ORNL/TM-12507

Engineering Physics and Mathematics Division

Mathematical Sciences Section

PARALLEL ALGORITHMS FOR THE SPECTRAL TRANSFORM METHOD

Ian T. Foster *
Patrick H. Worley †

* Argonne National Laboratory
Mathematics and Computer Science Division
Argonne, IL 60439-4801

† Oak Ridge National Laboratory
Mathematical Sciences Section
P. O. Box 2008
Oak Ridge, TN 37831-6367

Date Published: April 1994

Research was supported by the Atmospheric and Climate Research Division and by the Applied Mathematical Sciences Research Program, both of the Office of Energy Research, U.S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

Contents

PARALLEL ALGORITHMS FOR THE SPECTRAL TRANSFORM METHOD

Ian T. Foster
Patrick H. Worley

Abstract

The spectral transform method is a standard numerical technique for solving partial differential equations on a sphere and is widely used in atmospheric circulation models. Recent research has identified several promising algorithms for implementing this method on massively parallel computers; however, no detailed comparison of the different algorithms has previously been attempted. In this paper, we describe these different parallel algorithms and report on computational experiments that we have conducted to evaluate their efficiency on parallel computers. The experiments used a testbed code that solves the nonlinear shallow water equations on a sphere; considerable care was taken to ensure that the experiments provide a fair comparison of the different algorithms and that the results are relevant to global models. We focus on hypercube- and mesh-connected multicomputers with cut-through routing, such as the Intel iPSC/860, DELTA, and Paragon, and the nCUBE/2, but also indicate how the results extend to other parallel computer architectures. The results of this study are relevant not only to the spectral transform method but also to multidimensional FFTs and other parallel transforms.

1. Introduction

The spectral transform method is widely used for fluid dynamics problems in spherical geometry, in such areas as climate modeling, weather modeling, astrophysics, and reactor design. In this paper, we examine the problem of implementing the spectral transform method on massively parallel computers. Such computers comprise 10^2–10^4 processors, each with local memory and able to access other processors' memory via an interconnection network. When designing algorithms for these computers, important considerations include minimizing nonlocal memory accesses, organizing interprocessor communication to make efficient use of the network, masking communication latency, and minimizing load imbalances.

The spectral transform method as used in climate models comprises a Fourier transform phase, in which fast Fourier transforms (FFTs) are applied to each latitude of a latitude/longitude grid, and a Legendre transform phase, in which Gaussian quadrature is used to approximate the Legendre transform (LT) applied to each longitude (now wavenumber) of the same grid [4]. Efficient parallel FFT and LT algorithms have been the topic of intensive research (e.g., see [17, 27, 30, 31]). The spectral transform is nevertheless deserving of special study, first because the matrices involved are typically much smaller than usual for Fourier and Legendre transforms (e.g., 64–1024 in each dimension, rather than tens of thousands), second because the two phases interact in interesting ways on certain architectures, and third because the importance of the spectral transform makes even small performance improvements valuable.

Parallel spectral transform algorithms have been investigated previously by several researchers. We and colleagues at Argonne and Oak Ridge national laboratories have developed a *parallel transform* approach based on parallel FFT and quadrature algorithms [14, 34, 37]; this work has been incorporated in a parallel implementation [7, 8] of the National Center for Atmospheric Research (NCAR)'s Community Climate Model (CCM2) [20]. Other researchers have examined a *transpose* approach, in which communication requirements are encapsulated in a matrix transpose operation. This approach is used, for example, in the European Center for Medium-Range Weather Forecasts spectral weather model [6] and in Loft and Sato's data parallel implementation of CCM2 [24]. It has also been explored by Kauranne and Barros [23], Pelz and Stern [28], and Gärtel, Joppich, and Schüller [18].

In addition to the transform and transpose approaches, a variety of hybrid algorithms are possible that combine aspects of both. A comprehensive comparison of these algorithms has not previously been attempted. (Both [16] and [23] provide a qualitative analysis of some algorithms, but not detailed quantitative results or performance models.) Hence, it is difficult to evaluate the performance tradeoffs that arise when choosing a parallel algorithm for a particular application.

In this paper, we describe analytic and empirical studies that we have conducted to determine

1. whether there is a best algorithm (on a given platform, for a given problem size, etc.);
2. the sensitivity of the choice of optimal algorithm to problem size, number of processors, and platform specifics; and

3. the benefits of optimizing for a given platform or problem size.

In the analytic studies, we develop models that characterize the performance of the various spectral transform algorithms by relating communication requirements and load imbalances to problem size, processor count, and other parameters.

The empirical studies utilize a parallel shallow water equation solver designed specifically for these experiments [38]. Considerable care has been taken to ensure that experiments are as fair as possible, that is, that one algorithm is not unduly favored through choice of data structures, greater optimization, etc. In addition, the code structure mimics that of general circulation models, maximizing the applicability of results to these models.

The contributions of this paper are as follows. First, the analytic models provide a qualitative characterization of the performance of numerous parallel algorithms for the spectral transform, including both parallel algorithms developed previously and new algorithms developed in the course of this work. Second, the empirical results provide a detailed understanding of the performance characteristics of these algorithms on the target platforms. Third, we identify robust algorithm combinations for various problem size and machine characteristic regimes. These include a number of new, hybrid parallel algorithms for the spectral transform method.

The rest of this paper is as follows. Sections 2, 3, and 4 provide background information on the shallow water equations solved by our testbed code, the spectral transform method, and parallel computation. Sections 5 and 6 describe the parallel algorithms that we examine in the Fourier and Legendre phases of the transform. In §7, we use these models to make qualitative comparisons between the algorithms and to identify performance tradeoffs on different parallel computer architectures. Section 8 describes empirical studies conducted on a range of scalable parallel computers, and relates the results to algorithm and machine characteristics. Section 9 describes issues not addressed in this study, and directions for future work. Section 10 presents our conclusions.

2. The Shallow Water Equations

The nonlinear shallow water equations on a rotating sphere constitute a two-dimensional atmospheric-like fluid prediction model that exhibits many of the features of more complete models [36]. These equations are frequently used to investigate and compare numerical methods because they present many of the difficulties found in simulating the horizontal dynamics in three-dimensional global atmospheric models [5].

The algorithms used to solve the shallow water equations via the spectral transform method are similar to those employed in the NCAR Community Climate Model to handle the horizontal dynamics component of the primitive equations [20]. Hence, a model that solves the shallow water equations on multiple (independent) levels during each timestep of the simulation provides a framework in which the performance of CCM2's horizontal dynamics can be studied in isolation from the other aspects of the full model. While this framework is not a completely reliable predictor of the performance of the parallel algorithms in the full model, it allows us to determine accurately the relative merits of the different parallel approaches.

For completeness, we now describe the shallow water equations in the form that we solve using the spectral transform method. The shallow water equations on a sphere consist of equations for the conservation of momentum and the conservation of mass. Let $\mathbf{i}, \mathbf{j}$, and $\mathbf{k}$ denote unit vectors in spherical geometry, $\mathbf{V}$ denote the horizontal velocity, $\mathbf{V} = \mathbf{i}u + \mathbf{j}v$, Φ denote the geopotential, and f denote the Coriolis term. Then the horizontal momentum and mass continuity equations can be written as [35]

$$\begin{aligned} \frac{D\mathbf{V}}{Dt} &= -f\mathbf{k} \times \mathbf{V} - \nabla\Phi \\ \frac{D\Phi}{Dt} &= -\Phi\nabla \cdot \mathbf{V}, \end{aligned} \tag{1}$$

where the substantial derivative is given by

$$\frac{D}{Dt}(\;) \equiv \frac{\partial}{\partial t}(\;) + \mathbf{V} \cdot \nabla(\;)\ . \tag{2}$$

The spectral transform method does not solve these equations directly; rather, it uses a streamfunction-vorticity formulation in order to work with scalar fields. Define the vorticity η and the horizontal divergence δ by

$$\begin{aligned} \eta &= f + \mathbf{k} \cdot (\nabla \times \mathbf{V}) \\ \delta &= \nabla \cdot \mathbf{V}\ . \end{aligned}$$

To avoid the singularity in velocity at the poles, let θ represent latitude, and also redefine the horizontal velocity components as

$$(\,U\,,\,V\,) = \mathbf{V}\cos\theta\ .$$

Then, after some manipulation, the equations can be written in the form

$$\frac{\partial\eta}{\partial t} = -\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial\lambda}(U\eta) - \frac{1}{a}\frac{\partial}{\partial\mu}(V\eta) \tag{3}$$

$$\frac{\partial\delta}{\partial t} = +\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial\lambda}(V\eta) - \frac{1}{a}\frac{\partial}{\partial\mu}(U\eta) - \nabla^2\left(\Phi + \frac{U^2+V^2}{2(1-\mu^2)}\right) \tag{4}$$

$$\frac{\partial\Phi}{\partial t} = -\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial\lambda}(U\Phi) - \frac{1}{a}\frac{\partial}{\partial\mu}(V\Phi) - \bar{\Phi}\delta\ . \tag{5}$$

Here a is the radius of the sphere; the independent variables λ and μ denote longitude and $\sin\theta$, respectively; and Φ is now a perturbation from a constant average geopotential $\bar{\Phi}$.

Finally, U and V can be represented in terms of η and δ through two auxiliary equations expressed in terms of a scalar streamfunction ψ and a velocity potential χ:

$$U = \frac{1}{a}\frac{\partial \chi}{\partial \lambda} - \frac{1-\mu^2}{a}\frac{\partial \psi}{\partial \mu} \tag{6}$$

$$V = \frac{1}{a}\frac{\partial \psi}{\partial \lambda} + \frac{1-\mu^2}{a}\frac{\partial \chi}{\partial \mu} \ , \tag{7}$$

where

$$\eta = \nabla^2 \psi + f \tag{8}$$

$$\delta = \nabla^2 \chi . \tag{9}$$

In the spectral transform method, we solve Equations (3)–(5) for η, δ, and Φ, and use Equations (6)–(9) to calculate U and V.

3. The Spectral Transform Method

In the spectral transform method, fields are transformed at each timestep between the physical domain, where the physical forces are calculated, and the spectral domain, where the horizontal terms of the differential equation are evaluated. In the three-dimensional atmospheric models that we wish to emulate, all coupling between vertical levels is also calculated in the physical domain.

The spectral representation of a field variable ξ on a given vertical layer above the surface of a sphere is defined by a truncated expansion in terms of the spherical harmonic functions $\{P_n^m(\mu)e^{\mathrm{i}m\lambda}\}$:

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

where

$$\begin{aligned} \xi_n^m &= \int_{-1}^{1} \left[\frac{1}{2\pi} \int_0^{2\pi} \xi(\lambda,\mu) e^{-\mathrm{i}m\lambda} d\lambda \right] P_n^m(\mu) d\mu \\ &\equiv \int_{-1}^{1} \xi^m(\mu) P_n^m(\mu) d\mu \ . \end{aligned}$$

Here $\mathrm{i} = \sqrt{-1}$, $\mu = \sin\theta$, θ is latitude, λ is longitude, m is the wavenumber or Fourier mode, and $P_n^m(\mu)$ is the associated Legendre function. The spherical harmonic functions are the eigensolutions of the Laplacian operator in spherical coordinates and constitute a complete and orthogonal expansion basis for square integrable functions on the sphere. Additional properties of these functions can be found in [25].

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. This fact is used to reduce both computational complexity and storage requirements by calculating only spectral coefficients for nonnegative modes.

In each vertical layer of the physical domain, fields are approximated on an $I \times J$ longitude-latitude grid, where the I longitude grid lines are evenly spaced and the J latitude grid lines are placed at the Gaussian quadrature points $\{\mu_j\}$ in $[-1, 1]$. Transforming from physical coordinates to spectral coordinates involves first performing a Fourier transform for each line of constant latitude, generating the values $\{\xi^m(\mu_j)\}$ on an $M \times J$ wavenumber-latitude grid that we will refer to as the *Fourier grid.* This is followed by integration over latitude for each line of constant wavenumber, approximated using J-point Gaussian quadrature, to obtain the spectral coefficients,

$$\xi_n^m = \sum_{j=0}^{J-1} \xi^m(\mu_j)\, P_n^m(\mu_j)\, w_j \ .$$

Here w_j is the Gaussian quadrature weight corresponding to the Gaussian latitude μ_j. The point values are recovered from the spectral coefficients by computing

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

for each m, followed by inverse Fourier transforms to calculate $\xi(\lambda, \mu)$. When the spectral transform method is applied in a three-dimensional atmospheric model, the principal data structures are as shown in Fig. 1. P denotes the physical grid, F the Fourier grid, and S the spectral grid.

For the purposes of this paper, we will refer to the transformation from physical to spectral coordinates as the forward spectral transform, and its two phases as the forward Fourier and forward Legendre transforms, respectively. Similarly, the transformation from spectral to physical coordinates will be referred to as the inverse spectral transform, and the two phases as the inverse Legendre and inverse Fourier transforms, respectively. (With the exception of the Fourier transform, there is some variation in the literature as to which direction is the forward transformation and which is the inverse.)

In the shallow water equation code [21], each timestep begins by calculating the nonlinear terms $U\eta$, $V\eta$, $U\Phi$, $V\Phi$, and $\Phi + (U^2 + V^2)/(2(1-\mu^2))$ on the physical grid. Next, the nonlinear terms and the state variables η, δ, and Φ are Fourier transformed. The forward Legendre transforms of these fields are then combined with the calculation of the tendencies used in advancing η, δ, and Φ in time (essentially evaluating the right-hand sides of Equations (3)–(5)) and the first step of the time update. This approach decreases the cost, when compared with calculating transforms individually and then calculating the tendencies, and generates spectral coefficients for only three fields instead of eight. Next, the time updates of η, δ, and Φ on the spectral grid are completed. Finally, the inverse Legendre transforms of η, δ, and Φ are combined with the calculation of the fields U and V (solving Equations (6)–(9)), followed by inverse Fourier transforms of these five fields. Note that the associated Legendre polynomial values used in

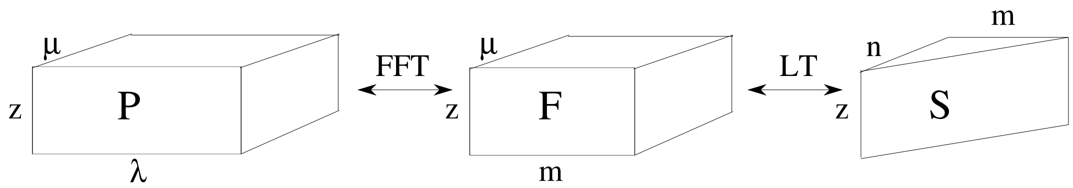


Figure 1: Principal Data Structures in Spectral Transform

the transforms are precomputed, decreasing the cost of the transforms at the cost of memory for two additional three-dimensional fields, one for the Legendre polynomials and one for the derivatives of the Legendre polynomials. This follows the practice used in CCM2.

Without significant loss of generality, we assume a triangular spectral truncation in this paper: $N(m) = M$ and the (m, n) indices of the spectral coefficients for a single vertical layer form a triangular grid. For a triangular truncation, exact, unaliased transforms of quadratic terms are obtained if $I \geq 3M + 1$ and if $I = 2J$ [35]. In this work we also use a fast Fourier transform (FFT) algorithm that requires I to be a power of two. As is commonly done, for a given M we choose I to be the minimum power of two satisfying $I \geq 3M + 1$, and set $J = I/2$. With these assumptions, the value of M can be used to characterize the horizontal resolution of the grids, and the term "TM" is used to denote a particular discretization. For example, for T85 we have $M = 85$, $I = 256$, and $J = 128$, and the number of spectral coefficients (N_{spec}) calculated per field for a single vertical layer is

$$N_{\text{spec}} \;=\; \sum_{m=0}^{M} \sum_{n=m}^{M} 1 \;=\; \frac{1}{2}(M+1)(M+2) \;=\; 3741.$$

The number of vertical levels is determined primarily by the physical processes that are being modeled and is chosen independent of M in current meteorological models. The term "TMLK" will be used to denote a model with a TM horizontal grid resolution and K vertical levels.

In subsequent discussion, we denote the index set of the physical grid by a triple $(\mathcal{I}, \mathcal{J}, \mathcal{K})$, with $\mathcal{I}$ corresponding to longitude, $\mathcal{J}$ to latitude, and $\mathcal{K}$ to the vertical. We denote the index set of the Fourier grid by the triple $(\mathcal{M}, \mathcal{J}, \mathcal{K})$, with $\mathcal{M}$ corresponding to wavenumbers. We denote the index set of the spectral grid by the triple $(\mathcal{M}, \mathcal{N}, \mathcal{K})$, with $\mathcal{N}$ corresponding to polynomial degree. (Note that in a triangular truncation, the index set $\mathcal{N}$ is dependent on the wavenumber.) We assume that computation is performed on a two-dimensional logical grid of $P = P_X \times P_Y$ processors. We denote an individual processor by an index pair (x, y).

Different phases of a parallel spectral transform algorithm may employ different decompositions of the computational grids onto the processor grid. We describe these by a triple, for example, of the form $(\mathcal{I}_a, \mathcal{J}_b, \mathcal{K}_c)$, where a, b, and c are X, meaning that indices in the subscripted dimension are partitioned over processors in the X plane of the processor grid; Y, meaning that indices are partitioned over processors in the Y plane; or null, meaning that indices are not partitioned. Analogous notations are used to represent the decompositions of the Fourier and spectral grids. We never decompose the domain over more than two dimensions. We assume that the physical space grid is always decomposed as $(\mathcal{I}_X, \mathcal{J}_Y, \mathcal{K})$ and that the physical domains of all fields are decomposed (and mapped to processors) in the same way, so that computations in vertical columns can proceed without communication. (These computations are not considered here but are an important and complex part of a climate model that are difficult to parallelize efficiently.) Unless otherwise noted, we also assume that all fields use the same Fourier and spectral domain decompositions.

We do not incorporate the vertical coupling in spectral space that is introduced by the use of a semi-implicit solver; this coupling is unnatural in a shallow water setting. We do not expect the interprocessor communication required to support the vertical coupling or the additional computation in spectral space to contribute significantly to total costs as it involves only a single spectral space field. This expectation has been verified empirically in the parallel version of the Integrated Forecast System developed at the European Centre for Medium-Range Weather Forecasting (personal communication, S. Barros, 1994) and in the parallel version of CCM2.

4. Parallel Algorithms and Architectures

Scalable parallel computers generally comprise a number of independent computers and an interconnection network. Each computer has its own processor and memory, can execute a sequential program, and can *send* and *receive* messages to and from other computers. In the absence of concurrent computation and communication, the run time for a parallel program on such a system can be characterized on a per-processor basis as

$$T = T_{\text{comp}} + T_{\text{comm}} \ , \tag{10}$$

where T_{comm} is the time spent (actively) communicating or waiting for messages to arrive and T_{comp} is the time spent computing (i.e., not communicating).

In all of the algorithms described in this paper, each send request is closely preceded or followed by a receive request for a message of approximately the same size. In current multiprocessors, the cost of such a send/receive pair can often be modeled with reasonable accuracy as

$$T_{\text{comm}} = t_s + N t_w \, , \tag{11}$$

where t_s is the time to initiate the communication requests, N is the size of the messages in words, and t_w is the time to transfer a single word of data into the network and transfer another word out of the network. By choosing t_s and t_w to reflect intrinsic system performance characteristics and defining T_{comm} to be the sum of the cost of these send/receive pairs, (10) becomes a lower bound on the execution time of the form

$$T = T_{\text{comp}} + \sum_i (t_s + N_i t_w) \ . \tag{12}$$

Costs omitted in this lower bound—for example, idle time waiting for messages to arrive or buffer copying associated with message passing—are generally proportional to the number of messages or to the message lengths in each of our algorithmic phases: physical domain computations, FFT, LT, and spectral domain computations. (This is due to the nature of our algorithms and does not hold in general.) Hence, by fitting (12) for a given phase to empirical data, system- and algorithm-dependent values for t_s and t_w can often be derived for which (12) is valid for a large range of problem sizes and numbers of processors.

Whether as a lower bound or as an empirically-fitted performance model, (12) is often sufficient to make accurate qualitative comparisons between parallel algorithms, and will be used in the algorithm analysis to follow. There are also two generalizations of this model that are important for some of the multiprocessor platforms included in this study, incorporating the impact of *computation/communication overlap* and *network bandwidth limitations*, respectively. In the following, we use an example to illustrate the simple model; we then introduce the generalizations.

4.1. Parallel Algorithm Example

We use the fast Fourier transform (FFT) to illustrate the use of (12). The Fourier transform, $\mathbf{y} = \{y_k\}$, of a sequence of D values $\mathbf{x} = \{x_j\}$ is given by

$$y_k = \sum_{j=0}^{D-1} x_j e^{2\pi \mathrm{i} jk/D} ,$$

where s $= \sqrt{-1}$. The FFT exploits symmetry to perform this computation in $\log_2 D$ steps, each involving $\Theta(D)$ operations. Assume that $\mathbf{x}$ is partitioned over Q processors by blocks, and let $D = 2^d$ and $Q = 2^q$ for some integer d and q, $d \geq q$. Each of the first q stages of the FFT involves a pairwise exchange of 2^{d-q-1} data or intermediate results with another processor, while the last $d - q$ stages can proceed without further communication after a single additional exchange [17, 19, 27, 30, 31]. Thus each processor engages in $1 + \log_2 Q$ communication operations, each involving the transfer of $D/2Q$ data, and communication costs are

$$T_{\text{ßcomm}} = (1 + \log_2 Q) \left(t_s + \frac{D}{2Q} t_w \right) . \tag{13}$$

The parallel and sequential algorithms perform exactly the same computation. As this computation is partitioned evenly among the Q processors, our performance model predicts that the time taken by the parallel code is

$$T = \frac{T_{\text{seq}}}{Q} + T_{\text{comm}}, \tag{14}$$

where T_{seq} is the time taken by the sequential code on a single processor. Notice that this parallel algorithms leaves the Fourier coefficients unordered. This avoids the need for an additional communication step, and (as discussed in §6.1) has the additional advantage when incorporated in the spectral transform of improving load balance during the LT step.

4.2. Computation/Communication Overlap

Some computers allow the effective cost of interprocessor communication to be reduced by overlapping computation with some of the operations performed to send or receive a message or with the time spent waiting for a message to arrive. A simple lower bound on the execution time when exploiting overlap is

$$T = \max\{T_{\rm comp}\, , \, T_{\rm comm}\};$$

hence, overlap at most halves the nonoverlap performance and does not change asymptotic behavior. We do not model explicitly the effect of overlap, but note when it can be used to reduce communication cost.

4.3. Network Bandwidth Limitations

Equation (12) assumes that the cost of sending a message is independent of the number of processors that are communicating at the same time. However, some interconnection network/algorithm combinations may result in multiple processors attempting to send messages over the same wire simultaneously. The impact of this behavior on performance can often be modeled with reasonable accuracy by assuming that the processors share available bandwidth, that is, by scaling the data volume term of our communication cost model by S, the number of processors sending concurrently:

$$T_{\rm comm\_bandwidth\_limited} = t_s + SNt_w \, . \tag{15}$$

The value S depends on properties of both the parallel algorithm and the underlying communication network. For example, the FFT described above can be organized to execute without competition for bandwidth on a hypercube [22]. In contrast, on a 1-D mesh of Q processors, each processor generates messages that must traverse 1, 2, ..., $2^{q-1}, 1$ hops distant in the $q+1$ steps of the algorithm [14, 19]. The total number of hops traversed by these messages is $Q(1+\sum_{i=0}^{q-1} 2^i) = Q^2$. This represents the number of wires to which a processor requires exclusive access during the FFT. Because a 1-D bidirectional mesh provides only $2(Q-1)$ wires, the algorithm cannot possibly proceed in less than Q/2 steps, rather than $(1+\log_2 Q)$ steps as supposed previously. Hence, the following model is a lower bound on communication costs:

$$T_{\rm fft\_mesh\_1d} = (1+\log_2 Q)t_s + \frac{D}{4} t_w \ . \tag{16}$$

5. Parallel Fourier Transform Algorithms

We now present the parallel spectral transform algorithms that we evaluate in this paper. By assumption, the physical and spectral domains for each field (η, δ, Φ, U, and V) are decomposed and mapped onto processors in the same fashion. Thus, the calculation of the nonlinear terms and the completion of the time update of η, δ, and Φ can proceed independently on each processor, and the computations will be load balanced if the decompositions are equipartitions of the index sets. These calculations have $\Theta(IJK)$ complexity, compared with $\Theta(IJK \log_2 J)$ for the Fourier transforms and $\Theta(I^2JK)$ for the Legendre transforms. Since any load imbalances will also reflect load imbalances in the Fourier or Legendre transforms, the effect of load imbalances on performance can be compared qualitatively by considering the transforms only. Hence, we discuss only parallel Fourier and Legendre transform algorithms.

For each algorithm that we consider, we develop performance models based on (12). We also consider the impact of bandwidth limitations in mesh architectures. On a hypercube we assume that the two-dimensional *logical* processor mesh of size $P_X \times P_Y = 2^q \times 2^r = 2^p$ is mapped into a hypercube of dimension p in such a way that each processor row and column is mapped to a subcube of dimension q and r, respectively [22]. Hence, performance analysis reduces to the problem of determining the cost of an FFT or LT in a hypercube. On a 2-D mesh computer, we assume that the $P_X \times P_Y$ logical processor mesh is mapped to an equivalent physical mesh. Thus, each FFT and LT algorithm executes in a 1-D processor array. Although this means that at most one half of the available wires are used in each communication phase, experiments suggest that this mapping is close to optimal when $P_X \approx P_Y$. Increasing the connectivity for one phase (e.g., FFT) of the spectral transform in order to improve performance generally decreases the performance of the other phase (e.g., LT) to a degree that offsets the earlier gain. Moreover, as will be shown later, $P_X \approx P_Y$ is generally optimal, so this assumption will not unduly affect the qualitative analysis.

We first consider parallel algorithms for the first phase of the spectral transform, in which real FFTs are performed on each row of the physical grid. The test case used in these experiments involves one forward FFT for each of 8 fields, J latitudes, and K vertical levels, and one inverse FFT for each of 5 fields, J latitudes, and K levels, per timestep. As $I = 2J$, we must perform $13JK$ FFTs per timestep, each on a vector of length $2J$. As noted in §3, we assume that the physical grid is initially decomposed as $(\mathcal{I}_X, \mathcal{J}_Y, \mathcal{K})$. We also assume that the $\mathcal{I}$ index set is partitioned over the P_X row processors in P_X equal-sized blocks and that P_Y divides J evenly. We will relax the latter assumption when considering load imbalances.

Our models and implementations assume that a single FFT and a single inverse FFT are performed per time step: that is, that the various fields, latitudes, and levels are blocked. This technique is efficient, but it requires additional memory for the FFT work array and can complicate the code when retrofitting an existing application. The reader can use our performance models to estimate the cost implications of performing additional FFTs, for example one FFT for each field. Note that the parallel version of CCM2 can be run with either a single pair of block FFTs per timestep or $J/2$ pairs of block FFTs per timestep, not blocking over latitude.

Table 1: Communication Characteristics of Parallel FFT Algorithms

Algorithm	Messages	Data Volume
Distributed	$4 + 2\log_2 P_X$	$13\frac{J^2K}{P}(2+\log_2 P_X)$
Overlapped Distributed	$8 + 4\log_2 P_X$	$13\frac{J^2K}{P}(2+\log_2 P_X)$
$\Theta(Q)$ transpose	$2(P_X - 1)$	$26\frac{J^2K}{P}\frac{P_X-1}{P_X}$
$\Theta(\log Q)$ transpose	$2\log_2 P_X$	$13\frac{J^2K}{P}\log_2 P_X$

Using the fully blocked FFTs is significantly faster and is used operationally.

An unordered real FFT is used in all experiments. This is cheaper than an ordered FFT, especially for the parallel FFT of §4, which would require additional communication to effect the ordering. It also provides some load balancing during the LT phase, as will be described in §6.

5.1. Distributed FFT

Our first FFT algorithm assumes $(\mathcal{I}_X, \mathcal{J}_Y, K)$ and $(M_X, \mathcal{J}_Y, K)$ decompositions of the physical and Fourier grids, respectively. Hence, both its input and output are decomposed across P_X processors, and we can use the algorithm presented in §4. There is no load imbalance if P_Y divides J evenly, and there is no redundant work. The real FFT used in the code requires an additional pairwise exchange over that described in (13), for a communication cost of

$$T_{\text{ßcomm}} = (2 + \log_2 Q)\left(t_s + \frac{D}{2Q}t_w\right). \tag{17}$$

Each row of P_X processors is responsible for transforming $1/P_Y$ of the physical grid, that is, computing $8KJ/P_Y$ forward FFTs and $5KJ/P_Y$ inverse FFTs. The forward and inverse FFTs are each computed as a block, so the number of messages is that required for two single transforms. As each FFT is applied to a vector of length I, the two block FFTs transform $8KIJ/P_Y = 16KJ^2/P_Y$ and $5KIJ/P_Y = 10KJ^2/P_Y$ data per processor row, respectively. Substituting the data volume values for D and P_X for Q and using $P = P_XP_Y$, we obtain from (17) the communication cost expression in Table 1.

Computation/Communication Overlap. To exploit overlap, the single-block FFT can be divided into two, allowing one block's communication to be overlapped with the other's computation [34]. Only the first swap involving the first block is not overlapped with computation.

Table 2: FFT Performance Models Specialized for 1-D Mesh (where they differ)

Algorithm	Revised Data Volume
Distributed	$\frac{13}{2}\frac{J^2K}{P}P_X$
$\Theta(Q)$ transpose	$\frac{13}{3}\frac{J^2K}{P}(P_X+1)$
$\Theta(\log Q)$ transpose	$\frac{13}{2}\frac{J^2K}{P}P_X$

This process requires twice as many messages, as indicated in Table 1, but has been shown to be cost effective on some multiprocessors.

Bandwidth Limitations. Both the one- and two-block algorithms can be mapped to a hypercube without competition for bandwidth [17]. As noted in §4.3, they will suffer from bandwidth limitations on a 1-D mesh. Applying (16) to the shallow water code, we obtain the expression in Table 2. This omits the additional exchange required in the real FFT, but is still a reasonable lower bound.

Algorithm Limitations. The basic operation in the standard power-of-two FFT is a "butterfly" transform involving two complex values. This corresponds to four real values in the real FFT, and at least four longitudes must be assigned to each processor to avoid redundant computation. Thus, if $I = 2^d$, we are restricted to $P_X \leq 2^{d-2}$. The distributed FFT algorithm used in this study also requires that P_X be a power of two.

Load Balance. As will be discussed in §6, the choice of Legendre algorithm determines whether the $\mathcal{J}$ index set is partitioned over the P_Y column processors in P_Y or $2P_Y$ equal-sized blocks. Two blocks are assigned per processor column in the latter case. If P_Y does not divide J (or $J/2$) evenly, load is somewhat unbalanced, with the processor row with maximum load computing $13K\lceil J/P_Y\rceil$ (or $26K\lceil J/(2P_Y)\rceil$) FFTs. This imbalance increases both data volume and computation cost proportionally. See Table 5.

5.2. Transpose FFT

An alternative algorithm reorganizes the physical grid from $(\mathcal{I}_X, \mathcal{J}_Y, \mathcal{K})$ to $(\mathcal{I}, \mathcal{J}_Y, \mathcal{K}_X)$ prior to the forward FFT so that each latitude row is stored within a single processor [1, 3, 6, 24, 28]. This eliminates the need for communication during the FFT, but requires communication within the *transpose* used for the reorganization. After the transform, the Fourier grid is decomposed as $(\mathcal{M}, \mathcal{J}_Y, \mathcal{K}_X)$. The inverse FFT proceeds similarly, requiring a transpose after the transform to reorganize from $(\mathcal{I}, \mathcal{J}_Y, \mathcal{K}_X)$ to $(\mathcal{I}_X, \mathcal{J}_Y, \mathcal{K})$.

The transpose requires that each processor exchange information with the other P_X processors in the same row of the processor grid. The two primary implementation approaches require $\Theta(P_X)$ and $\Theta(\log P_X)$ communication steps, respectively.

$\Theta(Q)$ Transpose. The first algorithm proceeds in $Q-1$ steps on Q processors: at each step, each processor sends $1/Q$ of its data to another processor [12, 22, 30]. Communication cost is as follows.

$$T_{\text{linear transpose}} = (Q-1)\left(t_s + \frac{D}{Q^2}t_w\right) \tag{18}$$

Substituting appropriate values for D and Q and counting both the forward and inverse FFTs, we obtain the expression in Table 1.

Note that for this algorithm to be efficient, and for (18) to hold, some care must be taken with the order of the data communication. For example, significant contention can result if all processors send to processor i in the ith step. The schedules used in our experiments send at most one message to each processor during a given step.

$\Theta(\log Q)$ Transpose. The transpose can be performed in $(\log_2 Q)$ communication steps at the cost of increased communication volume [13, 29]. We first partition processors into two sets. Each processor sends to the corresponding processor in the other set a single message containing all the data that it possesses that is destined for processors in the other set. This partitioning and communication process is repeated $\log Q$ times until each set contains a single processor. Each message has size $D/(2Q)$, so the total communication volume is $(\log_2 Q)/2$ times greater than in the $\Theta(P)$ algorithm, and communication costs are

$$T_{\text{log transpose}} = \log_2 Q \, \left(t_s + \frac{D}{2Q}t_w\right). \tag{19}$$

When applied to the FFTs in the shallow water code, communication costs are as in Table 1.

Computation/Communication Overlap. Overlap can be introduced in the transpose algorithms by breaking up a one-block transform comprising F vectors into B blocks of size F/B. After the first block is completely transposed, the transpose of a block can (potentially) be overlapped with the transform of the block preceding it.

This algorithm has not proven to be efficient in practice. Large B minimizes F/B, the size of the block whose transpose is not overlapped, but the number of messages grows by a factor of B, and not all message startup costs can be overlapped. Also, the transform must be divided into the same number of stages as the transpose algorithm to allow for interleaving. This restriction may diminish the computational rate.

Bandwidth Limitations. Neither transpose algorithm suffers from significant bandwidth limitations on hypercubes [12, 17, 22], but both do so on mesh architectures. In the $\Theta(Q)$ transpose, a total of $(Q^3-Q)/3$ hops are traversed on $2(Q-1)$ wires, requiring that the data volume be scaled by $Q(Q+1)/6$ instead of $Q-1$. The scaling factor for the $\Theta(\log Q)$ transpose is the same as that used for the distributed FFT. See Table 2.

Algorithm Limitations. Both transpose algorithms decompose the vertical dimension and thus require that $P_X \leq K$ if whole processor rows are not to be idle during the FFT. As K can be significantly smaller than I, this restriction is limiting for the transpose algorithms. One approach to mitigating this problem is to decompose also over the field "dimension" (8 for the forward FFT and 5 for the inverse) [23]. Many of these fields must be reunited for the LT phase, however, resulting in other performance problems. This generalization and the associated problems are discussed in §9. The $\Theta(\log Q)$ transpose algorithm requires that P_X be a power of two.

Load Balance. If P_Y does not divide J or $J/2$ evenly, load is unbalanced as in the distributed FFT algorithm. There is also load imbalance if P_X does not divide K evenly; some processor columns must compute FFTs for as many as $\lceil K/P_X \rceil$ vertical levels. See Table 5. An analogous load imbalance does not occur in the distributed FFT because I and P_X are both required to be powers of two.

6. Parallel Legendre Transform Algorithms

We next consider parallel algorithms for the second phase of the spectral transform, in which Legendre transforms (LT) are performed on each column of the Fourier grid. We define a single forward transform to be the calculation of the set of spectral coefficients $\{\xi_n^m \,|\, n = |m|, \ldots, N(m)\}$ for a given wavenumber m and field variable ξ, and an inverse transform to be the calculation of the set of Fourier coefficients $\{\xi^m(\mu_j) \,|\, 1 \leq j \leq J\}$ for a given wavenumber m and field variable ξ. Thus the number of spectral coefficients output (input) for each field in the forward (inverse) transform is a function of the wavenumber and of the spectral truncation used. Since we assume a triangular truncation, $M - m$ spectral coefficients are generated for wavenumber m.

At each timestep, the shallow water code performs one forward LT for each of three fields, M wavenumbers, and K vertical levels, and one inverse LT for each of five fields, M wavenumbers, and K vertical levels. Eight fields of Fourier coefficients are used to produce the three fields of spectral coefficients, and these three spectral fields are used to produce the five fields of Fourier coefficients. As we assume that $J \approx (3M+1)/2$ and $I = 2J$, the total number of spectral coefficients produced/consumed in these transforms is $3N_{\mathrm{spec}}K = (3/2)(M+1)(M+2)K \approx (1/3)(J+1)(2J+5)K$.

We describe four Legendre transform algorithms. The first two use distributed vector sum algorithms to complete the LT, while the third and fourth use the transpose algorithms of the preceding section. Each algorithm can be used with any FFT algorithm, but load balance may vary. To simplify the exposition, we assume initially that the distribution of spectral coefficients between the different processor columns is uniform, that is, approximately $(1/3)(J+1)(2J+5)K/P_X$ spectral coefficients per processor column.

6.1. Distributed LT

The first two LT algorithms assume either $(\mathcal{M}_X, \mathcal{J}_Y, \mathcal{K})/(\mathcal{M}_X, \mathcal{N}_Y, \mathcal{K})$ decompositions of Fourier and spectral space, respectively, or $(\mathcal{M}, \mathcal{J}_Y, \mathcal{K}_X)/(\mathcal{M}, \mathcal{N}_Y, \mathcal{K}_X)$ decompositions. The simple forward LT is computed as

$$\xi_n^m = \sum_{j=0}^{J-1} \xi^m(\mu_j) P_n^m(\mu_j) w_j = \sum_{y=0}^{P_Y-1} \left(\sum_{j \in \mathcal{J}_y} \xi^m(\mu_j) P_n^m(\mu_j) w_j \right) \equiv \sum_{y=0}^{P_Y-1} T_n^m(y) \ . \tag{20}$$

Each partial sum $T_n^m(y)$ can be evaluated within a processor (x, y) without interprocessor communication. The final calculation of the spectral coefficient ξ_n^m requires the summation of P_Y partial sums distributed over P_Y processors. A "column-wise" distributed vector sum algorithm can be used to perform this summation in a block fashion for all spectral coefficients, fields, and vertical levels associated with a given processor column. The same approach can also be used with the more complicated transforms producing ξ_n^m from multiple fields of Fourier coefficients.

The simple inverse LT is computed as

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

Each processor can calculate its associated Fourier grid values independently if the (distributed) vector of spectral coefficients $\{\xi_n^m\}$ is first replicated on all P_Y processors in the given processor column. This requires a broadcast operation prior to the inverse LT. The same approach also works when more than one field of spectral coefficients is needed to evaluate the Fourier grid values.

For ease of coding and interprocessor communication efficiency, we have found it useful to combine the distributed vector sum and broadcast in a single operation. Thus, at the end of the forward LT, all processors in a given column have the same spectral coefficients, and the decomposition of the spectral grid is $(\mathcal{M}_X, \mathcal{N}, \mathcal{K})$, a one-dimensional rather than two-dimensional decomposition. A disadvantage of this approach is that a small amount of computation that modifies the spectral coefficients between the forward and inverse LTs must be performed redundantly on the replicated coefficients. But the complexity of this computation is of a lower order and has a smaller constant than that involved in the LT operations. In our experiments, the savings due to improved communication efficiency easily outweigh the cost of the redundant computation. The redundant computation is ignored in subsequent analysis.

Ring Sum. We now describe the first of two LT algorithms based on this structure (distributed vector sum and broadcast). These algorithms differ only in the mechanisms used to

Table 3: Communication Characteristics of Parallel LT Algorithms

Algorithm	Messages	Data Volume
Ring sum	$2(P_Y - 1)$	$\frac{4}{3}\frac{(J+1)(2J+5)K}{P}(P_Y - 1)$
Butterfly sum	$2\log_2 P_Y$	$\frac{4}{3}\frac{(J+1)(2J+5)K}{P}(P_Y - 1)$
$\Theta(Q)$ transpose	$2(P_Y - 1)$	$\frac{52}{3}\frac{(J+1)JK}{P}\frac{P_Y - 1}{P_Y}$
$\Theta(\log Q)$ transpose	$2\log_2 P_Y$	$\frac{26}{3}\frac{(J+1)JK}{P}\log_2 P_Y$

Table 4: LT Performance Models Specialized for 2-D Mesh (where they differ)

Algorithm	Revised Data Volume
Butterfly sum	$\frac{2}{3}\frac{(J+1)(2J+5)K}{P}P_Y\log_2 P_Y$
$\Theta(Q)$ transpose	$\frac{26}{9}\frac{(J+1)JK}{P}(P_Y + 1)$
$\Theta(\log Q)$ transpose	$\frac{13}{3}\frac{(J+1)JK}{P}P_Y$

sum the vectors of partial sums T_n^m and to replicate the results. In the *ring sum* algorithm, data flows around a logical ring of processors. A summation involving Q processors proceeds in $Q-1$ steps, with each processor receiving D/Q data from its left neighbor and sending D/Q data to its right neighbor at each step. Upon completion, the vector of D spectral coefficients is evenly distributed over the Q processors. This process is reversed (without the summations) to broadcast the result. Communication costs are

$$T_{\text{ring sum}} = 2(Q-1)\left(t_s + \frac{D}{Q}t_w\right). \tag{21}$$

In the shallow water code, $Q = P_Y$ and $D \approx (2/3)(J+1)(2J+5)K/P_X$ (because the spectral coefficients are complex (two-word) values), giving the expression in Table 3.

Butterfly Sum. The *butterfly sum* algorithm is a hybrid of two algorithms [33]. For long vectors, we use a recursive halving algorithm [17] that utilizes a butterfly communication pattern like the distributed FFT. Each processor communicates (and sums) $D/2$ data in the first stage, half as much ($D/4$) in the second, and so on, so that each processor communicates a total of $D(Q-1)/Q$ data in ($\log_2 Q$) steps. The global sum is then complete, and the vector of D spectral coefficients is evenly distributed over the Q processors. This process is reversed (without the summations) to broadcast the result. Total communication cost is as follows:

$$T_{\text{butterfly sum}} = 2 \log_2 Q \; t_s + 2D \frac{Q-1}{Q} t_w \ . \tag{22}$$

When the vector becomes small, the hybrid algorithm switches to an exchange algorithm in which each processor communicates all the remaining data at each subsequent step. This eliminates some of the broadcast communication. The vector length at which the hybrid algorithm switches is a machine-dependent constant, and the communication cost of the butterfly sum is well characterized by (22). This approximation is used for the expression in Table 3.

Computation/Communication Overlap. The computation of the local sums $\{T_n^m(y)\}$ can be interleaved with stages of the distributed vector sum algorithms. Similarly, the broadcast can be delayed until the computation of the inverse LT, and the stages of the broadcast interleaved with computation. This eliminates the redundant computation on the spectral grid, because the broadcast is delayed, and does not change the number of messages or data volume.

When the interleaving is organized so that the communication of one stage of the algorithm is overlapped with the computation of the next stage, the ring sum is able to perform $\Theta(J^4/P_Y)$ computation while communicating $\Theta(P_Y J^3)$ data [37]. This overlapping can be highly effective for small P_Y and/or large J, decreasing the cost of communication significantly.

Overlap is less effective for the butterfly sum. Interleaving only applies to the recursive halving phase of the algorithm, and the communication of a vector of length $D/2^i$ is overlapped with the computation of local sums for a vector of length $D/2^{i+1}$, i.e., half the size, rather than the same size as in the ring sum. Due to time constraints, we have evaluated the overlap technique only in the ring sum algorithm.

Bandwidth Limitations. A ring can be embedded in a hypercube or bidirectional mesh, and ring sum does not suffer from bandwidth limitations on either interconnection topology.

The exchange and recursive halving components of the butterfly sum algorithm have the same communication structure as the FFT and, thus, suffer from bandwidth limitations on a mesh. But the recursive halving component suffers less than the distributed FFT. In the first step, $D/2$ data are exchanged with nearest neighbors without competition. In the second step, $D/4$ data are exchanged with processors 2 hops distant and 2 processors compete for each wire. In the fourth step, 4 processors must send $D/8$ data over the same wire, and so on, with the result that data volume must be scaled by $(1/2)\log_2 Q$. As before, we use the value for the recursive halving algorithm in qualitative comparisons, giving the expression in Table 4.

Algorithm Limitations. In order to exploit symmetry (i.e., to avoid computing spectral coefficients for negative wavenumbers), corresponding latitudes from the northern and southern hemispheres are paired. Hence, the $\mathcal{J}$ index set is partitioned over the P_Y column processors into $2P_Y$ equal-sized blocks, and two blocks are assigned to each processor column. Thus,

Table 5: Relative Increase in Computation and Communication Costs in Fourier Transforms As a Result of Load Imbalances

Algorithm	Computation Cost	Data Volume
Dist. FFT/Dist. LT	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)}$	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)}$
Trans. FFT/Dist. LT	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)} \frac{\lceil K/P_X\rceil}{K/P_X}$	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)} \frac{\lceil K/P_X\rceil}{K/P_X}$
Dist. FFT/Trans. LT	$\frac{\lceil J/P_Y\rceil}{J/P_Y}$	$\frac{\lceil J/P_Y\rceil}{J/P_Y}$
Trans. FFT/Trans. LT	$\frac{\lceil J/P_Y\rceil}{J/P_Y} \frac{\lceil K/P_X\rceil}{K/P_X}$	$\frac{\lceil J/P_Y\rceil}{J/P_Y} \frac{\lceil K/P_X\rceil}{K/P_X}$

$P_Y \leq J/2$ if whole processor rows are not to be idle during the LT. Similarly, $P_X \leq M+1 \approx (2/3)(J+1)$, if whole processor columns are not to be idle.

Load Balance. Load imbalance arises if P_Y does not divide $J/2$ evenly, with the processor row with the maximum load computing $2c\lceil J/(2P_Y)\rceil$ flops per spectral coefficient instead of $c(J/P_Y)$, for some constant c. The communication volume does not change because spectral coefficients are being communicated, not Fourier coefficients.

The performance of the distributed LT algorithms is also affected by the FFT algorithm used. As the Fourier transform is unordered, the distributed FFT algorithm assigns blocks of permuted Fourier coefficients to the P_X processor columns. This assignment approximately balances the assignment of "short" Legendre transforms (large wavenumbers) and "long" Legendre transforms (small wavenumbers) [34], but the load balance is not perfect and some processor columns have more work than others. A simple (over)estimate of the maximum number of spectral coefficients assigned to a processor column is

$$\frac{3}{2}\frac{(M+1)^2K}{P_X} + \frac{3}{2}(M+1)K + \frac{3}{8}KP_X \quad \text{for} \quad P_X \leq (M+1)$$

and $3(M+1)K$ for $P_X \geq M+1$, in contrast to $(1/3)(J+1)(2J+5)K/P_X$ for a load balanced assignment. If a transpose FFT is used, then the wavenumber dimension is not partitioned. The maximum number of spectral coefficients assigned to a processor column is $(3/2)(M+1)(M+2)\lceil K/P_X\rceil$, and there is load imbalance if P_X does not divide K evenly.

Load imbalance in the assignment of spectral coefficients affects both the communication and computation costs, scaling both proportionately, as indicated in Tables 5 and 6.

Memory Requirements The distribution $(\mathcal{M}_X, \mathcal{J}_Y, K)$ allows the $M.N.J$ Legendre polynomials $(P_n^m(\mu_j))$ to be distributed in two dimensions, and hence avoids replicating this large

Table 6: Relative Increase in Computation and Communication Costs in Legendre Transforms As a Result of Load Imbalances (worst-case approximations)

Algorithm	Computation Cost	Data Volume
Dist. FFT/Dist. LT:		
$P_X < \frac{2}{3}(J+1)$	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)} \left(1+\frac{3}{4}\frac{P_X}{J+1}\right)^2$	$\left(1+\frac{3}{4}\frac{P_X}{J+1}\right)^2$
$P_X \geq \frac{2}{3}(J+1)$	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)} \frac{6P_X}{2J+5}$	$\frac{6P_X}{2J+5}$
Trans. FFT/Dist. LT	$\frac{\lceil J/(2P_Y)\rceil}{J/(2P_Y)} \frac{\lceil K/P_X\rceil}{K/P_X}$	$\frac{\lceil K/P_X\rceil}{K/P_X}$
Dist. FFT/Trans. LT:		
$P_X < \frac{2}{3}(J+1)$	$\frac{\lceil K/P_Y\rceil}{K/P_Y} \left(1+\frac{3}{4}\frac{P_X}{J+1}\right)^2$	$\frac{\lceil K/P_Y\rceil}{K/P_Y} \left(1+\frac{3}{4}\frac{P_X}{J+1}\right)^2$
$P_X \geq \frac{2}{3}(J+1)$	$\frac{\lceil K/P_Y\rceil}{K/P_Y} \frac{6P_X}{2J+5}$	$\frac{\lceil K/P_Y\rceil}{K/P_Y} \frac{6P_X}{2J+5}$
Trans. FFT/Trans. LT:		
$P_Y < \frac{2}{3}(J+1)$	$\frac{\lceil K/P_X\rceil}{K/P_X} \left(1+\frac{9}{4}\frac{P_Y^2}{(J+1)^2}\right)$	$\frac{\lceil K/P_X\rceil}{K/P_X} \left(1+\frac{9}{4}\frac{P_Y^2}{(J+1)^2}\right)$
$P_Y \geq \frac{2}{3}(J+1)$	$\frac{\lceil K/P_X\rceil}{K/P_X} \frac{6P_Y}{2J+5}$	$\frac{\lceil K/P_X\rceil}{K/P_X} \frac{6P_Y}{2J+5}$

data structure. In contrast, the distribution $(\mathcal{M}, \mathcal{J}_Y, K_X)$ requires that the Legendre polynomials be replicated in each set of processors assigned a different set of vertical levels. This increases memory requirements for the Legendre polynomial values by a factor of P_X.

Similarly, the algorithms that combine the broadcast with the distributed vector sum use a $(\mathcal{M}_X, \mathcal{N}, \mathcal{K})$ or $(\mathcal{M}, \mathcal{N}, \mathcal{K}_X)$ decomposition of spectral space instead of a $(\mathcal{M}_X, \mathcal{N}_Y, \mathcal{K})$ or $(\mathcal{M}, \mathcal{J}_Y, K_X)$ decomposition. These one-dimensional decompositions require the replication of the spectral coefficients among processors assigned the same set of wavenumbers, increasing memory requirements for the spectral fields by a factor of P_Y.

For large problems, large numbers of processors, and/or small memory systems either of these increases can be problematic.

6.2. Transpose LT

The transpose algorithms of §5.2 can also be used to reorganize the Fourier grid so that the LT can proceed without further communication. If the FFTs are computed using a distributed algorithm, then the forward LT requires that the Fourier grid first be reorganized from $(\mathcal{M}_X, \mathcal{J}_Y, K)$ to $(\mathcal{M}_X, \mathcal{J}, \mathcal{K}_Y)$. The transform then produces a $(\mathcal{M}_X, \mathcal{N}, \mathcal{K}_Y)$ decomposition of the spectral grid. For the inverse LT, no further reorganization is needed, but the Fourier grid must be returned to the original $(\mathcal{M}_X, \mathcal{J}_Y, K)$ decomposition before the inverse FFT algorithm can begin. If the FFTs are computed using a transpose algorithm, then the Fourier grid must be reorganized from $(\mathcal{M}, \mathcal{J}_Y, K_X)$ to $(\mathcal{M}_Y, \mathcal{J}, \mathcal{K}_X)$ before the forward LT, and from $(\mathcal{M}_Y, \mathcal{J}, \mathcal{K}_X)$ to $(\mathcal{M}, \mathcal{J}_Y, K_X)$ after the inverse LT.

In both cases, the reorganizations require P_X independent transposes, each involving P_Y processors. Note that these transpositions involve the truncated Fourier grid rather than the spectral coefficients: that is, $(M+1) \times J \times K$ complex values. Hence, assuming perfect load balance, $D = 16(M+1)JK/P_X$ for the transpose preceding the forward LT and $D = 10(M+1)JK/P_X$ for the transposition following the inverse LT. Adapting (18) and (19) to this situation and using $2J \approx 3M+1$, we obtain the expressions in Table 3 for LT-related communication costs when using transpose algorithms.

The transpose LT algorithms, like the transpose FFT algorithms, become less efficient when modified to overlap computation with communication. Bandwidth limitations on a mesh affect the transpose LT algorithms in the same ways as the transpose FFT algorithms, as indicated in Table 4.

Algorithm Limitations. The distributed FFT/transpose LT algorithm decomposes the vertical dimension before computing the LT and, hence, requires $P_Y \leq K$ to avoid idle processor rows. For processor columns not to be idle requires $P_X \leq M+1 \approx (2/3)(J+1)$.

Conversely, the transpose FFT/transpose LT algorithm decomposes the wavenumber dimension before computing the LT and requires $P_Y \leq M+1$ to avoid idle processor rows. For processor columns not to be idle requires $P_X \leq K$.

Load Balance. Load imbalances occur in the distributed FFT/transpose LT algorithm if P_Y does not divide K evenly, and in the transpose FFT/transpose LT if P_X does not divide K evenly.

The distributed FFT/transpose LT is also subject to load imbalance as a result of the distribution of spectral coefficients generated by the distributed FFT, as described for the distributed FFT/distributed LT algorithm. Because the transpose FFT/transpose LT algorithm also partitions the wavenumber dimension, it also suffers from load imbalance. Since all equipartitions incur the same communication costs in the transpose algorithms, we minimize load imbalance by using the partitioning strategy described by Barros and Kauranne [3]. This pairs "short" transforms with "long" transforms in the assignment, and there is no load imbalance when P_Y divides $(M+1)/2$ evenly. Let

$$\alpha = \frac{M+1}{2P_Y} - \left\lfloor \frac{M+1}{2P_Y} \right\rfloor .$$

With the Barros-Kauranne strategy, the maximum number of spectral coefficients for a given vertical level assigned to a processor column is

$$\frac{3}{2}\frac{(M+1)(M+2)}{P_Y} \;+\; 6P_Y\alpha(1-\alpha)-3\alpha \;\text{ for }\; P_Y \leq (M+1)$$

and $3(M+1)$ for $P_Y \geq M+1$. Load imbalances in the assignment of spectral coefficients affect both the communication and computation costs, scaling both proportionately. See Tables 5 and 6.

Memory Requirements The distributions $(\mathcal{M}_X, \mathcal{J}, K_Y)$ and $(\mathcal{M}_Y, \mathcal{J}, K_X)$ produced by the two transpose algorithms both require replication of the Legendre polynomial values, in much the same manner as discussed in §6.1.

7. Qualitative Analysis

The three FFT algorithms described in §5 perform exactly the same computations. The four LT algorithms described in §6 perform essentially the same computations, modulo different partial orders and some redundant computations on the spectral grid. The FFT and LT algorithms are distinguished primarily by their communication performance and their load balance. In this section, we use the communication cost and load balance models (Tables 1–6) to make simple qualitative comparisons.

7.1. Parallel FFT Algorithm Comparisons

Data Transfer Costs. $\Theta(Q)$ transpose communicates the least data: $\Theta(J^2K/P)$ per processor versus $\Theta((\log_2 P_X)J^2K/P)$ for $\Theta(\log Q)$ transpose and distributed FFT. Hence, $\Theta(Q)$ transpose should perform better on large problems, particularly if data transfer costs (t_w) are high relative to message startup costs (t_s).

Message Startup Costs. Distributed FFT and $\Theta(\log Q)$ transpose send fewer messages than $\Theta(Q)$ transpose: $\Theta(\log Q)$ rather than $\Theta(Q)$. Hence, they should perform better when message startup costs are large relative to data transfer costs, and on problems that are small relative to the number of processors: that is, when J^2K/P is small.

Computation/Communication Overlap. Distributed FFT communicates the most data but is the most efficient at overlapping communication with computation and, hence, should perform better on computers that support computation/communication overlap.

Bandwidth Limitations. Distributed FFT and $\Theta(\log Q)$ transpose suffer less than $\Theta(Q)$ transpose from bandwidth limitations on mesh networks; when these are taken into account, their effective data volumes differ only by constant factors from that of the $\Theta(Q)$ transpose.

7.2. Parallel LT Algorithm Comparisons

Data Transfer Costs. For large P_Y, $\Theta(Q)$ transpose communicates the least data: $\Theta(J^2K/P)$ per processor versus $\Theta((\log_2 P_Y)J^2K/P)$ for $\Theta(\log Q)$ transpose and $\Theta(P_Y J^2K/P)$ for ring and butterfly sum. Hence, $\Theta(Q)$ transpose should perform better on large problems, particularly if data transfer costs are high relative to message startup costs.

On smaller numbers of processors, the coefficient as well as the complexity of the data transfer term needs to be considered. As the $\Theta(Q)$ transpose operates on the truncated Fourier grid, each processor communicates at least $(52/3)J^2K(P_Y-1)/(PP_Y)$ words. The summation algorithms operate on the spectral coefficients. Thus, ring and butterfly sum need move only about $(8/3)(J^2K)(P_Y-1)/P$ words, and, for $P_Y < 7$, these algorithms communicate less data than the transpose.

Message Startup Costs. As butterfly sum and $\Theta(\log Q)$ transpose send the fewest messages ($2\log_2 P_Y$ versus $2P_Y$ for the others), they should be superior on machines where message startup costs are large and on problems that are small relative to the number of processors. Again, the data volume term for $\Theta(\log Q)$ transpose is asymptotically smaller than for the butterfly sum, but is larger for $P_Y < 14$.

Computation/Communication Overlap and Bandwidth Limitations. Ring sum can overlap computation and communication when P_Y is small or J is large. It does not suffer from bandwidth limitations on mesh computers and has the smallest data volume on mesh computers.

Memory Requirements Algorithms that decompose spectral space in the vertical require replication of the Legendre polynomials. The butterfly and noninterleaved ring sum algorithms duplicate spectral coefficients and require more memory than the interleaved ring sum algorithm.

7.3. Parallel Spectral Transform Algorithm Comparisons

A parallel spectral transform algorithm must specify not only a parallel FFT and LT algorithm but also a processor grid aspect ratio (P_X and P_Y) for a given number of processors. P_X and P_Y are the processors used in the FFT and LT, respectively.

Aspect Ratio. Algorithmic comparisons are complicated by the fact that different combinations of FFT and LT algorithms perform best with different aspect ratios. For example, consider the algorithm combination $\Theta(Q)$ transpose FFT and $\Theta(Q)$ transpose LT. Here, a square grid is most efficient, as comparable amounts of data are moved in each phase in the same way, and overlap is not exploited, so the difference in computational cost between the two phases is not an issue. In contrast, for distributed FFT/ring sum LT it is most efficient to (a) apply all processors to the LT for small P; (b) use remaining processors for the FFT until communication costs in each phase are comparable; and (c) use an aspect ratio that favors the FFT increasingly for large P. The reason is that for small P the ring sum moves less data than does the distributed FFT and permits more computation/communication overlap. For large P, the ring sum sends more messages and more data, and the amount of computation available to overlap communication is small. This analysis changes on meshes, where the ring sum is favored if message startups do not dominate communication cost.

These considerations suggest that it is not sufficient to perform separate studies of FFT and LT algorithms, with the goal of selecting an optimal FFT and an optimal LT algorithm for inclusion in a parallel spectral transform. Instead, we must consider all possible pairs of algorithms at all possible aspect ratios.

Load Balance. Load balance issues arise when evaluating different FFT/LT algorithm combinations. Load balance is sensitive to problem size and number of processors, so general comparisons are difficult to make. While load balance problems can often be avoided by intelligent choices of the number of processors and aspect ratio of the logical grid, these choices implicitly represent algorithm limitations for the given algorithm combination.

There are two situations in which load imbalance is difficult to avoid:

1. The number of vertical levels is relatively small in climate models. Hence, transpose algorithms tend to need to apply relatively more processors in the other dimensions to avoid idle processors. For example, a transpose FFT implies a need for more LT processors. Because communication costs increase with P, LT performance will be worse than if an equal number of processors were applied to both FFT and LT.
2. The partitioning of the spectral coefficients usually introduces some load imbalance. The distributed FFT suffers the most from this phenomenon, but transpose FFT/transpose LT is subject to it also.

The effects of load imbalances are summarized in Tables 5 and 6; we see that transpose FFT/distributed LT is slightly less sensitive to load imbalance problems than other algorithms.

7.4. Summary

The qualitative comparisons suggest that no single algorithm is likely to be optimal in all situations. The choice of algorithm depends on a variety of factors such as problem size, type of network, number of processors, available memory, and communication parameters.

8. Empirical Studies

As indicated in preceding sections, the analytic models introduced in Tables 1–4 can provide insights into performance issues. The models can also be used to evaluate scalability and to make rough performance estimates [14, 16, 23]. For definitive algorithm comparisons, however, empirical studies are required to calibrate and validate the models. We expect constant factors to matter for a large range of multiprocessor sizes, and the relative efficiency of the implementations of the different algorithms can also play a crucial role. For example, the $\Theta(\log Q)$ transpose requires more data copying than the other algorithms, increasing its "effective" t_w value,

In this section we describe our experimental vehicle, methodology, and results. We demonstrate that the best algorithms do vary with architecture, number of processors, and problem size. We compare the optimal algorithms with a robust and asymptotically optimal algorithm, to indicate the importance of optimizing on the different platforms. We also use the results of the algorithmic comparison to identify the performance-critical aspects of each platform and to make simple scaling predictions.

8.1. PSTSWM: A Testbed Code

To permit a fair comparison of the suitability of the various algorithms for atmospheric models, we have incorporated the algorithms in a single testbed code called PSTSWM (for parallel spectral transform shallow water model).

PSTSWM is a message-passing parallel implementation of the sequential Fortran code STSWM 2.0 [21]. STSWM uses the spectral transform method to solve the nonlinear shallow water equations on a rotating sphere; its data structures and implementation are based directly on equivalent structures and algorithms in CCM2.

PSTSWM differs from STSWM in one major respect: vertical levels have been added to permit a fair evaluation of the transpose algorithms. This is necessary because in a one-layer model, a transpose algorithm reduces to a one-dimensional decomposition of each grid and hence can utilize only a small number of processors. The addition of vertical levels also has the advantage of modeling more accurately the granularity of the dynamics computation in atmospheric model. In all other respects we have changed the algorithmic aspects of STSWM as little as possible. In particular, we did not change loop and array index ordering. Although such changes would probably improve performance of some algorithms, our goal was to have a code as similar to a real atmospheric model as possible.

PSTSWM is structured so that a variety of different algorithms can be selected by runtime parameters. The FFT can be calculated using the distributed, $\Theta(Q)$ transpose, or $\Theta(\log Q)$ transpose algorithms. The LT can be calculated using either the ring sum, butterfly sum, $\Theta(Q)$ transpose, or $\Theta(\log Q)$ transpose algorithms. In addition, the distributed FFT can use either the two-block algorithm that permits computation/communication overlap or the one-

Table 7: Parallel Computers Used in Empirical Studies

Name	OS	Processor	Network	P
nCUBE/2	VERTEX R3.2	nCUBE 2	hypercube	1024
iPSC/860	NX 3.3.2	i860	hypercube	128
DELTA	NX/M R1.5	i860	16×32 mesh	512
Paragon	OSF/1 R1.1.2	i860SP	16×32 mesh	512
Paragon	SUNMOS	i860SP	16×32 mesh	512

Name	t_s (μsec)	t_w (μsec)	Single-processor MFlops/sec
nCUBE/2	240	2.3	1.2
iPSC/860	200	1.4	9.8
DELTA	240	0.84	9.8
Paragon (OSF)	350	0.18	11.6
Paragon (SUNMOS)	230	0.04	11.6

block algorithm, and the ring sum LT can use either the overlap or nonoverlap algorithms. Additional parameters select a range of variants of each of these major algorithms [38]. Note that all parallel algorithms were carefully implemented, eliminating unnecessary buffer copying and exploiting our knowledge of the context in which they are called. At the present time, this allows us to achieve better performance than can be achieved by calling available vendor-supplied routines. Hence, it provides a fairer test of the parallel algorithms.

8.2. Target Computers

We performed experiments on the five parallel computer systems listed in Table 7. These systems have similar architectures and programming models, but vary considerably in their communication and computational capabilities. Our values for t_s and t_w differ from those reported by most researchers [10, 11] because we measure the time required to swap floating-point values between two processors rather than the time to send bytes from a source to a destination. Also, the computational rate is measured by running PSTSWM on a single node and so is an achieved rather than a peak rate.

The Paragon experiments used the OSF-based R1.1.2 operating system and the low-overhead SUNMOS operating system from Sandia National Laboratories and the University of New Mexico. As both systems are still evolving, any conclusions as to their performance will be short-lived. They are interesting for this study, however, because they have significantly different performance characteristics.

8.3. Methodology

PSTSWM incorporates too many algorithmic variants to permit a comprehensive study of all possible combinations of parameters, problem size, computer, and processor count. Hence, we proceeded in two stages: algorithm selection and algorithm comparison.

Algorithm Selection. We first performed a series of tuning experiments to identify "optimal" communication parameters for each FFT and LT algorithm variant on each computer. For example, these parameters specify whether to use blocking or nonblocking sends and receives, or what schedule to use when the order of communication requests is not fixed by the algorithm. These experiments were performed using one-dimensional decompositions ($P_X = 1$ or $P_Y = 1$), allowing FFT and LT algorithms to be studied in isolation. Problem dimensions were reduced to provide the correct computation and communication granularities. For example, when evaluating the ring sum algorithm for a 16×8 processor grid at T85L32 resolution, a 1×8 processor grid was used with the number of vertical levels reduced by a factor of 16 (from 32 to 2).

We initially evaluated communication parameters only on "largest" and "mid-sized" computer configurations: for example, $P = 512$ and $P = 128$ on the Paragon. If one set of communication parameters proved consistently superior, no further experiments were performed. We expected the performance impact of communication parameters to be insensitive to problem granularity and number of processors, and we found this to be true in most cases. When a difference was significant, we selected the parameters that worked best for the larger configurations. In all cases, subsequent experiments for a given platform and algorithm used a fixed set of communication parameters. These experiments were also used to eliminate noncompetitive FFT and LT algorithmic variants.

Algorithm Comparisons. A second set of experiments compared all possible combinations of the remaining FFT and LT algorithms on all possible aspect ratios for each power of two number of processors supported by each computer. For example, on the Intel DELTA and Paragon, we used 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512 processors; for 32 processors, we tried the aspect ratios 1×32, 2×16, 4×8, 8×4, 16×2, and 32×1.

To measure the importance of the algorithm tuning and comparison, we repeated these experiments using a reference algorithm comprising the $\Theta(Q)$ transpose FFT and LT algorithms. The reference algorithm uses a particularly simple and portable communication protocol and is asymptotically optimal in the sense that it has the smallest data volume (t_w); as the problem size grows, this term comes to dominate communication costs in all of the parallel algorithms.

All experiments used the performance benchmark described in [36]: global steady state nonlinear zonal geostrophic flow. Experiments were performed for problem sizes T21L8, T42L16, and T85L32.

8.4. Results: Algorithm Selection

In presenting the results of the algorithm selection experiments, we do not discuss the communication parameters studied (see [38] for details) but focus on the algorithms. Table 8 summarizes both the algorithms considered and those selected for further consideration on different machines. For the most part, the table is self-explanatory. We always selected at least one distributed algorithm and one transpose algorithm for both the LT and FFT. In some cases, two distributed or transpose algorithms were selected, indicating that both were competitive for at least some of the problem sizes and processor counts being investigated. The number of distinct parallel spectral transform algorithms selected for each platform is also indicated in Table 8. For example, on the iPSC/860, seven algorithms were selected: 2 parallel FFT algorithms $\times$ 3 parallel LT algorithms plus the reference algorithm.

8.5. Results: Algorithm Comparisons

Tables 9 and 10 list the best algorithm for each computer, problem size, and processor count. Table 9 lists the best "FFT algorithm/LT algorithm" pair, where the FFT and LT algorithms are denoted by the keys listed in the first column of Table 8. If the best algorithm uses a one-dimensional decomposition (i.e., either the FFT or the LT is not parallelized), then no algorithm is listed for the unparallelized transform. Table 10 lists the aspect ratio associated with the best algorithm (i.e., the number of processors allocated to the FFT and LT, respectively). We see considerable variety, with 12 of the 30 algorithm combinations being optimal in some situations, as well as a variety of different aspect ratios. The variation in the aspect ratios stems both from the use of different algorithms in different situations and from limitations on transpose algorithms because of the number of vertical levels.

The tables do not indicate how much difference there is between different algorithms. Figures 2 and 3 provide some of this information. They shows on each machine at T42 and T85 resolution the performance of the reference algorithm and three other "interesting" algorithms: normally those algorithms that proved to be optimal for some processor count on that machine and problem size. (In a few cases, an algorithm that is optimal for just one processor count is omitted, if another algorithm has similar performance.) Performance is given relative to the performance of the best algorithm at each processor count.

Specific comments on the empirical results follow:

1. The reference algorithm is never optimal, and in some cases is ninety per cent worse than the best algorithm.

2. Some form of transpose forms part of the optimal algorithm combination in almost all cases on 16 or more processors. On maximal processor configurations, the algorithm T/T ($\Theta(Q)$ transpose for both FFT and LT) is either optimal or nearly optimal in almost all cases. Notice that this algorithm is identical with the reference algorithm except that its communication parameters have been tuned for the particular machine.

3. The algorithm combination that is optimal in the largest number of configurations is T/R: $\Theta(Q)$ transpose FFT and (overlapped) ring summation LT. This seems a good candidate for a standard algorithm, although its performance degrades for large P, particularly on the Paragon. This situation may change when the message coprocessor on

Table 8: Parallel Algorithms Considered in Algorithm Selection Studies, and Algorithms Selected for Algorithm Comparison Studies on nCUBE/2 (N), iPSC/860 (I), DELTA (D), Paragon-OSF (P), and Paragon-SUNMOS (S). A dash indicates a noncompetitive algorithm that was not considered for further study. The reference algorithm is included in the number of algorithm combinations.

Key	Phase	Algorithm	Variant	N	I	D	P	S
D	FFT	Distributed	no overlap	–	–	Y	Y	Y
O			overlap	Y	Y	–	–	–
T		$\Theta(Q)$ transpose		Y	Y	Y	Y	Y
L		$\Theta(\log Q)$ transpose		–	–	–	Y	Y
N	LT	Ring sum	no overlap	–	–	–	–	–
R			overlap	Y	Y	Y	Y	Y
B		Butterfly sum		Y	Y	Y	Y	Y
T		$\Theta(Q)$ transpose		Y	Y	Y	Y	Y
L		$\Theta(logQ)$ Transpose		–	–	–	Y	Y
Number of algorithm combinations				7	7	7	13	13

Table 9: Best Parallel Algorithms as a Function of Machine/OS, Problem Size, and Processors

Machine	Problem		Processors								
Type	T	L	2	4	8	16	32	64	128	256	512
nCUBE/2	21	8	–/R	–/R	–/R	–/R	T/R	T/R	T/B	O/B	—
nCUBE/2	42	16	—	—	–/R	–/R	T/R	T/R	T/R	T/R	T/R
nCUBE/2	85	32	—	—	—	—	—	O/R	T/R	T/R	T/R
iPSC/860	21	8	–/R	–/B	–/B	T/R	T/R	T/B	T/B	—	—
iPSC/860	42	16	—	–/R	–/R	–/T	T/R	T/R	T/R	—	—
iPSC/860	85	32	—	—	—	—	T/R	T/R	T/R	—	—
DELTA	21	8	–/B	–/B	–/B	T/B	T/B	T/B	T/B	D/B	—
DELTA	42	16	–/R	–/R	–/B	–/T	T/B	T/B	T/B	T/B	T/B
DELTA	85	32	—	—	—	–/R	–/T	T/R	T/B	T/R	T/T
PG-OSF	21	8	–/B	–/B	–/B	T/R	T/R	T/B	L/B	L/L	—
PG-OSF	42	16	—	–/B	–/B	–/T	T/R	T/R	T/T	T/T	L/L
PG-OSF	85	32	—	—	—	—	T/R	T/R	T/R	T/T	T/T
SUNMOS	21	8	–/B	–/B	–/B	L/B	T/B	L/B	L/B	L/L	—
SUNMOS	42	16	–/B	–/B	–/B	–/T	L/B	T/R	T/R	T/B	L/L
SUNMOS	85	32	—	—	—	—	–/T	D/T	L/R	T/R	L/T

Table 10: Best Logical Aspect Ratios as a Function of Machine/OS, Problem Size, and Processors

Problem T	Problem L	2	4	8	16	32	64	128	256	512
nCUBE/2										
21	8	1×2	1×4	1×8	1×16	8×4	8×8	8×16	16×16	—
42	16	—	—	1×8	1×16	8×4	8×8	16×8	16×16	16×32
85	32	—	—	—	—	—	4×16	16×8	32×8	32×16
iPSC/860										
21	8	1×2	1×4	1×8	4×4	8×4	8×8	8×16	—	—
42	16	—	1×4	1×8	1×16	8×4	16×4	16×8	—	—
85	32	—	—	—	—	4×8	16×4	16×8	—	—
DELTA										
21	8	1×2	1×4	1×8	8×2	8×4	4×16	8×16	16×16	—
42	16	1×2	1×4	1×8	1×16	8×4	16×4	16×8	16×16	16×32
85	32	—	—	—	1×16	1×32	32×2	32×4	32×8	8×64
Paragon-OSF										
21	8	1×2	1×4	1×8	8×2	8×4	8×8	8×16	8×32	—
42	16	—	1×4	1×8	1×16	16×2	16×4	8×16	16×16	16×32
85	32	—	—	—	—	4×8	16×4	16×8	16×16	32×16
Paragon-SUNMOS										
21	8	1×2	1×4	1×8	2×8	4×8	8×8	8×16	8×32	—
42	16	1×2	1×4	1×8	1×16	8×4	16×4	16×8	16×16	16×32
85	32	—	—	—	—	1×32	2×32	16×8	32×8	16×32

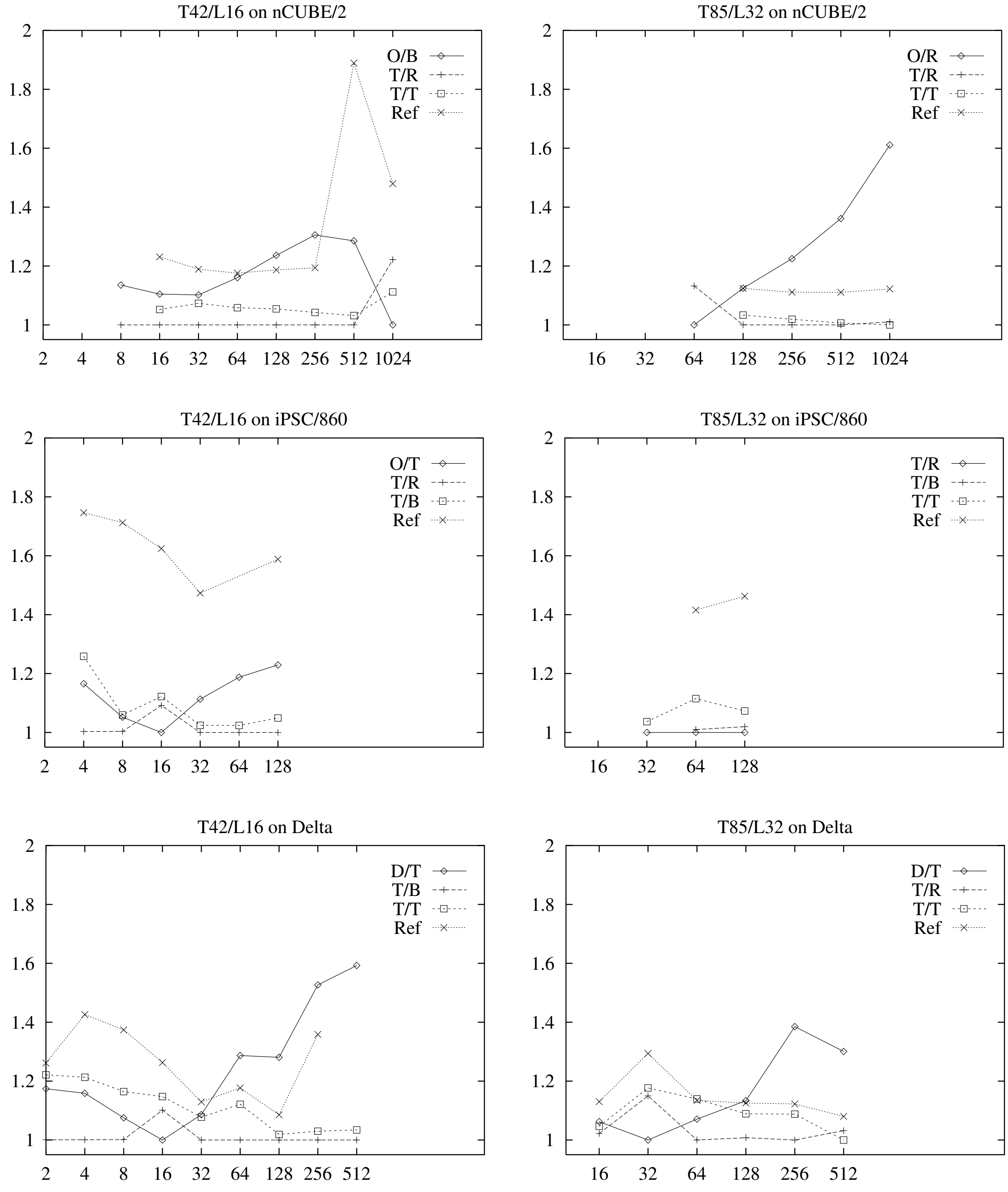


Figure 2: Performance of various parallel algorithms on the nCUBE/2, iPSC/860, and Intel DELTA. In each graph, the X-axis represents processor count and the Y-axis the solution time per time step, expressed as a ratio to the solution time for the best algorithm at that processor count.

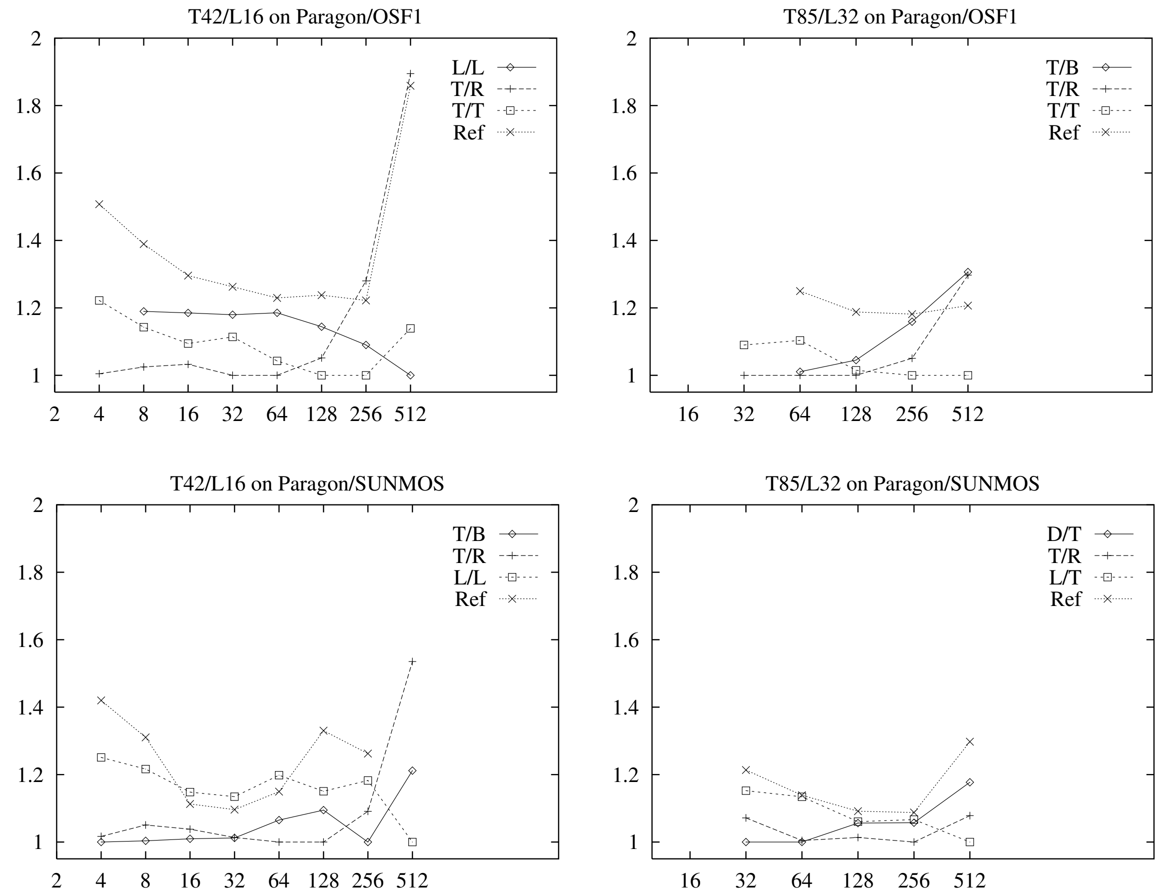
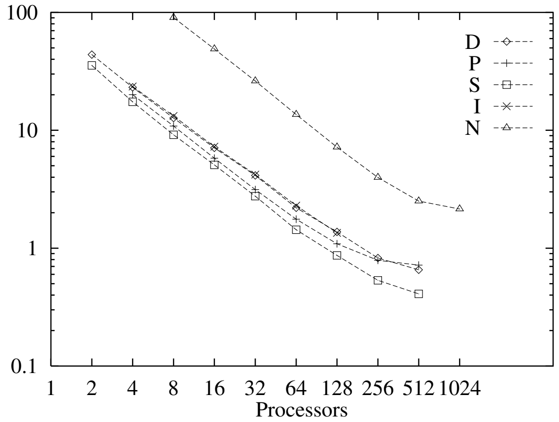


Figure 3: Performance of various parallel algorithms on the Intel Paragon. In each graph, the X-axis represents processor count and the Y-axis the solution time per time step, expressed as a ratio to the solution time for the best algorithm at that processor count.



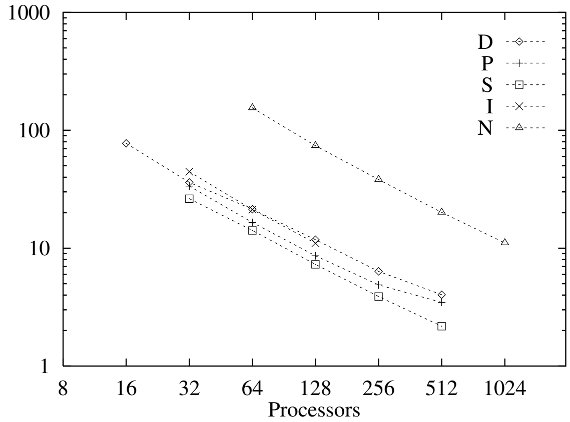


Figure 4: Execution time for 12 time steps (in seconds) as a function of processor count, at T42 resolution (left) and T85 resolution (right). Each time is for the best algorithm at the particular processor count and computer

the Paragon is enabled, decreasing message startup costs and better supporting computation/communication overlap.

4. Because the FFT involves more data than the LT, optimal algorithms on small numbers of processors (16 or less) mostly decompose data structures in a single dimension so as to avoid communication in the FFT, and use either the ring summation or butterfly summation algorithm for the LT. When the FFT is parallelized, transpose algorithms are almost always superior to distributed FFTs. Algorithm combinations such as O/R and D/T are optimal in a few configurations, but are not consistent in their performance.

Finally, Fig. 4 give the execution time for the best algorithm on each computer as a function of P, for problem sizes T42 and T85. We see considerable variation in execution times, with the nCUBE slower than the other machines by an order of magnitude, and the Paragon under SUNMOS fastest in almost all situations. The 512-processor SUNMOS time for T85L32 represents a computational rate of 4.3 GFlops/second.

8.6. Discussion

These results demonstrate the limitations of asymptotic analysis: the asymptotically optimal transpose algorithms are not the most efficient in many situations, particularly for smaller P. The results also demonstrate the importance of tuning algorithms to the communication characteristics of a particular machine. In some cases, tuning makes a greater difference than the choice of algorithm.

A parallel spectral transform code designed for portability should probably incorporate several parallel algorithms. The testbed code PSTSWM indicates that this is feasible. The most useful algorithms seem to be the two transpose algorithms for both FFT and LT, and the overlapped ring sum LT algorithm. A distributed FFT algorithm would also be needed if the number of vertical levels is small. A program designed to execute on a small number of processors (16 or less) can decompose data structures in one dimension only, and use butterfly summation or overlapped ring sum LT algorithms.

9. Caveats and Generalizations

We have attempted to make our empirical studies relevant and comprehensive. But the generality of the study required some simplifying assumptions, and certain algorithms were not examined. In this section, we briefly discuss some of these issues. Algorithm comparisons may need to be repeated if problems of interest differ drastically from our simplifying assumptions. Benchmark codes like PSTSWM make this feasible.

Problem size. For these experiments, problems sizes and processor counts were all powers of two. All of the algorithms work best in under these conditions. Some, like the $\Theta(\log Q)$ transpose and distributed FFT algorithms, do not work at all on a nonpower-of-two number of processors. Other algorithms suffer performance degradation. Nonpower-of-two problem dimensions also cause load imbalances, and the amount of performance degradation is strongly algorithm dependent.

Real weather and climate models often use a number of vertical levels significantly smaller than the other dimensions of the problem. For example, T213L31 is used in some operational weather-forecast models [23]), corresponding to a $640 \times 320 \times 31$ physical grid. The transpose FFT algorithms suffer because they must use a larger number of processors for the LT than an algorithm that uses a distributed FFT.

Decomposing "field" dimension. As mentioned in §5.2, one technique for applying the transpose algorithms when there are few vertical levels is to partition the state variables among the processors also. (This issue did not arise in our experiments, because our example problems had sufficient vertical levels relative to other problem dimensions.) This technique can be used in two ways in a transpose FFT/distributed LT algorithm:

1a. Starting with the usual $(\mathcal{I}_X, \mathcal{J}_Y, \mathcal{K})$ distribution of the physical grid, we transpose within processor rows over both K levels and 8 fields. We compute the FFTs, then transpose back (again within rows) to a $(\mathcal{M}_X, \mathcal{J}_Y, \mathcal{K})$ decomposition of the Fourier grid. We then proceed with the LT. A similar approach is used for the inverse transform, although only 5 rather than 8 fields are available. This approach performs twice as many transposes as the transpose FFT/distributed LT algorithm, but can use 5 times more processors without load imbalance. It has been used successfully in the message-passing version of CCM2 [7].

1b. We can avoid the double transpose at the cost of redundant work and some other additional communication by duplicating one field and decomposing over K levels and 3 sets of 3 fields. After the FFT, we then have separate distributed LT calculations for δ, ϕ, and ζ for the forward transform. For the inverse transform, we have $5K$ LT calculations to distribute over (fields δ, ϕ, ζ, U, V), and the U and V calculations require the updated δ and ϕ fields. The simplest approach, assigning U and V to the δ and ϕ "columns," requires duplication of δ and ϕ between processor columns and significant load imbalance and redundant work. A better load-balancing strategy would require something equivalent to an additional transpose and would still not eliminate all redundant work. Note that load imbalance in the inverse LT due to assignment of fields also implies load imbalance in the inverse FFT. We gain (at best) the ability to use 3 times more processors with this approach.

There is a single approach to distributing fields during the transpose FFT in a transpose FFT/transpose LT algorithm:

2. Again duplicate one field and decompose over $3K$ sets of fields, transposing across processor rows. After the FFT, transpose within processor columns, with each processor column computing forward LTs for either δ, ϕ, or ζ. For the inverse transform, we again need to assign the U and V calculations, duplicating the updated δ and ϕ fields. Similar communication and computation costs arise.

The double transpose FFT cannot be applied here because the transpose LT requires that the field and vertical dimensions remain decomposed if all processors are to be utilized.

We feel that the most promising of these approaches is (1a). It is simple, is relatively independent of problem size and number of processors, and incurs no additional computation cost.

Serial FFT algorithm. FFT routines that can handle vector lengths with factors of 2, 3, 4, and 5 allow a larger set of problem sizes to be treated. In addition, exploiting a factor of 4 is approximately 30 per cent faster than two factors of 2. These routines can be exploited directly in transpose FFTs. The distributed FFT can also be generalized, but some efficiency is lost.

Generalized $\Theta(\log Q)$ algorithms. A range of hybrid algorithms combining aspects of the $\Theta(\log Q)$ and $\Theta(Q)$ transpose algorithms can be defined that trade off message counts and communication volume in different ways. For example, the $\Theta(\log Q)$ algorithm can be modified to use $\log_4 Q$ stages by communicating with 3 other processors at each stage, assuming that the problem size and number of processors support this. Another approach is to use a switch, taking a few steps of the $\Theta(\log Q)$ algorithm, then switching to the $\Theta(Q)$ algorithm, analogously to the butterfly sum algorithm.

The distributed FFT can also be modified to use $\log_4 Q$ stages and can then exploit factors of 4 to reduce computation costs. And it is possible to apply a transpose-like algorithm within the FFT itself [9].

These hybrid algorithms can improve performance somewhat in regimes where message startup costs and data volume costs are comparable. However, they place additional requirements on problem size and processor counts.

Mesh-based algorithms. We have restricted ourselves to algorithms designed for one-dimensional processor meshes. In cases where nonsquare logical meshes were mapped to approximately square physical grids, it would be possible in principle to utilize specialized algorithms that exploit the extra connectivity [2, 32]. Because our experiments show that "optimal" processor grids are mostly close to square, we believe that these algorithms would not change our results. This issue will be addressed in further research.

Physics computations Realistic spectral models such as CCM2 include substantial grid-space computation used to simulate processes such as clouds, radiation, moist convection, the planetary boundary layer, and surface processes that are not not directly concerned with fluid dynamics. These processes typically have no horizontal data dependencies and, hence, can be parallelized easily. However, they can be subject to significant spatial and temporal variations in the computational load per grid point. These load variations need to be accounted for when developing a parallel model, and can influence our choice of spectral transform algorithm. As our experience suggests that load variation is very model dependent [26, 15], we have not considered these effects in this paper. However, note that transpose FFT algorithms can combine the redistribution required for load balancing with the transpose. A separate load balancing redistribution of work can also be performed before the grid-space computations, and a corresponding mapping back to a standard decomposition performed afterwards. This allows both spatial and temporal load imbalances to be addressed without decreasing the efficiency of the (optimized, one-dimensional, static) transpose or distributed FFT communication patterns. This load balancing step is likely to be cost-effective even in conjunction with transpose FFT algorithms if only local or a small number of global communications are required for the load balancing, and is more amenable to modification and experimentation since decoupled from the spectral transform phases.

Future work. In order to perform empirical investigation of some of the issues discussed in this section, we plan to incorporate into PSTSWM both distributed and transpose versions of the 2-3-4-5 parallel FFT, the double transpose FFT, and the hybrid $\Theta(Q) - \Theta(\log Q))$ transpose algorithms. These add additional capabilites for problem and machines sizes that we have not yet examined, but should not change our preliminary conclusions. Given the success of the overlap ring sum algorithm, we will also implement the overlap butterfly sum algorithm. This may increase the range of optimality of the transpose FFT/distributed LT algorithms on some machines.

10. Conclusions

We have conducted a detailed analysis and empirical investigation of parallel algorithms for the spectral transform method. This study has allowed us to identify optimal algorithms for various problem size and machine parameter regimes. This information should be directly useful to developers of parallel spectral-transform-based climate and weather models.

Most of the observed performance trends can be explained using our analytic performance models; this gives us confidence both that these models are correct and that the parallel algorithm implementations incorporated in our testbed code are efficient. It also provides a basis for extrapolating the results obtained here to other regimes. However, the models as described here are not sufficiently detailed to provide detailed performance predictions. In future work, we will investigate to what extent the empirical studies can be used to generate performance models that can be used for prediction.

This exhaustive study of alternative algorithms, communication techniques, and aspect ratios suggests some conclusions regarding parallel libraries. It is common practice in parallel computing to select parallel algorithms on the basis of asymptotic analysis, and then to incorporate these algorithms in portable libraries that are used unchanged on different computers. The results of this study emphasize three limitations of this approach. First, asymptotically suboptimal algorithms may be superior in many interesting regimes. Second, reference implementations of parallel algorithms designed for portability can be considerably less efficient than implementations tuned for a particular machine. Third, interactions between algorithms can impact performance; hence, for peak performance, it can be important to optimize algorithm combinations rather than individual algorithms.

Our work suggests three techniques that can be used to overcome these limitations. First, detailed analytic models that take into account constant factors and issues such as load imbalance can be used to develop improved understandings of algorithmic tradeoffs. Second, libraries can be defined to incorporate multiple algorithmic options selectable at runtime. This allows codes to be tuned for different problem or machine characteristics, either by the programmer or automatically on the basis of runtime performance data. Third, testbed codes such as PSTSWM can be used to explore algorithmic alternatives.

Acknowledgments

We are grateful to members of the CHAMMP Interagency Organization for Numerical Simulation, a collaboration involving Argonne National Laboratory, the National Center for Atmospheric Research, and Oak Ridge National Laboratory, for sharing codes and results; to Brian Toonen for his tremendous help with the computational experiments; and to the SUNMOS development team for help in getting PSTSWM running under SUNMOS on the Intel Paragon.

This research was performed using an Intel iPSC/860 multiprocessor system at Oak Ridge National Laboratory, the nCUBE/2 and Intel Paragon systems at Sandia National Laboratories, the nCUBE/2 system at Ames National Laboratory, the Intel Touchstone DELTA System operated by Caltech on behalf of the Concurrent Supercomputing Consortium, and the Intel Paragon XP/S 5 and XP/S 35 systems located in the Oak Ridge National Laboratory Center for Computational Sciences, funded by the Department of Energy's Mathematical, Information, and Computational Sciences Division of the Office of Computational and Technology Research.

11. References

[1] D. H. Bailey, *FFTs in external or hierarchical memory*, J. Supercomputing, 4 (1990), pp. 23–45.

[2] M. Barnett, R. van de Geijn, S. Gupta, D. G. Payne, L. Shuler, and J. Watts, *Interprocessor collective communication library (InterCom)*, in Proc. Scalable High Performance Computing Conf., IEEE Computer Society Press, Los Alamitos, CA, 1994, (in press).

[3] S. Barros and K. T, *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.

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

[5] G. L. Browning, J. J. Hack, and P. N. Swarztrauber, *A comparison of three numerical methods for solving differential equations on the sphere*, Mon. Wea. Rev., 117 (1989), pp. 1058–1075.

[6] 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.

[7] 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: 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. 500–513.

[8] J. B. Drake, I. T. Foster, J. J. Hack, J. G. Michalakes, B. D. Semeraro, B. Toonen, D. L. Williamson, and P. H. Worley, *PCCM2: A GCM adapted for scalable parallel computer*, in Fifth Symposium on Global Change Studies, American Meteorological Society, Boston, 1994, pp. 91–98.

[9] A. Dubey, M. Zubair, and C. E. Grosch, *A general purpose subroutine for fast Fourier transform on a distributed memory parallel machine*, Parallel Computing, (to appear).

[10] ——, *Performance of the Intel iPSC/860 and the Ncube 6400 hypercubes*, Parallel Computing, 17 (1991), pp. 1285–1302.

[11] T. H. Dunigan, *Communication performance of the Intel Touchstone DELTA Mesh*, Tech. Report ORNL/TM-11983, Oak Ridge National Laboratory, Oak Ridge, TN, December 1991.

[12] A. Edelman, *Optimal matrix transposition and bit reversal on hypercubes: all-to-all personalized communication*, J. Par. Dist. Comp., 11 (1991), pp. 328–331.

[13] J. O. Eklundh, *A fast computer method for matrix transposing*, IEEE Trans. Comput., C-21 (1972), pp. 801–803.

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

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

[16] 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.

[17] 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.

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

[19] A. Gupta and V. Kumar, *The scalability of FFT on parallel computers*, IEEE Trans. Par. Dist. Sys., 4 (1993), pp. 922–932.

[20] 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 Tech. Note NCAR/TN–382+STR, National Center for Atmospheric Research, Boulder, Colo., 1992.

[21] J. J. Hack and R. Jakob, *Description of a global shallow water model based on the spectral transform method*, NCAR Tech Note NCAR/TN-343+STR, National Center for Atmospheric Research, Boulder, CO, February 1992.

[22] S. L. Johnsson and C.-T. Ho, *Algorithms for matrix transposition on Boolean N-cube configured ensemble architectures*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 419–454.

[23] T. Kauranne and S. Barros, *Scalability estimates of parallel spectral atmospheric 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. 312–328.

[24] R. D. Loft and R. K. Sato, *Implementation of the NCAR CCM2 on the Connection Machine*, 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. 371–393.

[25] B. Machenhauer, *The spectral method*, in Numerical Methods Used in Atmospheric Models, vol. II of GARP Pub. Ser. No. 17. JOC, World Meteorological Organization, Geneva, Switzerland, 1979, ch. 3, pp. 121–275.

[26] J. Michalakes, *Analysis of workload and load balancing issues in the NCAR Community Climate Model*, Tech. report ANL/MCS-TM-144, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1991.

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

[28] 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.

[29] H. S. Stone, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-21 (1971), pp. 153–161.

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

[31] 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.

[32] S. S. Takkella and S. R. Seidel, *Complete exchange and broadcast algorithms for meshes*, in Proc. Scalable High Performance Computing Conf., IEEE Computer Society Press, Los Alamitos, CA, 1994, (in press).

[33] R. A. van de Geijn, *Efficient global combine operations*, in The Sixth Distributed Memory Computing Conference Proceedings, Q. F. Stout and M. Wolfe, eds., IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 291–294.

[34] 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.

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

[36] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber, *A standard test set for numerical approximations to the shallow water equations on the sphere*, J. Computational Physics, 102 (1992), pp. 211–224.

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

[38] P. H. Worley and I. T. Foster, *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, CA, 1994, (in press).