ORNL/TM-13304

Computer Science and Mathematics Division

Mathematical Sciences Section

A PARALLEL PERFORMANCE STUDY OF THE CARTESIAN METHOD FOR PARTIAL DIFFERENTIAL EQUATIONS ON A SPHERE

John B. Drake
Oak Ridge National Laboratory
and
Matthew P. Coddington
Swarthmore College

Date Published: January 14, 1997

Research sponsored by the U.S. Department of Energy CHAMMP Program of the Office of Health and Environmental Research.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Lockheed Martin Energy Research Corp.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-96OR22464

Contents

Acknowledgments

This report is one of a series of documents describing the use of parallel computers for global climate modeling. The work reported is sponsored by the CHAMMP program of the Department of Energy's Office of Energy Research, Environmental Sciences Division. We gratefully acknowledge the support of the CHAMMP program and the ORNL PIP program.

We would like to thank our collegues Paul Swarztrauber and David Williamson of the National Center for Atmospheric Research for their work in developing the Cartesian method. The ORNL Center for Computational Sciences (CCS) computers were used for this study under the auspices of the CHAMMP program.

A PARALLEL PERFORMANCE STUDY OF THE CARTESIAN METHOD FOR PARTIAL DIFFERENTIAL EQUATIONS ON A SPHERE

John B. Drake
Oak Ridge National Laboratory
and
Matthew P. Coddington
Swarthmore College

Abstract

A 3-D Cartesian method for integration of partial differential equations on a spherical surface is developed for parallel computation. The target computer architectures are distributed memory, message passing computers such as the Intel Paragon. The parallel algorithms are described along with mesh partitioning strategies. Performance of the algorithms is considered for a standard test case of the shallow water equations on the sphere. We find the computation times scale well with increasing numbers of processors.

1. INTRODUCTION

The solution of partial differential equations (PDEs) forms the core of many scientific applications. The flow equations in a rotating spherical geometry, for example, are important to weather and climate modeling. This report describes a new method for solving PDEs on a spherical domain and a parallel algorithm implementing the method on a massively parallel computer.

The use of triangular meshes for the solution of PDEs on a spherical domain is attractive for several reasons. Triangles allow nearly uniform meshes, while rectangular meshes suffer the problem of varying resolution near the poles for the standard spherical coordinates. Secondly, triangles require only a simple data structure for use with adaptive mesh techniques or for meshes that resolve irregular features. Adapting a mesh to fit a coastline is an obvious example. Our renewed interest in these methods springs from advances in computing and numerical analysis. The granularity of tasks that can be performed in parallel is advantageous for the finite difference and finite element methods and offers the possibility of effective use of many processors of a parallel computer.

Williamson and several others investigated the use of icosahedral-triangular meshes in a series of early papers [7, 14, 15, 16, 17, 19]. The shallow water equations on the sphere served as a primary equation set for testing the numerical methods because of their relevance in atmospheric flow models. The review article by Williamson [18] gives further references.

The Cartesian form of the shallow water equations was proposed by Swarztrauber in [20]. This formulation avoids the singularity in the velocity at the pole by expressing velocities in a 3-D Cartesian form instead of in spherical coordinates. It was used for the calculation of derivatives using a spectral vector harmonic method in [10]. In this paper we consider the Cartesian formulation for the calculation of derivatives using a stencil of points located on an icosahedral grid. We focus on the computational performance and the parallelism of the method.

In section 2, the shallow water equations and the Cartesian formulation are introduced. The numerical algorithm and the discretization of the differential operators on the sphere are discussed in section 3. Parallelism is addressed in section 4 with a description of the mesh partitioning algorithms and the structure of the parallel code. Section 5 gives performance results for a particular shallow water flow simulation on an Intel Paragon Parallel Computer.

2. THE SHALLOW WATER EQUATIONS ON A SPHERE

The equations representing conservation of momentum and mass for a fluid on the surface of a sphere of radius a can be written in advective form

$$\frac{d\mathbf{v}}{dt} = -f\mathbf{k} \times \mathbf{v} - g\nabla h + \mathbf{F}_{\mathbf{v}}, \tag{1}$$

and

$$\frac{dh^*}{dt} = -h^* \nabla \cdot \mathbf{v} + F_h. \tag{2}$$

The (material) derivative, given by

$$\frac{d}{dt}(\ ) \equiv \frac{\partial}{\partial t}(\ ) + \mathbf{v} \cdot \nabla(\ ), \tag{3}$$

is the rate of change as seen from a particle moving with the fluid. The velocity is referred to a rotating Cartesian frame and the components of $\mathbf{v} = (u, v)$, are in the longitudinal (λ) and latitudinal (θ) directions, respectively. The height of the fluid layer above the reference surface

$(r = a)$ is denoted $h = h^* + h_s$. The bottom surface height can be used to specify orography and is given by the time invariant function h_s. Thus, h^* represents the depth of fluid from top to bottom. The external forcing, if present, is included in $\mathbf{F_v} = (F_u, F_v)$ and F_h.

The velocity vector can be extended to a three dimensional (spherical) vector $\boldsymbol{v}_s = (w, v, u)^T$, where for motion on the sphere $w = 0$. If we define $\mathbf{V} = (X, Y, Z)^T$ as the velocity in Cartesian coordinates (x, y, z) then

$$\mathbf{v}_s = \mathbf{Q}\mathbf{V} \tag{4}$$

where

$$\mathbf{Q} = \begin{pmatrix} \cos\theta\cos\lambda & \cos\theta\sin\lambda & \sin\theta \\ -\sin\theta\cos\lambda & -\sin\theta\sin\lambda & \cos\theta \\ -\sin\lambda & \cos\lambda & 0 \end{pmatrix}. \tag{5}$$

It is shown in [11] that the equivalent Cartesian form of the momentum equations is

$$\frac{\partial \mathbf{V}}{\partial t} + \mathbf{Q}^T(\boldsymbol{\Lambda} + \boldsymbol{\gamma} + \boldsymbol{\Delta}) = 0, \tag{6}$$

where $\boldsymbol{\gamma}$ contains the forcing terms,

$$\mathbf{Q}^T\boldsymbol{\Delta} = \frac{(\zeta + 2\Omega z)}{a^2}\begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \tag{7}$$

and

$$\mathbf{Q}^T\boldsymbol{\Lambda} = \mathbf{P}\nabla_c(gh + \frac{\mathbf{V}\cdot\mathbf{V}}{2}). \tag{8}$$

The vorticity ζ is defined in the spherical coordinate system as $\zeta \equiv \mathbf{k}\cdot\nabla\times\mathbf{v} \equiv \mathbf{k}\cdot\nabla_c\times\mathbf{V}$ where $\mathbf{k}$ is the unit vector in the direction normal to the sphere. In terms of Cartesian derivatives the *curl* is the standard,

$$\nabla_c \times \mathbf{V} = \begin{pmatrix} \frac{\partial Z}{\partial y} - \frac{\partial Y}{\partial z} \\ \frac{\partial X}{\partial z} - \frac{\partial Z}{\partial x} \\ \frac{\partial Y}{\partial x} - \frac{\partial X}{\partial y} \end{pmatrix}. \tag{9}$$

Similarly, the conservation of mass equation in Cartesian form is

$$\frac{\partial h^*}{\partial t} + \mathbf{V}^T\mathbf{P}\nabla_c h^* + h^*\nabla_c\cdot\mathbf{V} = F_h. \tag{10}$$

The matrix P projects an arbitrary Cartesian vector onto a plane that is tangent to the sphere at the point (x, y, z).

3. DISCRETIZATION ON A SPHERICAL GRID

The spatial discretization is developed as in [11] using the spherical harmonics in a Cartesian trivariate polynomial representation. From the spatial discretization a set of ordinary differential equations are obtained and are integrated using a three time-level, explicit scheme.

3.1. Leapfrog Explicit Update

The time integration of the mass and momentum conservation equations uses the leapfrog method, a centered in time, second order approximation. At each grid point $1 \le j \le N_g$ the solution vector can be written $U_j = (X_j, Y_j, Z_j, h_j^*)$. Collecting all the terms but the time derivative on the right hand side of equations (6) and (10) and denoting the vector $\boldsymbol{U} =$

$(U_1, U_2, \ldots, U_{N_g})^T$, the system of ordinary differential equations can be written,

$$\frac{\partial \boldsymbol{U}}{\partial t} = \boldsymbol{R}(t, \boldsymbol{U}). \tag{11}$$

The three time levels of the leap frog method are denoted by superscripts $n-1, n, n+1$. If the solution vector is known at all points for time level $n-1$ and n, the solution at time level $n+1$ can be computed using the formula,

$$\boldsymbol{U}^{n+1} = \boldsymbol{U}^{n-1} + 2\Delta t \boldsymbol{R}(t^n, \boldsymbol{U}^n). \tag{12}$$

The time step Δt is held constant over the integration period. The update step is repeated, incrementing the simulation time by Δt, until the simulation ending time is exceeded.

3.2. The Local Cartesian Spectral Approximation

The spherical harmonic functions form a basis for functions defined on the surface of the sphere. In one sense, the harmonics are the natural basis for functions on the sphere since they are the complete orthonormal basis associated with the Laplace operator on the sphere [13]. For each eigenvalue α_n^m of the Laplacian,

$$\nabla^2 Y_n^m = \alpha_n^m Y_n^m. \tag{13}$$

The spherical harmonic, Y_n^m is defined with the normalized associated Legendre functions $\bar{P}_n^m(\theta)$ by

$$Y_n^m = e^{im\lambda} \bar{P}_n^m, (0 \le n, -n \le m \le n). \tag{14}$$

The normalized associated Legendre polynomials can be defined from Rodrigues' formula [9]

$$\bar{P}_n^m(\theta) = (-1)^m \left[\frac{2n+1}{2}\frac{(n-m)!}{(n+m)!}\right]^{1/2} \frac{1}{2^n n!}(\sin\theta)^m \frac{d^{m+n}}{dz^{m+n}}(z^2-1)^n \tag{15}$$

where $z = \cos\theta$ and θ is colatitude. Equations (14) and (15) are combined to give a formula for the Cartesian representation of the spherical harmonics [13].

$$Y_n^m(x, y, z) = C_n^m (x+iy)^m \frac{d^{m+n}}{dz^{m+n}}(z^2-1)^n, \tag{16}$$

where

$$C_n^m = \frac{(-1)^m}{2^n n!}\left[\frac{2n+1}{2}\frac{(n-m)!}{(n+m)!}\right]^{1/2}. \tag{17}$$

3.3. Discrete Operator Formulas

One way of calculating the derivative at a point is to fit an interpolating formula to the surrounding data and then differentiate the formula. Though this is a general method we have taken a different approach. Alternatively, one can approximate the differential operators directly by requiring that the (approximate) discrete operators act correctly on a selected set of basis functions. Given a cluster of M points p_l, $l = 0, ..., M-1$ on the surface of the sphere and a tabulation $U(p_l)$ about the point p_0, then we determine coefficients c_l such that

$$L(U)(p_0) \approx \sum_{l=0}^{M-1} c_l U(p_l). \tag{18}$$

q	Grid Points	Triangles	h_{min}(km)	h_{max}(km)	h_{ave}(km)	h_{min}/h_{max}
-	12	20	6699.0	6699.0	6699.0	1.0000
0	42	80	3482.0	3938.0	3710.0	0.8843
1	162	320	1613.0	2070.0	1901.0	0.7792
2	642	1280	761.1	1049.0	956.2	0.7255
3	2562	5120	368.4	526.3	478.8	0.7001
4	10242	20480	181.2	263.4	239.5	0.6878

Figure 1: Geometric information for icosahedral grids

The sense of the approximation ($\approx$) will be described shortly. To this end we require (18) to hold for all spherical harmonics through some degree n, including the first N harmonics,

$$L(r^n Y_n^m)(p_0) \cong \sum_{l=0}^{M-1} c_l r^n Y_n^m(p_l). \qquad (19)$$

This system is then solved for the stencil coefficients, c_l. This procedure is followed for each of the linear operators $L(U) = \frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}, \frac{\partial U}{\partial z}$, and the Laplacian, $\nabla^2 U$. This approach is general and is applicable to any distribution of points on the sphere.

The sense of the approximation in (18) is determined by the exact formulation and sense of the approximation in (19). We order the spherical harmonics Y_n^m, so that with increasing number the degree increases, (see the Appendix of [11] for a listing of the harmonics as trivariate polynomials). This formulation leads to a linear least squares problem. The problem can be stated in matrix form: find $\mathbf{c}$ which minimizes

$$||\mathbf{Hc} - \mathbf{d}||^2 \qquad (20)$$

where $\mathbf{H}$ is a $N \times M$ matrix of the lower order spherical harmonics evaluated at points of the cluster, and $\mathbf{d}$ are the analytic derivatives of hte harmonics evaluated at the center point.

The least squares problem is solved using the singular value decomposition (SVD) [4]. The choice of N and M determine the formal accuracy and smoothness of the derivative approximations. In the work reported here, we choose M points from an icosahedral spherical grid nearly symmetric about the point p_0. Table 1 gives geometric information about the different icosahedral meshes. The number of points used in the stencil will determine the efficiency of the method because evaluation of the derivatives requires a combination of values from these neighboring points. For the icosahedral grid, each point has 5 or 6 immediate neighbors and $M = 7$.

One further parameter that can be introduced is the truncation level in the SVD solution. The least squares problem has minimum norm solutions if the number of points is greater than N. We have found it advantageous to use the same truncation at all points. For $M = 7$, we truncate at six because the primary points of the icosahedron have only five neighbors.

Higher order methods are obtained by adding more points to the stencil. Values of $M = 13$ and $N = 16$ or $M = 19$ and $N = 25$ are good candidates for third order and fourth order versions of the Cartesian method with the icosahedral grid. A higher order version would incur a cost proportional to the product of M and the number of grid points, thus linear in the number of grid points. But the extra accuracy of the higher order methods may offer substantial benefit to the quality of the simulation.

3.4. Operation Counts

The operation count of the Cartesian method is low in comparison with the spectral transform method, but comparable to other grid point methods. For example, the floating point operation count per grid point, per time step in a spectral shallow water model [3] is $4.2N_{lat} + 107 \log N_{lat}$. (A floating point operation is usually defined as one multiplication plus one addition.) The grid in a spectral model is $N_{lat} \times 2N_{lat}$. For a triangular truncation of spectral coefficients with the maximum degree of the Legendre polynomials at 42 (T42), the value of N_{lat} is 64. Thus, a T42 resolution spectral model requires 713 floating point operations per grid point, per time step. The promising spectral element method [12] offers some of the advantages of the spectral method with an operation count that scales like a grid point model. For an 8x8 grid on elements, the spectral element method for the shallow water equations requires 192 operations per grid point.

The approximation of the terms of the shallow water equations using a stencil with M points requires 15 derivative approximations or $15M$ floating point operations. The total operation count for a time step is $15MN + 17N$. For $M = 7$, there are 122 floating point operations per grid point.

It is somewhat problematic to compare the operation count of different methods. For example, the method of Masuda [5] studied in [1, 2] uses a similar stencil to the Cartesian method, and consequently, should have a similar operation count. But the Masuda formulation solves the pole problem by introduction of the *scalar* stream function and velocity potential. With this formulation there are two elliptic equations to solve each timestep. These can be efficiently solved using the multi-grid method. But since this is an iterative method and may require a variable number of steps to converge, the operation count is problematic. It is best to compare computation times between methods as in [1].

The spectral method also gains an advantage when a semi-implicit time discretization is used. The resulting Helmholtz equation can be solved in spectral space in $O(N_{lat}^2)$ time. If the semi-implicit spectral model is able to take timesteps up to 10 times as long as the explicit methods then there is no advantage to be gained.

4. PARALLEL ALGORITHM FOR TIME INTEGRATION

The parallel algorithm is derived by partitioning the global nodes among processors. The partitioning of the nodes among a set of P processors then defines the division of tasks to be performed in parallel. Each processor performs the computation required to update values at the nodes in its partition. In the context of the integration of the partial differential equation, an update is performed each time step. Since the update of equations at each node depends on values at neighboring nodes, and some neighboring nodes are on another processor, information must be sent between processors each time step. A node adjacent to a node on another processor is called a *boundary node*. The set of adjacencies (or graph edges) between boundary nodes is called the *cut set*. For each edge in the cut set, communication is required. By minimizing the number of edges in the cut set a parallel algorithm is obtained with low communication cost. For an efficient parallel algorithm, the computational load must also be balanced between the processors. As a graph partitioning problem, this requirement is that the number of nodes in each partition be nearly equal.

4.1. Numerical Mesh Example and Organization of Section

The graph theoretic terminology is useful in describing the parallel algorithm and so will be developed further in this section. Figure 3 is an example of a triangular mesh partitioned between two processors, P_1 and P_2. The set of boundary nodes for P_1 is $\{3, 8, 10\}$, and for P_2,

G	Number of global nodes
L	Number of nodes on local processor
P	Number of processors
P_x	Processor x
adj	In parallel code, array of node adjacencies
end	In parallel code, an array pointing to the end of each node's adjacencies in the adj array. Node i's adjacencies start at index $end(i\text{-}1)+1$ and end at $end(i)$.

Figure 2: Notation used in section 4

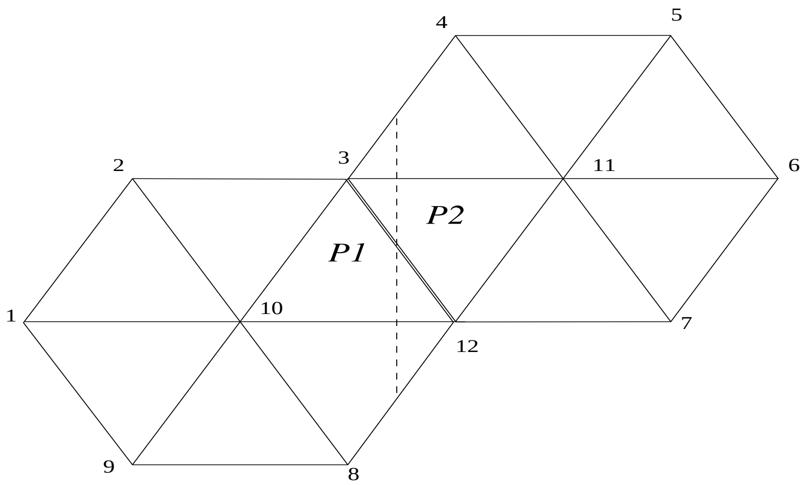

Figure 3: Sample mesh divided between two processors

$\{4, 11, 12\}$. Adjacency information is stored in an array *adj* that lists the neighbors of a given node. The global *adj* array for fig. 3 starts $\{2, 10, 9, 1, 10, 3, ...\}$ since node 1 is adjacent to nodes 2, 10, and 9, and node 2 is adjacent to nodes 1, 10, and 3. Since node 1's adjacency list ends at $adj[3]$, $end[1]= 3$.

In the parallel code, the generation of the global adjacency information as well as the partitioning is done in a master code. The master code spawns a parallel process for each partition and sends information about their set of nodes. The partitioning algorithm (see section 4.2) is part of in the master code Fig. 3 is partitioned so that each processor has an equal workload.

After receiving data from the host, each process reindexes its information into local arrays and executes initialization routines before beginning time stepping (section 4.3). During the time stepping, messages are sent with tags labeling their time step. This process ensures that no lagging processor receives information from the wrong time step.

4.2. Partitioning of the Global Mesh

We considered two ways to partition the global mesh: first minimizing the number of sends, and, second, minimizing the length of the messages. Minimizing the number of sends is the more efficient method because the large bandwidths of modern parallel networks make it possible to send large packages of data without drastically reducing parallel efficiency. Both partitioning algorithms are discussed below.

By minimizing the cut set of the partitions, we minimize the number of neighbor nodes that need to be sent between processors. To obtain a minimum cut set while maintaining a load balance among processors, the spectral bisection method can be used [6, 8]. This method has become a standard for irregularly meshed regions. It is general, but can also be expensive to apply. The recursive spectral bisection algorithm of [8] uses a modified Lanczos algorithm to compute an eigenvalue of the "mesh Laplacian" matrix to split the mesh in two. Further divisions of each of these parts are obtained by recursive application of the processes. For the icosahedral grid there is enough regularity to suggest reasonable partitions without the application of the spectral bisection algorithm. We leave to future work the application of these methods on irregular structured meshes.

To minimize the number of sends, a partition was developed in which processors take nodes on specific latitudes, or strips around the globe. This results in a partition such that no one processor should do more than two sends: one send to the processor to the north and one send to the processor to the south. More sends may be required if a set of nodes does not stretch all the way around the globe, but, in this case, there are probably too many processors for good efficiency. To form this partition the global nodes are reordered from minimum to maximum latitude, and then within each latitude, from minimum to maximum longitude.

After reordering, the nodes are divided up as equally as possible between processors (so that each processor has the same amount of work to do each timestep). Processor n is given node

$$(\frac{(n-1)\times G}{P})+1 \tag{21}$$

through

$$\frac{n\times G}{P} \tag{22}$$

Any remainders are divided so that the difference between the number of nodes per processor is no greater than one.

```
bnode_count=1             % count of all nodes sent
loop i=1,L
  loop x=end(i-1)+1,end(i)
    if (adj(x) is not a local node)
      sp=which_processor(x)
      send latitude, longitude, node index, to proc. sp
      smat(1,bnode_count)=global node index of x
      smat(2,bnode_count)=sp
      bnode_count=bnode_count+1
    endif
  endloop
endloop
```

Figure 4: *Initialize* $-$ *send* algorithm

4.3. Indexing and Data Storage

Every processor starts with a list of its local nodes' longitudes and latitudes along with the global node number index array. They also receive from the host a local adjacency array and an array containing pointers to the position in the global index array where each processors' nodes end. Using its local adjacency array, the processor is able to find the local nodes it must send to other processors. It determines which processor to send each node to (the $which_processor$ function called in Figure 4) by accessing its array pointing to processor positions within the global node array. The processor then sends the information of these nodes (at this point just their longitudes and latitudes) to the appropriate processor. While this send is occurring, the sending processor is writing a matrix, *smat* that contains a global node number and destination processor number for each boundary node. The matrix is later reordered so that the nodes sent to one processor are grouped together. This matrix is accessed during the time steps for efficient sending of data between processors (see also Section 4.4): the processor reads straight through its reordered *smat* matrix and packs the information it needs to send to one processor into a buffer and then sends it using only one send call. It then moves on to the next processor and repeats the process.

An important value generated after this initial sending and receiving is the number of boundary nodes that a processor must receive from neighbors. This is not necessarily the same number as the number of sends, although both of these numbers are based on *bnode_count* from Figure 4. The receive number is found by finding and removing multiple cases of a single global node being received by that processor in the initial send (this will happen if that global node is adjacent to more than one of the processor's nodes) and subtracting from the number of nodes originally received. A similar process, based on finding and removing nodes sent to a single processor more than once, is done to find the number of boundary nodes each processor must send each time step. The count of nodes to receive is the number of unique adjacent nodes that a processor needs but does not have locally. This number must be added to many of the loops in the sequential code. For example, a processor must unpack latitude and longitude values and have x,y, and z values for its local nodes as well as its non-local boundary nodes to find initial conditions, calculate the stencil coefficients and do updates.

```
loop i=1,send_count
  x=smat(1,i)    %global index of node to send
  p=smat(2,i)    %proc. to send to
  j=g2l(x)       %local index of node to send
  pack information into buffer
  if(i=send_count)
    send buffer
    exit loop
  endif
  if(p not equal to send(2,i+1))  %all nodes to p packed
    send buffer
  endif
endloop
```

Figure 5: $Send - data$ algorithm

```
count=0
while(count<recv_count)
    receive buffer
    unpack x nodes
    count=count+x
endwhile
```

Figure 6: $Receive - data$ algorithm

4.4. Overlap Updates

Sending boundary nodes to other processors is discussed above in section 4.2. Pseudo-code for the send algorithm is given in Figure 5.

Receiving nodes is a bit more difficult since the processor does not have information about which outside processor will send boundary updates. It is possible, however, to find the number of incoming nodes in each buffer. The total number of nodes that will be received can then be used to end the receive loop. Pseudo-code for a receive algorithm is shown in Figure 6.

5. PERFORMANCE RESULTS FOR A STEADY, ZONAL FLOW TEST

Test case 2 is a steady, non-linear zonal flow as proposed in [20]. It tests the ability of the code to maintain a steady state solution independent of the grid orientation and gives a good idea of the accuracy of the methods. The local spherical harmonic approximations for the derivative operators are able to capture the steady state solution well. Figures 7 and 8 show the error, as a function of time, in the velocity (using the relative RMS error with the exact steady solution) and the RMS error in the height field, respectively. The $q = 2$ icosahedral mesh used an integration time step of 1200 seconds for the 5 day (120 hour) simulations while the $q = 3, 4$ meshs used a 600 second timestep.

A contour plot of the absolute geopotential error is given in figure 9. The error is measured

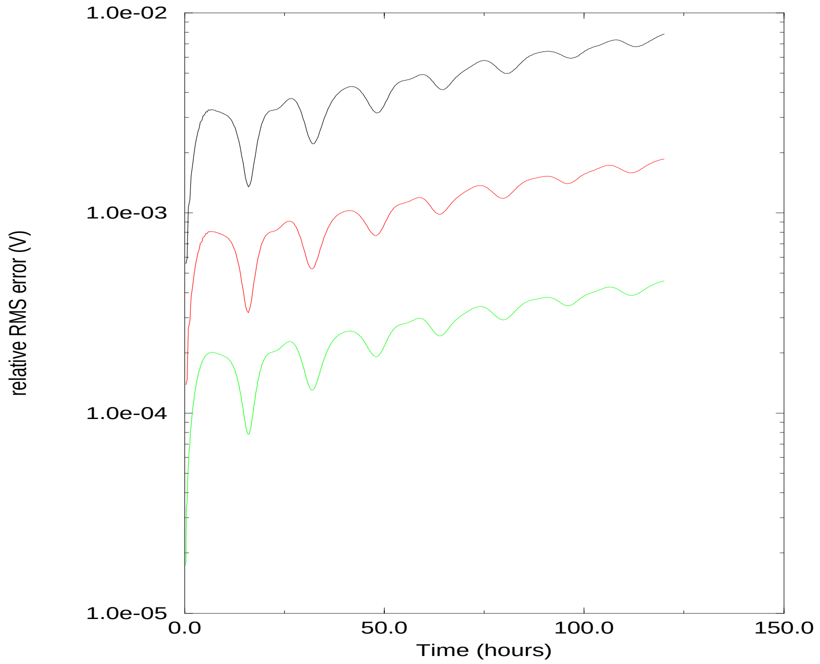

Figure 7: Relative RMS Error in Velocity, Test Case 2, $q = 2, 3, 4$.

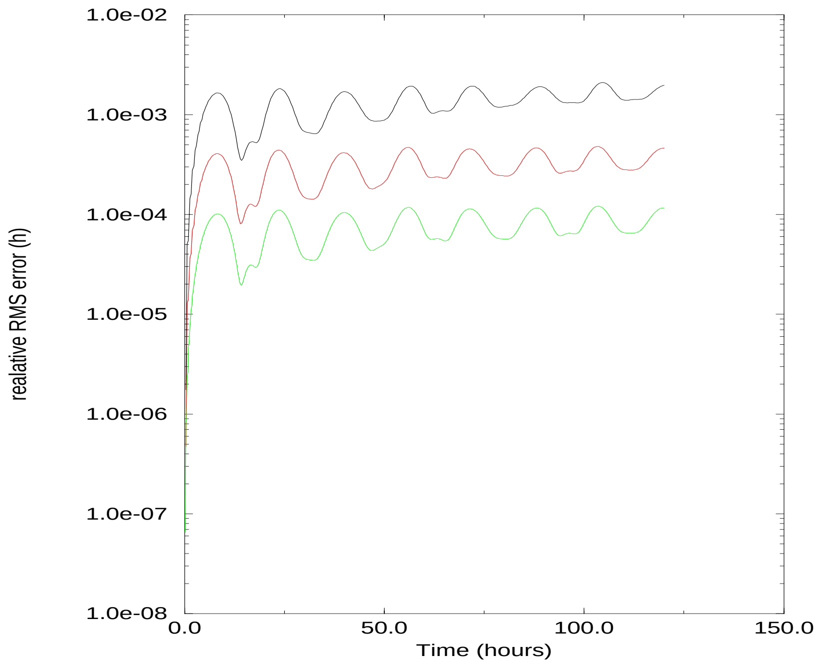

Figure 8: Relative RMS Error in Height, Test Case 2, $q = 2, 3, 4$.

at 5 days. Clearly evident are the base points of the icosahedral grid where the difference stencil involves 6 rather than 7 points.

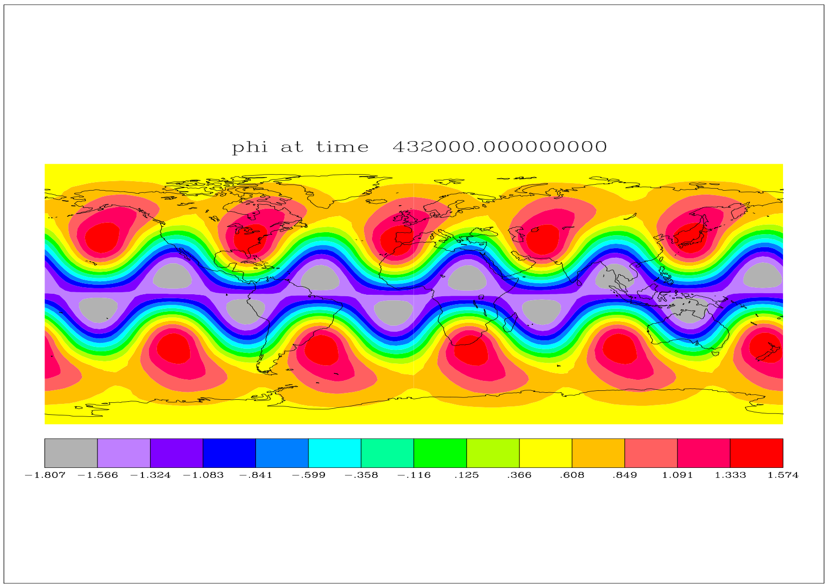

Figure 9: Error in Geopotential at 5 days, Test Case 2, $q = 4$.

The five day integration is timed for several different mesh sizes and with different numbers of processors involved in the computation. The time for a given time step may be divided into computational time (t_{comp}) and communication time (t_{comm}). The computation time is related to the number of grid points assigned to a given processor. The communication time is related to the number of messages sent and the size of the messages sent between processors for the update of boundary points.

Times were measured on the Intel Paragon XPS35 at Oak Ridge National Laboratory in the Center for Computational Sciences, a mesh connected massively parallel processor consisting of 512 Intel i860 nodes. The parallel performance is reported as speedup $S_p \equiv \frac{T_1}{T_p}$, ratio of the total time to execute on one processor to total time using p processors. Since memory constraints prevented running some of the test cases on small numbers of nodes, we designated the lowest number of processors run on as T_1. Also the parallel efficiency, $E_p \equiv \frac{S_p}{p}$, is reported. The total time on p processors, T_p, reflects both communication and computation time along with other miscelaneous overhead. Time is measured using a synchronization point (barrier) after the grids and stencil coefficients are initialized. It reflects the total time spent in time stepping. The time spent in computation represents the accumulation of times for each call to the time update routine. The time in communication is obtained similarly, by starting a timer before calling the routine to update the boundary nodes. The system clock used on the Intel Paragon was the *gettimeofday* utility. Times are reported in seconds.

Figure 11 contains information that affects the times in Figure 10. The number of nodes on the local processor is given by n_{local}. N_s is the average number of nodes sent per processor per time step, and P_m is the maximum number of processors that any one processor was adjacent to (the maximum number of sends that any processor must make).

The code was developed with the Parallel Virtual Machine (PVM) message passing constructs. This allowed code development on a network of workstations and only the performance

Processors	t_{comp}	t_{comm}	Total	S_p	E_p	$\frac{t_{comp}}{n_{local} \times n_{steps}}$
			Icosahedral Grid $q = 2$			
1	14.9	0.4	29.1	1	1.00	6.4e-5
2	7.6	22.4	38.3	0.76	0.38	6.6e-5
4	4.0	42.1	51.7	0.56	0.14	6.9e-5
8	2.1	42.9	49.4	0.59	0.07	7.2e-5
16	1.2	44.0	49.0	0.59	0.04	8.1e-5
32	0.7	34.0	38.1	0.76	0.02	9.3e-5
			Icosahedral Grid $q = 3$			
1	118.2	0.9	212.0	1	1.00	6.4e-5
2	59.7	102.7	209.0	1.01	0.51	6.5e-5
4	30.1	174.4	230.8	0.92	0.23	6.5e-5
8	15.4	168.1	199.7	1.06	0.13	6.7e-5
16	8.0	164.9	184.9	1.15	0.07	6.9e-5
32	4.3	168.3	181.4	1.17	0.04	7.4e-5
64	2.4	148.0	157.9	1.34	0.02	8.1e-5
			Icosahedral Grid $q = 4$			
4	120.0	468.9	675.4	4	1.00	6.5e-5
8	60.5	397.7	504.4	5.36	0.67	6.6e-5
16	30.4	360.4	417.2	6.48	0.41	6.6e-5
32	15.8	342.1	374.2	7.22	0.23	6.8e-5
64	8.2	335.5	355.1	7.61	0.12	7.1e-5
128	4.4	267.9	281.2	9.61	0.08	7.5e-5

Figure 10: Parallel Performance on the Intel Paragon

Processors	n_{local}	N_s	P_m
	Icosahedral Grid $q = 2$		
1	642	0	0
2	321	41	1
4	161	63	2
8	81	70	2
16	41	71	4
32	21	57	7
	Icosahedral Grid $q = 3$		
1	2562	0	0
2	1281	81	1
4	641	123	2
8	321	135	2
16	161	139	2
32	81	140	4
64	41	111	7
	Icosahedral Grid $q = 4$		
4	2562	248	2
8	1281	265	2
16	641	274	2
32	321	278	2
64	161	278	4
128	81	213	7

Figure 11: Parallel Performance on the Intel Paragon

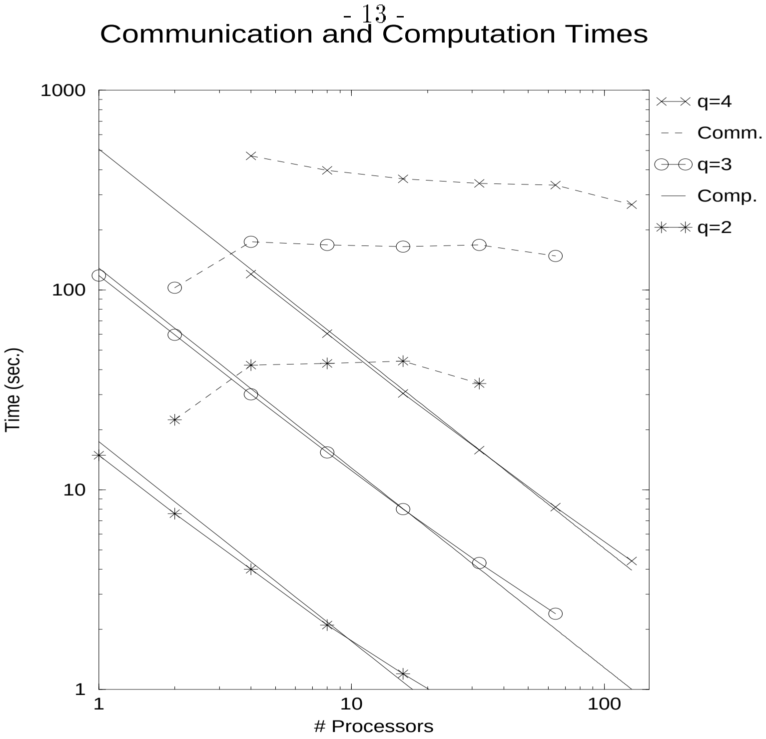

Figure 12: Computation and Communication Times on the Intel Paragon

runs were made on the Paragon. Because PVM is not highly optimized for the Paragon, the communication times reported in Figure 10 are excessive. Nevertheless, they show the correct trends: constant cost with increasing numbers of processors and increased cost with increased resolution. The computational times scale very well as expected. Other partitioning strategies and optimized message passing routines could improve the results presented. These improvements are necessary before extending the model to a more realistic 3-D baroclinic model.

6. References

[1] Ross Heikes and David A. Randall. Numerical integration of the shallow-water equations on a twisted icosahedral grid. part i. *Mon. Wea. Rev.*, 123:1862–1880, 1995.

[2] Ross Heikes and David A. Randall. Numerical integration of the shallow-water equations on a twisted icosahedral grid. part ii. *Mon. Wea. Rev.*, 123:1881–1887, 1995.

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

[4] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

[5] Y. Masuda and H. Ohnishi. An integration scheme of the primitive equation model with an icosahedral-hexagonal grid system and its application to the shallow water equations. In *Short- and Medium-Range Numerical Weather Prediction*, pages 317–326, 1986.

[6] A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[7] R. Sadourny, A. Arakawa, and Y. Mintz. Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere. *Mon. Wea. Rev.*, 96:351–356, 1968.

[8] H. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[9] I. A. Stegun. Legendre functions. In M. Abramowitz and I. A. Stegun, editors, *Handbook of Mathematical Functions*, chapter 8, pages 332–353. Dover Publications, New York, 1972.

[10] P.N. Swarztrauber, D.L. Williamson, and J.B. Drake. Spectral transform methods for solving the shallow-water equations on the sphere. *Mon. Wea. Rev.*, 124(4):730–744, 1996.

[11] P.N. Swarztrauber, D.L. Williamson, and J.B. Drake. The cartesian method for solving of PDE's in spherical geometry. *DAO*, to appear.

[12] Mark Taylor, Joseph Tribbia, and Mohamed Iskandarani. The spectral element method for the shallow water equations on the sphere. *Mon. Wea. Rev.*, 1996.

[13] A.N. Tikhonov and A.A. Samarskii. *Equations of Mathematical Physics.* Dover Publications, New York, 1963.

[14] D. L. Williamson. Integration of the barotropic vorticity equation on a spherical geodesic grid. *Tellus*, 20:642–653, 1968.

[15] D. L. Williamson. Integration of the primitive barotropic model over a spherical geodesic grid. *Mon. Wea. Rev.*, 98:512–520, 1969.

[16] D. L. Williamson. Numerical integration of fluid flow over triangular grids. *Mon. Wea. Rev.*, 97:885–895, 1969.

[17] D. L. Williamson. A comparison of first- and second-order difference approximations over a spherical geodesic grid. *J. Comp. Phys.*, 7:301–309, 1971.

[18] D. L. Williamson. *Numerical Methods Used in Atmospheric Models*, chapter 2, pages 51–120. GARP Pub. Ser. No. 17. JOC, WMO, Geneva, Switzerland, 1979.

[19] D. L. Williamson and G. L. Browning. Comparison of grids and difference approximations for numerical weather prediction over a sphere. *J. Appl. Meteor.*, 12:264–274, 1973.

[20] D. L. Williamson, J.B. Drake, J.J. Hack, Rudiger Jakob, and P.N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations on the sphere. *J. Comp. Phys.*, pages 211–224, 1992.

ORNL/TM-13304

INTERNAL DISTRIBUTION

1. E. F. D'Azevedo
2–3. J. B. Drake
4. M. R. Leuze
5. C. E. Oliver
6–10. S. A. Raby
11. B. D. Semeraro
12. R. F. Sincovec
13. P. H. Worley
14. K-25 Applied Tech. Library
15. Y-12 Technical Library
16. Laboratory Records - RC
17–18. Laboratory Records Dept.
19. Central Research Library
20. ORNL Patent Office

EXTERNAL DISTRIBUTION

21. David C. Bader, Atmospheric and Climate Research Division, Office of Health and Environmental Research, Office of Energy Research, ER-76, U.S. Department of Energy, Washington, DC 20585

22. Dominique Bennett, CERFACS, 42 Avenue Gustave Coriolis, 31057 Toulouse Cedex, FRANCE

23. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307

24. Ray Cline, Sandia National Laboratories, Livermore, CA 94550

25. Jean Coté, RPN, 2121 Transcanada Highway, Suite 508, Dorval, Quebec H9P 1J3, CANADA

26. William Dannevik, Lawrence Livermore National Laboratory, P. O. Box 808, L-16, Livermore, CA 94550

27. John Dukowicz, Los Alamos National Laboratory, Group T-3, Los Alamos, NM 87545

28. Ian Foster, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

29. Dr. Rhys Francis, Div. of Information Technology, CSIRO, 723 Swanston Street, Carlton, Vic. 3053, AUSTRALIA

30. Paul O. Frederickson, ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545

31. John Gustafson, 236 Wilhelm, Ames Laboratory, Iowa State University, Ames, IA 50011

32. Phil Gresho, Lawrence Livermore National Laboratory, L-262, P. O. Box 808, Livermore, CA 94550

33. James J. Hack, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

34. Michael Henderson, Los Alamos National Laboratory, Group T-3, Los Alamos, NM 87545
35. Lennart Johnsson, Thinking Machines Inc., 245 First Street, Cambridge, MA 02142-1214
36. J.R. Jump, ECE Dept., Rice University, P.O. Box 1892, Houston, TX 77251
37. Tom Kitchens, ER-7, Applied Mathematical Sciences, Scientific Computing Staff, Office of Energy Research, Office G-437 Germantown, Washington, DC 20585
38. Rich Loft, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
39. Michael C. MacCracken, Lawrence Livermore National Laboratory, L-262, P. O. Box 808, Livermore, CA 94550
40. Norman D. Malmuth, Science Center, Rockwell International Corporation, 1049 Camino Dos Rios, P.O. Box 1085, Thousand Oaks, CA 91358
41. Robert Malone, C-DO/ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545
42. Len Margolin, Los Alamos National Laboratory, Los Alamos, NM 87545
43. Hal Marshall Laboratory for Scientific Computation, Rm. 271 Cooley Bld., University of Michigan, Ann Arbor, MI 48109-2104
44. Frank McCabe, Department of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2BZ, ENGLAND
45. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550
46. Prof. Lisa Meeden, Swarthmore College, Dept. Computer Science, 500 College Ave., Swarthmore, PA 19081
47. John G. Michalakes, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
48. Richard Pelz, Dept. of Mechanical and Aerospace Engineering, Rutgers University, Piscataway, NJ 08855-0909
49. Andrew Priestley, Institute for Computational Fluid Dynamics, Reading University, Reading RG6 2AX, ENGLAND
50. Lee Riedinger, Director, The Science Alliance Program, University of Tennessee, Knoxville, TN 37996
51. Richard Smith, Los Alamos National Laboratory, Group T-3, Mail Stop B2316, Los Alamos, NM 87545
52. Peter Smolarkiewicz, National Center for Atmospheric Research, MMM Group, P. O. Box 3000, Boulder, CO 80307
53. Jurgen Steppeler, DWD, Frankfurterstr 135, 6050 Offenbach, WEST GERMANY
54. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

55. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

56. Wei Pai Tang, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

57. David L. Williamson, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

58. Samuel Yee, Air Force Geophysics Lab, Department LYP, Hancom AFB, Bedford, MA 01731

59. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600

60–69. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831