The Parallel Scalability of the Spectral Transform Method

Ian Foster, William Gropp, Rick Stevens
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439–4801, USA

Abstract

This paper investigates the suitability of the spectral transform method for parallel implementation. The spectral transform method is a natural candidate for general circulation models designed to run on large-scale parallel computers due to the large number of existing serial and moderately parallel implementations. We present analytic and empirical studies that allow us to quantify the parallel performance, and hence the scalability, of the spectral transform method on different parallel computer architectures. We consider both the shallow-water equations and complete GCMs. Our results indicate that for the shallow-water equations parallel efficiency is generally poor because of high communication requirements. We predict that for complete global climate models, the parallel efficiency will be significantly better; nevertheless, projected Teraflop computers will have difficulty achieving acceptable throughput necessary for long-term regional climate studies.

1 Introduction

Current general circulation models (GCMs) are rarely run at resolutions finer than 100 km when used for weather modeling (Simmons 1990) and 400 km when used for climate modeling (Williamson 1990). Yet simulation of weather and climate on a regional scale requires resolution measured in the small tens of kilometers (DOE 1990). This will require substantially greater computational resources than are available today on even the fastest vector computers. For example, the European Center for Medium-range Weather Forecast's (ECMWF) model requires one hour per simulated day when run at 100-km resolution on a CRAY-2 computer (Gates et al. 1990). At this rate, a 100-year simulation would take four years. Computational requirements also increase rapidly when resolution is increased.

Recognizing the limitations of conventional supercomputers, the U.S. Department of Energy has sponsored a multiyear effort called CHAMMP (Computer Hardware, Advanced Mathematics, and Model Physics) to investigate the feasibility of developing GCMs capable of exploiting the large-scale parallel machines that will become available in the mid 1990s. These machines will have several thousand processors, each as powerful as a CRAY-1.

One aspect of the CHAMMP program is the investigation of numerical methods for use in parallel models. Candidate methods must not only have good numeric properties but must also be scalable: that is, they must be able to exploit large numbers of processors with reasonable efficiency. Despite well-known deficiencies (Williamson and Rasch 1989), the spectral transform method is the most widely used in current models because of its computational efficiency (Bourke 1972, Browning 1989). Hence, it is important to understand the suitability of this method for parallel implementation. In this paper, we report analytic and empirical studies designed to quantify the scalability of the spectral transform method.

Parallel computers are often touted as the solution to problems with vast computational requirements. However, it is by no means obvious that a particular algorithm will perform well on a parallel computer. As more and more processors are added, the cost of coordinating the computation and sharing data among the processors increases. It is the rate at which these costs increase that determines an algorithm's scalability.

The costs of coordinating computation and sharing data depend critically on the way in which the algorithm is decomposed, that is, how its data and computation are distributed among the processors of a parallel computer. In this paper, we consider two alternative decompositions of the spectral transform method and mention others that have proved less promising (Foster and Gropp 1991). The rejected decompositions offer substantially inferior performance for practical problem sizes. Hence, the results of this paper tell us not only how efficient and scalable the spectral transform method is, but how it should be implemented.

The performance of parallel programs is typically studied in one of two ways. Asymptotic analysis considers the behavior of an algorithm on very large numbers of processors; empirical studies seek to extrapolate from results obtained on a small number of processors to predict large-scale parallel performance. Unfortunately, it is rare that either approach provides accurate predictions of program performance. As simple asymptotic analysis drops lower-order cost terms, it is often inaccurate on realistic numbers of processors. Extrapolation from empirical results without guidance from analytic studies is always dangerous.

We combine aspects of both asymptotic and empirical analysis to obtain results that we believe to be better than could be attained using either method alone. We first analyze the communication and computation costs of a parallel algorithm, but only drop lower-order terms that we can demonstrate are always negligible in the problem sizes of interest. We then calibrate our models with performance data obtained from empirical studies.

In this paper, we consider only horizontal decompositions; that is, we assume that the vertical structure of the model is left unchanged. This assumption permits us to ignore the model physics component of GCMs, which in existing models is generally column oriented, and focus on the dynamics component. To simplify empirical studies of scalability, we initially restrict ourselves to the shallow-water equations. We discuss how our results can be extended to deal with GCMs in Section 9.

A parallel spectral model is in regular use on a multi-processor CRAY at the European Center for Medium-range Weather Forecasts (ECMWF) (Simmons 1990). However, a completely different set of issues must be addressed when designing algorithms capable of exploiting thousands of processors. Kauranne has previously provided a quick survey of some current thinking about parallelism in weather models (Kauranne 1990b) and a high-level discussion of achievable parallelism (Kauranne 1990a). The present paper is the first to attempt a detailed analysis of the scalability of the spectral transform method.

This paper is divided into three parts. Tbe first part, comprising Sections 2–4, provides a quick review of parallel computers, the shallow water equations, and the spectral transform method. This material serves to describe our notation and, in addition, includes a simple example of the approach to performance analysis that is the heart of this paper. The second part of the paper, comprising Sections 5 and 6, looks at the time and space complexity of the spectral transform algorithm applied to the shallow water equations on uniprocessors and distributed memory parallel computers. The third part of the paper, comprising Sections 7–10, is concerned with estimating the performance of parallel implementations of the spectral transform method for global climate models. Section 7 compares our complexity model with computations on an Intel iPSC/860 parallel computer; we use this comparison both to validate our model (by checking the shape of the curves) and to determine values for those coefficients that depend on algorithm implementation details. Section 8 discusses the parallel performance of the spectral transform method applied to the shallow water equations on several parallel computers. Section 9 predicts the performance of the spectral transform method applied to general circulation models. Section 10 presents some conclusions and directions for further work.

2 The Machine Model

In this paper, we focus on a single class of parallel computer: distributed-memory MIMD (multiple instruction, multiple data) computers, or *multicomputers*. The distinguishing feature of this kind of parallel computer is that individual processors work independently of each other and exchange data via an interconnection network. Computers in this class include Intel's iPSC and Ncube Corporation's NCUBE.

We do not consider shared-memory multiprocessors, such as those produced by Cray and Sequent, in

our analysis. In these computers, processors exchange data via a common memory. This organization provides rapid access to data but is unlikely to scale economically to large numbers of processors.

2.1 Basic Notation

We assume that a parallel computer consists of p processors, or *nodes*, each running at the same speed and able to exchange data by means of messages sent via an interconnection network. In the execution of a parallel algorithm, each processor will spend some time doing essential computation and some time in overhead: communication, idling, or work that is not performed by a good sequential algorithm. We define the *sequential time*, T_e, to be the time taken by a good sequential algorithm and the *overhead*, T_o, to be the sum of the overheads incurred at each processor. We also define speedup (or utilization), S, and efficiency, E:

$$S = \frac{pT_e}{T_e + T_o}, \tag{1}$$

$$E = \frac{S}{p} = \frac{1}{1 + \frac{T_o}{T_e}}. \tag{2}$$

Both T_e and T_o are defined to be times taken in a single time step.

2.2 Machine Parameters

The performance of a multicomputer depends on more than just the floating point speed of its component processors. As processors must coordinate their activities and exchange data, the cost of sending messages must also be considered. There are two principal components of this cost: the cost of transferring a message to the interconnection network, and the cost incurred while in transit which is related to the distance the message must travel in the network. The cost to move a message to the interconnection network reflects details of the software and hardware implementation. The cost while in the network is determined in part by how the processors are connected to one another.

Two popular interconnects are a 2-D mesh, where each nonboundary processor is connected to its four immediate neighbors, and a hypercube, where each processor is connected to $\log p$ other processors. (All logs used in this paper have base 2.) The key parameter here is the maximum distance from any processor to any other processor. This is called the *diameter* of the network. The diameter of a nonperiodic mesh is $2\sqrt{p}$ and of a hypercube $\log p$. An important feature of a hypercube is that it contains a mesh (it may be considered a mesh with additional, long-distance connections). In this paper, we concentrate on the mesh network; at points where the additional connectivity of a hypercube can be exploited, we shall comment on how that changes our results.

The cost of actually transmitting a message between any two processors can be represented with three major parameters: the message startup time, which represents the time required to format the message and transfer it to the communication hardware (t_s); the per-hop time, which represents the time required to move from one processor to a neighbor (t_h); and the transfer time per word, which represents the physical bandwidth of the channel (t_w). Multicomputers commonly employ *cut-through routing*; this permits a message to be pipelined when its destination is more than one "hop" away. The time required to send a message of size s words to a processor h hops distant is then

$$t_s + t_h h + t_w s.$$

This expression is approximate, as on some computers the full bandwidth is realized only for larger s. In addition, it is important to realize that channel bandwidth must generally be shared by all processors sending on a channel. For example, if k processors must send messages through the same channel, then the effective bandwidth available to each processor is $1/k$ of the true bandwidth. This effect, which can be modeled by increasing t_w by a factor of k, proves to be important in pipelined communication algorithms discussed below.

Table 1: Target Machine Characteristics (times in μsec)

Machine	Interconnect	Processors	C	t_s	t_h	t_w
GAMMA	hypercube	128	1.0	136	2.0	1.6
DELTA	2-D mesh	528	0.8	50	0.05	0.32
SIGMA	2-D mesh	2,048	0.5	10	0.03	0.04
NCUBE-2	hypercube	8,096	12	60	2.0	1.6
FastMesh	2-D mesh	2,048	0.1	5	0.01	0.04
BigCube	hypercube	65,536	2.0	20	0.6	0.4

All three terms need to be maintained in the cost expression. As t_s is generally much larger than t_w, the t_s term can dominate in applications that send mostly small messages. The t_h term can also be significant on large computers if an application performs many nonlocal communications.

2.3 Target Machines

We can sketch a broad class of multicomputers that are likely to be available in a two- to five-year timeframe. Intel Corporation's Supercomputer Systems Division has installed examples of its Touchstone GAMMA and DELTA computers, containing 128 and 528 Intel i860 microprocessors respectively, and has announced a follow-on SIGMA computer with 2,048 similar microprocessors. Each i860 processor has a peak performance of 60 Mflops. Ncube Corporation manufactures an NCUBE-2 machine with up to 8,192 processors. The DELTA will have 16 MB of memory per processor, while the NCUBE-2 typically has 4 MB. An important difference between these machines is that the DELTA and SIGMA are mesh connected whereas the GAMMA and NCUBE-2 use a hypercube interconnect.

We summarize in Table 1 the machine parameters (t_s, t_h, t_w, each in μsec) used in our complexity estimates. We also specify the type of interconnect, the maximum machine size, and a computation scaling factor C. The scaling factor is applied to the sequential time (T_e) in a performance model; it reflects the performance differential between a machine's base processor and compiler and the i860 and preliminary compiler used in the calibration studies reported in Section 7. It is the relatively large size of these parameters, compared to the time for a floating-point operation, that makes a careful analysis of the communication times necessary. Note also that $50 \leq t_s/t_w \leq 250$ in Table 1 (with an even greater range for t_s/t_h); the size of this range is the reason that our model or communication needs all three of the terms t_s, t_h, and t_w.

The machine parameters for the GAMMA and DELTA were obtained by experiment. The parameters for the SIGMA are estimates provided by Intel. The scaling factor is intended to account for a 25% increase in i860 performance resulting from better compilation. The scaling factor of 0.5 for the SIGMA reflects the fact that this machine will be built with a future generation of Intel components that we have estimated to be 100% faster than the current i860. This estimate is based on likely improvements in both processor speed and compiler technology. The machine parameters for the NCUBE-2 are based on figures provided by Ncube; the scaling factor reflects the fact that the NCUBE-2 processor has a peak speed that we estimate is one twelfth that of the i860.

It is difficult to predict characteristics of the "next generation" of parallel machines, the "Teraflop" machines that will be available in a five- to ten-year timeframe. However, to permit preliminary evaluation of the suitability of the spectral transform method for these machines, we present parameters for two possible configurations: a mesh-based machine, with a relatively small number of very fast processors, and a hypercube, with a large number of slower processors. We consider these machines, which we name *FastMesh* and *BigCube*, in Section 9.

Note that our models do not attempt to represent the effect of memory speeds on sequential and parallel

performance. This can be expected to become an increasingly important factor in future computers, as processor speeds are increasing faster than memory speeds.

2.4 Example Analysis: Ring Pipeline

We use a simple example to illustrate the use of scalability analysis techniques. Assume that a vector of size n has been distributed evenly among p processors. A parallel algorithm requires that the processors are connected in a virtual ring and that all processors engage in $p-1$ computation/communication steps. In each step, each processor performs some computation using its local data, sends a copy of its local data to its neighbor in the ring, and receives a message of the same type from its other neighbor. How do we expect the performance of this *ring pipeline* algorithm to change as the number of processors increases?

Assume that the total amount of essential computation performed in this algorithm is Cn, where C is a coefficient that can be determined by empirical studies and n is the vector size. No redundant computation is performed by the parallel algorithm. Hence, the only overhead is that due to communication. At each step, each processor sends a message of size n/p to its neighbor in the ring. If p is even, then a ring can be embedded in a mesh in such a way that each processor's neighbors are exactly one hop distant. Hence, the cost to send this message is

$$t_s + t_h + t_w \frac{n}{p},$$

and the efficiency of the algorithm is (cf. Eq. 2)

$$E = \left(1 + \frac{p(p-1)(t_s + t_h + t_w \frac{n}{p})}{Cn}\right)^{-1}. \tag{3}$$

A great deal can be learned about the behavior of the algorithm by a simple qualitative analysis of this expression. For example, we see that communication costs are proportional to both p^2 and np, while computation is proportional only to n. Hence, we may expect this algorithm to scale badly: efficiency drops rapidly as p is increased.

Such qualitative analysis is useful but does not allow us to make predictions of actual performance. To make such predictions, we must first calibrate Eq. 3 by finding values for the computation coefficient C and the machine parameters t_s, t_h, and t_w on a parallel computer of interest. C can be determined by timing an implementation of the algorithm on a single processor; the machine parameters are determined by experimentally measuring the message-passing performance.

3 The Shallow-Water Equations

For completeness, we provide the shallow-water equations in the form that we solve using the spectral transform method (to be described in Section 4). Let

$$\begin{aligned} \mathbf{V} &= \text{velocity on sphere} \\ \mathbf{i}, \mathbf{j}, \mathbf{k} &= \text{unit vectors} \\ \Phi &= \text{geopotential} \\ f &= \text{coriolis term} \\ a &= \text{radius of sphere} \end{aligned}$$

where $\mathbf{V} = \mathbf{i}u + \mathbf{j}v$. Then the equations are (Washington and Parkinson 1986)

$$\begin{aligned} \frac{d\mathbf{V}}{dt} &= -f\mathbf{k} \times \mathbf{V} - \nabla\Phi \\ \frac{d\Phi}{dt} &= -\Phi\nabla \cdot \mathbf{V}, \end{aligned} \tag{4}$$

where

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \mathbf{V} \cdot \nabla,$$

and ∇ in spherical coordinates is

$$\nabla = \frac{\mathbf{i}}{a(1-\mu^2)} \frac{\partial}{\partial \lambda} + \frac{\mathbf{j}}{a} \frac{\partial}{\partial \mu},$$

with λ the longitude and $\mu = \sin\phi$, where ϕ is the latitude.

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

$$\begin{array}{lrcl} \text{vorticity}: & \eta & = & f + \mathbf{k} \cdot (\nabla \times \mathbf{V}) \\ \text{divergence}: & \delta & = & \nabla \cdot \mathbf{V} \end{array}$$

and

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

so that U and V are continuous scalars at the poles. Then, after some manipulation, the equations can be written in the form

$$\begin{aligned} \frac{\partial \eta}{\partial t} &= -\frac{1}{a(1-\mu^2)} \frac{\partial}{\partial \lambda}(U\eta) - \frac{1}{a} \frac{\partial}{\partial \mu}(V\eta) \\ \frac{\partial \delta}{\partial t} &= +\frac{1}{a(1-\mu^2)} \frac{\partial}{\partial \lambda}(V\eta) - \frac{1}{a} \frac{\partial}{\partial \mu}(U\eta) - \nabla^2 \left(\Phi + \frac{U^2+V^2}{2(1-\mu^2)} \right) \\ \frac{\partial \Phi}{\partial t} &= -\frac{1}{a(1-\mu^2)} \frac{\partial}{\partial \lambda}(U\Phi) - \frac{1}{a} \frac{\partial}{\partial \mu}(V\Phi) - \bar{\Phi}\delta. \end{aligned} \tag{5}$$

Finally, U and V are represented in terms of η and δ through two auxiliary equations expressed in terms of the streamfunction ψ and velocity potential χ:

$$\begin{aligned} \eta &= \nabla^2 \psi + f \\ \delta &= \nabla^2 \chi \end{aligned} \tag{6}$$

and

$$\begin{aligned} U &= \frac{1}{a} \frac{\partial \chi}{\partial \lambda} - \frac{1-\mu^2}{a} \frac{\partial \psi}{\partial \mu} \\ V &= \frac{1}{a} \frac{\partial \psi}{\partial \lambda} + \frac{1-\mu^2}{a} \frac{\partial \chi}{\partial \mu}. \end{aligned} \tag{7}$$

Equation 5, like Eq. 4, is in three unknowns. The velocity terms have been replaced with a vorticity and a divergence term; the velocities are found by solving Eq. 6 and then evaluating Eq. 7.

4 Spectral Transform Method

The spectral transform method used to solve Eq. 5 maintains prognostic variables U, V, and Z in a computationally uniform physical grid with coordinates (λ_i, μ_j), where $1 \le i \le I$ and $1 \le j \le J$. In contrast, the scalar quantities η, ϕ, and δ are represented as sets of spectral coefficients.

An arbitrary scalar field is approximated by a truncated series of its spectral coefficients a_n^m as follows:

$$a(\lambda, \mu) = \sum_{m=-M}^{m=M} \sum_{n=|m|}^{N(m)} a_n^m P_n^m(\mu) e^{im\lambda}, \tag{8}$$

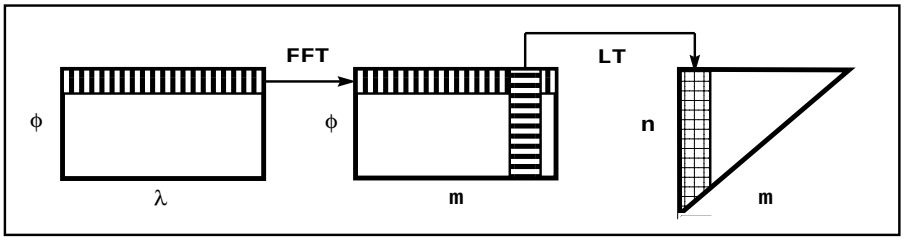

Figure 1: Spectral Transform Method Data Dependencies

where P_n^m are the associated Legendre functions. $N(m)$ specifies the form of the truncation of coefficients; this is discussed below.

Computation is performed in both the physical and spectral data spaces: Physics and nonlinear terms are evaluated in physical space; time stepping is performed in spectral space. At each time step, data is transferred between the two spaces by means of forward and inverse spectral transforms.

As Eq. 8 suggests, the spectral transform can be implemented by a Fourier transform followed by a Legendre transform. The operation of the two transforms is illustrated in Figure 1. The Fourier transform, which can be implemented with the Fast Fourier Transform (FFT), operates on each grid space latitude independently to produce a set of intermediate quantities. The Legendre transform then operates on each column of the intermediate array independently to produce the spectral coefficients. (The inverse spectral transform operates in the reverse sequence.) In Figure 1, the areas shaded with vertical and horizontal bars represent the input and output of a single FFT and Legendre transform, respectively.

Note that Eq. 5 contains linear and quadratic terms. To prevent aliasing of the quadratic terms in the numerical approximation, the number of points in both directions is chosen to be larger than the degree of the expansion. For example, the number of points in longitude $I \geq 3M+1$, where M is the highest Fourier wave number. Thus, we use a standard discrete Fourier transform but truncate its output to $2M+1$ points. The number of terms in the Legendre expansions is similarly truncated.

We provide in Figure 2 a summary of the spectral transform method used to solve Eq. 5. Note the four steps in the algorithm; these will be referred to in subsequent sections. The FFT and its inverse (IFFT) are represented in the figure. Here, $\eta(,\mu_j)$ is a vector (over i) for each μ_j, and $A^m(\mu_j)$ is also a vector (over m) for each μ_j. Each of the FFTs and IFFTs is of length I. The sums that the IFFTs are applied to are evaluated for each $-M \leq m \leq M$; these give the $2M+1$ Fourier coefficients.

The inverse Legendre transform requires the computation of an integral; this is done by using Gaussian quadrature. Thus, the latitude points μ_j are picked as Gaussian grid points (the longitude points λ_i are ordinarily picked as uniformly spaced to simplify the Fourier transforms). The Gaussian quadrature has weights w_j; these appear in Figure 2. (The term H_n^m, which also appears in Figure 2, is the derivative of P_n^m.)

It is useful to define some intermediate quantities. These terms appear in several places in Eq. 5, and their computation as a separate step is an important optimization.

$$\begin{aligned} A &= U\eta \\ B &= V\eta \\ C &= U\Phi \\ D &= V\Phi \\ E &= \frac{U^2+V^2}{2(1-\mu^2)}. \end{aligned}$$

At each time step:

1. Inverse transform spectral quantities to physical space:

$$Z(,\mu_j) = \text{IFFT} * \sum_{n=|m|}^{N(m)} \eta_n^m P_n^m(\mu_j)$$

2. Determine physical quantities U and V:

$$U(,\mu_j) = -\text{IFFT} * \sum_{n=|m|}^{N(m)} \frac{a}{n(n+1)}(im\delta_n^m P_n^m(\mu_j) - \eta_n^m H_n^m(\mu_j))$$

$$V(,\mu_j) = -\text{IFFT} * \sum_{n=|m|}^{N(m)} \frac{a}{n(n+1)}(im\delta_n^m P_n^m(\mu_j) + \eta_n^m H_n^m(\mu_j))$$

3. Evaluate nonlinear products in physical space and transform to spectral space

$$A^m(\mu_j) = \text{FFT} * Z(\lambda_i,\mu_j)U(\lambda_i,\mu_j)$$

$$B^m(\mu_j) = \text{FFT} * Z(\lambda_i,\mu_j)V(\lambda_i,\mu_j)$$

$$C^m(\mu_j) = \text{FFT} * \phi(\lambda_i,\mu_j)U(\lambda_i,\mu_j)$$

$$D^m(\mu_j) = \text{FFT} * \phi(\lambda_i,\mu_j)V(\lambda_i,\mu_j)$$

$$E^m(\mu_j) = \text{FFT} * \frac{U^2(\lambda_i,\mu_j) + V^2(\lambda_i,\mu_j)}{2(1-\mu_j^2)}$$

4. Time-step spectral quantities:

$$\frac{\partial \eta_n^m}{\partial t} = -\sum_{j=1}^{J}(imA^m(\mu_j)P_n^m(\mu_j) - B^m(\mu_j)H_n^m(\mu_j))\frac{w_j}{a(1-\mu_j^2)}$$

$$\frac{\partial \delta_n^m}{\partial t} = -\sum_{j=1}^{J}(imB^m(\mu_j)P_n^m(\mu_j) + A^m(\mu_j)H_n^m(\mu_j))\frac{w_j}{a(1-\mu_j^2)} + \frac{n(n+1)}{a^2}(E^m(\mu_j) + \Phi^m(\mu_j))P_n^M(\mu_j)w_j$$

$$\frac{\partial \Phi_n^m}{\partial t} = -\sum_{j=1}^{J}(imC^m(\mu_j)P_n^m(\mu_j) - D^m(\mu_j)H_n^m(\mu_j))\frac{w_j}{a(1-\mu_j^2)} - \bar{\Phi}\delta_n^m.$$

Figure 2: Spectral Transform Method

4.1 Truncation

Equation 8 represents a finite truncation of the infinite spectral transform for a scalar field. A *triangular truncation* is frequently adopted; in this case

$$N(m) = M.$$

Other truncations are also employed (e.g., rhomboidal). However, the choice of truncation does not have a significant impact on our analysis of computational requirements. Hence, without significant loss of generality we consider only triangular truncations here. This allows us to fix I and J in terms of the truncation number n: henceforth, we assume that $I = 3.2n$ and $J = 1.6n$. We can also compute the number of spectral coefficients. Three sets of coefficients must be maintained, for three fields: δ, η, and Φ. Each field has size $(4 + 3n + n^2)/2$ (Foster and Gropp 1991), giving a total number of $(12 + 9n + 3n^2)/2$ spectral coefficients. For the problem sizes of interest here, $3n^2 \gg 12 + 9n$; hence, we approximate the number of spectral coefficients as

$$\frac{3n^2}{2}. \tag{9}$$

4.2 Alternative approaches

It is known that the computational cost of the spherical transform method is dominated asymptotically by the Legendre transform, which requires $\mathcal{O}(n^2)$ operations per longitude. The authors are not aware of any spectral climate model using an algorithm different than that described above. However, there are a number of possible replacements that are less computationally expensive per grid point. For example, Dilts describes an approach based on 2-dimensional FFTs (Dilts 85). In a recent paper (Alpert and Rokhlin 91), it is shown how to compute the Legendre transform (involving P_n) in time $\mathcal{O}(n \log n)$. Unfortunately, the extension of that method to $P_n^m(x)$, as required by the spherical transform method, remains an active research problem.

Orszag describes a method for evaluating a wide class of eigenfunction transforms (Orszag 86). This method is $\mathcal{O}(n \log^2 n / \log \log n)$ but has a rather large constant, making it slower than the direct method for modest sized n. As the resolution of climate models increases, an approach that uses Orszag's method will become faster than the current spectral transform method.

Finally, Boyd (Boyd 90, Boyd 91) has described two approaches for the fast evaluation of pseudospectral methods. The first uses the Fast Multipole Method (Greengard 88) to evaluate pseudospectral transforms on arbitrary grids (Boyd 90). The second takes advantage of weights of alternating sign in the sinc pseudospectral method to produce a fast method for evaluating this transform (Boyd 91). These recent results suggest that fast versions of either the spectral transform method, or an alternate pseudospectral method, may appear in the near future. Such methods will become increasingly competitive as the resolution of climate model increases.

Once alternative methods become established in a climate model, a scalability analysis of the sort described in this paper will need to be conducted to ascertain the best decomposition of the algorithm onto a parallel computer. The interested reader can gain an approximate understanding of the potential parallel performance of these algorithms by replacing the cost of the Legendre transform in the complexity equations given in subsequent sections with the cost of an FFT (for both arithmetic and communication). Note however that the precise computation and communication requirements of some alternative methods can be quite different: for example, the arithmetic complexity of the Fast Multipole Method is linear in the number of mesh points, while the communication cost of other replacement algorithms may well be higher than that of the Legendre transform.

5 Sequential Complexity

The first step in the development of a parallel performance model for a parallel algorithm is to determine the sequential time complexity of the algorithm, that is, the time requirements of the method on a uniprocessor computer, as a function of problem size. Here, we adopt the truncation number n as the problem size.

A simple inspection of the spectral method equations (Figure 2) shows that asymptotically the method performs $\mathcal{O}(n^3)$ operations. Hence, at sufficiently large n we can expect to be able to model the computational requirements of this method by a function of the form Cn^3, C a constant.

Unfortunately, the spectral method is never likely to be used in the asymptotic execution time regime, because of its enormous computational requirements for large n. In practice, we need to consider relatively small n: for example, in the range 40 to 500. This in turn requires that we perform a more detailed analysis, as lower-order terms can be significant at small n if the associated constants are large. To obtain an accurate picture of the time complexity of the spectral method, we relate the mathematical formulation given in Section 4 to a Fortran implementation of the method. This code, provided by the National Center for Atmospheric Research (NCAR) (Hack 1990), has been carefully tuned to obtain good performance on a vector computer. A detailed analysis of the Fortran program suggests that the number of operations performed is approximately

$$8 + 15.6n + 195n^2 + 550n^2 \log n + 35.2n^3.$$

We do not expect this expression to provide accurate predictions of program execution time, as the operations being counted are of different types. However, it suggests that we need to consider both $n^2 \log n$ and n^3 terms: $550n^2 \log n$ is actually larger than $35n^3$ for $n < 106$. Hence, we model sequential time as

$$T_e = C_1 n^2 \log n + C_2 n^3. \tag{10}$$

The coefficients C_1 and C_2 will be determined by fitting with experimental data (cf. Section 7).

6 Parallel Complexity

We now analyze the time requirements of parallel decompositions of the spectral transform method. In preparing this paper, we considered a large number of alternative decompositions (Foster and Gropp 1991). Here, we present just two. The first, one-dimensional decomposition, is used to introduce our approach and to motivate the empirical investigations described in Section 7. The second, two-dimensional decomposition, represents the most promising decomposition that we have investigated.

We assume in the following analysis that FFTs always involve a number of points that is an integer power of two. We also assume that decompositions always allocate the same number of points to every processor. These simplifying assumptions must be kept in mind when applying the analysis to real situations.

6.1 Data Dependencies and Data Decompositions

The time complexity of a parallel algorithm depends on the time complexity of the corresponding sequential algorithm (which we have already determined in the present case) plus the parallel overhead resulting from communication, idle time, and redundant computation. It is frequently possible to trade off redundant computation for communication: it may be cheaper to recompute a data value locally rather than fetch it from another processor. However, neither of the algorithms considered here involves significant redundant computation.

The communication requirements of a parallel algorithm depend on how data is allocated to processors (the *data decomposition*) and on the data values that are required when computing a new value for a piece of data (the *data dependencies*). The principal data dependencies inherent in the spectral transform method can be deduced from Figure 1. Each point in the intermediate array depends on all the values in the corresponding grid space latitude (row). Each point in spectral space depends on all the values in the

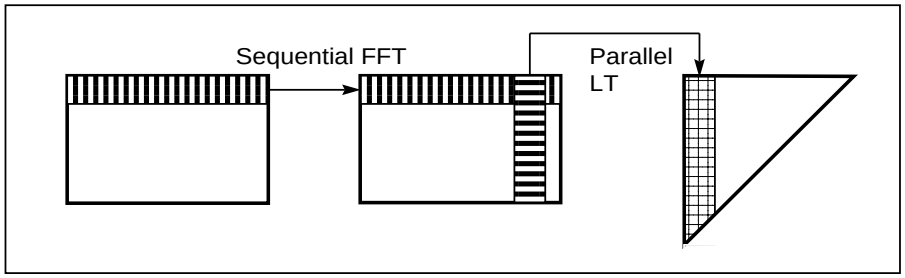

Figure 3: Decomposition by Latitude

corresponding column of the intermediate array. We shall explore two alternative data decompositions in subsequent sections.

6.2 Decomposition by Latitude

We first consider a simple decomposition by latitude, in which grid space variables from $J/(2p)$ latitude lines (grouped for reasons of computational efficiency as north/south pairs) and $1/p$ of the spectral coefficients are allocated to each of p processors. This decomposition is illustrated in Figure 3: the grid space variables and spectral coefficients allocated to a single processor are shaded with vertical bars.

It should be clear from the discussion in Section 4 that a decomposition by latitude permits FFTs to proceed without communication but requires communication for the forward and reverse Legendre transforms in Steps 1 and 4 (Figure 2). This is illustrated in Figure 3: updates to a single processor's spectral coefficients (C) are computed from a complete intermediate column (shaded in B). As this column is partitioned among all p processors, global communication is required. We shall assume that this communication is structured in terms of a ring pipeline (cf. Section 2.4).

The ring pipeline requires that each processor participate in $2(p-1)$ communication steps. Recall that there are approximately $3n^2/2$ spectral coefficients at truncation n (Eq. 9). We assume that a complex spectral coefficient occupies two words; hence, each message sent in the pipeline has size $3n^2/p$ words. If p is even, then communications can be organized in a mesh so that each message travels exactly one hop and no link carries more than one message in each step. The total communication cost incurred at a single processor is then

$$t_s 2(p-1) + t_h 2(p-1) + t_w \frac{p-1}{p} 6n^2.$$

Recall that efficiency is given by $E = (1 + T_o/T_e)^{-1}$ (Eq. 2). In the present case, T_o consists of the communication cost multiplied by the number of processors, p, and T_e is given by Eq. 10. Hence,

$$E = \left(1 + \frac{t_s 2p(p-1) + t_h 2p(p-1) + t_w(p-1)6n^2}{C_1 n^2 \log n + C_2 n^3}\right)^{-1}. \tag{11}$$

This equation also holds for a hypercube interconnect as a nearest neighbor embedding of a ring is also possible in this architecture. However, it may be more efficient on a hypercube to use a different algorithm based on spanning trees, particularly for large p (Foster and Gropp 1991).

In Section 7, we show how empirical studies can be used to calibrate this equation and predict the performance of the decomposition on a variety of parallel computers. The results there show that the communication terms in Equation 11 are significant for modest resolutions on current parallel computers.

A 1-D decomposition by longitude is also possible (Foster and Gropp 1991); in this case, the FFT requires communication while the Legendre transform can proceed without communication. However, as

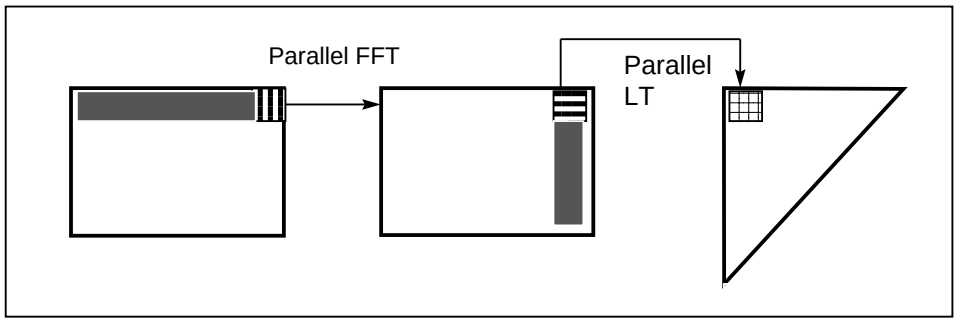

Figure 4: Decomposition by Latitude and Longitude

the 2-D decomposition considered in the next section proves to be better than any 1-D decomposition, we do not consider the longitudinal decomposition here.

6.3 Decomposition by Latitude and Longitude

The latitudinal decomposition has two disadvantages: it has a high communication overhead and can utilize at most $J/2 = 0.8n$ processors. We now consider a 2-D decomposition that requires significantly less communication and in addition can employ up to $(I/2)(J/2) \approx 1.3n^2$ processors.

Assume that p processors are partitioned into p/q sets of q processors each. We refer to these sets as q-sets. Assume also that the grid space variables (grouped as latitude pairs) and the spectral coefficients are partitioned and allocated to q-sets. Within each q-set, the variables are further partitioned by longitude, so that each processor is allocated I/q longitudes of each of the Jq/p latitudes allocated to its set. Longitudes are paired in such a way that the amount of computation performed in spectral space is the same on each processor. This decomposition is illustrated in Figure 4: as in Figure 3, the grid space variables and spectral coefficients allocated to a single processor are shaded with vertical bars and the data required from other processors when performing a forward transform are colored grey.

We see from Figure 1 that both the FFTs and the Legendre transforms require communication when this decomposition is used. The Legendre transforms are achieved with a ring pipeline as before and require 2(p/q-1) communications. The FFTs require $\log q$ communications. The cost of these communications depends on how the q-sets are arranged on a mesh.

We have identified two particularly promising organizations (Foster and Gropp 1991). In the first, each of the p/q q-sets is allocated as a sub-mesh, and processors responsible for corresponding spectral subvectors are scattered across the mesh. In the second, the processors responsible for corresponding spectral subvectors are concentrated, and processors in the same q set are scattered. We present the first of these alternatives here. For simplicity, we assume that $q = r^2$, for some integer r, and that the processors in each q-set are organized in a $r \times r$ submesh. The organization is illustrated in Figure 5, with $p = 64$ and $q = 16$. Processors allocated to one of the four q-sets are outlined; processors holding corresponding spectral subvectors are shaded.

Ring Pipeline. A total of $2q$ ring pipeline operations must be performed, each involving p/q processors (one from each q-set) and requiring $(p/q) - 1$ communications of approximately $3n^2/p$ words (cf. Section 6.2). The q-sets can be organized in such a way that the processors involved in a particular pipeline form a mesh in which neighboring processors are $\sqrt{q}$ distant. This organization is illustrated in Figure 5; the processors involved in one of the pipelines are distinguished by grey shading. Each communication must traverse $\sqrt{q}$ hops; hence, in the absence of competition for bandwidth, communication costs from the

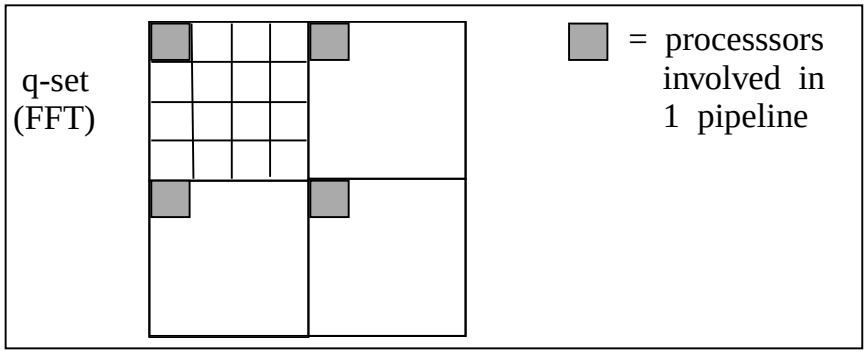

Figure 5: Mesh Organization

pipeline are

$$2\left(\frac{p}{q}-1\right)\left(t_s + t_h\sqrt{q} + t_w\frac{3n^2}{p}\right).$$

In practice, q pipelines must operate concurrently. Assume that communications are organized so that all four links connecting each processor are fully utilized. Then at each communication step, each link must carry $\sqrt{q}/4$ messages; if $\sqrt{q} \geq 4$, the effective bandwidth must be reduced by this factor to reflect this load. Hence, the total communication cost from the global sum is

$$2\left(\frac{p}{q}-1\right)\left(t_s + t_h\sqrt{q} + t_w\frac{3n^2\sqrt{q}}{4p}\right) \quad \text{when} \quad (\sqrt{q} \geq 4).$$

Parallel FFT. An unordered FFT on r points using q processors, with r and q integer powers of 2, can be achieved in $\log q$ steps in which each processor exchanges r/q points with another processor, followed by a local computation step (Pease 1968). The q processors can organized on a mesh in such a way that the communication steps involve processors in the same row or column, 2^0, 2^1, ..., $2^{\log(\sqrt{q})-1}$ hops distant, for a total of $2\sqrt{q}$ hops. It can be shown that the total communication cost of the FFT with this organization is approximately (Gupta and Kumar 1990; Foster and Gropp 1991),

$$t_s \log q + t_h 2(\sqrt{q}-1) + t_w 2\frac{r(\sqrt{q}-1)}{q}.$$

The data transfer term is $2r(\sqrt{q}-1)/q$, not $2r\log q/q$ as might be expected, because competition for bandwidth increases the time required for data transfer. On some computers, it may be possible to reduce the data transfer overhead by overlapping some computation with other processors' communication. However, we do not consider this possibility here.

The decomposition considered here allocates $Jq/p = 1.6nq/p$ latitudes to each q-set. A parallel FFT must be performed for each latitude; each FFT involves a vector of size $I = 3.2n$ words. We show in (Foster and Gropp 1991) that a total of 10 sets of FFTs is required (3, 2, and 5 in Steps 1, 2, and 3, respectively), giving a total FFT size of $r = 10(1.6nq/p)(3.2n) \approx 51qn^2/p$. As all FFTs in each step can be performed concurrently, the different FFTs can be bundled into a total of three transforms. Hence, the total communication cost from FFTs is

$$t_s 3\log q + 2(\sqrt{q}-1)(t_h 3 + t_w\frac{51n^2}{p}).$$

Efficiency. Let T_e be defined by Eq. 10. The overhead, T_o, consists of the communication cost associated with both the ring pipelines and the parallel FFTs, multipled by p. Hence for $\sqrt{q} \geq 4$,

$$E = \left(1 + \frac{2p\left(\frac{p}{q} - 1\right)\left(t_s + t_h\sqrt{q} + t_w\frac{3n^2\sqrt{q}}{4p}\right) + t_s 3p\log q + 2(\sqrt{q} - 1)(t_h 3p + t_w 51n^2)}{C_1 n^2 \log n + C_2 n^3}\right)^{-1}. \quad (12)$$

Somewhat different results are obtained for a hypercube interconnect. If each q-set is allocated to a subcube, both the FFT and the ring pipeline can proceed with only nearest neighbor communication and no competition for bandwidth. In this case

$$E = \left(1 + \frac{2p\left(\frac{p}{q} - 1\right)\left(t_s + t_h + t_w\frac{3n^2}{p}\right) + 3\log q\,(t_s p + t_h p + t_w 17n^2)}{C_1 n^2 \log n + C_2 n^3}\right)^{-1}. \quad (13)$$

As in the 1-D case, we note that it may be more efficient on a hypercube to use a different algorithm based on spanning trees, particularly for large p.

The optimal value of q depends on various characteristics of a computer such as interconnect and machine parameters. In a mesh, our performance model shows that it is typically about $\sqrt{p}$. In a hypercube, where the cost of the FFT communications is greatly reduced, q should be as large as possible.

7 Calibration

The analytic models developed in preceding sections are intended to capture as accurately as possible the behavior of parallel algorithms on architectures of interest. However, the models still need to be calibrated to account for variables such as compilers, processor speed, and additional communication costs (e.g., formatting of message buffers) not accounted for by the models. We achieve this calibration by fitting them with data obtained from actual executions.

Recall that the performance model that we have developed for the 1-D decomposition includes five implementation- and machine-specific coefficients: C_1, C_2, t_s, t_h, and t_w. We compute values for these coefficients by fitting the model to run-times from a parallel shallow-water equation code developed by P. Worley (Worley and Drake 1991) at Oak Ridge National Laboratory, using the decomposition described in Section 6.2. This code is based on the sequential code discussed in Section 5 (Hack 1990), and is run here on an Intel GAMMA, a multicomputer with 128 i860 processors. A least squares fit to this data gives the following values:

$$C_1 = 36\ \mu\text{sec},\ C_2 = 2.4\ \mu\text{sec},\ t_s + t_h = 230\ \mu\text{sec},\ t_w = 1.2\ \mu\text{sec}.$$

The fit is illustrated in Figure 6, with observed speedup indicated by points and predicted speedup by solid lines, for a variety of different n. Note that the proportion C_1 / C_2 (20.1) is in reasonably good agreement with the ratio (15.6) between the $n^2 \log n$ / n^3 operation counts obtained in Section 5.

Because the time complexity for the floating-point computation contains terms with the same asymptotic form as the principal communication terms, it is impossible to determine an exact correspondence between the t_s, t_h, and t_w values obtained by fitting and the "best achievable" communication parameters given in Table 1 ($t_s = 136\mu$sec, $t_h = 2.0\mu$sec, $t_w = 1.6\mu$sec.). However, the similarities in the values give us confidence that our model is accurate and the parallel implementation is efficient. The disparities may be artifacts of the fitting process or may reflect additional operations (e.g., buffer formatting and management) or optimizations (e.g., overlapping of communication and computation) not dealt with by our model.

The disparity between "best achievable" and observed communication costs poses problems when models are used for performance predictions. We choose to use the t_s, t_h, and t_w values given in Table 1 when making performance predictions. This means that it may be possible to achieve slightly better performance than we predict in some circumstances. However, we expect overlapping of communication and computation to be less significant on future parallel computers (in which channel bandwidth will be higher). The

Figure 6: Predicted (lines) and Observed (data points) Speedups

interested reader is invited to substitute the more aggressive observed values into our models and compare predictions. The qualitative nature of our conclusions is not changed by these small variations in machine parameters.

8 Forecasting Model Performance

We use our analytic models and empirical studies to predict the performance of spectral transform-based shallow-water equation models on future parallel computers. For concreteness, we focus primarily on computers that are likely to become available in the next two to five years; these were described in Section 2.3.

We employ two different measures of parallel performance: speedup and throughput. The first of these measures is commonly used in parallel computing; the second is useful because, when plotted as a function of resolution, it makes apparent the relationship between problem size and parallel performance.

8.1 Speedup

We first develop projections for achievable speedups on the first three target computers. Recall that speedup $S = pT_e/(T_e + T_o)$. These forecasts use the 2-D performance model developed in Section 6 (Eq. 12). (The 1-D model is not considered here, as it cannot utilize sufficient processors to be interesting.) The coefficients C_1 and C_2 in Eq. 12 are those obtained in the calibration studies reported in Section 7, scaled according to the scaling factors specified in Table 1. The machine parameters are also obtained from Table 1.

Performance predictions for the DELTA, SIGMA, and NCUBE-2 are given in Figures 7, 8, and 9 respectively. The parameter q is chosen to be $\sqrt{p}$, which gives close to optimal performance. Note that the 2-D decomposition is able to employ all 2,048 SIGMA processors when $n > 40$ and all 8,096 NCUBE-2 processors when $n > 80$.

Recall that parallel efficiency is defined as speedup divided by the number of processors (Equation 2).We see that in each case, reasonable efficiencies are achieved at high resolutions (e.g., 73%, 77%, and 71% at T213 for the DELTA, SIGMA, and NCUBE-2, respectively). However, at lower truncation numbers the results are poor: at T80, the DELTA, SIGMA, and NCUBE-2 achieve 50%, 35%, and 23% efficiency, respectively.

These disappointing results can easily be accounted for: the high communication requirements of the spectral method come to dominate execution time as the number of processors is increased. (Note that all three machines do relatively well when the number of processors is small.) The hypercube achieves better speedup than the DELTA and SIGMA for two reasons: the hypercube interconnect scales better, and the node processors are slower. However, we shall see in the next section that the DELTA and SIGMA still achieve better total throughput.

8.2 Throughput

The results obtained in the preceding section make it clear that the performance of the parallel spectral transform depends critically on model resolution. Achievable resolution depends in turn on the throughput that is required from a parallel model. We now quantify the relationship between achievable resolution and required throughput for different parallel computers.

Let throughput R be expressed in simulated years per real-time hour. We have

$$R = \frac{pE}{YT_e}(3.6 \times 10^9 \ \mu\text{sec/hour}) = \frac{3.6 \times 10^9 p}{Y(T_e + T_o)}, \tag{14}$$

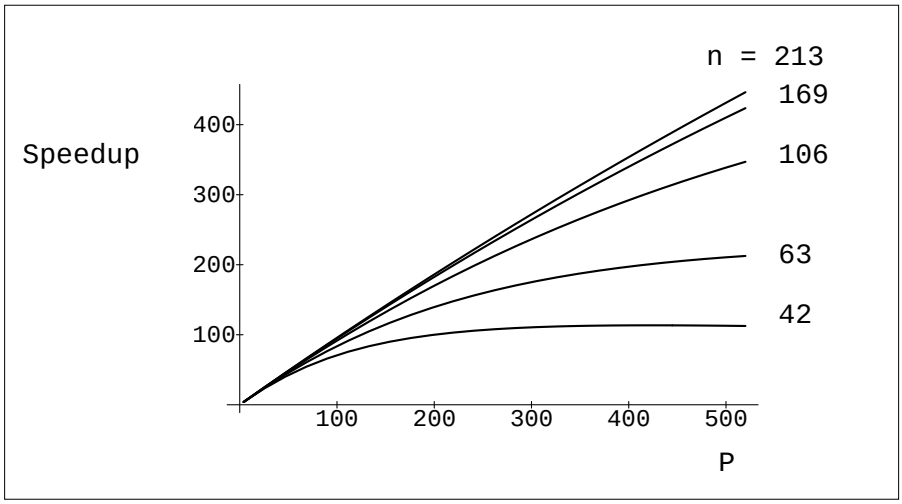

Figure 7: DELTA Forecast: 2-D Decomposition

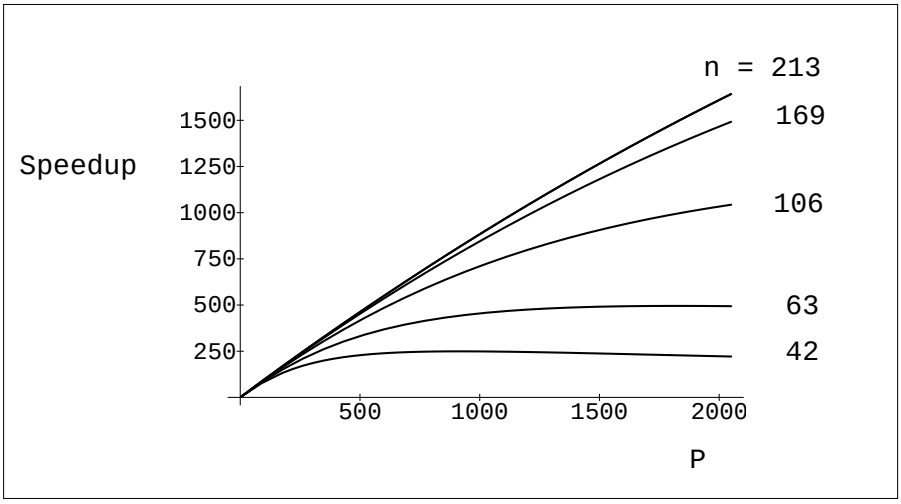

Figure 8: SIGMA Forecast: 2-D Decomposition

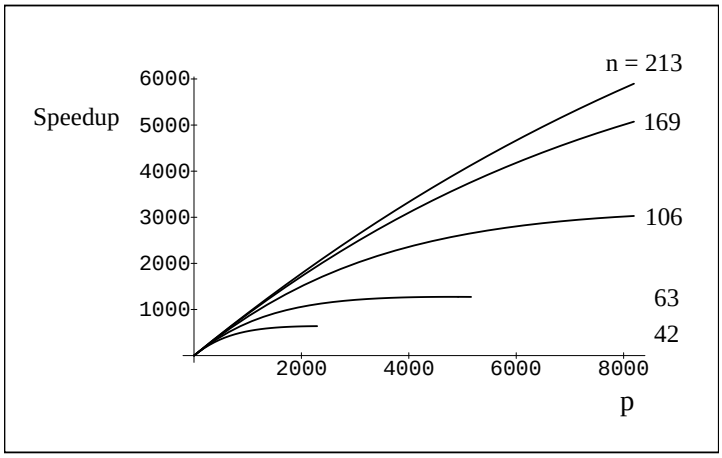

Figure 9: NCUBE-2 Forecast: 2-D Decomposition

where Y is the number of time steps required for a one-year simulation. The timestep is normally a function of resolution, but we fix it here to a constant 0.5 hours, giving $Y = 18,000$.

T_e is given by Eq. 10; T_o is easily obtained from Eq. 2. These quantities, plus the machine parameters and sizes presented earlier, allow us to compute throughput (R) as a function of truncation (n). We plot this function in Figure 10, for the DELTA, SIGMA, and NCUBE-2 (on 528, 2,048, and 8,192 processors, respectively).

A number of interesting phenomena are apparent in Figure 10. We see that rapid increases in computational requirements lead to a decrease in throughput as resolution increases. However, this effect is balanced somewhat by an increase in parallel efficiency at higher resolutions. NCUBE-2 throughput decreases less rapidly than that of the DELTA or SIGMA, because of its hypercube interconnect, and matches but does not exceed the DELTA at high resolutions.

Figure 10 may be used to determine maximum achievable resolution as a function of desired throughput rate. For example, assume that we wish to complete a 100-year shallow-water equation simulation in ten hours. We predict that this task can achieved at about T120, T120, and T275 resolution, on the NCUBE-2, DELTA, and SIGMA, respectively.

9 Relevance to General Circulation Models

There are several important differences between the shallow-water equations considered in preceding sections of this paper and the general circulation models used in weather and climate modeling. Here, we discuss how three of these differences — the full primitive equations, multilayer representations of the atmosphere, and physics — effect our analysis. We also develop estimates for the overall computational requirements and scalability of a GCM. These estimates are based on a number of assumptions (listed in the text) which may or may not hold in practice. We encourage the reader who disagrees with our assumptions to explore the impact of alternative choices by modifying the basic model appropriately.

There are two important aspects of full GCMs that we do *not* consider in this paper: I/O and coupling with other (e.g., ocean) models. We expect I/O requirements to be a determining factor in the scalability and usability of parallel GCMs. Coupling also introduces difficult problems. However, as these issues are essentially orthogonal to numerical methods, we choose to ignore them here.

Finally, we do not consider communication requirements associated with local coupling of grid space values, resulting, for example, from the use of advanced physics modules with horizontal coupling or semi-

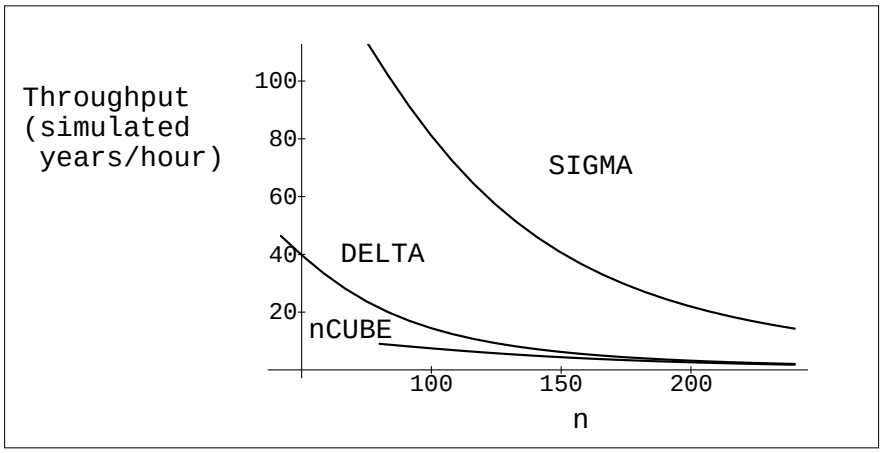

Figure 10: Throughput vs. Truncation

Lagrangian transport (Williamson and Rasch 1989).

9.1 Effects of GCM Differences

The substitution of the primitive equations for the shallow-water equations requires extra computation (and, in a parallel code, communication) but does not otherwise impact our analysis. We shall not attempt to model the additional computation directly. Instead, we introduce a scaling coefficient into our shallow-water equation model and determine the value of this coefficient by calibration with execution times reported for operational weather and climate models. Neither do we develop a detailed model of primitive equation communication costs. Instead, we simply double the communication volume communicated in each level, to approximate overhead due to forcing and additional data fields.

As noted earlier, we assume here that the vertical structure of a GCM is not changed in a parallel implementation: that is, all levels for a single mesh point are placed on the same processor. This means that the volume of computation and communication will scale linearly with the number of levels, but the actual number of communication operations can stay fixed, as data from all levels may be sent simultaneously. Hence, the presence of multiple layers in the GCM will tend to actually improve overall parallel efficiency. However, as the data volume term (t_w) already dominates communication costs in the machines considered here, we do not expect a significant improvement.

The physics component of GCMs frequently dominates computational requirements, at least at low resolutions. The physics component of GCMs is commonly implemented as a purely local (column-oriented) operation that does not require additional communication. Hence, this part of a GCM would exhibit perfect speedup if the same computational effort were required at each grid point. In practice, the physics computation performed at a grid point depends on aspects of model state such as the time of day, season, or water vapor concentration. However, detailed studies conducted on CCM1 suggest that the overall load imbalance caused by these factors is relatively small (Michalakes 1990). For example, these studies show that a parallel efficiency of 91% would be achieved in the physics component of CCM1 at T42 resolution, if eight grid points were located on each processor. The magnitude of physics load imbalances in future models remains to be seen. In general, however, it would seem that physics computation will tend to increase overall parallel efficiency.

This discussion suggests that the scalability of spectral-transform-based GCMs is constrained primarily by the spectral transform method used to compute the dynamics. At low resolutions, the additional physics computation will tend to reduce throughput but increase parallel efficiency. The scalability analysis developed in Section 6 should be directly applicable, if corrected appropriately for the increased computation

and communication requirements.

9.2 Computational Requirements of GCMs

In order to estimate the computational requirements for a GCM, we must estimate what fraction of the work is part of the dynamics (which we can model with our shallow-water estimates) and what is "other" work (work that is nearly or perfectly parallelizable). We can use some published results from existing GCMs to help estimate the relative costs of different parts of a GCM without restricting ourselves to a detailed model of a specific approach.

We develop a simple model for the total computational requirements of a spectral-transform-based GCM. Let D and P represent the computational requirements of the dynamics and physics components of a GCM in a single time step, and Y the number of time steps in a year. Then the computation required for a one-year simulation, T_y, is given by

$$T_y = (D + P)Y.$$

We make a number of simplifying assumptions in order to obtain values for D, P, and Y. First, we assume that the computation required for the dynamics component is the time required by the shallow-water equations, multiplied by both the number of vertical layers and a scaling coefficient C_m reflecting the additional cost of the primitive equations. Reports by users of a number of GCMs suggest that the number of vertical layers is often chosen to be approximately $10 + n/4$, where n is the truncation. In order to predict execution times on an i860, we substitute the the values obtained from the calibration studies reported in Section 7 for the execution time coefficients C_1 and C_2 (cf. Eq. 10). This yields

$$D = C_m(36n^2 \log n + 2.4n^3)(10 + n/4)\ \mu\text{sec}.$$

Second, we assume that the physics computation is evenly distributed over all processors and that the physics component of the model takes time proportional to the product of n^2 and the number of vertical layers. That is,

$$P = C_p n^2 (10 + n/4)\ \mu\text{sec},$$

where C_p is a coefficient representing the cost per grid point. In the ECMWF model, the physics component accounts for 51% of the computation at T106 resolution (Simmons 1990). We use this information to determine an approximate value for C_p: 526 μsec. As in Section 8.1, we assume that the number of time steps per year is $Y = 18,000$. Hence, we obtain

$$T_y = C_m 18,000(10 + \frac{n}{4})(526n^2 + 36n^2 \log n + 2.4n^3)\ \mu\text{sec}. \tag{15}$$

We calibrate this model with execution times reported for the ECMWF and CCM1 models. We assume that the i860 processor used to run the shallow-water equations achieves a sustained rate of 8 Mflops, and obtain good fits with $C_m = 4.4$. That is, it seems that typical primitive equation models requires 4.4 times more computational resources per vertical layer than NCAR's shallow-water equation code. The points fitted to, and the predicted values, are shown in Figure 11. The T213 value is scaled by a factor of 2.5 to compensate for its unusually low vertical resolution.

9.3 Throughput

As noted in Section 9.1, we can expect the scalability of a spectral-transform-based parallel GCM to be similar (probably somewhat better) than that of a shallow-water equation model. On the other hand, throughput of a GCM will be significantly lower. Here, we develop estimates for throughput of a parallel GCM on the target computers listed in Table 1. These estimates were obtained using a model similar to that presented as Eq. 12. However, the computation time estimates are those from the preceding section and the transfer time (t_w) terms are scaled by the number of vertical layers. In addition, total

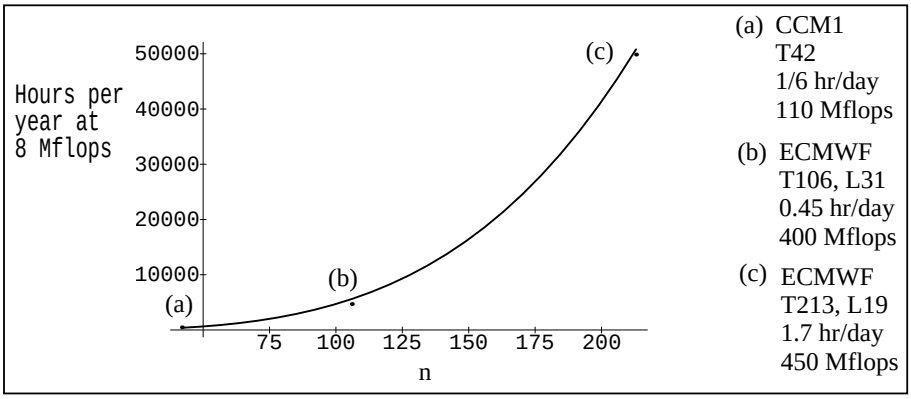

Figure 11: Fitting GCM Performance Predictions. The solid line represents a fit to T_y for the three data points, which have been normalized to a 8 Mflop CPU.

communication costs are scaled by a factor of two to approximate additional communication requirements due to the primitive equations, forcing, etc. Specifically, the computational time is given by the formula

$$\begin{aligned} T &= (D + P + 2O)Y \\ D &= (10 + n/4)T_e \\ O &= 2(\frac{p}{q} - 1)\left(t_s + t_h\sqrt{q} + t_w\frac{3(10 + n/4)n^2\sqrt{q}}{2p}\right) + \\ &\qquad 3t_s \log q + 2(\sqrt{q} - 1)\left(3t_h + t_w 52(10 + n/4)\frac{n^2}{p}\right) \end{aligned}$$

where $10 + n/4$ approximates the number of vertical layers.

We present in Figure 12 estimated throughput (in simulated years per hour) as a function of resolution (i.e., truncation) (cf. Section 8.2), for the DELTA, SIGMA, and NCUBE-2. Figure 14 presents the same information for the Teraflop computers characterized in Table 1. One interesting phenomenon is that the NCUBE-2 performs proportionally better than in the shallow equation case, relative to the DELTA; we attribute this result to the higher communication volumes. In addition, the hypercube interconnect allows the *BigCube* machine to achieve best overall performance at high resolutions. However, it can keep all 65,536 processors busy only at resolutions higher than 224.

The most striking aspect of Figures 12 and 14 is the low throughput rates achieved. For example, assume that we wish to complete a 100-year simulation in a week. This requires that we achieve a throughput of 0.6 simulated years/hour. The highest resolutions at which this throughput can be achieved are approximately T47 (DELTA), T100 (SIGMA), and T140 (*FastMesh*). The NCUBE-2 and *BigCube* machine cannot achieve this throughput rate. At these resolutions, we predict that the former three machines all achieve around 90% efficiency.

Alternatively, we may wish to achieve a 10-year run in a week. In this case, the highest resolutions that can be achieved are approximately T118 (DELTA and NCUBE), T220 (SIGMA), T300 (*FastMesh*), and T410 (*BigCube*).

These results suggest that when used for climate modeling, parallel spectral-transform-based GCMs will not be run at very high resolutions (probably not more than T300) even on large parallel machines. Hence, Eq. 15 suggests that physics computation will make a significant contribution to execution time. To clarify the relationship between resolution and relative physics/dynamics execution times, we show in Figure 15 the predicted proportions of time spent in physics, dynamics, and parallel overhead on the Intel SIGMA as a function of resolution. We see that physics never dominates total execution time. Nevertheless, it is apparent that the dynamics, despite its $O(n^3)$ asymptotic cost, becomes a major burden

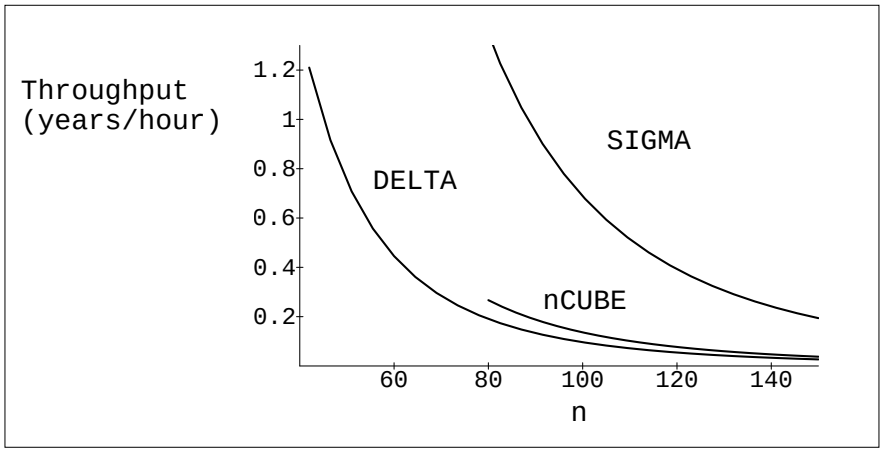

Figure 12: GCM Throughput vs. Truncation (I)

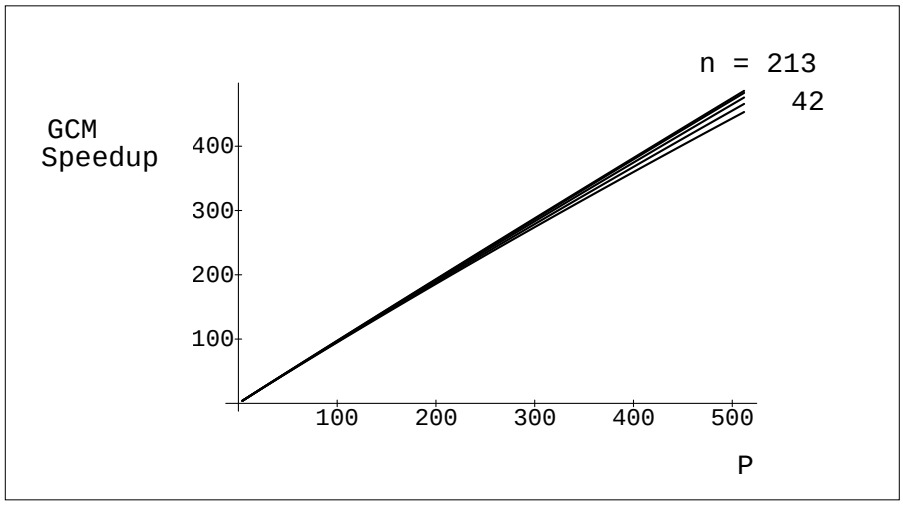

Figure 13: GCM Speedup on Intel DELTA

only at resolutions much higher than we can expect to run in the foreseeable future. Finally, we see that when physics is included, communication overhead accounts for a large proportion of total execution time only at very low resolutions.

The focus of discussion in this section has been on the throughput required for large-scale climate modeling runs. Our models can also be used to analyze weather modeling scenarios, in which throughput requirements are lower and higher resolutions can be considered. However, the memory requirements of the spectral transform method must also be evaluated carefully at higher resolutions.

In Figure 13 we have computed the estimated speedups for a full GCM on a 512 node DELTA. We see that high parallel efficiency is achieved even for modest problem sizes. This is largely attributable to the addition of physics computations, which increase the amount of work but do not change the parallel overhead, and the increase in the amount of work for the dynamics computations.

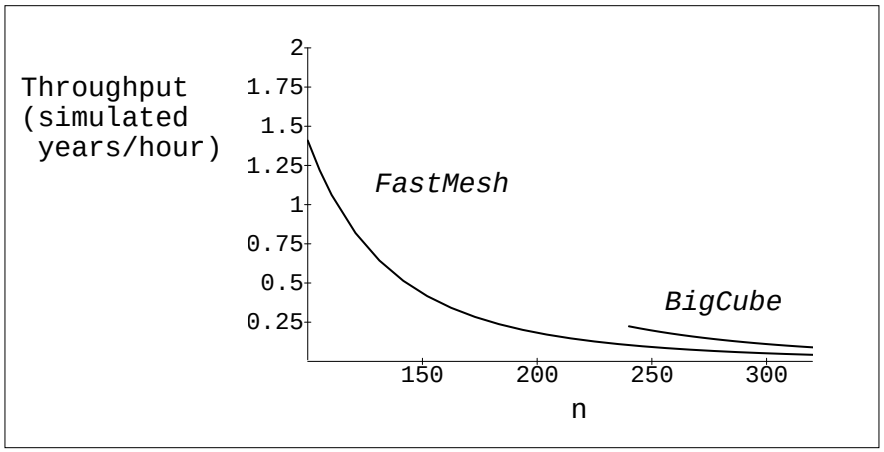

Figure 14: GCM Throughput vs. Truncation (II)

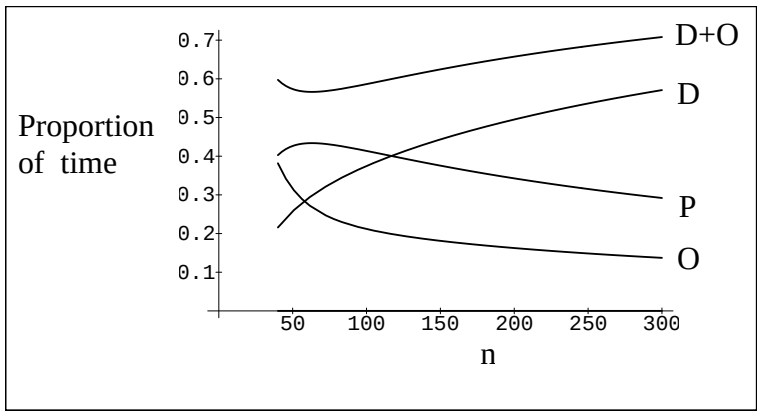

Figure 15: GCM Time Breakdown (SIGMA): D=dynamics, P=physics, O=overhead

10 Conclusions

The spectral transform method is a natural candidate for general circulation models designed to run on large-scale parallel computers. In this paper, we have presented analytic and empirical studies intended to elucidate the feasibility of constructing such models. In particular, we have attempted to quantify the *scalability* of the spectral transform method: that is, the extent to which a parallel implementation of this method can profitably utilize increasing numbers of processors.

The bulk of the paper has focused on the shallow-water equations rather than the primitive equations used in GCMs. This simplification has permitted us to calibrate our performance models with empirical data. We have investigated a variety of parallel spectral transform algorithms; in this paper, we have presented performance models for two of these, based on 1-D and 2-D decompositions of the principal data structures.

The performance models can be used to predict performance for shallow-water equation models on a variety of multicomputers. We have developed predictions for several machines. We find that parallel efficiency is generally poor, because of the high communication requirements of the parallel spectral transform. This is in spite of the fact that the high asymptotic complexity of the spectral transform method provides a great deal of computation that can be done in parallel.

We have also discussed how our results apply to full GCMs. We account for primitive equations, vertical layering, and physics components in our performance model by changing coefficients on computation and communication terms. We use this extended model to develop a rough analysis of computational requirements and throughput. We emphasize that the results obtained are highly approximate, as a number of assumptions in our model may not hold in practice. Furthermore, our projections probably underestimate communication requirements and definitely ignore potential computational and communication demands of grid-space coupling, I/O, coupled ocean and biosphere models, and smaller time steps. Nevertheless, we believe that our estimates of the relationship between compute power, required throughput, and achievable resolution provide useful insights into the feasibility of spectral-transform-based parallel GCMs.

We find that the parallel efficiency of spectral-transform-based GCMs is likely to be considerably better than that of the shallow-water equation code, even at the same resolution. This effect is due to increased computational requirements in GCMs, associated with both more complex dynamics and the introduction of physics. However, despite this increased efficiency we find that even projected Teraflop computers will have difficulty achieving acceptable throughput at the high resolutions believed necessary to resolve regional climate. For example, consider our (optimistic) analysis of the performance of the *FastMesh* Teraflop architecture. This predicts a maximum resolution of only T185 (66km) being achieved if a 100 year simulation is to be completed within a week. In practice, achieved resolutions may be even lower. These results have significant consequences for designers of both physics parameterizations and parallel models.

It is commonly believed that the spectral transform method is impractical for large-scale parallel models because of its elevated computational and communication requirements. However, it is only at high resolutions that the cost of dynamics computations dominates the cost of physics computations. At these resolutions, the cost of physics alone makes extended (100 year or more) simulations impractical, even on Teraflop computers. At lower resolutions, both dynamics costs and parallel efficiency are reasonable. Thus, for long term climate simulations, we conclude that parallel efficiency is not a reason to reject the spectral transform method as a basis for the implementation of parallel GCMs. In contrast, for short term simulations where there is adequate computational power to handle the model physics at much higher resolution, the $\mathcal{O}(n^3)$ time complexity of the spectral transform method will dominate the computation time and the apparent parallel efficiency will be due to the serial inefficiency (relative to fast $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log n)$ methods).

Further analytic and empirical studies are required to refine the models and predictions presented in this paper. In particular, we need to measure more accurately the communication performance of parallel FFTs, the communication requirements of primitive equation models, and the effect of issues such

as semi-Lagrangian transport, coupled models, and I/O on parallel performance. Also to be investigated are methods that use parallel transpose operations with one-dimensional decompositions; these allow both the FFT and the Legendre transform to be computed without any additional communication. The high computational requirements of parallel GCMs (for both dynamics and model physics) suggests that adaptive refinement methods and nested global/mesoscale models may also be promising avenues for further research.

In summary, we note that the development of GCMs for massively parallel computers can be an expensive exercise. The approach to performance analysis advocated in this paper can reduce the potential for serious error, by guiding our implementation efforts, uncovering performance problems in software and hardware, and allowing us to evaluate intelligently alternatives in numerical methods, algorithms, and hardware. We believe that performance analysis should be an essential component of the design and development process for any parallel model.

Acknowledgments

This research was supported by the Atmospheric and Climate Research Division and the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy. 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 and for useful discussions.

References

Alpert, B., and Rokhlin, V., 1991: A fast algorithm for the evaluation of Legendre Expansions, *SIAM J. Sci. Stat. Comp.*, 12(1), 158–179.

Boyd, J. P., 1990: Multipole expansions and pseudospectral cardinal functions: a new generalization of the fast Fourier transform, Preprint, University of Michigan.

Boyd, J. P., 1991: Sum-accelerated pseudospectral methods: the Euler-accelerated sinc algorithm, *Appl. Num. Math.* 7, 287–296.

Bourke, W., 1972: An efficient, one-level, primitive-equation spectral model, *Mon. Wea. Rev.* 102, 687–701.

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

Dilts, G. A., 1985: Computation of spherical harmonic expansion coefficients via FFTs, *J. Comp. Phys.*, 57(3), 439–453.

DOE, 1990: U.S. Department of Energy, Building an Advanced Climate Model: Program Plan for the CHAMMP Climate Modeling Program, Publication DOE/ER-0479T (available from National Technical Information Service).

Foster, I., and Gropp, W., 1991: Unpublished information.

Gates, W., Potter, G., Phillips, T., and Cess, R., 1990: An overview of ongoing studies in climate model diagnosis and intercomparison, *Energy Sciences Supercomputing 1990*, UCRL 53916, DOE.

Greengard, L., 1988: The rapid evaluation of potential fields in particle systems, MIT Press, Cambridge.

Gupta, A., and Kumar, V., 1990: The scalability of FFT on parallel computers, Tech. Report, University of Minnesota.

Hack, J., 1990: Personal communication.

Kauranne, T., 1990a: Asymptotic parallelism of weather models, in *The Dawn of Massively Parallel Processing in Meteorology*, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 303-314.

Kauranne, T., 1990b: An introduction to parallel processing in meteorology, in *The Dawn of Massively Parallel Processing in Meteorology*, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 3–20.

Michalakes, J., 1990: Analysis of workload and load balancing issues in NCAR Community Climate Model, Tech. Report ANL/MCS-TM-144, Argonne National Laboratory (available from the DOE Office of Scientific and Technical Information).

Orszag, S. A., 1986: Fast eigenfunction transforms, in *Science and Computers, Advances in Mathematics Supplementary Studies*, G. C. Rota, ed., Academic Press, New York, 23-30.

Pease, M., 1968: An adaptation of the fast Fourier transform for parallel processing, *JACM*, 15(2), 252–264.

Simmons, A., 1990: Some computational aspects of numerical weather prediction, *Europhysics Conference Abstracts*, 14F, 21. (available from DOE Office of Scientific and Technical Information).

Washington, W., and Parkinson, C., 1986: *An Introduction to Three-Dimensional Climate Modeling*, University Science Books.

Williamson, D., ed., 1990: CCM progress report — July 1990, NCAR Tech. Note 351.

Williamson, D., and Rasch, P., 1989: Two-dimensional semi-Lagrangian transport with shape-preserving interpolation, *Mon. Wea. Rev.* 117(1), 102–129.

Worley, P., and Drake, J., 1991: Parallelizing the spectral transform method—Part I, Tech. Rep. ORNL/TM-11747, Oak Ridge National Laboratory, Oak Ridge, Tenn. (available from DOE Office of Scientific and Technical Information).

List of Figures

List of Tables