Spectral and Multigrid Spherical Helmholtz Equation Solvers on Distributed Memory Parallel Computers

Saulo R. M. Barros
University of São Paulo

Tuomo Kauranne
European Centre for Medium-range Weather Forecasts

January 7, 1992

Abstract

We compare implementations of a multigrid method and a spectral method for solving the Helmholtz equation on the sphere on the Intel iPSC/2 distributed memory parallel computer. Solving Helmholtz equations is the heart of any semi-implicit time-stepping scheme in a global weather model. The multigrid solver employs a full multigrid solution scheme and longitudinal line relaxation smoothing. The spectral method uses spherical harmonics as basis functions, employing rhomboidal truncation and symmetric-antisymmetric decomposition. Benchmark results, together with analysis of parallel speedup and efficiency, are presented for both scalar and vectorized versions of the codes.

1. INTRODUCTION

Weather forecasting has always required a lot of computer power. Weather centres have been using the most powerful computers available from the pioneering experiments on the ENIAC till current supercomputers. Today, global weather forecasts are being computed on vector supercomputers, with a small number of concurrent processors sharing a global memory and delivering a performance up to a few Gigaflops (Simmons and Dent 1989 , Dent 1992). Recent developments in computer technology suggest that in the relatively near future the most powerful machines will make use of more massive parallelism. They may well have on the order of several thousand interconnected processors equipped with local memories and their peak performance may soon reach a Teraflops. This all motivates the study of parallelization strategies

for algorithms relevant to weather models, and their implementation on distributed memory systems.

The weather models themselves have undergone several developments during the last two decades. One important direction has been the movement towards more implicit time-stepping schemes. By now, semi-implicit discretizations of both grid-point and spectral models (Robert 1969 , Kwizak and Robert 1971) have been widely adopted in operational models. Spectral models have been adopted in many centres, due to their accuracy and the absense of pole problems. These models have been recently improved by combining semi-Lagrangian advection techniques with the semi-implicit discretization (Robert 1981, Robert et al. 1982, Côté and Staniforth 1988, Ritchie 1988, 1990). Semi-Lagrangian techniques combined with a better handling of pole problems are also renewing the interest in grid-point models (McDonald and Bates 1989, Bates et al. 1990). A common feature of all semi-implicit models is the necessity of the solution of at least one Helmholtz-type equation every timestep.

The purpose of the present work is to examine techniques for the parallelization of spectral as well as grid-point Helmholtz solvers on distributed memory parallel machines. This can be viewed as a kernel problem for the parallelization of global weather models. Our aim here is not to exhaust all possible strategies for parallelization, but rather to derive reasonable approaches based on general considerations. Our implementation makes use of some parallelization tools (Hempel 1987, Bomans and Hempel 1990), which simplify the job and give some portability to the codes. We did not attempt to specifically tune the programs for running on the iPSC/2, the machine used for benchmarking.

The Helmholtz solvers chosen for our experiments were a spectral method, using spherical harmonics with a rhomboidal truncation (Machenhauer 1979 , Orszag 1970) and a full multigrid algorithm (Barros 1991). The choice of the spectral method is standard, with the exception of the truncation. Full rhomboidal truncation was preferred (instead of a 2/3-triangular truncation) for convenience, since it leads to a perfect load balance and since in a linear problem like the Helmholtz equation, aliasing is not a problem. Multigrid was chosen as the grid-point solver because of its optimal complexity (which makes it very fast) and because of its broader applicability, since it is not very sensitive to small changes in the equation form (Barros et al 1990). These solvers are described in Section 2, followed by a description of the parallelization strategies in Section 3. Our benchmark results on the iPSC/2 are presented in Section 4. Some conclusions are presented in Section 5.

2. THE HELMHOLTZ SOLVERS

2.1 The Full Multigrid Solver

In this section we describe the multigrid method (cf. Barros 1991) used in our

experiments. The Helmholtz equation

$$-\Delta u + cu = f \tag{1}$$

on the unit sphere is discretized through centered finite differences on a latitude-longitude grid, which includes the poles as grid-points. Written in spherical coordinates (λ, θ), (1) reads

$$-\left(\frac{1}{\cos^2\theta}\frac{\partial^2 u}{\partial\lambda^2} + \frac{1}{\cos\theta}\frac{\partial}{\partial\theta}\left(\cos\theta\frac{\partial u}{\partial\theta}\right)\right) + cu = f \tag{2}$$

In spherical coordinates the mesh is chosen to be rectangular and uniform. The mesh parameter h in spherical coordinates is defined by the relations $h = 2\pi/N_\lambda = \pi/N_\theta$, where the numbers N_λ and N_θ denote the number of grid-points on each latitude and longitude. N_λ and N_θ are chosen as powers of two, which obeys the relation $N_\lambda = 2N_\theta$ implied above. This is convenient for the coarsening process in the multigrid scheme and favors a good load balancing on hypercubes. The discretization leads to five-point stencils on the regular grid-points. At the poles, an integral form of the equation is used for the discretization. The resulting formulas relate the average of the solution at the first latitude line closest to the poles to the solution value at the pole (see Barros 1991 for details).

In the multigrid method a sequence of coarser grids is introduced, each one obtained by doubling the meshsize of the previous finer grid. We denote these grids as G_1 (finest grid) to G_m (coarsest grid). G_m is chosen as a uniform 45 degree grid, having thus 8×4 grid points.

The equations are solved to the level of the truncation error through a full multigrid scheme. In this procedure, the equation is first solved on the coarsest grid G_m through several relaxation sweeps. The solution is then interpolated (through bicubic interpolation) to G_{m-1}, where it is used as a first approximation to the solution. A multigrid V(1,1)-Cycle is then performed to compute the solution on this grid. This process of interpolating the solution from G_i to obtain a first guess on G_{i-1} and then computing the solution on G_{i-1} through a V(1,1)-Cycle is repeated until the solution on the finest grid G_1 is obtained. The fast convergence of the V(1,1)-Cycles (one cycle reduces the residual to less than 10% of its initial value, independently of the meshsize) allows the use of just one cycle on each level of the full multigrid procedure to obtain a solution with an error smaller than the discretization error. The computational work of the whole procedure is linearly proportional to the number of unknowns on the finest grid.

A V(1,1)-Multigrid cycle (on grid G_i, $i < m$) comprises the following steps:

- Application of one relaxation sweep to the current approximation $\hat{u}_i$ on G_i (producing $\tilde{u}_i$)
- Evaluation of the residual $r_i = f_i - L_i\tilde{u}_i$ on G_i (L_i stands for the discrete operator on G_i)

- Residual transfer to G_{i+1}, producing $f_{i+1} = I_i^{i+1} r_i$, where I_i^{i+1} is the transfer operator
- Solution of the equation $L_{i+1} u_{i+1} = f_{i+1}$ either through one V(1,1)-cycle starting with $\hat{u}_{i+1} = 0$ (if $i+1 < m$) or through several relaxation sweeps (if $i+1 = m$)
- Interpolation and addition of the coarse grid correction u_{i+1}, producing $\bar{u}_i = \tilde{u}_i + I_{i+1}^i u_{i+1}$
- Application of one relaxation sweep to $\bar{u}_i$, producing u_i

Our multigrid solver employs a zonal line-relaxation, where we solve simultaneously for all the unknowns common to a latitude circle. A cyclic tridiagonal system has to be solved for each line. Each relaxation sweep is performed zebrawise: the odd-indexed lines (including the poles and the equator) are relaxed first, followed by the even-indexed lines. This order not only gives a better smoothing of the error, but also enables the parallelization of this part of the algorithm. The relaxation can be performed independently on each odd-index line, and then on each even-index line. After a complete relaxation sweep the residuals vanish (and therefore do not need to be computed) at the even lines. This fact is used in the evaluation and transfer of residuals. The transfer operator is a full-weighting procedure - a weighted average of the residuals at the nine fine-grid-points surrounding a coarse-grid-point is transferred to the coarse grid. The transfer of corrections from coarse to fine grids is performed through bilinear interpolation.

2.2 The Spectral Solver

In the spectral method (Machenhauer 1979, Orszag 1970), the solution u of the Helmholtz equation (1) is expanded in spherical harmonics. The solution process takes advantage of the fact that spherical harmonics are eigenfunctions of the Laplacian on the sphere. Employing a rhomboidal truncation, u is written as:

$$u(\lambda,\theta) = \sum_{m=-M}^{M} \sum_{n=|m|}^{|m|+M} \hat{u}_n^m Y_n^m(\lambda,\theta) \quad , \tag{2}$$

where $Y_n^m(\lambda,\theta) = e^{im\lambda} P_n^m(\theta)$ are the spherical harmonics, λ is the longitude, θ is the latitude and P_n^m is the Legendre polynomial of degree n and order m. M is the truncation parameter. When the equation to be solved is nonlinear, M is typically given a value satisfying $M \leq 2N_\theta/3$. Such a truncation is sufficient to eliminate aliasing due to quadratic nonlinearities (Orszag 1970). Because the Helmholtz equation is linear, we chose not to truncate and used the value $M = N_\theta$.

The relation

$$-\Delta Y_n^m + cY_n^m = (c + n(n+1))Y_n^m \tag{3}$$

enables, once the right-hand-side expansion

$$f(\lambda,\theta) = \sum_{m=-M}^{M} \sum_{n=|m|}^{|m|+M} \hat{f}_n^m Y_n^m(\lambda,\theta) \tag{4}$$

is known, the direct computation of the spectral coefficients of u by dividing the spectral transform of the data $\hat{f}_n^m$ by the corresponding eigenvalue of the Helmholtz operator $c + n(n+1)$:

$$\hat{u}_n^m = \hat{f}_n^m / (c + n(n+1)) \ . \tag{5}$$

The first step in the spectral solution process is the evaluation of the spectral coefficients of the right-hand-side

$$\hat{f}_n^m = \frac{1}{4\pi} \int_0^{2\pi} \int_{-1}^{1} f(\lambda,\mu) \bar{Y}_n^m(\lambda,\mu) d\mu d\lambda \ , \tag{6}$$

where $\mu = \sin\theta$, and the overbar denotes complex conjugation. This is done numerically in two steps, with the aid of a Gauss-Legendre quadrature which is computed on a Gaussian grid $\{(\lambda_i,\mu_j),\ i = 0,\ldots,2N_\theta - 1,\ j = 1,\ldots,N_\theta\}$, where the λ_i's are uniformly spaced and the μ_j's are the roots of the Legendre polynomial $P_{N_\theta}^0(\mu)$. We first compute the direct Fourier transforms by

$$f^m(\mu_j) = \frac{1}{2N_\theta} \sum_{l=0}^{2N_\theta - 1} f(\lambda_l,\mu_j) e^{-im\lambda_l} \tag{7}$$

for all j and m and then compute the direct Legendre transforms by

$$\hat{f}_n^m = \sum_{j=1}^{N_\theta} w(\mu_j) f^m(\mu_j) P_n^m(\mu_j) \tag{8}$$

for all m and n, where $w(\mu_j)$ are the Gauss-Legendre quadrature weights. Relation (5) is used for the computation of $\hat{u}_n^m$. The solution values on the Gaussian grid are obtained by computing the inverse Legendre transforms by

$$u_m(\mu_j) = \sum_{n=|m|}^{|m|+M} \hat{u}_n^m P_n^m(\mu_j) \tag{9}$$

for all j and m followed by the inverse Fourier transforms

$$u(\lambda_l,\mu_j) = \sum_{m=-M}^{M} u_m(\mu_j) e^{im\lambda_l} \ . \tag{10}$$

We notice that the whole solution process consists of performing (Fast) Fourier transforms in steps (7) and (10) and discrete Legendre transforms in steps (8) and (9).

The code employs the FFT routine FFT771 from (Temperton 1983). The coefficients and the roots of the Legendre polynomials are computed using routines from the software package Spherepack by Adams and Swarztrauber (Swarztrauber 1984).

3. PARALLELIZATION STRATEGIES

3.1 The Multigrid Method

A standard technique for parallelizing grid-point problems, such as those arising from discretized partial differential equations, is grid partitioning. It is also a natural choice when parallelizing multigrid schemes (cf. Hempel and Schüller 1989, Brandt 1981). The idea is to divide the computational domain into subdomains, each processor being responsible for computing the solution in one of them. For two dimensional applications on rectangular computational domains, one usually subdivides the domain into $N_x \times N_y$ subrectangles of equal size, with $N_p = N_x \times N_y$ being the total number of processors. The parallelism comes from the locality in the computations to be performed. For example, when computing the residuals (from a finite difference operator) at a grid-point, information is needed only from the surrounding grid-points. Thus, with the possible exception of grid-points lying near the boundaries of the subdivision, all the information needed is already within the processor. It is therefore rather convenient to provide each processor with some extra grid-lines near its boundaries, where copies of the data lying in neighbouring processors can be stored. In this way, all information required for the computations, also near the boundaries, will be available and the programming will be simplified. Communication is needed to update the information in these *overlapping regions*, whenever the corresponding data is modified in one of the processors. A grid partition and the corresponding overlapping regions are represented in Figure 1.

The actual choice of a grid partition should be based on efficiency considerations. To illustrate this point, consider a pointwise Jacobi-relaxation. The use of a stripwise partition ($1 \times N_p$ or $N_p \times 1$) minimizes the number of messages to be sent (each processor then has only two neighbours) for updating the overlapping regions, but maximizes the message lengths, which are proportional to the perimeter of the subrectangles. On the other hand, an $N_p{}^{1/2} \times N_p{}^{1/2}$ partition would minimize the message lengths, but maximize the number of messages. The best choice thus depends on the problem size, the number of processors, the communication start-up latency and the communication bandwidth of the system used.

In the full multigrid algorithm we mainly deal with local computations, with the exception of the line-relaxation, which couples the unknowns common to a latitude circle in a cyclic tridiagonal system. The solution of tridiagonal systems on distributed memory multiprocessor systems has been extensively studied by Krechel et al 1990. They showed that the simultaneous solution on N_p processors of several tridiagonal systems requires on the order of $\log N_p$ communication steps with message lengths linearly proportional to the number of systems. We made estimates based on their results and on actual communication and computation times on the iPSC/2 (see also Bomans and Roose 1989), in order to choose the best approach for parallelizing the full multigrid algorithm. According to these estimates, the best strategy is to use a longitudinal stripwise partition, thus avoiding the communication for solving the

tridiagonal systems, which will then lie completely within a single processor. The same conclusion is valid also when vectorization is taken into account, although a meridional partition may be competitive in this case.

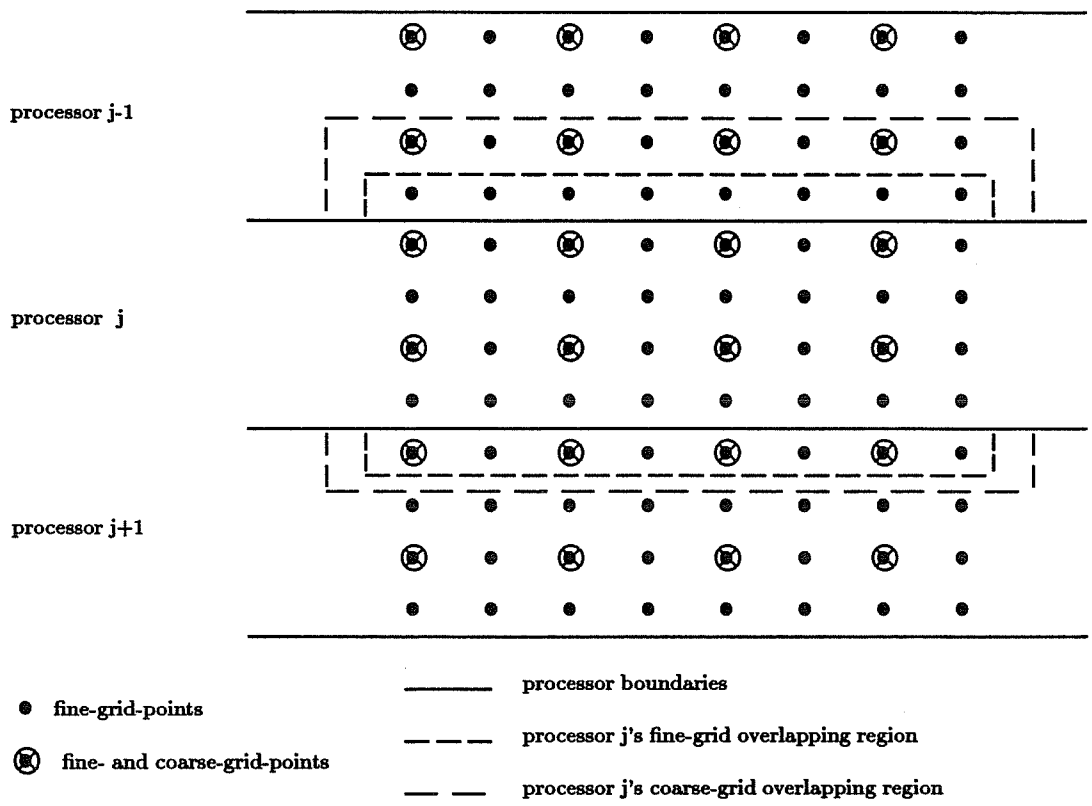

Figure 1 - Grid-point allocation to processors

The solution of the cyclic tridiagonal systems is the most time consuming part of the algorithm. In our scalar parallel computations we apply a block Gaussian elimination (requiring 14 FLOPS per unknown) for this purpose. However, this routine itself is neither parallelizable nor vectorizable. We thus change to cyclic reduction (amounting to 17 FLOPS per unknown) when vectorizing the code. We make no use of precomputations in the solution. Using precomputed bidiagonal LU-factors would reduce the operation counts of both tridiagonal solvers and, therefore, would give a potential increase in the final efficiency (reduction in the wall clock time) of the full multigrid algorithm.

We now give a more detailed description of the parallel algorithm. Each of the $N_p = 2^{M_P}$ processors will be given $N_l = N_\theta/N_p = 2^{M_\theta - M_P}$ latitude circles to treat. The processors will be connected in a ring topology (easily imbedded in a hypercube), each one being directly connected to its northern and southern neighbour. The last processor treats N_l latitude lines plus the north pole, the first is responsible for the south pole and $N_l - 1$ latitude circles (incurring a small load imbalance). Every processor is provided with one extra latitude line at its north and its south for storage

of data from the neighbouring processors; we thus work with overlap-width equal to one (see Figure 1). We assume that at least two latitude circles are allocated to each processor.

In the initialization phase of the algorithm, each processor is provided with the information necessary for the communication, such as its own processor number and the number of the first and the last lines of latitude located in its immediate neighbour processors. A global numbering of the latitude lines - from south to north - is employed. Also the right-hand-side values on these lines are provided.

The algorithm commences with the relaxation process. The residual is smoothened with a relaxation sweep on the odd-indexed lines, which can all be treated independently. After this half-sweep the values on the overlapping regions must be updated. Each processor (but the first) sends its new first line values to its southern neighbour. Accordingly, every processor (but the last) receives an updated latitude line from its northern neighbour and stores these values on the corresponding overlapping line. The new values are needed for the second half-sweep of the relaxation on the even lines, after which another exchange of updated values is necessary. Every processor (but the last) sends now its last latitude line to its northern neighbour and receives information from the south to update its overlapping region (with the exception of the first processor).

The processes of interpolation and residual evaluation and transfer are completely local and therefore trivially parallelizable, since all the data needed for these is already stored in each processor after a complete relaxation sweep. When interpolating the correction to the next finer grid in the V(1,1)-Cycle, the values needed for bilinear interpolation are already available in every processor (see Figure 1). However, the bicubic interpolation employed in the full multigrid solution interpolation stage requires an extra coarse grid line near the processor boundaries. Therefore, in the full multigrid interpolation stage, every processor sends to its southern and northern neighbours also the coarse grid solution values of its second and next-to-last latitude line, respectively, thus providing each processor with a second extra line in the north and in the south, but only for coarse grid values. The bicubic interpolation can then be performed. In reality, only the values at even-indexed fine grid latitude lines are interpolated, since the odd line values would be immediately updated (and never used) by the subsequent relaxation sweep.

We have made the assumption that each processor is given at least two latitude lines. While this is a reasonable assumption for the finer levels in the multigrid process, we shall eventually reach a coarse grid where this rule will be violated, unless we stop the coarsening process too early and work with a relatively fine coarsest grid. In this case, however, computing the coarsest grid solution - directly or iteratively - becomes costly. It is better to reduce the number of active processors and keep coarsening the grid. Our choice is to use only one processor on the coarsest grid (a 45 degree uniform grid), four processors on the next finer grid, and then to double the number of active processors when passing to finer levels, until reaching the total number of available processors. When reducing the number of active processors (com-

ing from finer to coarser levels) we will need to collect data from 2 or 4 processors into one (*agglomeration process*) and then to redistribute the data (*deagglomeration process*) when reactivating nodes on the way back to finer grids. The agglomeration and deagglomeration processes are implemented with the aid of a special routine for this purpose provided in the Suprenum communications library (Hempel 1987).

Our parallel multigrid algorithm has been designed so that it exactly reproduces the results of a sequential version, independently of the number of processors employed. This has been very convenient in the debugging phase, since we could check the code for correctness during all stages of development.

3.2 The Spectral Method

We consider two strategies to parallelize the spectral method, both using a longitudinal stripwise data allocation to processors when computing the Fourier transforms. In this way, all the data needed for each individual FFT will be stored on the same processor and communication can be avoided at this stage of the algorithm. Parallelism is achieved by performing several FFT's simultaneously in different processors. The strategies differ in the treatment of the Legendre transforms.

The *rotation approach* uses a ring topology and rotates the data one full cycle during the computation of the Legendre transforms. The data is the Fourier transformed right-hand-side in the direct transform and the Legendre transformed spectral coefficients in the inverse transform. At each of the N_p stages of the cycle, each processor adds the contribution from the data it has currently in its local memory to the spectral coefficients. After completion of the Legendre transforms, the data for the spectral-to-grid-point FFT's will reside in the right processors and no more communication is needed.

In the *transposition approach*, a global transposition of data from longitudinal to latitudinal stripwise storage is performed between the direct Fourier and Legendre transforms, as well as a back transposition between the corresponding inverse transforms. In this manner, all the data needed for each individual transform will always reside in the same processor, see Figure 2 for an example of both decompositions.

In both approaches, the data from each equatorially symmetric latitude circle pair from the northern and the southern hemisphere will be stored in the same processor. This facilitates the use of the symmetric-antisymmetric properties of Legendre polynomials to reduce the computational work needed to compute each Legendre transform by almost a factor of two. In this respect, we follow the multitasked implementation of the ECMWF spectral model on Cray multiprocessors (Simmons and Dent 1989). The north-south pairs of latitude circles are then uniformly distributed to the processors. We assume that the number of latitudes of the Gaussian grid is a power of two, as well as the number of processors. We also assume that each processor will be assigned at least one pair of latitude circles.

In the first step of the algorithm, each processor has all the data for computing the FFT's on its own latitude circles stored in its local memory. No communication is necessary at this step.

In the rotation approach, the spectral coefficients $\hat{u}_n^m$ will be allocated to processors according to a uniform division on the range of n, so that every processor will compute $\hat{u}_n^m$ for all m and for those values of n allocated to it (see Figure 2). Two arrays of Legendre polynomials, to be used in the direct and inverse Legendre transforms, respectively, will be stored in each processor. One of them contains the values of the Legendre polynomials multiplied by the Gaussian weights and the inverses of the corresponding eigenvalues of the Helmholtz operator. This array will be stored for all values of μ and m, but only for the values of n in the processor's range. The second array stores the values of the Legendre polynomials for all m and n, but just for those latitude circles μ that have been assigned to the processor.

After the Fourier coefficients $f^m(\mu_j)$ have been computed, each processor knows the values of $f^m(\mu_j)$ only for those latitude circles assigned to it. The Legendre transform (8) will thus be evaluated in N_p (number of processors) steps. Each processor begins by accumulating the partial sums corresponding to its own latitude circles. After that it forwards its part of the matrix $f^m(\mu_j)$ to its successor in the ring and receives the part of the matrix contained in its predecessor. Each processor can then accumulate another part of the sums, forward the part of the matrix it now has to its sucessor and receive another part from its predecessor. It is obvious that after N_p steps each processor completes the sums. The inverse Legendre transforms (9) are evaluated in the same way, the parts of the matrix $\hat{u}_n^m$ contained in each processor being rotated around the ring. After the Legendre transforms no communication is needed, since all the data is already in place for evaluation of the Fourier transforms within individual processors.

In the transposition approach, another allocation of the spectral coefficients $\hat{u}_n^m$ is used, with a uniform division on the range of m (see Figure 2). For the Legendre transforms, two precomputed arrays of Legendre polynomials will again be stored (one with the values multiplied by the Gaussian weights and the inverses of the eigenvalues of the Helmholtz operator), for all values of n and μ, but with m restricted to the values assigned to the processor in the case of both arrays.

After the Fourier transforms (7), a complete rearrangement of the data in the processors is performed. The matrix $f^m(\mu_j)$, of which each processor stores the values corresponding to the latitude circles assigned to it, will be 'transposed', so that each processor will then store the matrix values for entire longitude strips, but just for the range of values of m allocated to it. In order to do that, every processor has to exchange a submatrix with every other. This can be accomplished in $N_p - 1$ steps in a ring topology. In step k, each node sends a submatrix to the processor that lies k positions ahead in the ring and receives a submatrix from the processor k positions behind. After this block-transposition, each processor is able to transpose its submatrices locally. After the transposition, the computation of both Legendre transforms (8) and (9) can be done within each processor. The transforms are followed

by a back transposition of the array $u_m(\mu_j)$, so that the FFT's (10) can again be performed within individual processors. This completes the solution process.

Both approaches require the same amount of computation, differing only in the communication. Rotation involves $2N_p$ messages sent per processor, N_p of them of size $2MN_\theta/N_p$ and the remaining N_p of size $2M^2/N_p$. The factor 2 emerges because the spectral and Fourier fields, of size M^2 and MN_θ, respectively, are complex valued. Assuming $M = N_\theta$, $4M^2$ words will be sent by each processor (always to direct neighbours on the ring). Transposition requires $2(N_p - 1)$ messages per processor; half of them of length $2MN_\theta/{N_p}^2$, half of them of length $2M^2/{N_p}^2$. Again with $M = N_\theta$, we obtain a total of $4M^2/N_p$ words to be sent by each processor. In this case, each processor communicates with every other processor and the messages will have to travel across the entire machine. For the iPSC/2, for which the communication bandwidth is not very sensitive to the number of hops between two communicating nodes (cf. Bomans and Roose 1989), the transposition approach is more efficient.

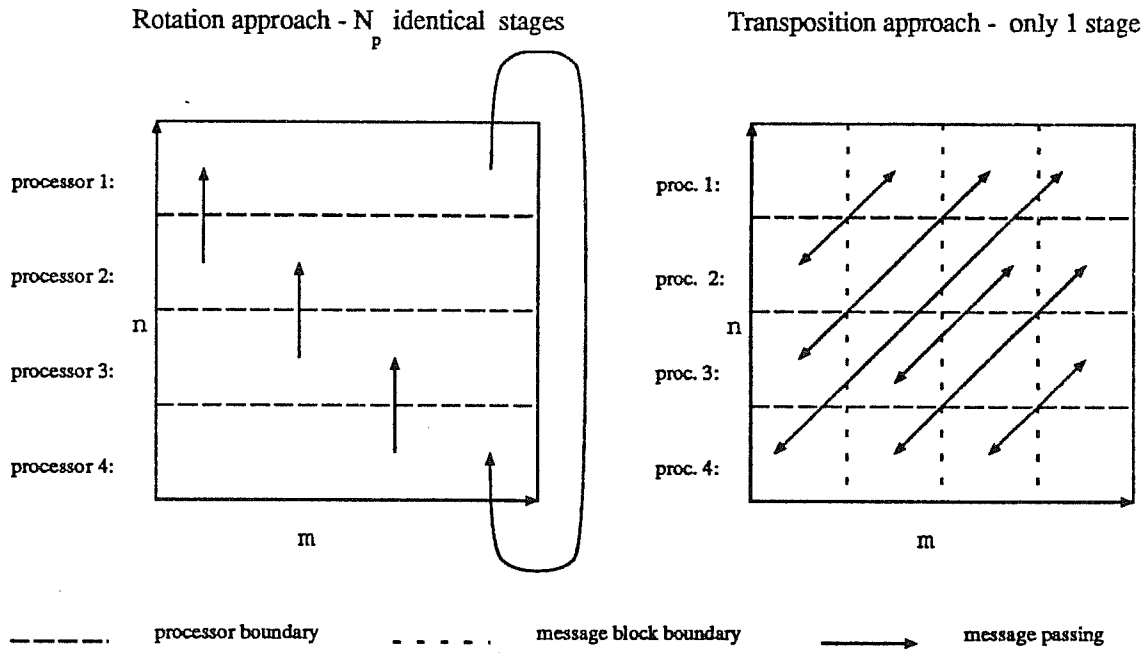

Figure 2 - Allocation and communication of spectral data in the rotation and transposition approaches.

4. PARALLEL BENCHMARKS

4.1 The programming environment and the characteristics of the iPSC/2

In our choice of parallelization strategies for the Helmholtz solvers we did not

rely on the particular parallel architecture of any machine (although we used the parameters of the iPSC/2 for estimates). We assumed that we would be working on a distributed memory machine with message passing communications. The actual implementation of the algorithms is also independent of the particular machine at hand to a certain extent, since it uses parallelization tools which are available on a number of parallel computers with different architectures.

For instance, mapping the problem onto the parallel topology was done with the aid of a routine from the Suprenum communications library (Hempel 1987). In this routine, the ring topology is mapped onto the machine and the neighbourhood relations between the processors are defined automatically. Each processor then knows his own adress and those of his successor and predecessor on the ring. Also, the initial information and the parameters defining the problem are made available to every node by the host.

Communication during the algorithms is implemented using macro-constructs (Bomans and Hempel 1990), which make commands like send and receive portable, albeit somewhat less efficient. By writing all the communication instructions either with the macros or with the aid of library routines, a certain amount of portability has been achieved. For instance, the same codes used for benchmarking on the iPSC/2 have run essentially unchanged on a prototype of the Suprenum computer (with up to eight nodes, within one cluster). We would also expect no difficulties in making the programs run on other parallel machines where the macro-constructs have been (or may be) implemented.

We now give some characteristics of the Intel hypercube we have used for the benchmarks presented here. It is an iPSC/2 with 32 node processors, each one being an Intel 80386 equipped with an 80387 numeric coprocessor and an AMD vector extension board. Each node has a local memory of 4 Mbytes in the scalar mode. When the vector board is used, the available memory is reduced to 1 Mbyte per node.

The vector board delivers a peak performance of 2.6 Mflops (measured) for DAXPY operations (cf. Intel 1988). The iPSC/2 has a relatively high $n_{1/2}$ number (minimal vector length for achieving 50% of the peak performance); it lies around 50 for all vector operations. Scalar double-precision performance is around 200 Kflops.

The nodes are interconnected in a hypercube topology by full-duplex channels. They are equipped with a Direct Connect Module (DCM) for handling message passing. The communication times can be modelled as an affine function of the message length. A feature of the iPSC/2 is that very short messages (up to 100 bytes) are handled in a different manner from longer messages. For short messages, communication has a 370 μs start-up time and a transfer rate of 1.5 μs per double-precision word. For longer messages (more than 100 bytes) the start-up time increases to 700 μs and the transfer rate decreases to 2.9 μs per double-precision word. The maximum communication bandwidth is around 2.7 Mbytes / sec (Bomans and Roose 1989). The use of the DCM for routing messages makes the iPSC/2 fairly insensitive to the number of hops between communicating processors. The overhead for communication over

several links in comparison to nearest neighbour communication is less than 17% for 104 byte messages, dropping quickly for longer messages (around 5% for a 4 Kbyte message).

4.2 Efficiency measurements

The parallel performance of an algorithm can be measured by its efficiency and speedup. The speedup on N_p processors is defined as

$$S(N_p) = T(1)/T(N_p) \quad , \tag{11}$$

the ratio between the CPU-times for sequential execution (in one processor) and for running on N_p processors. Correspondingly, the parallelization efficiency on N_p processors is given by

$$E(N_p) = T(1)/(N_p T(N_p)) \quad . \tag{12}$$

Normally, a linear speedup ($S(N_p) = N_p$) is the best achievable, corresponding to 100% efficiency. Evaluation of speedup and efficiency of a given problem on N_p processors may be complicated by memory constraints for the computation of $T(1)$, unless the problem size on N_p processors is restricted to be unrealistically small. One way to circumvent this problem is to estimate $T(1)$, based on measured CPU-times for smaller problems, which fit in one processor. This is rather simple to do for the multigrid solver, since its linear computational complexity allows for a simple linear extrapolation, with very accurate predictions. For the spectral code the extrapolation is somewhat more complicated, requiring separate estimates for Legendre and Fourier transforms, but still reasonable. However, if vectorization is used, these extrapolations will be useless, since the changes in vector lenghts (and consequently in performance) are not accounted for in such estimates. We prefer to consider the vector speedup separately from the parallel speedup. Vector speedup, as used here, only measures the extra gain achieved by vectorization of the problem. It is defined as

$$S_v(N_p) = T_s(N_p)/T_v(N_p) \quad , \tag{13}$$

the ratio between the scalar and vector performance of the algorithm on N_p processors.

All the quantities defined will only measure how well the algorithms are vectorized and parallelized. They say nothing about the absolute performance of the algorithms. Often the most easily parallelizable algorithms are not the fastests. The absolute CPU-time required to solve the problem is still the most fundamental measure of the performance of an algorithm.

When employing a supercomputer in scientific modelling, the aim is usually to solve the largest possible problem in the smallest possible time. This aim motivates the adoption of a new measure of parallel performance, namely *scaled speedup* (Gustafsson, Montry and Benner 1988), which is defined as

$$S_{scaled}(N_p) = rT_{m1}(1)/T_{mp}(N_p) \quad , \tag{14}$$

where r is the ratio between problem sizes $m1$ and mp, the largest problems fitting in one and in N_p processors, respectively. T_{m1} and T_{mp} denote the corresponding CPU-times.

Scaled speedup is defined with respect to problem size. Problem size may have different meanings depending on which problem parameters are held fixed and which are allowed to vary. In the case of the numerical solution of partial differential equations, at least three different measures of problem size can be employed: grid size, serial complexity and solution accuracy.

Grid size is probably the most common way to refer to problem size in numerical solution of partial differential equations. When the fastest serial method is used as a base line for the time T_{m1} at each grid size, the comparison gives credit to the methods that produce the best parallel speedup over the fastest serial method. A suboptimal serial complexity has therefore an equally severe impact on the scaled speedup score of a method as poor parallelizability. On the other hand, using grid size as a measure of problem size ignores the fact that some methods produce accurate solutions already on relatively coarse grids when the solution is sufficiently smooth. It may also be that there are other reasons for using a certain discretization technique, such as the ability to use existing software. If this is the case, the base line for the time T_{m1} should always be the same solution method whose speedup is being studied. This means that scaled speedup is measured with respect to serial complexity, rather than grid size.

Serial complexity is an appropriate measure of problem size when the task is to study the parallel efficiency of a certain solution method, rather than solving the given problem using any method. Using serial complexity to define scaled speedup often gives credit to 'brute force' methods that are easy to parallelize efficiently due to their simple structure. It ignores differences in both solution accuracy and serial complexity between methods. Using serial complexity to define problem size in scaled speedup may sometimes be misleading, since optimal methods in terms of both accuracy and serial complexity often have a more complex structure, and therefore poorer parallel efficiency at low resolutions, than simple but suboptimal methods.

Accuracy is probably the most appropriate measure of problem size of all, whenever it can unambiguously be defined. Defining an accuracy measure amounts normally to selecting a norm of the error or the residual. The appropriate norm depends, however, crucially on solution smoothness. This may be difficult to determine in, for example, the case of atmospheric fields. Smoothness of a field depends on the scale of atmospheric motion to be studied. It is also dependent on the smoothness of other fields, since atmospheric motion is described using a system of partial differential equations. The particular problem or set of problems used to compare different methods may also cause an undue bias to scaled speedup comparisons, if the problems used do not represent typical 'real-life' solutions.

In the following comparisons and the discussion, we try to pay some attention to the impact of different definitions of problem size on scaled speedup.

4.3 Benchmark results

4.3.1 Scalar results

We first present the results for purely scalar computations. The solution of the example problem used has approximately third order smoothness, but is wholly artificial. It has an analytic solution. As a measure of accuracy, the maximum deviation of the numerical solution from this analytic solution is given in each table.

Table 1 contains the CPU-times for solution of the Helmholtz equation with the spectral method. Several combinations of problem sizes and numbers of processors are considered. The parallelization efficiency (12) is given in parentheses. Some computing times in one processor had to be estimated (due to memory constraints), to allow for the evaluation of the efficiency of some problems. The transposition approach has been employed for the results of Table 1. In Table 2 we present results for the rotation approach. The rotation approach produces longer execution times in all cases. Table 3 presents the total execution times for the full multigrid solver, and Table 4a gives the scaled speedups of both algorithms with respect to grid size, Table 4b with respect to serial complexity. The multigrid numbers stay the same in both of the tables 4a and 4b, thanks to its linear complexity with respect to grid size. The spectral numbers scale by a factor between 7 and 8 (the exact scaling factor at each grid size was calculated from the serial complexity), instead of 4, with grid size because of the quadratic serial complexity of the Legendre Transform and the $\mathcal{O}(n \log n)$ serial complexity of the Fast Fourier Transform. In Figures 3 to 5 we present a graphical comparison between the CPU-times required by the schemes for the solution of the equations.

Problem Size	Number of Processors					
max error	1	2	4	8	16	32
16×8	34	21	15			
$8.17 \cdot 10^{-3}$		(80.9)	(56.7)			
32×16	197	105	63	42		
$1.33 \cdot 10^{-3}$		(93.8)	(78.2)	(58.6)		
64×32	1330	679	351	206	129	
$2.09 \cdot 10^{-4}$		(97.9)	(94.7)	(80.7)	(64.4)	
128×64	9198	4624	2331	1195	656	402
$3.15 \cdot 10^{-5}$		(99.5)	(98.6)	(96.2)	(87.6)	(71.5)
256×128	65700*			8423	4283	2291
$4.60 \cdot 10^{-6}$				(97.5)	(95.9)	(89.6)

Table 1- CPU-times (in milliseconds) for the parallel execution of the spectral method (transposition approach). Efficiency is given in parentheses. (* - estimated)

Problem Size	Number of Processors					
max error	1	2	4	8	16	32
16×8	34	24	24			
$8.17 \cdot 10^{-3}$		(70.8)	(35.4)			
32×16	197	110	71	62		
$1.33 \cdot 10^{-3}$		(89.5)	(69.4)	(39.7)		
64×32	1330	688	374	227	174	
$2.09 \cdot 10^{-4}$		(96.7)	(88.9)	(73.2)	(47.8)	
128×64	9198	4636	2405	1290	761	531
$3.15 \cdot 10^{-5}$		(99.2)	(95.6)	(89.1)	(75.5)	(54.1)
256×128	65700*				4727	
$4.60 \cdot 10^{-6}$					(86.9)	

Table 2- CPU-times (in milliseconds) for the parallel execution of the spectral method (rotation approach). Efficiency is given in parentheses. (* - estimated)

Problem Size	Number of Processors					
max error	1	2	4	8	16	32
128×64	4642	2572	1451	918	688	620
$6.80 \cdot 10^{-4}$		(90.2)	(79.9)	(63.2)	(42.2)	(23.4)
256×128	18502	9662	5056	2793	1706	1238
$1.71 \cdot 10^{-4}$		(95.7)	(91.5)	(82.8)	(67.8)	(46.7)
512×256	74008*	37739	19180	9963	5398	3212
$4.27 \cdot 10^{-5}$		(98.1)	(96.5)	(92.9)	(85.7)	(72.0)
1024×512	296032*			38222	19680	10526
$1.07 \cdot 10^{-5}$				(96.8)	(94.0)	(87.9)
2048×1024	1184128*					39732
$2.67 \cdot 10^{-6}$						(93.1)

Table 3- CPU-times (in milliseconds) for the parallel execution of the full multigrid algorithm. Efficiency is given in parentheses. (* - estimated)

Processors	2	4	8	16	32
Spectral Method	1.99	3.95	4.37	8.59	16.06
Multigrid Method	1.96	3.86	7.74	15.04	29.79

Table 4a- Scaled speedups of the spectral (transposition approach) and of the multigrid method with respect to grid size. (The values refer to scaling of the last entry in each column of tables 1 and 3.)

Processors	2	4	8	16	32
Spectral Method	1.99	3.95	7.80	15.33	28.67
Multigrid Method	1.96	3.86	7.74	15.04	29.79

Table 4b- Scaled speedups as in Table 4a, but with respect to serial complexity.

4.3.2 Vector results

The vectorization of the codes was done with the aid of the VAST-2 vectorizer, supported by compiler directives where necessary. The codes are still not fully optimized for vectorization. In the spectral solver, we did not attempt to vectorize the FFT's. The Legendre transforms were easily vectorized, running up to 7.7 times faster (on 32 processors, with M=127) than in scalar mode. In the multigrid scheme, the cyclic tridiagonal solver was only vectorized for systems with more than 32 unknowns (cyclic reduction was employed in this case). For up to 32 unknowns, Gaussian elimination on scalar mode was faster and thus preferred. This shows the effect of the large $n_{1/2}$ value of the iPSC/2. Tables 5 and 6 give the total execution times for the vectorized spectral and multigrid algorithms, respectively. In addition to CPU-times, also vector speedups (13) are given (in parentheses). Tables 7a and 7b contain the scaled speedups in vector mode, while a graphical comparison of the CPU-times required by both algorithms is presented in Figures 3 to 5.

Problem Size	Number of Processors					
max error	1	2	4	8	16	32
16×8	46	26	19			
$8.17 \cdot 10^{-3}$	(0.74)	(0.81)	(0.79)			
32×16	170	93	58			
$1.33 \cdot 10^{-3}$	(1.16)	(1.13)	(1.09)			
64×32	810	423	226	137	93	
$2.09 \cdot 10^{-4}$	(1.64)	(1.61)	(1.55)	(1.50)	(1.39)	
128×64			989	523	313	204
$3.15 \cdot 10^{-5}$			(2.36)	(2.28)	(2.10)	(1.97)
256×128						733
$4.60 \cdot 10^{-6}$						(3.13)

Table 5- CPU-times (in milliseconds) for the parallel execution of the vectorized spectral method (transposition approach). Vector speedups are given in parentheses.

Problem Size	Number of Processors					
max error	1	2	4	8	16	32
128×64	2851	1655	1002	695	575	553
$6.80 \cdot 10^{-4}$	(1.63)	(1.55)	(1.45)	(1.32)	(1.20)	(1.12)
256×128		4436	2450	1480	1040	886
$1.71 \cdot 10^{-4}$		(2.18)	(2.06)	(1.89)	(1.64)	(1.40)
512×256				3555	2170	1559
$4.27 \cdot 10^{-5}$				(2.80)	(2.49)	(2.06)
1024×512						3271
$1.07 \cdot 10^{-5}$						(3.22)

Table 6- CPU-times (in milliseconds) for the parallel execution of the vectorized full multigrid algorithm. Vector speedups are given in parentheses.

Processors	2	4	8	16	32
Spectral Method	1.91	3.28	6.19	10.35	17.70
Multigrid Method	2.58	4.64	12.80	20.96	55.68

Table 7a- Scaled speedups of the vectorized spectral (transposition approach) and of the multigrid method with respect to grid size. (The values refer to scaling of the last entry in each column of tables 5 and 6.)

Processors	2	4	8	16	32
Spectral Method	1.91	5.85	11.05	18.47	59.85
Multigrid Method	2.58	4.64	12.80	20.96	55.68

Table 7b- Scaled speedups as in Table 7a, but with respect to serial complexity.

5. DISCUSSION

The speedups attained show that both algorithms admit significant gains from parallel computation. For the largest problems considered, a parallel efficiency of more than 90% was attained with 32 processors for both algorithms. Both solvers therefore present a good degree of parallelism.

The two solution methods differ, however, in the grain of parallelism necessary to attain good performance (say, a parallel efficiency exceeding 50%). While for the spectral method only one pair of latitude lines allocated to each processor is sufficient to achieve this, four to eight latitude lines in each node were needed in the multigrid algorithm for the same efficiency. The larger granularity of parallelism necessary for the multigrid algorithm is compensated for by its more modest memory requirements: ($\mathcal{O}(n^2/N_p)$ words per processor against $\mathcal{O}(n^3/N_p)$ for the spectral method). Therefore, significantly larger two dimensional problems can be solved with multigrid.

The question of the granularity of parallelism becomes more important if extension of the algorithms to three dimensions is considered. An exploitation of parallelism in just one dimension, as is done at present with both algorithms, would be too restrictive for multigrid in 3D, leading to too small a grain size even on modestly parallel computers. For the spectral method, it seems possible that the one-dimensional parallelization strategy can still be succesfully applied in three dimensions. However, by allocating all data in a vertical column to one processor and by making the vertical dimension the innermost loop, the granularity of both algorithms could probably be kept at an acceptable level. On highly parallel computers, it would probably be most natural to amend parallelization over latitudes by parallelization in the vertical. This strategy would call for another data reorganization stage when computing physics - i.e. the right hand side - in real atmospheric models, though.

The memory constraints on the spectral method caused by the need to store the coefficients of the Legendre polynomials would also be relatively weakened when solving 3D problems, since the same polynomial values can be used on all vertical levels. However, a lot of memory would now be required by the matrices for the vertical eigendecomposition of the original 3D Laplace operator that gave rise to the 2D Helmholtz equations. Ideally, these full K by K matrices - one for each vertical column - where K is the number of vertical layers, should be stored on a fast access disk, and in fact, the I/O bandwidth may be critical for parallel systems. This important question has not been addressed here, partly because the parallel machine employed had no disks allocated to individual processors.

On a grid-point model, one could opt not to decouple the three-dimensional equation and instead, apply a three-dimensional multigrid solver. This has the potential advantages of reducing the computational complexity of the scheme (since the multigrid solver has linear complexity, cf. Barros 1991, and the decoupling process requires a computational work, which is quadratic in the number of vertical layers) and of a drastic reduction in the memory requirements, by avoiding the storage of the matrices for eigendecomposition. The parallel multigrid code should, however, exploit parallelism in at least two dimensions.

The vectorization of the codes brought speedups up to a factor three when solving large problems. These results were somewhat limited by the characteristics of the iPSC/2, which has a high $n_{1/2}$ value. Especially the multigrid solver would greatly benefit from a smaller $n_{1/2}$, since it employs a sequence of coarser grids and uses cyclic reduction in the solution of tridiagonal systems. The speedups achieved in the spectral method came from the easily vectorizable Legendre transforms. Improvements can be achieved by the vectorization of the FFT's, which has not been tried.

From the comparison between the spectral and the multigrid solver presented in Figures 3 to 5, we can see that for smaller grain parallelism the spectral code performs better. The spectral method also gains more from the vectorization on small problems. However, when going to larger problems and to a larger grain parallelism, multigrid quickly improves its performance, while the $\mathcal{O}(n^3)$ complexity of the Legendre transform begins to tell on the spectral method. With 32 processors,

the break-even point between the two vectorized algorithms is close to a 256×128 grid. By this problem size, the spectral method already achieves a parallel efficiency of more than 90% and an extra vector speedup of more than 3, while the multigrid still performs with 47% efficiency and obtains a vector speedup of 1.4. Halving the meshsizes increases the efficiency of multigrid to 72% and its vector speedup to more than 2. For a problem of this size, multigrid should be about twice faster than the spectral method, which could not be run at this grid size due to memory limitations.

The error norms in Tables 1 to 3 and 5 to 6 reveal another characteristic property of each method. The test problem used in the benchmarks has a solution with approximately third order smoothness. When the maximum errors of each method are plotted on a log-log scale as a function of grid size, the spectral method is seen to decrease the error at an approximate rate of $\mathcal{O}(h^3)$, whereas the multigrid method is limited to $\mathcal{O}(h^2)$ accuracy by the five-point stencil employed in the discretization. Hence, in the particular case of a solution with three bounded spatial derivatives, the spectral and multigrid methods are asymptotically equally effective in reducing the error: the suboptimal complexity of the spectral method and the suboptimal accuracy of the multigrid method with a second order finite difference operator cancel one another. This trend is also displayed in Figure 4.

Spectral methods can automatically utilize any degree of smoothness present in the solution. In the case of finite differences - like any fixed-order discretization method - accuracy is limited by the order of the discretization. Accuracy can be improved by choosing a higher order discretization, at the expense of a linear increase in the volume of both computation and communication. The impact of a spectrally accurate discretization on the efficiency in reducing the error depends, however, on the smoothness of the fields to be solved. Some atmospheric fields, like geopotential, may have something like three bounded derivatives (cf. Julian et al 1970, Speth and Madden 1983) at present resolutions (in the sense that third order polynomials seem to fit best on the spectrum of observational and spectral model fields up to present operational resolutions). Some other fields - like cloud liquid water - are only square integrable in the same sense, having therefore zeroth order smoothness. On such fields, spectral methods gain nothing from their accuracy. When the right hand side has more than five orders of smoothness - and therefore the solution more than three - the spectral method is always more efficient with respect to accuracy than any second order finite difference method. When the right hand side is less smooth, the break-even point between the methods moves to ever smaller grid sizes, and a finite difference scheme with a multigrid solver is always asymptotically the more efficient on sufficiently large problems.

A remarkable fact is, that the combination of the effects of parallelization and vectorization leads to a superlinear scaled speedup. By superlinear scaled speedup we understand that computationally N_p times more demanding a problem runs faster on N_p processors than the initial problem runs on one processor. Because of the linear complexity of the multigrid algorithm, it exhibits superlinear scaled speedup with respect to both serial complexity and grid size, whereas the spectral method only shows superlinear scaled speedup with respect to serial complexity, i.e. floating point

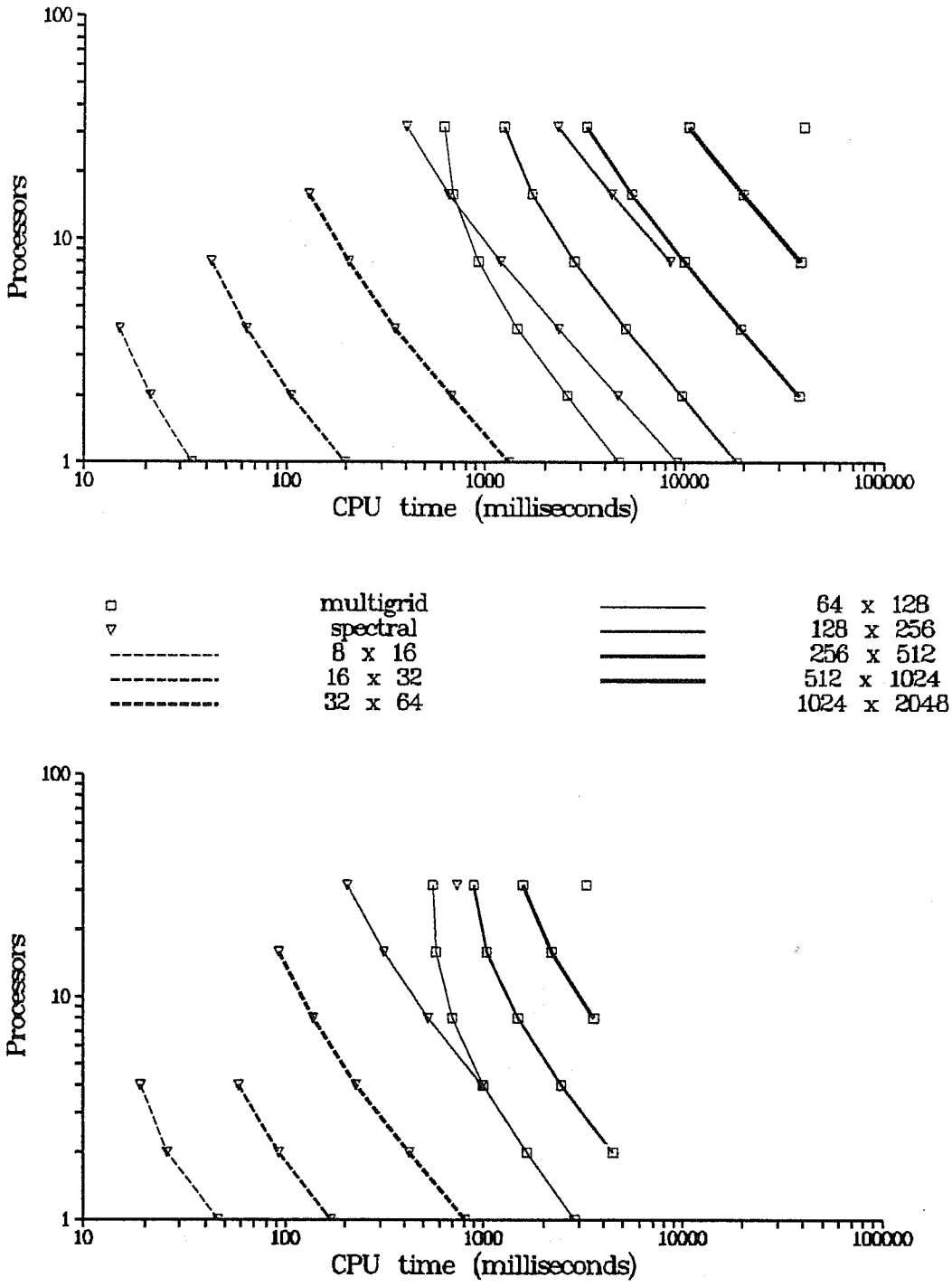

Figure 3 - Graphical representation of the CPU-times to solve a Helmholtz equation with the spectral and multigrid methods. The curves show the variation of the CPU-times when solving problems of a fixed grid size with an increasing number of processors (from 1 up to 32). Grid sizes from 16×8 up to 2048×1024 are considered. Scalar (above) and vector (below) results are represented.

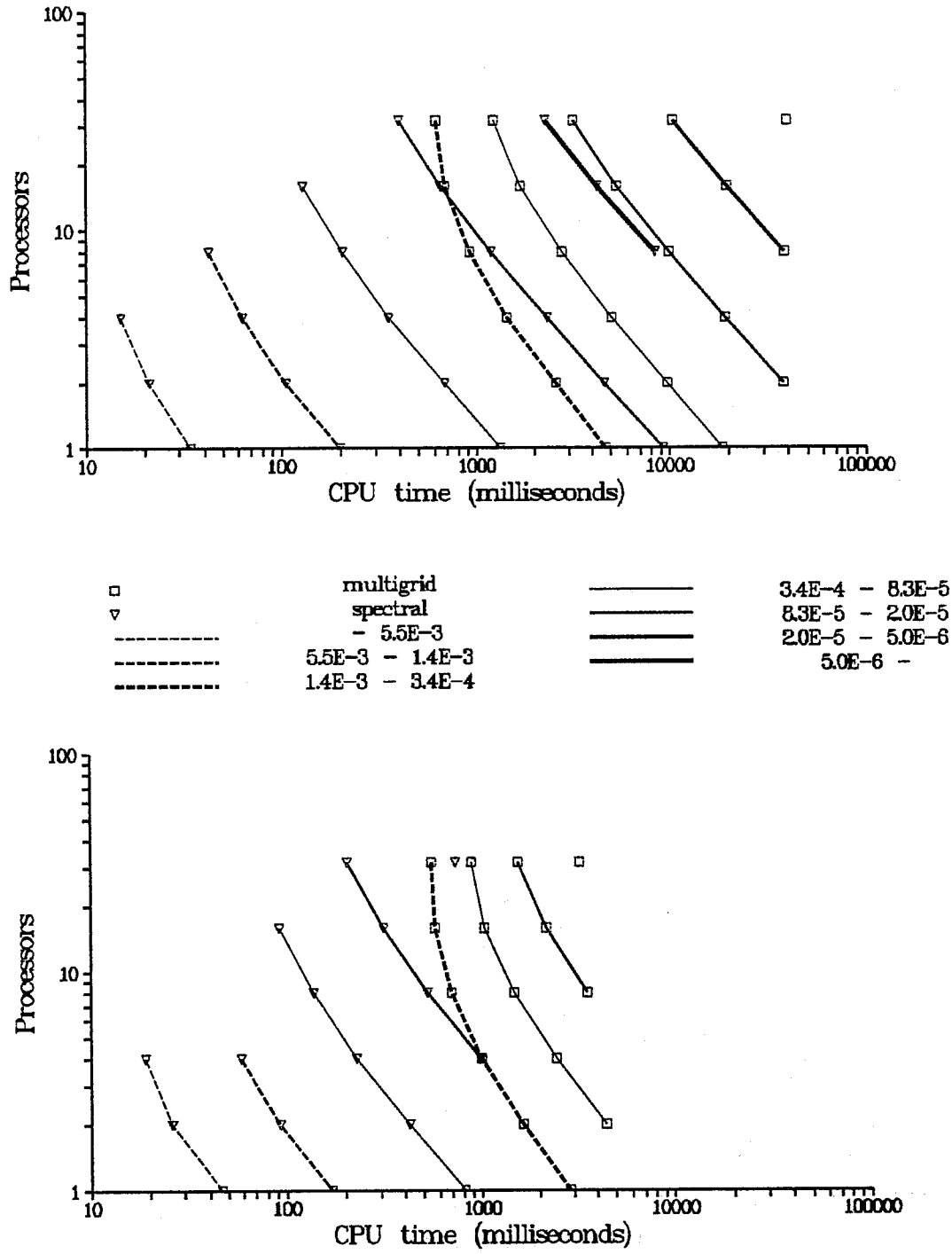

Figure 4 - Graphical representation of the CPU-times to solve a Helmholtz equation with the spectral and multigrid methods. The curves show the variation of the CPU-times when solving the example problem to within a fixed accuracy with an increasing number of processors (from 1 up to 32). Scalar (above) and vector (below) results are represented.

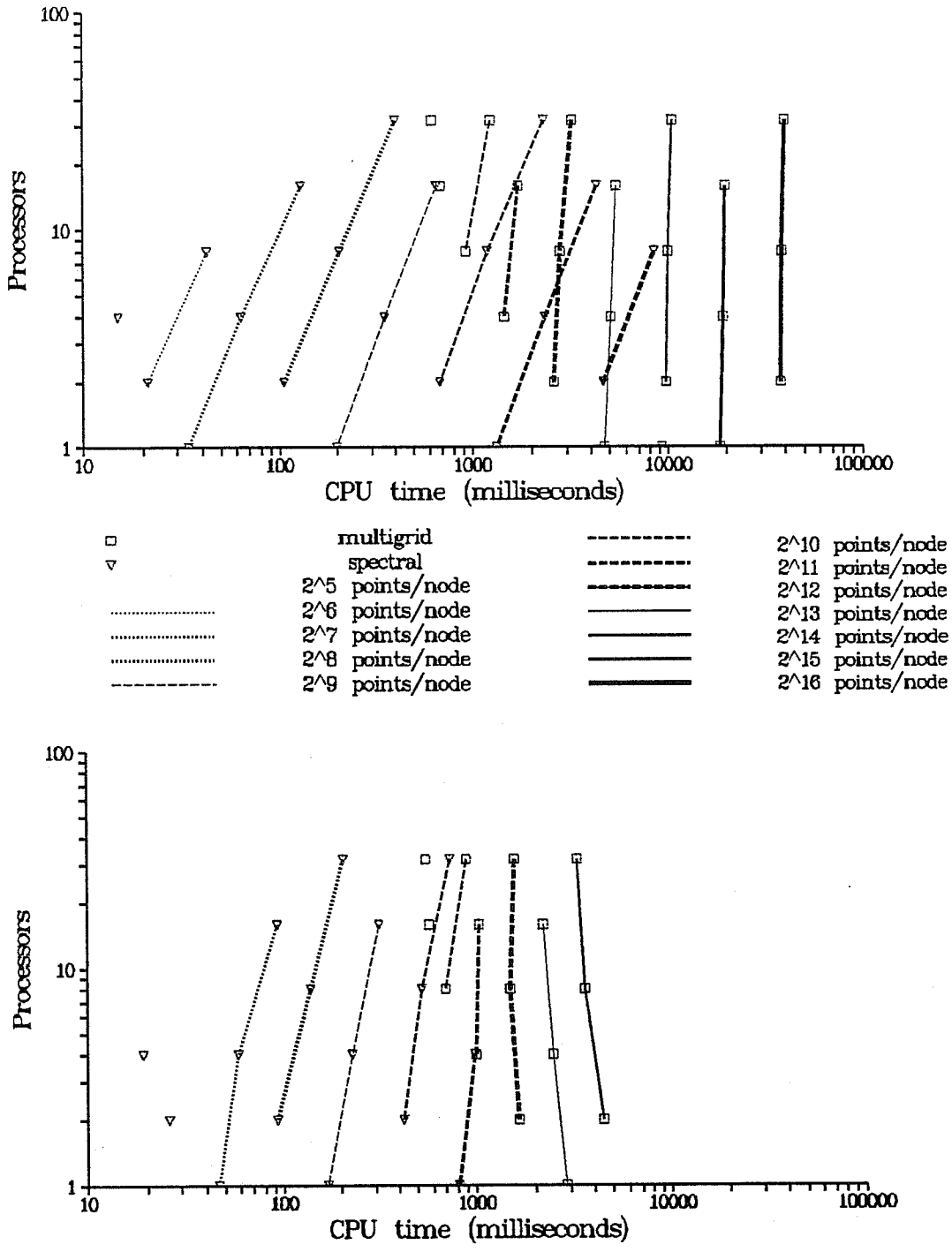

Figure 5 - Graphical representation of the CPU-times to solve a Helmholtz equation with the spectral and multigrid methods. The curves show the variation of the CPU-times when solving problems of increasing sizes (but keeping the number of grid points per processor fixed), with an increasing number of processors (from 1 up to 32). Scalar (above) and vector (below) results are represented. A linear scaled speedup would be represented by a vertical line. Lines sloping to the left indicate superlinear speedup attained by the vectorized multigrid algorithm.

operation count (see Figure 5). Superlinear scaled speedup can be explained by the fact that by scaling problem size up when more processors are employed, we obtain longer vectors within each processor, even though the number of unknowns in each processor remains the same. It is thus a consequence of the stripwise allocation of data. The gains due to a better vectorization more than compensate for the increase in communication, thus leading to a CPU time reduction. For example, a 256×128 problem is solved on 2 processors in 4.4 seconds, while a 1024×512 problem, which is 16 times larger, needs only 3.3 seconds to be solved on 32 processors with the vectorized multigrid method.

6. SUMMARY

The parallel solution of the Helmholtz equation on the surface of a sphere on distributed memory machines was studied. Spherical Helmholtz equations play an important role in all kinds of spectral or grid-point semi-implicit time stepping schemes in global atmospheric models, and can thus be viewed as a kernel problem for the parallelization of global weather models.

We investigate parallelization techniques for a spectral scheme which uses spherical harmonics and a rhomboidal truncation, and for a full multigrid (grid-point) solver, employing line-relaxation. A global data transposition strategy shows to be very convenient for the parallelization of the spectral method. With this approach, all data for a single Legendre or Fourier transform will be placed within the same processor by the time it is needed. For the multigrid algorithm we employ a grid-partitioning technique, with a longitudinal data allocation to processors. This data distribution allows the solution of the cyclic tridiagonal systems needed in the relaxation to be performed within individual processors. Because of the locality of the algorithm, the other components of the method are easily parallelized by grid partitioning.

Numerical benchmarks were calculated on an Intel iPSC/2 hypercube with 32 nodes. A high parallelization efficiency, up to more than 90% (for sufficiently large problems) was achieved. The parallel algorithms were implemented with the aid of the PARMACS macro package (Bomans and Hempel 1990), which gives the codes a certain amount of portability. We could, for example, run the same codes on a prototype Suprenum computer.

The algorithms have also been vectorized, with an extra speedup of up to a factor three. The relatively high $n_{1/2}$ value of the iPSC/2 required, however, fairly large problems for good vector performance. The combined use of vectorization and parallelization leads to remarkable speedups in some cases. For instance, a superlinear scaled speedup of both algorithms with respect to serial complexity has been obtained through the combination of parallelization and vectorization. The multigrid method exhibits superlinear scaled speedup also with respect to grid size.

Acknowledgements

Most of this work was carried out at the GMD (Gesellschaft für Mathematik und Datenverarbeitung), where we had access to their iPSC/2. We are indebted to several people there for the friendly atmosphere and the constant support on technical aspects, as well as for interesting discussions. We mention, in particular, Rolf Hempel, Arno Krechel, Tony Niestegge, Hans Plum and Hubert Ritzdorf.

The first author acknowledges partial support from the project on parallel computing from the BID/USP program. The second author has been supported by the Esprit II program P2702 *Genesis*, by the Academy of Finland and by the Finnish Technology Development Centre TEKES under the program FINSOFT III: Parallel Algorithms.

References

[1] **Barros, S. R. M. 1991**: *Multigrid methods for two- and three-dimensional Poisson-type equations on the sphere.* J. Comput. Phys. 92 (1991), 313-348.

[2] **Barros, S. R. M., Dee, D. P. and Dickstein, F.**: *A multigrid solver for semi-implicit global shallow-water models.* Atmos. Ocean 28 (1990), 24-47.

[3] **Bates, J. R., Semazzi, F. H. M., Higgins, R. W. and Barros, S. R. M. 1990**: *Integration of the shallow water equations on the sphere using a vector semi-Lagrangian scheme with a multigrid solver.* Mon. Wea. Rev. 118 (1990), 1615-1627.

[4] **Brandt, A. 1981**: *Multigrid solvers on parallel computers.* In Elliptic problem solvers. Schultz, M. H. (ed.). Academic Press, New York 1981.

[5] **Bomans, L. and Hempel, R. 1990**: *The Argonne/GMD Macros in Fortran for portable parallel programming and their implementation on the Intel iPSC/2.* Parallel Computing 15 (1990), pp. 119-132.

[6] **Bomans, L. and Roose, D. 1989**: *Communication benchmarks for the iPSC/2.* In Proc. first European workshop on hypercube and distributed computers. Andre, F. and Verjus, J.P (eds.). North-Holland, Amsterdam 1989.

[7] **Côté, J. and Staniforth, A. 1988**: *A two-time-level semi-Lagrangian semi-implicit scheme for spectral models.* Mon. Wea. Rev. 116 (1988), 2003-2012.

[8] **Dent, D. 1988**: *The ECMWF model: past, present and future.* In Multiprocessing in meteorological models. Hoffmann, G.-R. and Snelling, D. F. (eds.). Springer-Verlag, Berlin 1988.

[9] **Dent, D. 1992**: *The ECMWF model on the Cray Y-MP8.* In the proceedings of the Fourth ECMWF Workshop on the Use of Parallel Processors in Meteorology. Hoffmann, G.-R. and Kauranne, T. (eds.), submitted for publication.

[10] **Gustafsson, J.L., Montry, G.R. and Benner, R.E. 1988**: *Development of parallel methods for a 1024-processor hypercube.* SIAM J. Sci. Stat. Comp. 9 (1988) 2, pp. 312-326.

[11] **Hempel, R. 1987**: *The SUPRENUM Communications Subroutine Library for grid-oriented problems.* Argonne National Laboratory Technical Report ANL-87-23. Argonne 1987.

[12] **Hempel, R. and Schüller, A. 1988**: *Experiments with parallel multigrid algorithms, using the SUPRENUM Communications Subroutine Library.* GMD-Studie 141, St. Augustin 1988.

[13] **Intel 1988**: *iPSC/2 Fortran programmer's reference manual.* Intel Scientific Computers, Beaverton 1988.

[14] **Julian, P. R., Washington, W. M., Hembree, L. and Ridley, C. 1970**: *On the spectral distribution of large-scale atmospheric kinetic energy.* J. Atmos. Sci. 27 (1970), 376-387.

[15] **Krechel, A., Plum, H.-J. and Stüben, K. 1990**: *Parallelization and vectorization aspects of the solution of tridiagonal systems.* Parallel Computing 14 (1990), 31-49.

[16] **Kwizak, M. and Robert, A. 1971**: *A semi-implicit scheme for grid point atmospheric models of the primitive equations.* Mon. Wea. Rev. 99 (1971), 32-36.

[17] **Machenhauer, B. 1979**: *The spectral method.* In Numerical methods used in atmospheric models. GARP Publications Series No. 17 vol. II, 124-275. WMO, 1979.

[18] **McDonald, A. and Bates, J. R. 1989**: *Semi-Lagrangian integration of a gridpoint shallow water model on the sphere.* Mon. Wea. Rev. 117 (1989), 130-137.

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

[20] **Ritchie, H. C. 1988**: *Application of the semi-Lagrangian method to a spectral model of the shallow water equations.* Mon. Wea. Rev. 116 (1988), 1587-1598.

[21] **Ritchie, H. C. 1990**: *Application of the semi-Lagrangian method to a multilevel spectral primitive-equations model.* Q. J. R. Meteorol. Soc. 117 (1990), 91-106.

[22] **Robert, A. 1969**: *The integration of a spectral model of the atmosphere by the implicit method.* Proc. of WMO/IUGG Symp. on numerical weather prediction in Tokyo, 1968. Meteor. Soc. Japan, (VII-19)-(VII-24).

[23] **Robert, A. 1981**: *A stable numerical integration scheme for the primitive meteorological equations.* Atmos. Ocean 19 (1981), 35-46.

[24] **Robert, A., Yee, T. L. and Ritchie, H. C. 1985**: *A semi-Lagrangian and semi-implicit numerical integration scheme for multi-level atmospheric models.* Mon. Wea. Rev. 113 (1985), 388-394.

[25] **Simmons, A. and Dent, D. 1989**: *The ECMWF multi-tasking weather prediction model.* Comp. Phys. Reports 11 (1989), 165-194.

[26] **Speth, P. and Madden, R. 1983**: *Space-time spectral analyses of Northern Hemisphere geopotential heights.* J. Atmos. Sci. 40 (1983), 1086-1100.

[27] **Swarztrauber, P. 1984**: *Software for the spectral analysis of scalar and vector functions on the sphere.* In Large scale scientific computation. Academic Press 1984.

[28] **Temperton, C. 1983**: *Fast mixed-radix real Fourier transforms.* J. Comput. Phys. 52 (1983), pp. 340-350.