CONCURRENCY: PRACTICE & EXPERIENCE, **4**(4), 269-291 (JUNE 1992)

PARALLELIZING THE SPECTRAL TRANSFORM METHOD

PATRICK H. WORLEY* AND JOHN B. DRAKE*

Abstract. The spectral transform method is a standard numerical technique used to solve partial differential equations on the sphere in global climate modeling. In particular, it is used in CCM1 and CCM2, the Community Climate Models developed at the National Center for Atmospheric Research. This paper describes initial experiences in parallelizing a program that uses the spectral transform method to solve the nonlinear shallow water equations on the sphere, showing that an efficient implementation is possible on the Intel iPSC/860. The use of PICL, a portable instrumented communication library, and ParaGraph, a performance visualization tool, in tuning the implementation is also described.

The Legendre transform and the Fourier transform comprise the computational kernel of the spectral transform method. This paper is a case study of parallelizing the Legendre transform. For many problem sizes and numbers of processors, the spectral transform method can be parallelized efficiently by parallelizing only the Legendre transform.

Key words. hypercube multiprocessors, parallel programming tools, parallel Legendre transform, performance visualization, shallow water equations, spectral transform method

AMS(MOS) subject classifications. primary 65W05; secondary 65M60, 68M20

1. Introduction. In order to understand the effects of the increasing atmospheric concentrations of greenhouse gases, numerical simulations covering hundreds of years using high resolution models are needed. This is not computationally feasible using current climate models and computers. Developing the ability to adequately model the climate will require advances in both numerical algorithms and model physics, and massively parallel multiprocessors will be necessary to run such simulations in a reasonable amount of time [6]. In this regard, we are investigating the suitability for parallel computation of the numerical techniques currently used in climate models.

The spectral transform method [8], [26], [28] is a standard numerical technique for solving partial differential equations on the sphere in weather and climate modeling [1], [2], [27], [36]. In particular, it is used in CCM1, the Community Climate Model developed at the National Center for Atmospheric Research (NCAR) [36]. There are both numerical and algorithmic issues to be considered before using the spectral transform method in climate models with much finer resolutions. In the work described here, we restrict ourselves to investigating how efficiently the spectral transform method can be parallelized on distributed-memory MIMD multiprocessors, and how this performance is likely to scale as both the problem size and the number of processors increase. Some theoretical work has been done in this area [9], [14], [23], but empirical studies have been limited to shared-memory multiprocessors with small numbers of processors and to massively parallel SIMD multiprocessors [5], [19], [30], [31]. In this paper, we describe

* Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, Tennessee 37831-6367 (`worleyph@ornl.gov` or `bbd@ornl.gov`). This research was supported by the Atmospheric and Climate Research Division and the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

our initial experiences in parallelizing a serial program that uses the spectral transform method to solve the nonlinear shallow water equations on the sphere, showing that an efficient implementation is possible on the Intel iPSC/860. The shallow water equations constitute a simplified atmospheric-like fluid prediction model that has been used to benchmark a number of algorithms and machines [3], [18], [31], [33], [35]. This particular code is similar to a computational kernel of CCM2, the successor to CCM1 currently being developed at NCAR, and was provided courtesy of J. J. Hack at NCAR.

Throughout this paper, we emphasize our use of PICL [11], [12], [13], [39], a portable instrumented communication library, and ParaGraph [15], [16], [39], a performance visualization tool, when developing and tuning the parallel implementation, and when discovering previously unknown performance quirks of the iPSC/860. These tools enable us to visualize the algorithm's performance and simplify the modeling of the performance of the algorithm. This is very important because we are parallelizing a complex code and using a new multiprocessor, making it difficult to develop reliable performance models *a priori*.

2. Shallow water equations on a sphere. The shallow water equations on the sphere consist of equations for the conservation of momentum and the conservation of mass. With $\mathbf{V}$ denoting the horizontal velocity, $\mathbf{V} = \mathbf{i}u + \mathbf{j}v$, and Φ denoting the geopotential, the horizontal momentum and mass continuity equations can be written as [32]

$$\frac{d\mathbf{V}}{dt} = -f\mathbf{k} \times \mathbf{V} - \nabla\Phi \tag{1}$$

and

$$\frac{d\Phi}{dt} = -\Phi\nabla \cdot \mathbf{V}, \tag{2}$$

where the substantial derivative is given by

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

and f is the Coriolis term.

For the application of the spectral transform method a different form of the equations will be used. It is convenient to redefine the horizontal velocity components as $U \equiv u\cos\phi$ and $V \equiv v\cos\phi$, where ϕ is latitude, to avoid the singularity in velocity at the poles. The horizontal momentum can also be specified in terms of relative vorticity,

$$\zeta \equiv \mathbf{k} \cdot (\nabla \times \mathbf{V}), \tag{4}$$

and horizontal divergence,

$$\delta \equiv \nabla \cdot \mathbf{V}. \tag{5}$$

Using these new variables, the horizontal momentum equation (1) can be rewritten as [38]

$$\frac{\partial \eta}{\partial t} = -\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial \lambda}(U\eta) - \frac{1}{a}\frac{\partial}{\partial \mu}(V\eta) \tag{6}$$

$$\frac{\partial \delta}{\partial t} = \frac{1}{a(1-\mu^2)}\frac{\partial}{\partial \lambda}(V\eta) - \frac{1}{a}\frac{\partial}{\partial \mu}(U\eta) - \nabla^2(\Phi + \frac{U^2+V^2}{2(1-\mu^2)}), \tag{7}$$

where

$$\eta \equiv \zeta + f = \frac{1}{a(1-\mu^2)}\frac{\partial V}{\partial \lambda} - \frac{1}{a}\frac{\partial U}{\partial \mu} + f, \tag{8}$$

$$\delta = \frac{1}{a(1-\mu^2)}\frac{\partial U}{\partial \lambda} + \frac{1}{a}\frac{\partial V}{\partial \mu}, \tag{9}$$

a is the radius of the sphere, and the independent variables λ and μ denote longitude and $\sin\phi$, respectively. The mass continuity equation can also be rewritten as

$$\frac{\partial \Phi}{\partial t} = -\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial \lambda}(U\Phi) - \frac{1}{a}\frac{\partial}{\partial \mu}(V\Phi) - \bar{\Phi}\delta, \tag{10}$$

where Φ is now a perturbation from a constant average geopotential $\bar{\Phi}$.

The velocity vector can be related to a scalar stream function ψ and a scalar velocity potential χ by

$$\mathbf{V} = \mathbf{k} \times \nabla\psi + \nabla\chi. \tag{11}$$

These are related to vorticity and divergence, through application of the curl and divergence operators, by

$$\eta = \nabla^2\psi + f \tag{12}$$

and

$$\delta = \nabla^2\chi. \tag{13}$$

In the spectral transform method, we solve equations (6), (7), and (10) for η, δ, and Φ, respectively, and use equations (11)-(13) to calculate U and V.

3. Spectral transform method. The spectral transform method approximates the state variables of the shallow water equations in both physical space, where nonlinear terms are calculated, and spectral space, where the differential terms of the equations are evaluated. Representations of the state variables in physical space are the point values on a tensor-product grid whose axes represent longitude and latitude. Representations of the state variables in spectral space are the coefficients of a truncated expansion in terms of complex exponentials and associated Legendre polynomials,

$$\xi(\lambda,\mu) = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) e^{i \cdot m\lambda}, \tag{14}$$

where $P_n^m(\mu)$ is the (normalized) associated Legendre function [29] and $i = \sqrt{-1}$. The spectral coefficients are then determined by the equations

$$\xi_n^m = \int_{-1}^{1} \left[\frac{1}{2\pi} \int_0^{2\pi} \xi(\lambda,\mu) e^{-i \cdot m\lambda} d\lambda \right] P_n^m(\mu) d\mu \tag{15}$$

$$\equiv \int_{-1}^{1} \xi^m(\mu) P_n^m(\mu) d\mu \tag{16}$$

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

The tensor-product grid in physical space is rectangular with I grid lines evenly spaced along the longitude axis and J grid lines along the latitude axis placed at the Gaussian quadrature points in $[-1,1]$. To evaluate the spectral coefficients, a fast Fourier transform (FFT) is used to calculate $\xi^m(\mu)$ from $\xi(\lambda,\mu)$ for any given μ. The Legendre transform (16) is then approximated using a Gaussian quadrature rule. Denoting the Gauss points in $[-1,1]$ by μ_j and the Gauss weights by w_j,

$$\xi_n^m = \sum_{j=1}^{J} \xi^m(\mu_j) P_n^m(\mu_j) w_j. \tag{17}$$

(For simplicity, we will henceforth refer to (17) as the forward Legendre transform.) The point values are recovered from the spectral coefficients by computing

$$\xi^m(\mu) = \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) \tag{18}$$

for each m and μ (which we will refer to as the inverse Legendre transform), followed by FFTs to calculate $\xi(\lambda,\mu)$. To allow exact, unaliased transforms of quadratic terms the following relations are sufficient: $J \geq (3M+1)/2$, $I = 2J$, and $N(m) = M$ [28]. Using $N(m) = M$ is called a triangular truncation because the (m,n) indices of the spectral coefficients make up a triangular grid.

The algorithm for calculating the time update of the state variables begins by transforming the current approximation to the solution from spectral space to physical space, evaluating the nonlinear terms in the differential equations, and transforming these terms back to spectral space. The spatial derivatives in the differential equations are simply scalar multiplications in spectral space, and the time update is completed using a standard finite difference approximation to the time derivatives. See [38] for a more complete description of the algorithm.

Assume that a triangular truncation is used. Tranforming the state variables from spectral space to physical space requires calculating an inverse Legendre transform for each Fourier mode, $\Theta(J \cdot M^2)$ operations,[1] followed by an FFT at each latitude, $\Theta(J \cdot I \log I)$ operations, for each state variable. Computing the nonlinear terms on the physical grid requires $\Theta(J \cdot I)$ operations. Transforming these quantities back to spectral space requires calculating an FFT at each latitude, $\Theta(J \cdot I \log I)$ operations, followed by a forward Legendre transform for each Fourier mode, $\Theta(J \cdot M^2)$ operations, for each term. The complexity of the time update of a state variable is $\Theta(1)$ for each of the spectral coefficients, for a total complexity of $\Theta(M^2)$. Thus, the total complexity for a single timestep is $\Theta(J \cdot I \log I) + \Theta(J \cdot M^2)$. Note that if $J \approx (3M+1)/2$ and $I = 2J$, then the complexity becomes $\Theta(M^3)$, in agreement with the complexity reported in [3], and the Legendre transforms are the most time consuming portion of the calculation for large M.

4. Example problems. A triangular truncation of the spherical harmonic expansion is one of the three options, along with rhomboidal and trapezoidal truncations, found in CCM1 [36, pp. 41–45]. For a triangular truncation, it is common practice to choose J and I to be the smallest values satisfying $J \geq (3M+1)/2$ and $I = 2J$ for which I is a convenient number of longitudes over which to calculate an FFT. For such a model, the value of M is sufficient to characterize the resolution of the model, and is commonly referred to as TM.

It is unclear what resolution future climate models will use. The initial design point for CCM2 is T42 [37, p. 3]. A resolution of T213 is currently used for the European operational weather model [24]. In this paper we present results for T21, T42, T85, T169, and T340. These examples indicate how the parallel implementation scales as a function of problem size and number of processors. While this scaling analysis is directly applicable only to solving the shallow water equations, we describe some of the implications for a parallel implementation of CCM2 in §11. The T85 example, which is the largest of our examples that will fit in the memory of a single node on the Intel iPSC/860, will be used in examples detailing the performance of the algorithm.

As a minimal test of the correctness of our parallel implementations, initial data is used that represent a steady state zonal flow [34]. Note that the choice of initial data does not influence the execution time of the method once the other problem parameters have been fixed.

5. SSWMSB. `SSWMSB` is a FORTRAN program written by J. J. Hack at NCAR. It is a serial implementation of the algorithm described in §3 that was designed to solve the shallow water equations efficiently on scalar and vector computers. In particular, the associated Legendre polynomials $P_n^m(\mu)$ are precalculated for each spectral coefficient and for each latitude in the northern hemisphere, eliminating redundant calculations.

[1] In this discussion, the expression $\Theta(x)$ denotes a quantity whose leading order term is proportional to x. See Horowitz and Sahni [20, p. 31] for a formal definition of Θ notation.

The program also exploits symmetry in the approximation to the Legendre transform between northern and southern latitudes, effectively halving the operation count over the naive implementation.

We first ported `SSWMSB` to a Sun workstation, and then to a single processor on the IPSC/860. It is a reflection on the high quality of `SSWMSB` that this was a relatively simple task, requiring primarily the removal of calls to graphics routines and associated logic that are used in the post-processing of the results.

6. Intel iPSC/860. The Intel iPSC/860 is a state-of-the-art distributed-memory MIMD multiprocessor with up to 128 nodes. Each node is either an Intel i860 microprocessor with 8 megabytes of local memory or an Intel 80386 microprocessor with 1, 4, 8, or 16 megabytes of local memory. The nodes are physically interconnected by a hypercube network, but the communication hardware effectively emulates a fully connected network. See [7] and [22] for more details on the iPSC/860 architecture. The iPSC/860 is also known as the Touchstone Gamma prototype, since it represents an early phase of Intel's Touchstone project. Future Touchstone prototypes will scale up to 2048 nodes [25], and are candidate architectures for the next generation of global climate models.

The iPSC/860 at Oak Ridge National Laboratory, which was used for our performance measurements, has 128 nodes, each of which is a 40 MHz i860 processor with a peak execution rate of 80 Mflops for 32-bit floating point calculations. Thus, the aggregate peak performance is 10 Gflops. The peak execution rates are based on conditions that are difficult to realize (and sustain) in practice even for assembly-coded routines [17]. For most compiled FORTRAN or C programs, the performance is significantly less, ranging between 2 and 8 Mflops on a single node. All performance results reported here used standard FORTRAN 77 source code compiled using the Portland Group compiler with -O1 optimization. At the time of these experiments, invoking higher levels of optimzation did not consistently improve performance, and sometimes introduced errors.

7. PICL and ParaGraph. The Portable Instrumented Communication Library (PICL) is a subroutine library that implements a generic message-passing interface on a variety of multiprocessors [11], [12], [13], [39]. Programs written using PICL routines instead of the native commands for interprocessor communication are portable in the sense that the source can be compiled on any machine on which the library has been implemented. Correct execution is also a function of the parameter values passed to the routines, but standard error trapping is used to inform the user when a parameter value is not legal on a particular machine. Programs written using PICL routines will also produce timestamped trace data on interprocessor communication, processor busy/idle times, and simple user-defined events if a few additional statements are added to the source code.

ParaGraph is a graphical display system for visualizing the behavior of parallel algorithms on message-passing multiprocessor architectures [15], [39]. It takes as input execution trace data provided by PICL and replays it pictorially, thus providing a dynamic depiction of the behavior of the parallel program. ParaGraph provides over 20 distinct visual perspectives from which to view the same performance data, enabling the viewer to gain insights that might be missed by any single view.

We use PICL subroutine calls to implement interprocessor communication when parallelizing `SSWMSB`. This allows us to port the resulting code to similar message-passing multiprocessors, like the NCUBE family of hypercubes, with only a slight loss of performance. For example, when tracing is not enabled, the overhead of using PICL instead of the native commands is less than 1% for our implementations of `SSWMSB` on the iPSC/860. More importantly, the use of PICL allows us to collect trace data for analyzing and tuning the parallel implementation. This has been crucial in our efforts to produce an efficient parallel program. ParaGraph is an invaluable tool in the analysis since the amount of trace data collected in a typical run is very large. We use ParaGraph displays in the subsequent sections to illustrate the behavior of our parallel implementations. Note that ParaGraph displays are most detailed (and informative) when viewed in color, but monochrome displays are also supported, and these monochrome displays are used in this paper.

8. Parallel Implementation. Our initial parallel implementation of `SSWMSB` is based on equipartitioning both the physical and spectral domains, assigning data and work to processors consistent with the partitions. The physical space is partitioned into slabs, so that each processor is responsible for a specified range of latitudes. To exploit the symmetry in the Legendre transform, each processor is assigned two slabs, one for latitudes in the northern hemisphere and one for the "reflected" latitudes in the southern hemisphere. Thus, at most $J/2$ processors can be used, and each processor is responsible for at least two latitude lines. Using this partition of the physical space, all partitions of the spectral space that assign equal numbers of spectral coefficients to each process are equally good. The particular partition of the spectral space that we employ will be convenient in the next stage of our parallel implementation, partitioning over longitude in physical space. See [38] for the details.

Using these domain decompositions, each processor contains all the information needed to perform the FFTs for its latitudes, so an optimized serial FFT can be used for both forward and inverse transforms. The evaluation of a Legendre transform for a given line of constant longitude requires information from all grid points on that line. The communication required for this part of the algorithm is similar to that of the N-body problem [10], and can be performed efficiently using ring algorithms.

When transforming from spectral space to physical space, the spectral coefficients are moved around a "ring" of processors. At each step a processor sends its current block of data to the processor on its left (in the ring topology), uses the data to calculate contributions to the physical quantities it is responsible for, then receives a new block of data from the processor on the right. After P steps, the calculation is complete and each block of spectral coefficients has been returned to its source processor. The concurrent send (receive) volume of data is approximately $M^2/2$ complex numbers for each state variable, and much of the computation and communication is overlapped because the data is forwarded before it is used. We will refer to this as the *first* ring algorithm, or ring algorithm 1.

When transforming from physical to spectral space, the spectral values being calculated are circulated around the ring. Each processor calculates its contribution to the block of coefficients it knows that it will receive next, receives that block from the "right", adds in its contribution, sends the updated block to the "left", and begins the calculation for the next block. As before, the computation and communication are overlapped and the concurrent send (receive) volume is $M^2/2$ complex values per state variable. We will refer to this as the *second* ring algorithm, or ring algorithm 2.

As part of this parallel implementation, we also modified `SSWMSB` to combine the evaluations of all forward (inverse) transforms during each timestep and to combine the state vectors into a single array of one larger dimension. Combining the transforms increases the communication/computation overlap and decreases communication start-up cost. Combining the state vectors eliminates the need to pack and unpack message buffers.

While other approaches to parallelizing the Legendre transform in `SSWMSB` exist, equi-partitioning both the physical and spectral spaces among the processors and using the ring algorithms to transform between the physical and spectral spaces has numerous advantages. First, this parallel implementation is perfectly load balanced, in the following sense. The floating point operations in the inner loops of the original program are divided equally among the processors, and these are the only computations in the inner loops of the parallel implementation. Moreover, the same number of floating point operations are performed in each processor during each step of the ring algorithms. Second, the ring algorithms require only that the underlying architecture support a ring topology efficiently in order to minimize the communication cost, which may be important in the next generation of distributed-memory multiprocessors. Third, the communication volume generated by the ring algorithms is the minimum possible for a parallel Legendre transform that does not perform redundant calculations. Fourth, the organization of the ring algorithms maximizes the potential for communication/computation overlap. One disadvantage of the ring algorithms is that the communication start-up cost is $\Theta(P)$, which is not the minimum for a parallel Legendre transform. The communication/computation overlap (potentially) offsets the additional communication cost. Another disadvantage is that at most $J/2$ processors can be used, limiting the exploitable parallelism. But nothing in this implementation prevents exploitation of other directions of parallelism, and we are currently investigating parallelizing the FFTs as well as the Legendre transform.

8.1. Denormalized numbers.. Table 1 contains the performance numbers for our initial implementation of the algorithm described above. Here P is the number of processors used, T is the total execution time (in seconds) for 10 timesteps, S is the observed speed-up over the execution time on one processor, E is the parallel efficiency, C is the maximum communication time (in seconds) observed on a single processor, and C/T is the ratio of the maximum communication time to the total execution time. The communication time is the *observed* time during which a processor is sending a message, receiving a message, or waiting for a message. The time that a processor spends within

T85 (10 timesteps)					
Processors (P)	Time (T)	Speedup (S)	Efficiency (E)	Communication (C)	C/T
1	39.51	1.00	1.00	.00	0.00
2	21.57	1.83	.92	3.62	0.17
4	14.29	2.76	.69	5.23	0.37
8	8.03	4.92	.62	3.22	0.40
16	5.00	7.90	.49	2.41	0.48
32	3.31	11.94	.37	1.69	0.51
64	2.80	14.11	.22	1.76	0.63

TABLE 1
Performance of initial parallel implementation.

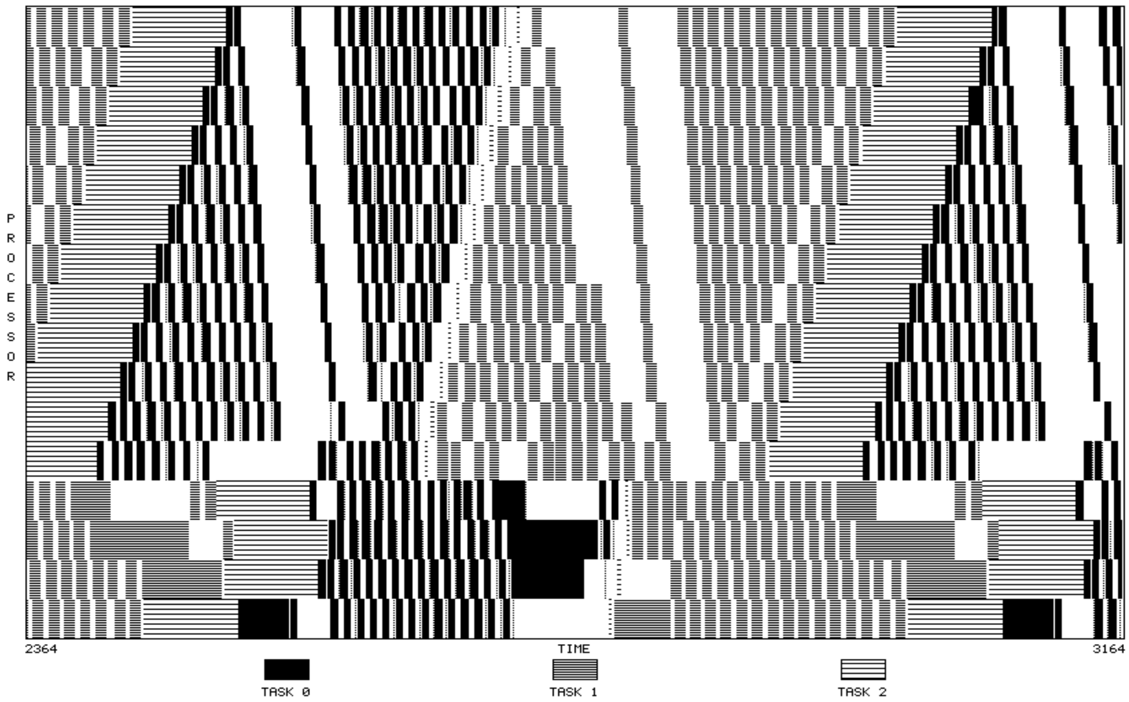

FIG. 1. *Task graph showing effect of denormalized numbers on ring algorithm.*

a system routine is observable, while cycle-stealing and memory contention caused by overlapping communication and computation are not.

These performance numbers are both disappointing and surprising. Since the parallel implementation is load balanced, the program should run in a relatively synchronous manner, in which case the communication time should be an increasing function of the number of sends (receives) and the volume of data being communicated, and a decreasing function of the amount of computation in each step of the ring algorithms. (The more computation there is in a step, the more likely it is that communication will be overlapped with computation during the step.) For each ring algorithm, the volume is independent of P, the number of sends (receives) is proportional to P, and the amount of computation per step is inversely proportional to P. In consequence, we expect the communication time to be an increasing function of P. Thus, it is surprising that the communication cost is largest *for small numbers of processors*.

One reason for the unexpected behavior of this implementation can be observed from the *task graph* in Fig. 1. The task graph indicates what each processor was doing at a given time. Three different tasks are indicated. Task 1 is the computational kernel of the first ring algorithm, calculating contributions to physical quantities from the current block of spectral coefficients. Thus, a sequence of P "task 1" blocks, separated by white blocks representing interprocessor communication, occurs for each processor during each timestep. Task 2 is the computation occurring between the two ring algorithms, primarily a sequence of FFTs. Task 0 is the computational kernel of the second ring algorithm, calculating contributions to the next block of spectral coefficients to arrive. Thus, each processor also has a sequence of P "task 0" blocks per timestep. Because only three tasks can be indicated in this monochrome display, the spectral coefficient update in the second ring algorithm is also called task 1 and the time update of the spectral coefficients is also called task 2. Both of these tasks take very little time, and are easily distinguished in the task graph.

The task graph in Fig. 1 displays a little more than one timestep when using 16 processors to solve T85. Since the latitudes and spectral coefficients are divided equally among the processors, each instance of a task should take approximately the same amount of time as every other instance of the same task. This is true for task 2, but the bottom four processors have some task 0 and task 1 blocks that are up to 20 times as long as they should be. The resulting load imbalance destroys the efficiency of the ring algorithm, causing processors to wait for long periods for messages. The reason for the load imbalance is the presence of denormalized numbers[2] in the precalculated values $P_n^m(\mu)$. The Intel iPSC/860 supports the IEEE floating point standard, but handles computation with denormalized single or double precision numbers in software, not hardware. Using our partition of the spectral coefficients and the latitudes, the denormalized numbers are not equidistributed over the processors, causing large load imbalances.

[2] A denormalized number is a non-zero floating-point number whose exponent is the minimum admitted and whose leading significant digit is zero [4].

8.2. Blocking sends. The solution to the problem of denormalized numbers is simple. For the resolution of the example problems used in our tests, the additional accuracy provided by denormalized numbers is unnecessary. Our solutions are just as accurate when the denormalized $P_n^m(\mu)$ values are set to zero. If higher accuracy is required, using double precision arrays to hold $P_n^m(\mu)$ eliminates the denormalized numbers when solving T85 and restores the load balance. For the larger examples, denormalized numbers will reoccur even when using double precision variables, and will still need to be set to zero. Note that, while using double precision arrays would normally significantly increase the execution time of `SSWMSB` when compared to using single precision arrays, the increase is partially offset by not doing arithmetic with denormalized numbers. For example, the execution time for solving T85 on one processor is approximately 30% greater when the arrays are double precision than when they are single precision, if denormalized numbers have been zeroed. When denormalized numbers are not set to zero, the double precision version is only 5% slower than the single precision version, and it becomes faster when more than one processor is used, due to the better load balance.

Table 2 contains the performance numbers when denormalized $P_n^m(\mu)$ values are set to zero. Note that speedup and efficiency are measured in terms of the execution time when using one processor with this implementation. Since the fastest serial execution time belongs to an implementation described later in this paper, we have not normalized the speedup and efficiency measures over all implementations. To compare between implementations, we use the raw execution times.

As expected, the improvement over the previous implementation is significant, but the communication cost is still higher than expected for small P. The *Gantt chart* in Fig. 2 indicates what the difficulty is. The Gantt chart indicates what state (busy, communicating, idle) each processor is in at any given time. Figure 2 is a magnified snapshot of the start of the second ring algorithm, for problem T85 when using 16 processors. Note the stagger of the shaded blocks. These blocks primarily represent time spent in the PICL routine `send0`, which initiates a message transmission. In most cases, a processor is not exiting `send0` until the processor it is sending to (the one immediately below it in the Gantt chart) has finished computing. This causes long delays for the sending processor.

The problem is due to the communication protocol on the iPSC/860 when using the native iPSC command `csend`, which is called within the PICL routine `send0`. On the Intel iPSC/2, if a message is sent to a processor that is not expecting it, the user process is interrupted and the message is copied into a system buffer. When the request for the message is finally made, the message is copied from the system buffer into the designated user buffer. In an attempt to cut down on buffer copies of long messages, the current operating system on the iPSC/860 does not interrupt the user process to handle the incoming message, but rather waits until control is relinguished to the operating system for other reasons. Since the iPSC command `csend` blocks until the message is on its way and the message buffer can be reused, the sending processor is blocked until a compute phase on the receiving processor is finished. The operating system

T85 (10 timesteps)					
P	T	S	E	C	C/T
1	36.58	1.00	1.00	.00	0.00
2	18.68	1.96	.98	.72	0.04
4	11.17	3.27	.82	2.08	0.19
8	6.06	6.04	.76	1.43	0.24
16	3.81	9.60	.60	1.11	0.24
32	2.71	13.50	.42	1.10	0.29
64	2.50	14.63	.23	1.40	0.56

TABLE 2
Performance of combined latitude/spectral coefficient partitioned code after removal of denormals.

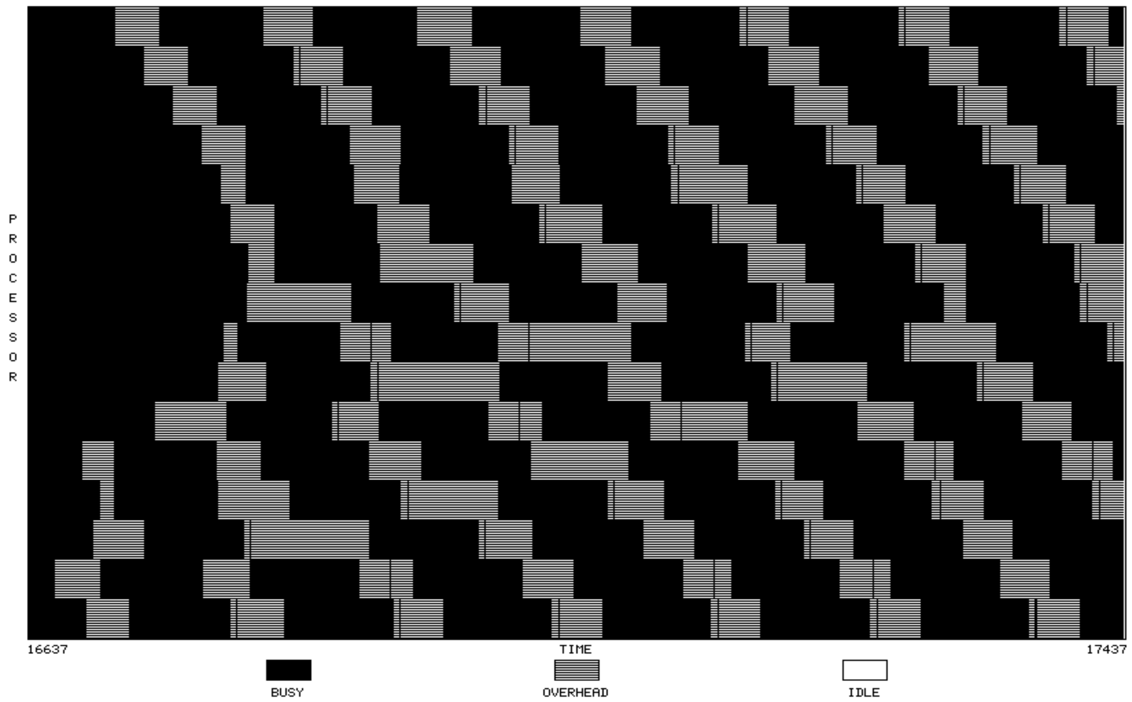

FIG. 2. *Gantt chart showing effect of blocking send on ring algorithm.*

designers are aware of this problem, and future revisions of the operating system will use a different communication protocol.

8.3. Blocked network protocol messages. To avoid the "blocking send" phenomenon, we replaced the calls to the native iPSC commands `csend` and `crecv`, which do not return until the message buffer can be used, with calls to the commands `isend` and `irecv`, which return immediately. The logic of the first ring algorithm is now (a) call `irecv` to indicate where the next message should be put, (b) call `isend` to send the current data to the neighbor to the left, (c) compute using the current data, and (d) wait until the `isend` and `irecv` calls have completed by calling `msgwait`. Similar logic is used for the second ring algorithm. This requires double buffering so that incoming data do not overwrite the data that is currently being used. Since `isend`, `irecv`, and `msgwait` are not available in the original version of PICL, new commands `sendbegin0`, `sendend0`, `recvbegin0`, and `recvend0` have been added that use these native commands.

This new logic not only avoids the blocking send, but also eliminates some of the (system-level) buffer copying inherent in the previous implementation. Other minor modifications were also made at this time to increase the potential for communication/computation overlap. Table 3 contains the performance numbers for this new implementation. First, note we achieve "superlinear" speed-up when $P = 2$. The performance of the i860 processor is very sensitive to the number of cache misses, and the decrease in the workload per processor as P increases from 1 to 2 permits a higher percentage of the data to reside in cache at one time. We notice this for the first time in this implementation because the double buffering also increases data locality, thus decreasing cache misses. The performance of this implementation is very good for small numbers of processors, and does not seem inappropriate for large P.

I. Foster, one of a group of researchers at Argonne National Laboratory who are examining the theoretical scalability of the spectral transform method [9], used our performance numbers to validate their analysis. The numbers fit very well *if the communication costs of the iPSC/860 are much higher than benchmarks indicate.* To clarify this anomaly, we used ParaGraph to examine the behavior of this implementation more closely. Figure 3 is the Gantt chart for the first part of the second ring algorithm when using 32 processors to solve T85. This figure indicates that processors are idle an unexpectedly large portion of the time, although it is unclear at first glance what is causing this behavior. Upon closer examination (of this figure and other ParaGraph displays), we discovered that the time required to send a message to a neighboring processor varies significantly during the computation. This inconsistency has the same effect as a load imbalance on the ring algorithms, causing a significant increase in the idle time when P is large. We queried P. Pierce at Intel, and received the following reply. The standard protocol for long messages is (a) to query the destination processor as to whether there is sufficient space for the message, (b) to wait for the reply, and (c) to begin the transmission of the message. In a ring algorithm, each processor is not only requesting to send a message to its "left", but it is also being requested to accept a message from its "right". If the transmission of the message from the right

T85 (10 timesteps)					
P	T	S	E	C	C/T
1	34.56	1.00	1.00	.000	0.00
2	16.94	2.04	1.02	.003	0.00
4	8.66	3.99	1.00	.006	0.00
8	4.96	7.37	.92	.022	0.00
16	2.73	12.66	.79	.064	0.02
32	2.07	16.70	.52	.408	0.20
64	2.14	16.15	.25	1.024	0.48

TABLE 3

Performance of combined latitude/spectral coefficient partitioned code after eliminating blocking send phenomenon.

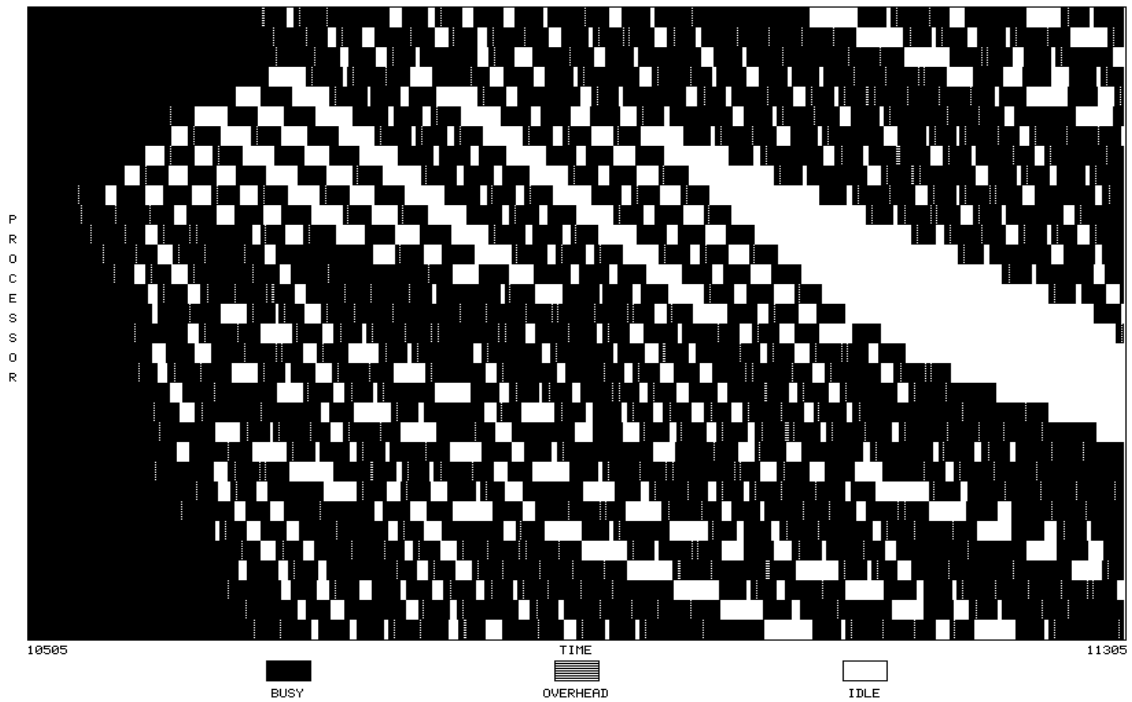

FIG. 3. *Gantt chart showing effect of three step message protocol on ring algorithm.*

begins before the acknowledgement from the left is received, this acknowledgement is blocked until the "incoming" message is completed. As P increases, and the amount of computation between communication calls decreases, this behavior becomes both more probable and more destructive to the efficiency of the ring algorithms.

8.4. Simplified network protocol. A simple solution to the performance problem caused by the communication protocol is to use a special message type available on the Intel iPSC/2 and iPSC/860. If the message type used to send a message is within a special range [21], then the destination processor is not queried as to whether there is room for the message. If the user has not "told" the destination processor to expect the message, by calling `crecv` or `irecv` for example, before the message arrives, then the message is thrown away. Some care is required to use these *force types* correctly, but the ring algorithms are deterministic, and all message types and sizes are known *a priori*. Thus, by increasing the amount of working storage allocated to message buffering, the request and acknowledgement messages of normal interprocessor communication can be eliminated. The performance of this implementation for T85 is indicated in Table 4. Figure 4 is the Gantt chart for the first part of the second ring algorithm when using 32 processors to solve T85. It is clear that using force types avoids the inefficient behavior illustrated in Fig. 3: almost all idle time has been eliminated and 50% more steps of the ring algorithm have been completed during the interval of time displayed in the figure.

9. Results and analysis. Table 5 contains the performance numbers of the final implementation for problems T21, T42, T85, T169, and T340. This table contains five new columns. The column marked J/P is the number of latitudes assigned to each processor for the given problem and number of processors. The column $(C+I)$ is the maximum (over all processors) of the sum of the communication time and the idle time at the end of the program spent waiting for the last processor to finish. (As the number of timesteps increases, the difference between $(C+I)$ and C becomes arbitrarily small, but it is still a factor in the results reported here.) The column T_{perf} is the execution time if the program were "perfectly parallel", i.e. the execution time when using one processor divided by P. The column T_{comp} is the execution time for a version of the program in which the subroutine calls that invoke interprocessor communication are commented out. The column T_{comm} is the execution time for a version of the program in which everything is commented out except the subroutine calls that invoke interprocessor communication and logic associated with interprocessor communication.

First, note that the efficiency is high ($\geq 80\%$) if at least eight latitude lines are assigned to each processor. Also note that the efficiency is almost solely a function of J/P across all of the problems. This "fact" is used to define the speedups and efficiencies for problems T169 and T340, since the executable of the program when compiled for these problems will not fit in the memory of a single processor.[3] The speedup and efficiency

[3] The size of the executables can be reduced dramatically by not precomputing the associated Legendre polynomials, but this is beyond the scope of our experiments here. Our parallel implementations of SSWMSB make no substantive changes to the original serial algorithm except reordering the computations.

T85 (10 timesteps)					
P	T	S	E	C	C/T
1	34.76	1.00	1.00	.000	0.00
2	16.94	2.05	1.02	.147	0.01
4	8.59	4.05	1.01	.125	0.01
8	4.59	7.57	.95	.102	0.02
16	2.62	13.27	.83	.117	0.04
32	1.68	20.69	.65	.198	0.12
64	1.28	27.16	.42	.359	0.28

TABLE 4
Performance of combined latitude/spectral coefficient partitioned code after introducing force types.

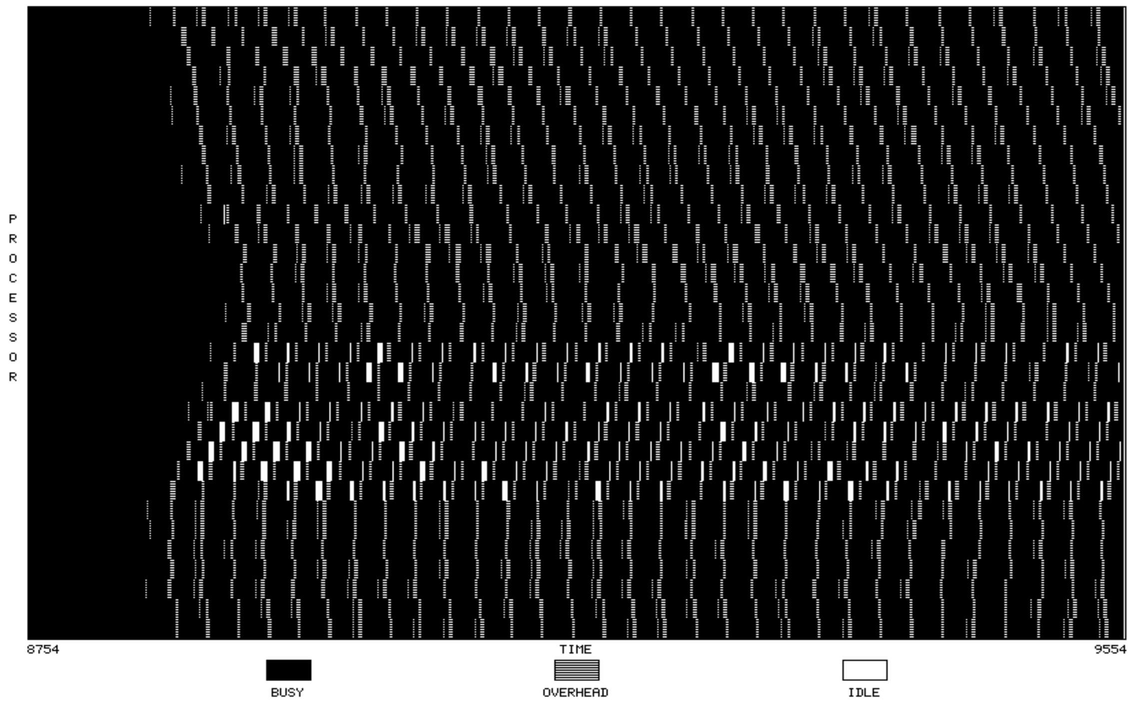

FIG. 4. *Gantt chart showing effect of using force types in the ring algorithms.*

T21 (30 timesteps)									
P	J/P	T	S	E	C	$(C+I)$	T_{perf}	T_{comp}	T_{comm}
1	32	2.40	1.00	1.00	.000	0.00	2.40	2.40	.002
2	16	1.24	1.94	.97	.033	0.033	1.20	1.20	.083
4	8	.72	3.33	.83	.046	0.047	.60	.638	.166
8	4	.51	4.71	.59	.108	0.108	.30	.370	.195
16	2	.49	4.90	.31	.175	0.175	.15	.239	.265

T42 (30 timesteps)									
P	J/P	T	S	E	C	$(C+I)$	T_{perf}	T_{comp}	T_{comm}
1	64	14.03	1.00	1.00	.000	0.00	14.03	14.03	.004
2	32	7.03	2.00	1.00	.105	0.105	7.02	6.90	.259
4	16	3.70	3.79	.95	.101	0.102	3.51	3.52	.432
8	8	2.12	6.62	.83	.152	0.152	1.75	1.86	.515
16	4	1.41	9.95	.62	.207	0.208	.88	1.07	.639
32	2	1.25	11.22	.35	.418	0.418	.44	.68	.771

T85 (10 timesteps)									
P	J/P	T	S	E	C	$(C+I)$	T_{perf}	T_{comp}	T_{comm}
1	128	34.76	1.00	1.00	.000	0.00	34.76	34.76	.014
2	64	16.94	2.05	1.02	.147	0.152	17.38	16.86	.309
4	32	8.59	4.05	1.01	.125	0.129	8.69	8.44	.549
8	16	4.59	7.57	.95	.102	0.109	4.35	4.36	.577
16	8	2.62	13.27	.83	.117	0.129	2.17	2.33	.635
32	4	1.68	20.69	.65	.198	0.201	1.09	1.31	.674
64	2	1.28	27.16	.42	.359	0.359	.54	.83	.807

T169 (10 timesteps)									
P	J/P	T	S	E	C	$(C+I)$	T_{perf}	T_{comp}	T_{comm}
8	32	26.07	"8.00"	"1.00"	.332	0.370	26.07	25.78	2.07
16	16	13.91	14.99	.94	.188	0.206	13.04	13.37	2.29
32	8	8.12	25.68	.80	.232	0.251	6.52	7.25	2.31
64	4	5.27	39.57	.62	.406	0.414	3.26	4.15	2.45
128	2	4.05	51.49	.40	.854	0.882	1.63	2.58	2.63

T340 (10 timesteps)									
P	J/P	T	S	E	C	$(C+I)$	T_{perf}	T_{comp}	T_{comm}
64	8	28.00	"51.20"	".80"	.472	0.519	22.4	24.74	9.21
128	4	18.19	78.81	.62	.850	0.889	11.2	14.31	9.36

TABLE 5
Performance of best combined latitude/spectral coefficient partitioned code.

for the smallest number of processors on which the executable will fit is extrapolated from the efficiency values observed in the smaller example problems for the same number of latitudes per processor. The extrapolated values are surrounded by quotes in Table 5. Subsequent speedups and efficiencies are calculated from these extrapolated values. Note that these subsequent values are consistent with the assumption that efficiency is solely a function of J/P.

As mentioned earlier, the parallel implementation described in this paper is perfectly load balanced, and we might expect the parallel execution time to be T_{perf}. But this naive analysis ignores (at least) two other costs, the *observable communication overhead*, reflecting the time spent in interprocessor communication routines, and the *computation overhead*, reflecting loop overhead, logic associated with managing interprocessor communication, and the redundant computation of intermediate results that are too costly to communicate around the ring.[4] An estimate for the observable communication overhead is given by C. The computation overhead can be estimated by subtracting T_{perf} from T_{comp}. Of these two, only the observable communication overhead is amenable to minimization in the parallel implementation, once serial inefficiencies are eliminated. We successfully avoided packing and unpacking message buffers in our implementations, and, modulo compiler improvements, the computation overhead represents a fixed cost. Thus, T_{comp} is a lower bound on the execution time of any parallel implementation of `SSWMSB`.

Without overlapping communication and computation, the observable communication overhead would be at least as large as T_{comm}. (It can be even larger since load imbalance or network contention appear as communication overhead. Note that T_{comm} is only a lower bound because our use of `isend`, `irecv`, and force types for interprocessor communication minimizes contention for network resources.) Thus, $T_{\mathrm{comm}} - C$ is a conservative estimate of the amount of communication that is *available* to be overlapped with computation, and

$$(T_{\mathrm{comm}} - C)/T_{\mathrm{comm}} \tag{19}$$

is a measure of how effective a parallel implementation is at *hiding* communication. Table 6 describes this measure for our implementation. Thus, our final parallel implementation is very effective at hiding communication costs.

Some of the hidden communication still represents an overhead in the parallel implementation, which we will refer to as the *hidden communication overhead*. While we conjecture that this overhead is due to cycle-stealing and memory contention, its magnitude is easily calculated from our performance numbers. For example, if there were no hidden computation overhead, then

$$T_{\mathrm{comp}} + (C + I) \tag{20}$$

[4] In general, both of these costs vary from processor to processor. Since the amount and scheduling of both computation and communication is the same on all processors, there should be little variation in costs between processors. The performance measurements for our final implementation support this, and we treat these costs as being processor independent in our discussions.

$(T_{\text{comm}} - C)/T_{\text{comm}}$					
P	T21	T42	T85	T169	T340
1	1.0	1.0	1.0	—	—
2	.60	.59	.52	—	—
4	.72	.77	.77	—	—
8	.45	.70	.82	.84	—
16	.34	.68	.82	.92	—
32	—	.46	.71	.90	—
64	—	—	.56	.83	.95
128	—	—	—	.68	.91

TABLE 6
Relative amount of hidden communication.

would be an overestimate of T. Table 7 describes this estimate and its error for T85. Thus, (20) is an overestimate only for $P = 2$, and it is an underestimate by as much as

T85 (10 timesteps)				
P	T	$T_{\text{comp}} + (C + I)$	error	error$/T$
1	34.76	34.76	0.0	0.0
2	16.94	17.01	.07	.004
4	8.59	8.57	−.02	−.002
8	4.59	4.47	−.12	−.03
16	2.62	2.46	−.16	−.06
32	1.68	1.51	−.17	−.10
64	1.28	1.19	−.09	−.07

TABLE 7
Comparison of execution time estimate to observed execution time.

10% for the other cases. The fraction of the hidden communication that contributes to the hidden communication overhead can be estimated by

$$\frac{T - (T_{\text{comp}} + (C + I))}{T_{\text{comm}} - C}. \tag{21}$$

The numerator is an estimate of the increase in computation time due to the hidden overhead, while the denominator is an estimate of the amount of communication that is hidden. This measure is indicated in Table 8. On the average, 30% of the hidden communication contributes to the hidden communication overhead when $P > 2$.

10. Performance model. The following rough analysis supports the strong dependence of the efficiency on J/P. The dominant terms in the serial complexity of the spectral transform method are of the form

$$T_s \;\equiv\; aJI\log_2 I \;+\; bJM^2$$

P	T21	T42	T85	T169	T340
1	0.0	0.0	0.0	—	—
2	.14	.16	−.44	—	—
4	.29	.24	.05	—	—
8	.37	.30	.25	−.05	—
16	.84	.31	.31	.16	—
32	—	.43	.36	.30	—
64	—	—	.20	.35	.31
128	—	—	—	.33	.35

TABLE 8
Estimate of fraction of hidden communication that contributes to hidden communication overhead.

for positive constants a and b. Since the parallel implementation is perfectly load balanced, the parallel complexity should be $T_p = T_s/P$ plus a term representing the *computation overhead* and a term representing the *communication overhead*, reflecting both the observable and hidden communication overheads. The dominant term in the computation overhead is proportional to the number of times the outer loops in the ring algorithms are executed, and has the form cM^2 for some positive c. The communication overhead is a function of both the number of sends (receives) and the volume of information sent (received) during the ring algorithms, and has the form $dP + eM^2$ for positive constants d and e if communication and computation are not overlapped (i.e. all communication is reflected in the observable communication overhead) and if there is no contention for network resources. The speedup of the resulting model is

$$S = \frac{aJI\log_2 I \; + \; bJM^2}{a(J/P)I\log_2 I \; + \; b(J/P)M^2 \; + \; (c+e)M^2 \; + \; dP},$$

and the efficiency is

$$\begin{aligned} E &= S/P \\ &= \frac{a(J/P)I\log_2 I \; + \; b(J/P)M^2}{a(J/P)I\log_2 I \; + \; b(J/P)M^2 \; + \; (c+e)M^2 \; + \; dP} \\ &= \frac{(a(I\log_2 I)/M^2 + b)(J/P)}{(a(I\log_2 I)/M^2 + b)(J/P) \; + \; c \; + \; e \; + \; d(P/M^2)}. \end{aligned}$$

Since $(I\log_2 I)/M^2$ and P/M^2 both go to zero as M grows, this expression for the efficiency is primarily a function of J/P for large M, converging to $b/(b+(c+e)P/J)$ when $M \rightarrow \infty$. If communication and computation are instead assumed to be completely overlapped with no hidden communication overhead, then the asymptotic formula becomes

$$\frac{b}{\max\{\, b + c(P/J)\, ,\; e(P/J)\,\}}.$$

Note that both of these models are too simplistic to model the observed behavior. They ignore the possibility of partially overlapped communication and of hidden communication overhead. But these models do partially explain the observed relationship between the efficiency and J/P.

11. Implications for global climate modeling. Unlike the shallow water equations, the dynamics equations of CCM2 involve vertical as well as horizontal velocities. Horizontal spectral approximations for neighboring vertical levels of the atmosphere are coupled using finite difference approximations. When using the spectral transform method in this context, `SSWMSB` represents a computational kernel that is computed for each level independently. The parallel spectral transform algorithm described here can be applied unchanged to each level in sequence, and similar efficiencies should be observed for this part of the code. But a similar parallel algorithm can be used to transform all levels simultaneously, either by interleaving the computation on the different levels or by blocking the computation across levels. Both of these variants will significantly increase the opportunity for overlapping communication and computation, especially for the smaller model resolutions.

CCM2 also models the physical processes associated with radiation, condensation, clouds and surface hydrology. These calculations, which are the dominant computational cost for small to moderate size resolutions, introduce coupling in the vertical direction only. Thus, for a purely horizontal decomposition, keeping associated vertical data local to a processor, the physics calculations require no interprocessor communication. When the physics and vertical velocity calculations are added to the horizontal dynamics, the computational load increases without any increase in communication cost. Thus, for a global climate model like CCM2, we expect the parallel efficiencies to be significantly better than reported here.[5]

Note that decomposing over only latitude in the physical space limits the number of processors that can be used to no more than half as many processors as there are lines of latitude. This will cause problems when scaling the problem size because the serial complexity grows as M^3 times the number of vertical levels for the dynamics calculations and M^2 times the number of levels for the physics calculations, per timestep. Even for current problem sizes, this restriction is unacceptable. For example, while T42 is the current design resolution for CCM2, the computational cost of this resolution will burden any present day supercomputer. Yet at most 32 processors can be used for this resolution, using the parallel implementation described here. Thus it is clear that more than a purely latitudinal decomposition will be necessary when parallelizing CCM2.

12. Conclusions. Our initial experiments in parallelizing the spectral transform method indicate that efficient implementations are possible on the iPSC/860 even when decomposing over only latitude in the physical space and parallelizing only the Legendre transform. Moreover, the performance of the resulting algorithm appears to be solely a function of the number of latitudes assigned per processor. Our next step is to

[5] This discussion ignores the issues of load imbalance in the physics calculations and the parallel efficiency of the semi-Lagrangian algorithm that is used to move moisture in the model, but initial studies indicate that neither of these represent performance bottlenecks. More problematic is meeting the I/O demands of a production code given the currently available hardware and software.

introduce a parallel FFT into SSWMSB and examine the tradeoffs between decomposing over latitude and longitude, and how the combined efficiency of the parallel Legendre transform and the parallel FFT varies as a function of the decomposition.

Note that the observed efficiencies are sensitive to compiler improvements or the introduction of assembly-coded inner loops. The observed Mflop rate for the one processor implementation of SSWMSB for T85 is 4.8, on a processor whose peak rate is 80 Mflops. As better compilers become available, the efficiency will drop.[6] For example, T_{comm} represents a lower bound on the execution time for any executable of the code, and it is already greater than 60% of the observed execution time when two latitude lines are assigned to each processor. Thus, while we achieve an aggregate Mflop rate of 340 when solving T340 using 128 processors, we can not do better than 735 Mflops for this problem with the current interconnection hardware, microprocessor, and operating system, no matter how good the compiler is.

Execution time, rather than efficiency, is the important issue. The goal of this work is to determine whether the parallel spectral transform method and the iPSC/860, or similar machines, can provide the turn-around time to run the next generation of global climate models. Our performance measurements indicate that our parallel implementation of the Legendre transform in SSWMSB is near optimal with respect to communication overhead, and that it will be useful in determining these issues.

The work described in this paper took five months to complete. Without PICL and ParaGraph, we would likely have stopped the work much sooner, out of simple frustration, and the results would have been much worse. These tools were crucial in our efforts to identify, understand, and correct performance bottlenecks. Note that not all parallel application codes that we have experience with are as sensitive to the details of the interprocessor communication protocol. Moreover, some of the performance problems we described in this paper will disappear as multiprocessor hardware and software mature. But a state-of-the-art multiprocessor will, by definition, never be a mature product, and performance measurement tools of the type described here will always be important when using new computer architectures.

Acknowledgments. We are grateful for the help and advice of the members of the CHAMMP Inter-Organizational Team for Numerical Simulation, a collaboration involving Argonne National Laboratory, the National Center for Atmospheric Research, and Oak Ridge National Laboratory. In particular, we thank J. J. Hack for making SSWMSB available to us, and for his willingness to answer our questions about the code, R. E. Flanery for helping port SSWMSB from NCAR to ORNL, and I. Foster for pointing out anomalies in our performance numbers. We also thank P. Pierce of Intel Scientific Computers for his prompt replies to queries and for his clear explanations of the network protocols used on the iPSC/860. Finally, we thank the anonymous referees for their helpful suggestions on the presentation of the material in this paper.

This research was supported by the Atmospheric and Climate Research Division

[6] Since this work was done, new compilers have become available that increase the one processor Mflop rate for T85 to 6.8.

and the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

REFERENCES

[1] A. P. M. Baede, M. Jarraud, and U. Cubasch, *Adiabatic formulation and organization of ECMWF's spectral model*, ECMWF Tech. Rept. No. 15, European Centre for Medium-Range Weather Forecasts, Reading, England, 1979.

[2] G. J. Boer, N. A. McFarlane, R. Laprise, J. D. Henderson, and J.-P. Blanchet, *The Canadian climate centre spectral atmospheric general circulation model*, Atmos.-Ocean, Vol. 22 (1984), pp. 226–243.

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

[4] W. J. Cody, J. T. Coonen, D. M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. N. Ris, and D. Stevenson, *A proposed radix- and work-length-independent standard for floating-point arithmetic*, IEEE Micro, Vol. 4, No. 4 (August 1984), pp. 86–100.

[5] D. Dent, *A modestly parallel model*, in The Dawn of Massively Parallel Processing in Meteorology, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 1990, pp. 21–31.

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

[7] T. H. Dunigan, *Performance of the Intel iPSC/860 hypercube*, Tech. Rep. ORNL/TM-11491, Oak Ridge National Laboratory, Oak Ridge, TN, June 1990.

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

[9] I. Foster, W. Gropp, and R. Stevens, *The parallel scalability of the spectral transform method*, Technical Report MCS-P215-0291, Argonne National Laboratory, Argonne, IL, 1991.

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

[11] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, *A machine-independent communication library*, in The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, J. L. Gustafson, ed., Golden Gate Enterprises, P.O. Box 428, Los Altos, CA, 1990, pp. 565–568.

[12] ——, *PICL: a portable instrumented communication library, C reference manual*, Tech. Rep.

ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.

[13] ——, *A users' guide to PICL: a portable instrumented communication library*, Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.

[14] J. J. Hack, *On the promise of general-purpose parallel computing*, Parallel Computing, Vol. 10 (1989), pp. 261–275.

[15] M. T. Heath, *Visual animation of parallel algorithms for matrix computations*, in The Fifth Distributed Memory Computing Conference, D. W. Walker and Q. F. Stout, eds., IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 1213–1222.

[16] M. T. Heath and J. A. Etheridge, *Visualizing the performance of parallel programs*, IEEE Software, Vol. 8, No. 5 (1991), pp. 29–39.

[17] M. T. Heath, G. A. Geist, and J. B. Drake, *Early experience with the Intel iPSC/860 at Oak Ridge National Laboratory*, Tech. Rep. ORNL/TM-11655, Oak Ridge National Laboratory, Oak Ridge, TN, September 1990.

[18] G.-R. Hoffman, P. N. Swarztrauber, and R. A. Sweet, *Aspects of using multiprocessors for meteorological modeling*, in Multiprocessing in Meteorological Models, G.-R. Hoffman and D. F. Snelling, eds., Springer-Verlag, New York, 1988, pp. 125–196.

[19] R. N. Hoffman and T. Nehrkorn, *Multiprocessing a global spectral numerical weather prediction model*, in Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 168–173.

[20] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, Maryland, 1978.

[21] Intel Scientific Computers, *iPSC/2 and iPSC/860 programmer's reference manual*, Beaverton, OR, June 1990.

[22] ——, *iPSC/2 and iPSC/860 user's guide*, Beaverton, OR, June 1990.

[23] T. Kauranne, *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, 1990, pp. 303–314.

[24] T. Kauranne and G.-R. Hoffmann, *Parallel processing: a view from ECMWF*. Oral presentation at Computers in Atmospheric Science, September 1990.

[25] S. L. Lillevik, *Touchstone program overview*, in The Fifth Distributed Memory Computing Conference, D. W. Walker and Q. F. Stout, eds., IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 647–657.

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

[27] B. J. McAvaney, W. Bourke, and K. Puri, *A global spectral model for simulation of the general circulation*, J. Atmos. Sci., Vol. 35 (1978), pp. 1557–1583.

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

[29] I. A. Stegun, *Legendre functions*, in Handbook of Mathematical Functions, M. Abramowitz and I. A. Stegun, eds., Dover Publications, New York, 1972, ch. 8, pp. 332–353.

[30] P. N. Swarztrauber and R. K. Sato, *Experiences on the connection machine.* Oral presentation at Computers in Atmospheric Science, September 1990.

[31] ——, *Solving the shallow water equations on the Cray X-MP/48 and the Connection Machine 2*, in The Dawn of Massively Parallel Processing in Meteorology, G.-R. Hoffman and D. K. Maretis, eds., Springer-Verlag, Berlin, 1990, pp. 260–276.

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

[33] D. L. Williamson, *Difference approximations for fluid flow on a sphere*, in Numerical Methods Used in Atmospheric Models, vol. II of GARP Pub. Ser. No. 17. JOC, World Meteorological Organization, Geneva, Switzerland, 1979, ch. 2, pp. 51–120.

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

[35] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber, *A standard test set for numerical approximations to the shallow water equations on the sphere*, Tech. Rep. ORNL/TM–11773, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.

[36] D. L. Williamson, J. T. Kiehl, V. Ramanathan, R. E. Dickinson, and J. J. Hack, *Description of NCAR community climate model (CCM1)*, NCAR Tech. Note NCAR/TN-285+STR, NTIS PB87–203782/AS, National Center for Atmospheric Research, Boulder, CO, June 1987.

[37] D. L. Williamson, Ed., *CCM progress report - July 1990*, NCAR Tech Note NCAR/TN–351+PPR, National Center for Atmospheric Research, Boulder, CO, July 1990.

[38] P. H. Worley and J. B. Drake, *Parallelizing the spectral transform method – part I*, Tech. Rep. ORNL/TM–11747, Oak Ridge National Laboratory, Oak Ridge, TN, February 1991.

[39] P. H. Worley and M. T. Heath, *Performance characterization research at Oak Ridge National Laboratory*, in Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 431–436.