM-5. Even though there are 16 times as many processors, the running time decreases by just less than a factor of three. At T42L18 there are insufficient computations to be done in some sections of the code to keep all 2048 vector units (4 per processor, 512 processors) busy all of the time.

In Fig. 2 we show time for one day of simulation (144 times steps) of CCM2 at resolution T85L18 on 128, 256, and 512 processors of a CM-5. It required 1.9 Gbytes, 2.5 Gbytes, and 3.7 Gbytes of memory respectively for these three different size machines[1]. In Fig. 3 we show time for one day of simulation (288 times steps) of CCM2 at resolution T170L18 on 512 processors of a CM-5. It requires 11.4 Gbytes of memory to run on 512 processors. This is the smallest configuration that was capable of running CCM2 at T170L18.

For highest resolution runs on larger numbers of processors the FFT, DYNAMICS and SLT part of the CCM2 take an increasing percent of the total time. This is because the physics calculations do not require interprocessor communication. At resolution T42 on 32 processors the physics takes 32% of the total time. At T170 on 512 processors the physics computations take only 12% of the total time.

[1] The total memory per processor on the CM5 at NCAR and at Los Alamos National Lab. is 32 Mbytes.

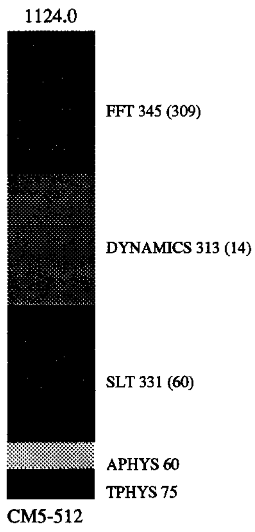

Fig. 3. Time for 1 model day at T170L18 resolution on the CM-5.

At higher resolutions the percentage of the execution time spent in communication increases significantly because the number of time steps per day is increasing but parts of the physics are only executed once each hour.

The data parallel version of CCM2 is currently being run in production at the T170L18 resolution on a 512 processor partition of the 1024 processor CM-5 at the Advanced Computing Laboratory at Los Alamos National Laboratory (LANL). Fig. 4 shows an instantaneous value of precipitable water (column integrated water) computed at 1 AM on January 1 of the first January for a simulation started September 1. The precipitable water is displayed as opacity using a linear grey scale. A one-year run requires approximately 120 hours of computer time using 512 processors. These simulations are part of a continuing study of the sensitivity of CCM2 to horizontal resolution [21].

4. Automated conversion of the CCM2 from F77 to CM Fortran

Manual conversion of a code as large and complex as CCM2 is a daunting task. The problem is compounded by the fact that CCM2 is continually evolving. Sections of the code are modified, added, or replaced for scientific experimentation. The physics sections of CCM2 are the largest in terms of lines of code, the most frequently modified, and fortunately, the most amenable to automated conversion. Any viable automated conversion must be able to translate the physics source quickly and easily.

We automated the translation process using a tool we developed called *fparser* [10,11] to supplement the capabilities of the Thinking Machines CMAX translation tool [3]. The fparser tool is a directive driven, recursive-descent parser used to

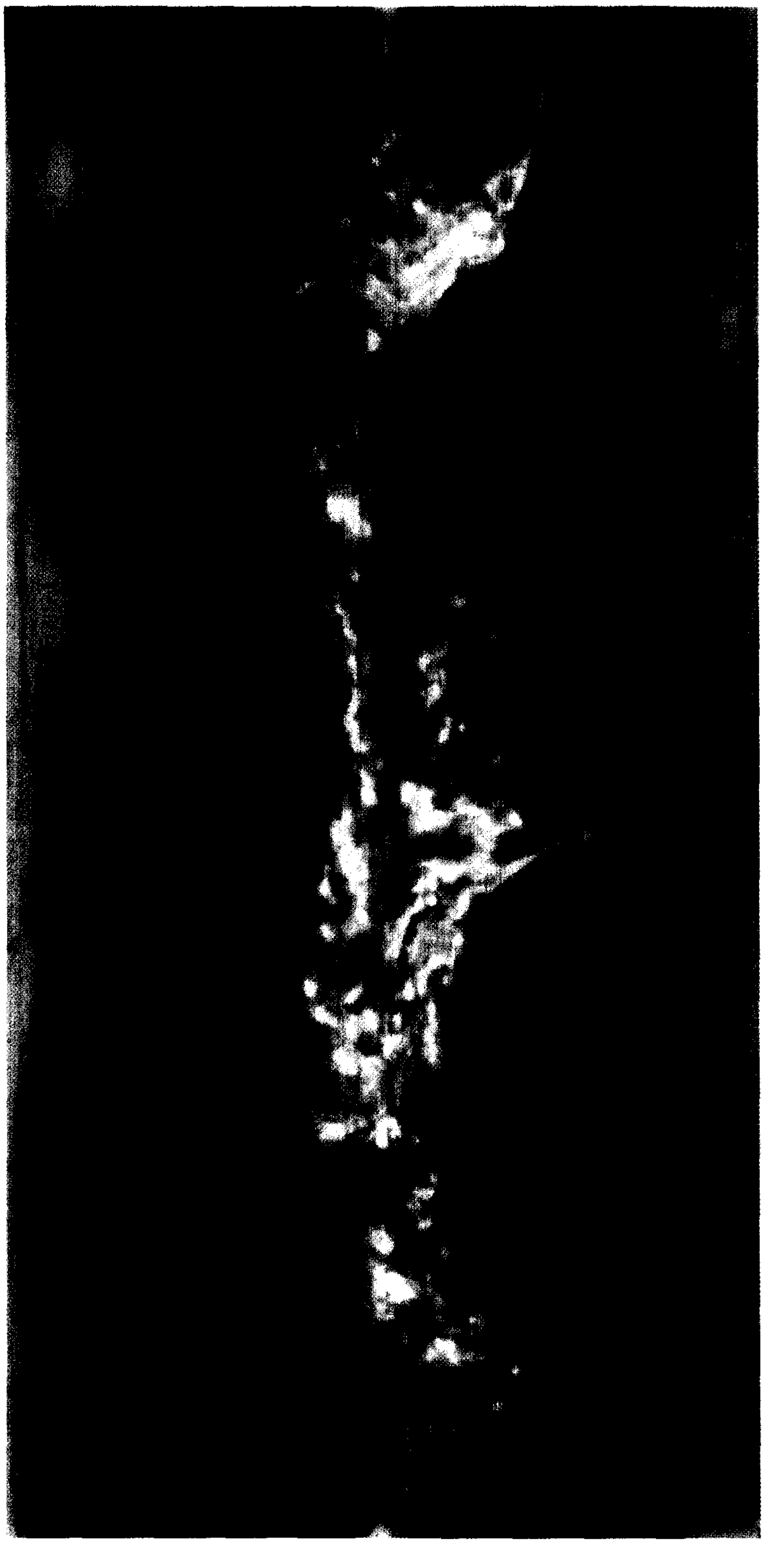

Fig. 4. Precipitable water in CCCM2 for a timestep in a simulated January.

overcome some limitations of CMAX (described below). It preprocesses Fortran 77 code and outputs scalable Fortran 77. Here we define scalable to mean that the data structures within the code contain enough elements to allow a large (or small) number of processors to execute the code efficiently. CMAX is the Connection Machine Automated Trans (X) lator supplied by Thinking Machines. It automatically converts Fortran 77 source into CM Fortran.

The Cray version of CCM2 is organized around a one dimensional latitude decomposition. In the physics routines, the fundamental data structure is a two dimensional level-longitude slice. The multi-tasked latitude loop is located at a high level in the code, outside of all the physics routines. This arrangement provides a satisfactory level of parallelism for multitasking on traditional vector multiprocessors with modest numbers of processors. However, it contains insufficient parallelism for MPPs. In a data parallel implementation of the code structure, at the standard resolution (T42L18), only the longitude axis with 128 elements can be executed in parallel.

Since latitude lines may also be operated on in parallel, the best approach for parallelizing the model to exploit a relatively large number of processors is to convert the fundamental data structure in the physics to a full three dimensional array (latitude by longitude by level). This forces one to push the latitude loop down inside *every* physics routine. Although CMAX has an automated loop pushing directive, it fares poorly in practice. It often produces unreadable, inefficient, and even incorrect code unless the subroutine is very simple.

Three other CMAX deficiencies were encountered in the translation of CCM2. First, we have found that it does a poor job of determining the most efficient array layout on its own. Second, it is common to encounter DO loops in which the results of array valued calculations are assigned to temporary scalar variables within the loop. Although CMAX is capable of promoting such scalar variables to arrays, it frequently promotes such loop temporaries to arrays with layouts that are inconsistent with the rest of the model. This causes unnecessary interprocessor communication that inhibits performance. Finally, it is desirable for performance reasons to permute the order of array indices and the nesting order of loops to achieve optimal data layouts. CMAX does not have this capability.

Using fparser preprocessing combined with CMAX enabled an automated port of the majority of the physics routines. We estimate that the use of automated translation tools reduced the amount of programmer hours to port CCM2 physics code by a factor of six. For example, the tendency physics (TPHYS) routines that previously took one programmer year to convert and debug took two months using the automated system. Finally, the performance of the automated physics code is comparable to or better than that of the previously hand ported versions.

5. Spherical harmonic transform method

As mentioned above, the spherical harmonic transform (spectral transform) method is employed to compute the dry dynamics of CCM2 [8,19]. It consists of computing the spherical harmonic function coefficient representation of the atmospheric state variables through a series of highly non-local operations. We give a brief summary of the major steps involved in the method to identify components of this method which must be considered for computational efficiency on MPP systems.

Numerically, the spherical harmonic transform method begins with a real to complex FFT in the longitudinal direction, producing a set of Fourier coefficients. Typically only two-thirds of the available wave numbers are kept after the FFT to prevent aliasing. Next the Fourier coefficients are symmetrized and anti-symmetrized about the equator to form distinct parity states. This step takes advantage of the parity identities of the associated Legendre polynomials (ALP's), reducing the storage required for the ALP's and reducing the operation count for the Legendre transform by a factor of two. Finally the spherical harmonic coefficient representation of the state variable is obtained by performing the Legendre transform, which amounts to performing multiple instances of matrix vector multiplies involving triangular matrices. The inverse steps that complete the spectral cycle are essentially identical: an inverse Legendre transform is followed by a reconstruction of the fields from the distinct parity states and a complex to real FFT.

This transform algorithm presents a special challenge for an MPP, as each of the operations are non-local along different axes, so interprocessor communication cannot be avoided. Furthermore, the number of spectral coefficients is approximately an order of magnitude less than the number of elements in the Gaussian polar grid. Therefore there is less parallelism to be exploited.

Here we describe a data parallel spherical harmonic transform method based on a scheme called the *transposition method* [12,13]. There have been other data parallel implementations of spectral transforms in atmospheric models based on parallel prefix operations [16]. In the transposition method, each computational step of the spectral transform (FFT and Legendre Transform) is performed in processor. Between each step, data is remapped to maximize locality. The notion of rearranging the data to improve data locality and performance for parallel spectral transforms has been known for some time. For example, the data flow of the original Cray implementation of the ECMWF Integrated Forecasting System (IFS) was organized around this principle [14].

For MPP's, it has also been demonstrated that 3-D transposition methods involving all axes of the state variables, have inherent scaling and communications advantages over other techniques in which the FFT and Legendre transforms are distributed across processors [13]. In addition, transposition methods also have significant practical advantages. First, since all transform computations are local, the results from the data mapping technique are bit for bit reproducible and machine size independent. This is an important requirement for climate modelers since climate simulations are sensitive to single bit perturbations of initial conditions. If the results depended on the hardware history of a particular run, then scientists can only verify results by exactly reproducing this history. Second, this method localizes computation. This technique also permits the use of efficient, widely available serial software libraries for the FFT and the Legendre transform, executing in parallel on each processor. Finally, the transposition method segregates the communication part of the problem from the computation. This simplifies the task of obtaining optimized communication libraries to accomplish the necessary data rearrangement.

In general, the data remappings in the 3-D transposition method are not simple matrix transposes. Load balancing or array padding considerations have lead to implementations in which data involved in the spectral transform method is rearranged in more complex ways. In this section we discuss the details of these mappings, and their impact on performance, from a data parallel perspective.

The first data parallel implementation of the 3-D transposition method in a spectral climate model was in CCM2 [12]. This transpose-based spectral code is written in CM Fortran but makes heavy use of the Connection Machine Scientific Software Library (CMSSL). In particular the complex to complex FFT, the matrix vector multiply, and the COMM-SEND communication routines from CMSSL [4,5] were used. Also, although not discussed here, we have developed a data parallel, multiple instance version of a radix 2,3,4,5, and general, real to complex FFT based on a serial code from FFTPACK [7,12]. Figs. 5 and 6 summarize the overall performance of the transposition method on the CM-5. In Fig. 5 we show the total time required to perform a complete transform cycle of a single 3-D variable for the T42L18, T85L18, and T17L18 resolutions.

The floating point performance of the spectral transform itself was obtained by dividing the operation count for each transform by the elapse time. For a grid with *LAT* latitudes and *LON* longitudes, the mathematical operation count of the power of two real to complex FFT implemented in the CMSSL is $5 \times (LON/2)$

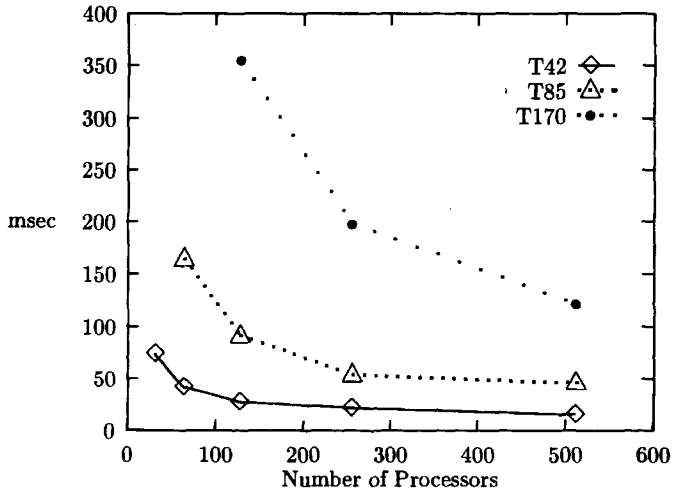

Fig. 5. Time in milliseconds required to complete a full spectral transform cycle for a three dimensional state variable. Data is shown for T42TL18, T85L18, and T170L18 resolutions for various numbers of processors on the CM-5.

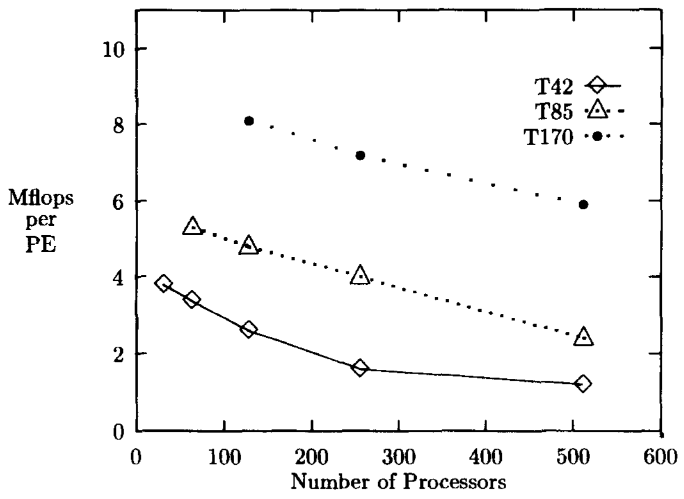

Fig. 6. Mflops per processor obtained performing a full spectral transform cycle for a 3 dimensional state variable. Data is shown for T42LL18, T85L18, and T170L18 resolution for various numbers of processors on the CM-5.

$\times LAT \times \log_2(LON/2)$. For the Legendre transform, the number of spectral coefficients for a triangular spectral truncation T, Ns, is given by $Ns = (T+1)^2 - T(T+1)2$. Therefore, the total number of floating point operations in the Legendre transform is given by $4Ns(LAT/2)$, where the factor of four is derived from taking the dot-product of complex vectors. Finally, the floating point operation count of the symmetrization/antisymmetrization step is given by $4 \times (T+1)$ LAT.

Fig. 6 shows the aggregate floating point performance attained for the spectral transform cycle using these operation counts. The best total performance, 3.05 Gflops, is attained for T170L18 on 512 processors.

5.1 Communications performance

In the data parallel programming model, particularly in CM Fortran, the concept of array shape is very important. The description of the data remappings required by our technique is best understood in terms of these shapes. There are five essential state variable shapes in our data parallel implementation of the spherical harmonic transform method. Our notation for these shapes represents the axes in standard Fortran order with the fastest varying index first. Axes within curly brackets are distributed across processors and operated on in parallel. The axis outside the curly brackets is serial within each processor. Also, the following axis naming convention is employed here: LON = number of longitudes, LAT = number of latitudes = LON/2, LEV = number of levels, and T = truncation wave number = (LON-1)/3. Thus the state variable beings in a "physical" shape which we denote {LON,LAT}LEV. This is then mapped into a "transposed physical" shape denoted LON{LAT,LEV}, which serializes the longitude axis in preparation for the FFT. The next shape called the "FFT" shape is the output of the FFT after the wavenumbers have been truncated to a value T, (T + 1){LAT,LEV}. The fourth shape is called the "transposed FFT" shape, (LAT/2){2,(T + 1),LEV}. It brings the latitude index into processor for the Legendre transform and creates a parity axis in preparation for symmetrization/antisymmetrization. Finally, the output of the Legendre transform step is called the "spectral" shape: (T/2 + 1){2,(T + 1),LEV}.

Described in this way, there are clearly four data mapping steps required to complete the spectral transform cycle. A pair of remappings (forward and back) are required to move from the physical to the transposed physical (pre FFT) shape. Also, a pair of remappings are required to move from the FFT shape to the transposed FFT (pre Legendre Transform) shape.

The most effective way to perform the one to one remapping of data from one parallel layout to another on the CM-5 is typically through the use of the COMM–SEND communication routine from the CMSSL. Even so, the communication takes up most of the time in the transpose-based spectral transform. For example, for resolution T42L18 interprocessor communication takes 73.3% and 66.5% of the time on 32 and 512 processors, respectively. For resolution T170L18 interprocessor communication consume 66.4% and 69.4% of execution time on 128 and 512 processors, respectively.

The performance of the COMM–SEND routine on the CM-5 is shown in Fig. 7. It shows the communication time on the CM-5 for the physical to transposed physical mapping. A linear fit of this data indicates an asymptotic bandwidth of 1.87 Mbytes/sec, and an apparent software latency of 1.97 milliseconds. For the inverse mapping, a linear fit produces an asymptotic bandwidth of 1.89 Mbytes/sec and an apparent software latency of 0.822 milliseconds.

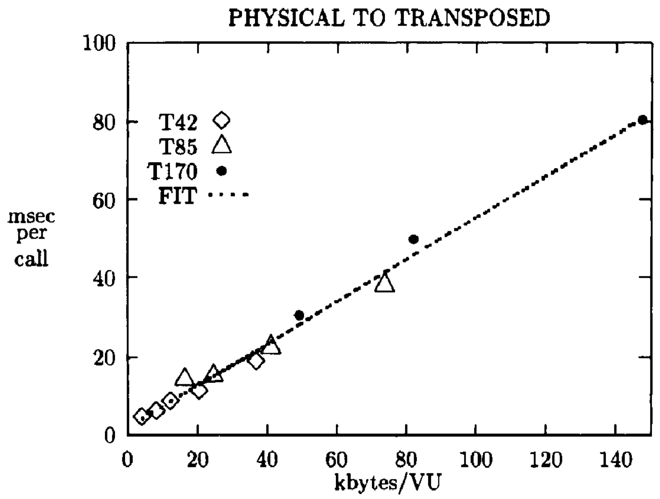

Fig. 7. Time required by the physical to transposed physical data mapping, on the CM-5, versus the padded CM-5 vector unit subgrid size in kilobytes.

5.2 FFT and Legendre transform performance

We have measured the performance of the real to complex FFT and the Legendre Transform on the CM-5 for the following resolution and machine size combinations: T42L18 on 32, 64, 128, 256, and 512 processors; T85L18 on 64, 128, 256 and 512 processors; and T170L18 on 128, 256, and 512 processors. These measurements were performed using 64-bit floating point arithmetic on the LANL CM-5. The FFT performance discussed here is computed using the floating point operation count of $5\ (LON/2)\log_2\ (LON/2)$. The Legendre Transform performance data takes into account the fact that the Associated Legendre Polynomials form a triangular matrix. Since we choose to pad this with zeros in a square matrix, we lose a factor of two in transform's performance.

The timing results for the real to complex FFT are displayed in Fig. 8. The Mflops per processor for the Legendre transform is displayed in Fig. 9. The degradation of the floating point performance of these local routines are the consequence of short vector lengths. In both cases, for a fixed number of processors and an increasing model resolution, the transforms achieve higher per node performance levels and display better scaling properties. The best aggregate performance for each transform is achieved for T170L18 on 512 nodes. In that case the local real to complex FFT attains 5.35 GFlops, and the local Legendre Transform attains 13.77 GFlops on the CM-5.

5.3 Combined parallel axes and padding effects

Many MPP's, including the CM-5, partition the computer into domains containing a power of two number of processors. The scalar spherical harmonic transforms

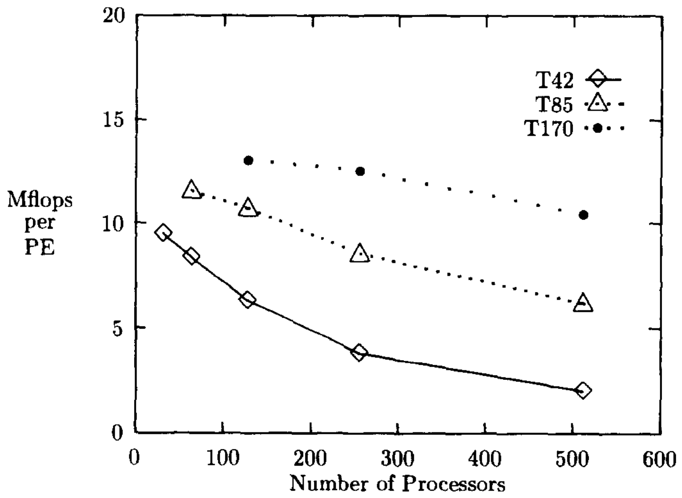

Fig. 8. Performance of the Local Real To Complex FFT in Mflops per processor for various resolutions and CM-5 partition sizes.

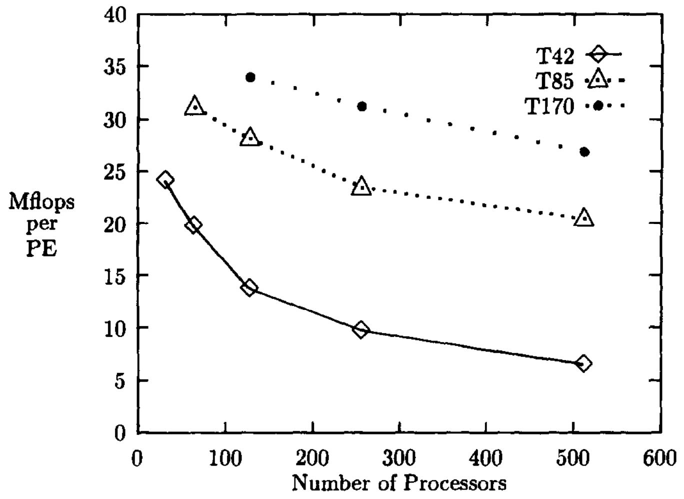

Fig. 9. Performance of the Local Legendre Transform in Mflops per processor for various resolutions and CM-5 partition sizes.

naturally parallelize along the wave number and verical level axis. Unfortunately, neither of these extents is typically a power-of-two. Therefore, one must consider the impact that the padding of the transformed arrays will have on performance.

We have investigated the scaling properties of three variants of the transposition method each using different array shapes. These three strategies differ in the degree to which they combine multiple parallel axes to achieve more efficient domain decompositions with minimal padding.

The three array shape strategies are shown in Table 2. Recall that axes enclosed in curly brackets are distributed across processors. We follow the Fortran convection of left justifying the fastest varying indices in processor memory. Method 1 does not combine axes throughout the transform cycle. Method 2 combines the parallel axes associated with the transposed FFT and the spectral shapes. Method 3 combines the parallel axes for all shapes except the physical grid. No effort was made to fold in the extent 2 odd/even parity axis as this prevents the use of the CSHIFT intrinsic to do this task. Also, it greatly complicates the symmetrization and antisymmetrization of model state variables in the data parallel implementation.

The number of levels was held fixed at the CCM2 value of 18 throughout this study. All calculations were performed in 64-bit floating point. The timings and rates reported here are the average of three runs on the CM-5 at LANL. The compiler used was CM Fortran 2.2 Final.

The speedup curves comparing the three methods are displayed in Fig. 10 and 11 for T42L18 and T170L18 respectively. At low resolutions there is no clear advantage obtained by combining axes. However, referring to Fig. 11 we see that at high resolution (T170L18), on large numbers of processors, a distinct advantage is achieved by combining parallel axes. Method 3, in which the parallel axes LAT/2

Table 2
Three parallel array shape strategies for the data mapping spherical harmonic transforms

Shape name	METHOD 1	METHOD 2	METHOD 3
Physical	{LON,LAT}LEV	{LON,LAT}LEV	{LON,LAT}LEV
Transposed physical	LON{LAT,LEV}	LON{LAT,LEV}	LON{2,(LAT/2)* LEV}
FFT	(T+1){LAT,LEV}	(T+1){LAT,LET}	(T+1){2,(LAT/2)* LEV}
Transposed FFT	(LAT/2){2,(T+1),LEV}	(LAT/2){2,(T+1)* LEV}	(LAT/2){2,(T+1)* LEV}
Spectral	(T/2+1){2,(T+1),LEV}	(T/2+1){2,(T+1)* LEV}	(T/21){2,{T+1}* LEV}

and LEV of the FFT layout and (T + 1) and LEV of the spectral layout have been combined, performs 18% better than the original Method 1, in which no axes are combined, on 512 processors. The magnitude of this difference is in agreement with the difference in padding fraction of the arrays in Method 3 (25%) compared to Method 1 (40%) in this case.

5.4 Parallelizing the level axis: Performance considerations

The Associated Legendre Polynomials (ALP's) and their derivatives are replicated for each parallel level employed in the transposition method. These ALP's represent a large allocation of memory. For example, for T42L18 on 32 processors, a single ALP array consumes 9.4 Mbytes of memory; for T85L18 on 64 processors, 73.2 Mbytes; and for T170L18 on 128 processors, 542.1 Mbytes. This presents about 1%, 3.5%, and 13.2%, respectively, of the total memory for these configura-

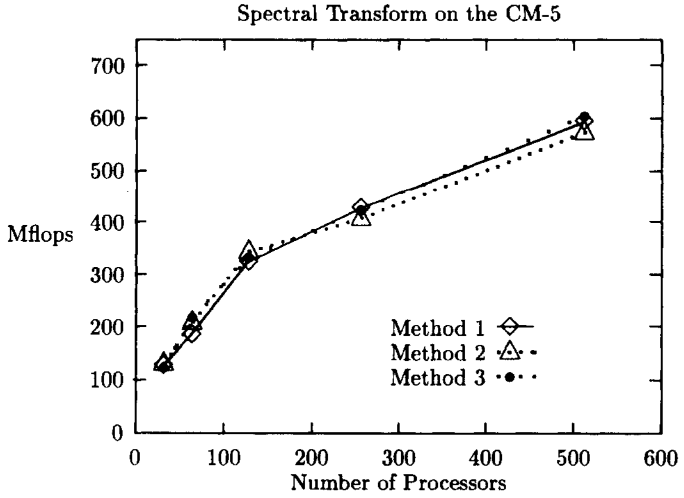

Fig. 10. A comparison of the performance at T42L18 for each of three variant array shape strategies described in the text.

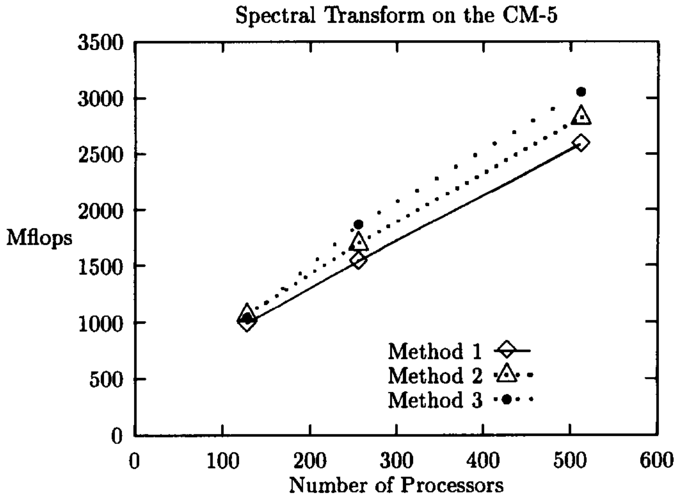

Fig. 11. A comparison of the performance at T170L18 for each of three variant array shape stategies described in the text.

tions. Consequently, per processor memory limitations restrict the maximum number of levels employed in this method. In contrast, a clear way to improve the speedup characteristics of this method is to maximize the number of levels operated upon in parallel.

This competition between memory limits and performance considerations suggest that one should experimentally determine when the benefits of increasing the number of parallel levels diminish to the point that the memory costs become a dominant concern. This balance point depends on resolution, machine size, and machine architecture.

We have done this tuning for the CM-5 by measuring the per processor performance as a function of the number of parallel levels employed in the transposition method. Fig. 12 shows the results taken from the most scalable method (Method 3) for a resolution of T42L18 on 32 processors. The number of layers involved in the spectral transforms was varied from 1 to 36. It is clear from the figure that increasing the number of layers involved in the spectral transform beyond about 8 layers does little to improve overall performance. The results for other numbers of CM-5 processors and model resolution combinations are similar.

Because per processor performance is flat for large numbers of levels, we have chosen not to increase the number of parallel levels employed in the spherical harmonics in CCM2 beyond the number of levels in a single three dimensional state variable. This choice results in readable code since 3-Dimensional variables are transformed as a unit.

5.5 Data remapping considerations for other systems

We are confident that the transposition method will run efficiently on other MPP platforms. On the Connection Machine, approximately 70% of the time for

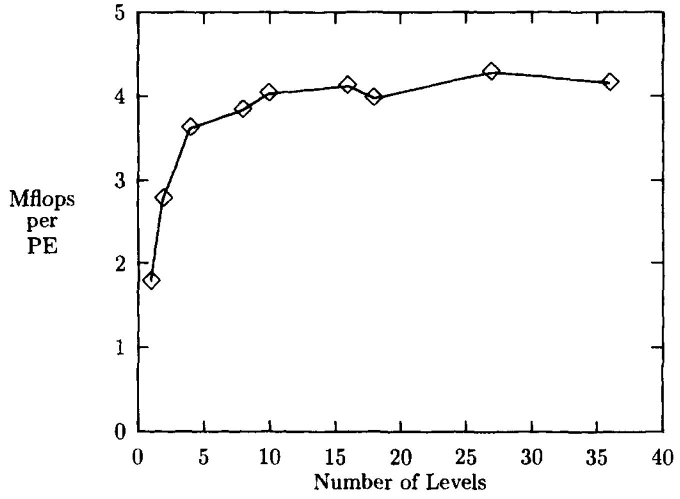

Fig. 12. Sensitivity of the transposition method performance on the CM-5 to the number of levels processed in parallel.

this method is spent in the data remapping operations. The performance data in Section 5.1 above indicates that the CMSSL COMM–SEND communication routine has a relatively high $O(2000)$ microsecond software latency and a relatively low asymptotic bandwidth (approximately 2.0 Mbytes/sec/proc). Many existing MPP networks have far lower latencies and higher bandwidths. Therefore, it should be straight forward to achieve better speedups and higher per processor performance figures on other platforms. To demonstrate this capability, we are currently working on a data mapping communication package for the Cray T3D based on the SHMEM communication library [18].

Finally these experiments can be used to illustrate both the strengths and weaknesses of using a data parallel language like CM Fortran. Because the runtime system provides automated, transparent domain decomposition, different implementation strategies can quickly be tested and evaluated. The range of experimentation is constrained however, because many of the details of the domain decomposition are controlled by the compiler. For example, irregularly sized and shaped domains are not supported under the CM Fortran data parallel programming model.

6. CCM2 Semi-Lagrangian transport

There are serious modeling disadvantages associated with the use of spherical harmonic transform methods for advecting constituent gases such as water vapor [15,20]. Spectral overshoots can cause negative values in spectrally transformed positive definite quantities. This results in non-physical phenomena in models, such as "spectral rain". This problem is a non-local effect and is not easily fixed. As a result, CCM2 uses a shape preserving Semi-Lagrangian Transport (SLT) scheme with a Hermite cubic interpolant to preserve monotonicity and eliminate overshoot [15,20]. The CCM2 SLT algorithm is also timesplit, with the two dimensional horizontal interpolation followed by a one dimensional vertical interpolation.

In the Cray Y-MP version of the code, the basic SLT data structure is an extended grid in both the longitude and latitude directions which regularizes the gathering of the stencil information required for the SLT interpolations. Therefore the SLT implementation on the Cray has three parts: grid extension, horizontal interpolation, and vertical interpolation. Of these, the most numerically and communication intensive is the horizontal interpolation.

6.1 Horizontal interpolation

The horizontal interpolation involves a different, general non-local gather communication every timestep. Therefore, optimizing the parallel horizontal interpolation algorithm was the focus of our effort. It in turn determined the structure of every other part of the CM-5 version of the SLT code. The horizontal interpolation may be encapsulated in the following code snippet involving 28 floating point operations per site:

```
do n=-1,2
  do j=1, LAT
    do k=1, LEV
      do i=1, LON
        fint(i,k,j,n)=f(idp(i,k),k,jdp(i,k)+n)*hl(i,k)+
                      f(idp(i,k)+1,k,jdp(i,k)+n)*hr(i,k)+
                      fxl(idp(i,k),k,jdp(i,k)+n)*dhl(i,k)+
                      fxr(idp(i,k),k,jdp(i,k)+n)*dhr(i,k)
      end do
    end do
  end do
end do
```

where

- `fint` is the horizontally interpolated field;
- `f` is the field to be interpolated;
- `fxl` is the left derivative of the field to be interpolated;
- `fxr` is the right derivative of the field to be interpolated;
- `idp` is the longitude index of the grid box containing the departure point;
- `jdp` is the latitude index of the grid box containing the departure point; and
- `hl, hr, dhl, dhr` are arrival point based interpolation factors derived from the longitude coordinate of the departure point.

Note that each departure point is contained within a grid box defined by a corner index (`idp, jdp`).

We indentify two basic approaches to parallelizing the horizontal interpolation, depending on where the interpolant is calculated. We call these the arrival point based and the departure point based methods The arrival point based algorithm consists of the following steps.

(1) Gather a two by four longitude, latitude stencil consisting of points with relative offsets (0:1, −1:2).
(2) Next, each arrival point (`i,j`) sends a message to departure point (`idp, jdp`) containing the latitude and longitude indices of the arrival point (just (`i,j`)) to the departure point grid box location (`idp, jdp`).
(3) Because there may be multiple departure points in the grid box defined by (`idp, jdp`), several such messages may be received at each site. The messages from the arrival points are queued at each departure point grid box location defined by (`idp, jdp`).
(4) Using the stencil information compiled in step 1, the departure point (`idp, jdp`) returns the 16 words of interpolation data to each arrival point in its receive queue.
(5) Finally, the interpolant is computed at the arrival point.

We now discuss the departure point based horizontal interpolation algorithm. The first step is the same as in the arrival based algorithm.

(1) One first gathers a two by four longitude, stencil consisting of the points with relative offsets (0:1, −1:2).
(2) Next, each arrival point (`i,j`) sends a message to the departure point grid box (`idp, jdp`) containing the latitude and longitude indices of the arrival point (just (`i,j`)) *and the logitudinal coordinate of the departure point* to the departure point grid box (`idp, jdp`).
(3) The messages from the arrival points are queued at each departure point grid box location defined by `(idp, jdp)`.
(4) The interpolants are computed at the departure point grid box location (`idp, jdp`).
(5) The interpolants (four words) are moved back to each arrival point).

One advantage of the arrival point method is that the computation of the interpolant `fint` is perfectly load balanced. A clear disadvantage of this method is the need to communicate *all* the interpolation data in step 4. A potential disadvantage of the arrival point method is that the cost of the communication (steps 2 and 4) depends on the departure point occupancy number in each box. The magnitude of this disadvantage can only be assessed by understanding the properties of atmospheric fluid flow in CCM2. The advantage of the departure point method is that fewer words need to be communicated. A clear disadvantage is that the computation of the interpolant is no longer load balanced. The departure point method also suffers the potential disadvantage that both the communication costs and the computational costs scale with the departure point occupancy number.

The relative merits of the two methods can be quantitatively compared using this information. Suppose that the maximum departure point occupancy number in any grid box is q, the departure point method will be faster if the following inequality holds:

$$\text{time to do } 28(q-1) \text{ flops} < \text{time to move } 11q \text{ words.} \tag{1}$$

To resolve this question we need information regarding the performance characteristics of the MPP in question and the maximum value of q in the climate model. It has been observed that the maximum departure point occupancy number in CCM2 are usually between four and five and occur in regions of large divergence [22]. The occupancy number also increases as the timestep increases for a fixed resolution, and as the resolution increases for a fixed timestep [22]. Table 3 shows the bandwidths for different operations on the CM-5. Note the non-local scatter moves approximately 2 Mbytes or 0.25 Mwords per second per processor. If we assume a nominal floating point rate of $O(10)$ Mflops/processor then there is a floating point computation to interprocessor communication ratio of roughly 40 Flops per word communicated. This indicates that 28 floating point operations can easily be performed faster than the interprocessor communication of 11 words.

This convinced us to implement the departure point based horizontal interpolation algorithm described above. Even with the departure point method, the performance of non-local communications on the Connection Machine was of great concern during the design of the CM-5 horizontal interpolation algorithm, since non-local gathers have the worst bandwidths (see Table 3).

Important observations from the climate model influenced our design. The CCM2 data indicates that the greatest departure point non-locality was in the longitude direction. This characteristic is illustrated in Fig. 13. It shows the zonal, vertical, and month-long maximum departure point grid box offset in the longitudinal direction for the month of January. The data was measured at a T42L18 resolution, with a 20 minute timestep.

We note that the lines of constant latitude near the north pole have maximum departure point longitudinal offsets of up to 14. This is understandable since the longitudinal points tend to bunch up near the poles. Also, the longitudinal non-locality is seasonally dependent and increases as the timestep increases [22].

Table 3
Measured CM-5 bandwidths in Mbytes per second per processor

Local memory reference	131.49
Regular shift operation	64.91
Local indirect address	43.07
Non-local scatter	1.94
Non-local gather	1.14

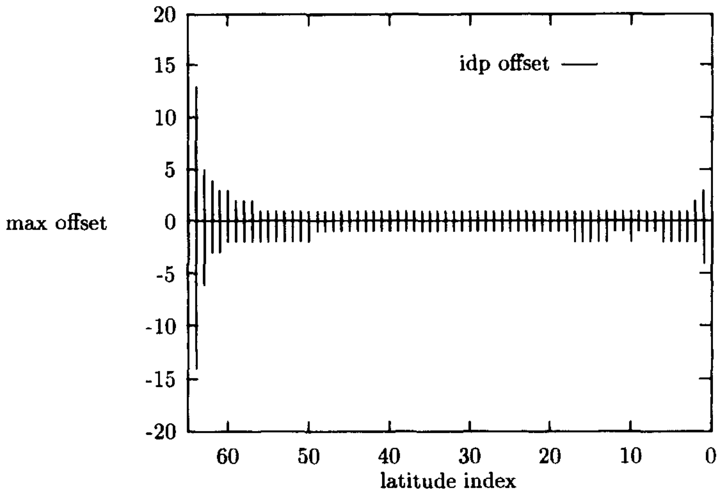

Fig. 13. An illustration of the strong non-locality of the longitudinal departure point locations in the CCM2 SLT. The north pole is located at latitude index 64 and the south pole at latitude index 1.

In contrast, the latitude offset of the departure point is always a small and a well defined distance from the arrival point, because the time step is restricted by the spectral dynamics. This has to do with the nature of the CCM2's Gaussian grid. Although the grid points tend to bunch up near the poles in the longitude direction, the Gaussian latitudes are relatively equally spaced.

We have therefore developed a quasi-local departure point based horizontal interpolation for the CM-5. In this method we use an "extended grid" array layout which exploits the difference in departure point locality in the two directions. It is reminiscent of the data mapping technique which we used to advantage in the spherical harmonic transform method. Using the nomenclature defined in Section 5 this layout may be described as LON{(LAT + 4) * LEV}. The extension includes one pole point and one transpolar point at each end of each latitude segment. Recall that in Table 3 we showed that non-local scatters and gathers are a factor of 22 and 37, respectively, slower than indirect addressing. By localizing the longitude axis in processor, we are able to take advantage of the 43 Mbyte/sec local indirect addressing bandwidth of the CM-5 processor. Communication between departure point grid boxes and arrival points with latitude offsets are accomplished by CSHIFT's, since latitude is the fastest varying value along the parallel axis of the extended grid layout. This method uses the three highest bandwidths in the CM-5 to get performance from the horizontal interpolation. Mapping the grid data exchanges relatively slow operations for faster ones at the expense of a fixed, regular communication pattern, which can be optimized with communication libraries, in this case the COMM-GET CMSSL function [5].

6.2 SLT performance results on the CM-5

We have measured the performance of the CCM2 SLT on the CM-5 for the following resolution and machine size combinations: T42L18 on 32, 64 and 128 processors; T85L18 on 128, 256, and 512 processors; T170L18 on 512 processors. These measurements were performed using 64-bit floating point arithmetic on the LANL CM-5. The number of milliseconds to advect one constituent one time step for these combinations are displayed in Fig. 14. For reference, the SLT implementation on the Cray Y-MP/1 requires 764 msec per timestep. Also, we derive the Mflop rate on the CM-5 based on the performance of this operation on a Cray Y-MP. It is shown in Fig. 15. The SLT implementation on the Cray Y-MP/1 achieves 146.5 Mflops.

Despite all of the optimizations described in the preceding section, the performance of the CCM2 SLT algorithm on the CM-5 is still dominated by the horizontal departure point interpolation. It consumes roughly 64% of the computation time for all combinations. For comparison, it is interesting to note that on the Cray Y-MP/1 (T42L18), the horizontal interpolation takes a slightly more than 71% of the time. Surprisingly the data remapping communications in the data parallel SLT require only about 16% of the time. This stands in dramatic contrast with the 70% figure observed in the spherical harmonic transform method. Finally, the vertical interpolation takes only about 4% to 6% of the compute time on the CM-5, compared to 22% on the Cray Y-MP/1.

6.3 General suitability of the transpose-based SLT to other platforms

The interpolations in the SLT requires a great deal of data motion. Implementing the transpose-based SLT on an other MPP platform will require revisiting the

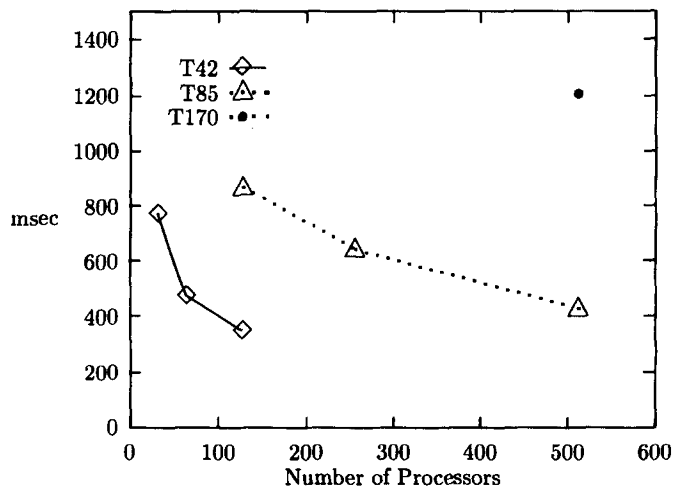

Fig. 14. Execution time in milliseconds required to advect one constituent one timestep for various resolutions and CM-5 partitions sizes.

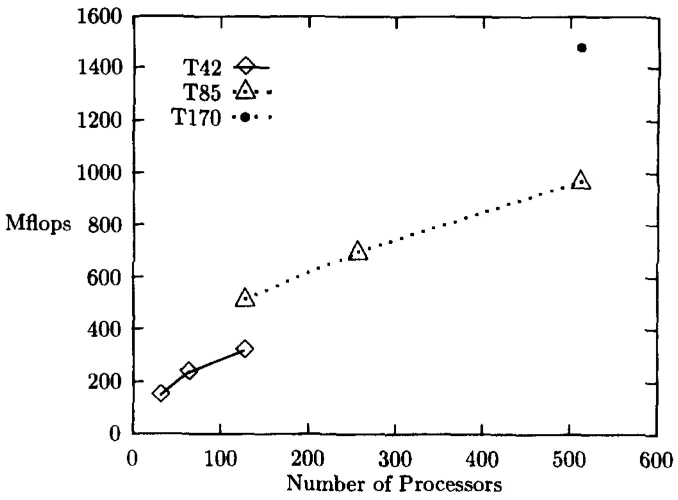

Fig. 15. Floating point performance of the data parallel SLT for various resolutions and machine sizes.

floating point and interprocessor inequality (1), to determine which interpolation strategy should be implemented. Any parallel machine should be able to do well with a data remapping approach provided the local indirect addressing and nearest neighbor communication bandwidth is high.

Finally, the technique which we have used to parallelize the CCM2 SLT is uniquely suited to the Gaussian polar grid on the sphere, the zonal nature of atmospheric dynamics, and the time splitting of the departure point interpolation method in the CCM2 SLT. Changing any of these ingredients in a different semi-Lagrangian setting would likely lead to a different optimal parallelization scheme. For example, a full three dimensional interpolation scheme would force a series of expensive non-local gather communications.

7. Scalable I/O - Issues and performance

Like many application areas, geoscience simulation codes such as CCM2 are output data intensive. CCM2 history files vary in size depending on model resolution and the number of constituent fields written. For the T170L18 resolution, output file size can be as large as 770 Mbytes. CCM2 history file output frequencies can range from every time step to each simulated model month. Large restart files are also saved at regular intervals to allow the model to be stopped and restarted gracefully. Although CCM2 has the capability to compress output data up to a factor of four, it is not uncommon in the course of execution for CCM2 to generate a megabyte of output data for analysis for every billion floating point operations performed. The input/output capabilities of any computer are crucial to the production performance of CCM2.

Here we list some requirements for scalable input and output on parallel computers, based on our experience with CCM2 on the CM-5. MPPs should provide a single logical file system which should be accessible to every computational node or group of nodes and be available to every running process and collection of processes for expeditious storage and retrieval. Each processor should be able to read from or write to a single image, as opposed to each processor writing a portion of the output to a file located on its own local disk.

In addition, output files should be machine size independent. A file written by an application running on P processors should be transparently read by a program running on Q processors (and visa-versa), where $P \neq Q$. Ideally it should be possible to read or write this same file using a workstation as well. This capability is especially important for restarting or debugging a code. Suppose that a simulation starts on P processors and is interrupted. Further, suppose that when the user wants to continue the simulation there are less than P processors available. If the restart data is specific to a particular number of processors, then the user must wait until they are available or perform some file conversion to be able to use the different set of processors. Also, consider the case of debugging a program. A user may be able to use fewer processors during debugging than would be desirable for a long production run. However, if the files needed are dependent on a specific number of processors then one will need a set of data files for each possible system configuration.

Finally, throughput should scale with the number of processors in the system. As a programmer uses more processors, the rate at which one is able to compute increases and the rate at which results are produced increases. Therefore, to keep I/O from being a bottleneck to the computation, I/O rates need to scale with the aggregate computational rate.

In Fig. 16 we show average I/O rates measured between the CM-5 and the scalable disk array (an attached disk system). The rates are for the three different size buffers on different numbers of processors. Each data point is the average of three separate tests made for each combination of buffer size and number of processors. The buffers are rank three arrays of 64-bit floating point numbers—31.9 Mb ($64 \times 128 \times 487$), 127.6 Mb ($128 \times 256 \times 487$), and 510.6 Mb ($256 \times 512 \times 487$), respectively. This simulates writing restart information for three different model resolutions, T42, T85, and T170. The first two dimensions are the horizontal grid and the third dimension is used for single level and multi-level fields.

Fig. 16 shows that for the 31.9Mb buffer, I/O rates vary between 17.8Mb/sec. on 32 processors to 62.8Mb/sec. on 1024 processors. There is only an increase of a factor of 3.5 in throughput even though we are using 32 times as many processors. For the 127.6Mb buffer, I/O rates range from 19Mb/sec. on 32 processors to 283Mb/sec. on 1024 processors, a factor of 14.9 increase for 32 times as many processors. Finally, for the 510.6Mb buffer, I/O rates vary between 72Mb/sec on 128 processors[2] and 427Mb/sec. on 1024 processors, a factor of 5.9 increase for

[2] The largest buffer would not fit on fewer than 128 processors.

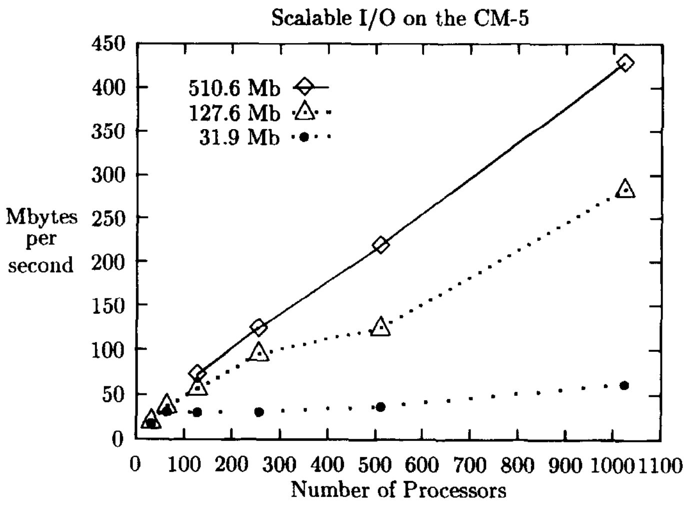

Fig. 16. Average I/O rates for three different buffer sizes and varying numbers of processors of the CM-5.

eight times as many processors. The I/O rate scales reasonably well for the medium and large size buffers. However, latency dominates the time for writing small buffers.

The buffers are written to a single file using an unformatted Fortran write statement. In general, file I/O operations on the CM-5 can be specified with standard Fortran `open`, `write`, and `close` syntax. They satisfy the ideal features listed above – single image, portability across different numbers of processors, and throughput scalability.

8. Conclusions

The original objective of the dual CHAMMP effort was to compare the relative efficiency and ease of use of the message passing and data parallel paradigms. This objective was soon overshadowed by the larger difficulties associated with development and testing of the necessary parallel algorithms, porting and maintaining the codes at the current version levels, and dealing with unstable and changing MPP hardware and software environments. In retrospect it is safe to say that neither approach was easy. A true comparison of the relative efficiency of the data parallel program versus message passing for CCM2 can only be assessed when efficient and mature versions of each programming model are available on the same MPP. We are unaware of any platform to date that satisfies this requirement.

Our results imply certain conclusions regarding the sorts of MPP architectures appropriate for spectral climate modeling. We observe that interprocessor communication associated with the transposition method in CCM2 never took more than 22% of overall time for any resolution or processor combination on the CM-5.

Thus increasing the *ratio* of the interprocessor bandwidth relative to the floating point performance observed for the CM-5 will result in a performance increased guaranteed to be less than this same small factor. A better approach is to ask why the nodal performance is so low O(5 Mflops/node) in the first place. We attribute this poor nodal performance to the vector architecture of the CM-5. As we observed in the text, CCM2 displays generally low levels of parallelism by MPP standards. This means that any spectral climate application which employs scalable 3-D transposition techniques will produce short vectors on large numbers of processors. Thus MPP's with vector nodal architectures tuned for moderately long vectors, such as the CM-5, will not scale well. We expect that MPP's with superscalar architectures will handle these conditions for better.

We have measured performance on the CM-5 for transposed based spherical harmonic transform method up to 3.05 GFlops on 512 processors at a T170L18 resolution. We have developed a performance model of Semi-Lagrangian Transport which allows us to predict whether interpolation calculation should be performed at the arrival point or the departure point grid point location. The data parallel version of the SLT code, which localizes the horizontal interpolation calculation through data transposition, achieves performance levels as high as 1.48 GFlops on 512 processors for a T170L18 resolution. We showed that one model day on a 32-processor CM-5 at T42L18 is 75% faster than the optimized serial code running on a single processor Cray Y-MP. Finally, we gave the performance and memory requirements of higher resolutions runs on larger numbers of processors.

Acknowledgements

We want to thank Jim Hack, Jim Rosinski, Paul Swaraztrauber, and Dave Williamson, all from NCAR, for their contributions to this work. Also, we want to thank Don Middleton from NCAR for rendering the image of the precipitable water in Fig. 4. This research was performed using the Connection Machines located at the Scientific Computing Division of the National Center for Atmospheric Research, Boulder, CO 80307 and at the Advanced Computing Laboratory of Los Alamos National Laboratory, Los Alamos, NM 87545. This work was sponsored in part by the Department of Energy CHAMMP initiative and the National Science Foundation.

References

[1] L.M. Bath, J.R. Rosinski and J. Olson, User's guide to (CCM2), Technical Report NCAR Technical Note/TN-382+IA, Climate and Global Dynamics Division, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307, 1992.

[2] Building the Advanced Climate Model, a draft plan for CHAMMP, Department of Energy, March 1990.

[3] Using the CMAX Converter Thinking Machines Corporation Manual, Version 2.0 (May 1994).

[4] CMSSL for CM Fortran, Vol. II, Version 3.2, Thinking Machines Corporation (April 1994), 486.

[5] CMSSL for CM Fortran, Vol II, Version 3.2, Thinking Machines Corporation (April 1994), 706.
[6] J. Drake, I. Foster, J. Michalakes and P. Worley, Design and performance of a scalable community climate model, *Parallel Computing*, this issue.
[7] P.N. Swarztrauber, Vectorizing the FFT's, *Parallel Computations* Rodrique, ed. (Academic Press, NY, 1982).
[8] 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), Technical Report NCAR Technical Note/TN-382 +STR, Climate and Global Dynamics Division, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307, 1993.
[9] J.J. Hack, J.M. Rosinski, D.L. Williamson, B.A. Boville and J.E. Truesdale, Computational design of the NCAR community climate model, *Parallel Computing*, this issue.
[10] S.W. Hammond, R.D. Loft, J.M. Dennis and R.K. Sato, Massively parallel atmospheric modelling at NCAR, *Proc. ECMWF's Sixth Workshop on Parallel Computing and Weather Forecasting* (Nov. 21–25, 1994).
[11] S.W. Hammond, R.D. Loft, J.M. Dennis and R.K. Sato, A data parallel implementation of the NCAR community climate model (CCM2), *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computations* (Feb. 1995), 125–130.
[12] R.D. Loft and R.K. Sato, Implementation of the NCAR CCM2 on the Connection Machine, *Parallel Supercomputing in Atmospheric Science,* G.R. Hoffman and T. Kauranne, eds. (World Scientific, 1993 371–393).
[13] S.R.M. Barros and T. Kauranne Scalability estimates of parallel spectral atmospheric models, *Parallel Supercomputing in Atmospheric Science*, G.R. Hoffman and T. Kauranne, eds. (World Scientific, 1993 312–328).
[14] D.W. Dent Parallelization of the IFS model, *Parallel Supercomputing in Atmospheric Science,* G.R. Hoffman and T. Kauranne, eds. (World Scientific, 1993 73–78).
[15] P.J. Rasch and D.L. Williamson, Computational aspects of mositure transport in global models of the atmosphere, *Quart, J. Roy. Meteor. Soc.*, 119 (1990) 1071–1090.
[16] J.G. Sela, P.B. Anderson, D.W. Norton and M.A. Young, Massive parallelization of NMC's spectral model, in preparation.
[17] H.G. Siegel and S. Abraham, Summary of the report of the NSF-sponsored Purdue workshop on grand challenges in computer architectures for the support of high performance computing, *Proc. Frontiers '92 Sym. on the Frontiers of Massively Parallel Computation* (IEEE Computer Press, 1992) 76).
[18] *SHMEM User's Guide for Fortran,* Revision 2.2 (Cray Research, Aug. 15, 1994).
[19] W.L. Washington and C.L. Parkinson, *An Introduction to Three-Dimensional Climate Modeling,* (University Science Books, 1986 181–205).
[20] D.L. Williamson and P.J. Rasch, Two-dimensional semi-Lagrange transport with shape preserving interpolation, *Mon, Wea. Rev.,* 117 (1989) 102–129.
[21] D.L. Williamson, J.T. Kiehl and J.J. Hack, Climate sensitivity of the NCAR community climate model (CCM2) to horizontal resolution. Technical Report NCAR Technical Note/Ms-0301/94-04, Climate and Global Dynamics Division, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307, Feb 1994.
[22] Private communications with D.L. Williamson.