A hybrid parallel algorithm for the Spectral Transform Method which uses functional parallelism

Edmund Chadwick
Hitachi Dublin Laboratory
Trinity College, University of Dublin

A new hybrid parallelisation technique which combines functional parallelism with the more usual spatial parallelism is described and implemented for a global shallow water model of the earth's atmosphere using the Spectral Transform Method.

Asymptotically, as the number of processors is increased, the functional parallelism shall be shown to have lower magnitude (than the spatial parallelism) for the number and volume of processor communications; also, the hybrid method shall be shown to give an extra freedom of choice in the parallelisation, as well as allowing a greater maximum number of processors for a given problem size; it is hoped that these advantageous factors will make the hybrid method a useful tool in the optimisation of parallel global models which use the Spectral Transform Method.

The new technique is implemented for resolutions T85 and T170 on the Hitachi SR2201, a distributed memory RISC massively parallel supercomputer, and is found to reduce the execution times at certain resolutions.

Key words: Spectral Transform Method, Global Models, Shallow Water Equations, Massively Parallel Processing.

1 Introduction

Many Global Weather and Climate Models use the Spectral Transform Method, such as the Integrated Forecasting System (IFS) [10] of the European Centre for Medium-range Weather Forecasts (ECMWF). IFS is a particularly important model, providing boundary conditions for limited area regional weather forecasting models such as the HIgh Resolution Limited Area Model (HIRLAM) [3] of the HIRLAM consortium. The computational requirements of weather/climate models are such that parallel programming is becoming increasingly important [7], and a recent special issue of Parallel Computing [2] gives the state of the art on parallel models of the ocean and atmosphere. However, Foster et al [4] find that the parallel efficiency of the Spectral Transform Method, in particular for global shallow water equation models, is in general rather poor.

In the present paper, a new technique for parallelising the Spectral Transform Method on a massively parallel distributed memory architecture (Hitachi SR2201) shall be presented. This technique uses functional parallelism as well as spatial (or data) parallelism, resulting in a reduction of the communication overhead associated with the Spectral Transform Method. It is therefore hoped that it will become a useful tool in the optimisation of parallel code for Global Weather/Climate Models. These two ways of parallelising code (functional versus data) shall be described and compared later in section 5. The new technique shall be implemented for resolutions T85 and T170 for up to 112 processors, and tested on a global shallow water equation model which is described in the following sections. Section 2 describes the shallow water equations, and section 3 outlines the Spectral Transform Method for numerically solving these equations.

2 The shallow water equations

The earth's atmosphere behaves approximately like a shallow, incompressible fluid, determined by the shallow water equations (examples of which are given in [11], [6] and [8]. These approximate the incompressibe Euler equations by assuming that the horizontal accelerations are independent of the vertical co-ordinate; $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are unit vectors in spherical geometry in the eastward, northward and vertical directions respectively (the spherical co-ordinates are (λ, θ, a) where λ denotes longitude, θ denotes latitude and a is the radius of the earth). In terms of the horizontal velocity $\mathbf{V} = u\mathbf{i} + v\mathbf{j}$, and the free surface geopotential Φ^*, the shallow water equations are given by

$$\begin{aligned} \frac{D\mathbf{V}}{Dt} &= -f\mathbf{k} \times \mathbf{V} - \nabla \mathbf{\Phi}^* \\ \frac{D\Phi^*}{Dt} &= -\Phi^* \vec{\nabla}.\mathbf{V}. \end{aligned} \tag{2.1}$$

f is the Coriolis force, and the substantial derivative is given by $\frac{D}{Dt}() = \frac{\partial}{\partial t}() + \mathbf{V}.\vec{\nabla}()$. ∇ is the gradient operator and $\vec{\nabla}()$ is the divergence operator $\frac{\mathbf{i}}{a\cos\theta}\frac{\partial}{\partial\lambda}() + \frac{\mathbf{j}}{a}\frac{\partial}{\partial\theta}()$.

The spectral method [6] and [8], uses the variables of vorticity η and horizontal divergence δ defined by

$$\begin{aligned} \eta &= f + \mathbf{k}.(\vec{\nabla} \times \mathbf{V}) \\ \delta &= \vec{\nabla}.\mathbf{V}. \end{aligned} \tag{2.2}$$

After some manipulation, letting $U = u\cos\theta$ and $V = v\sin\theta$ to avoid the singularity in the velocity at the poles, the equations can be written in the form

$$\begin{aligned} \frac{\partial\eta}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial A}{\partial\lambda} - \frac{1}{a}\frac{\partial B}{\partial\mu} \\ \frac{\partial\delta}{\partial t} &= \frac{1}{a(1-\mu^2)}\frac{\partial B}{\partial\lambda} - \frac{1}{a}\frac{\partial A}{\partial\mu} - \nabla^2 E - \nabla^2\Phi \\ \frac{\partial\Phi}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial C}{\partial\lambda} - \frac{1}{a}\frac{\partial D}{\partial\mu} - \bar{\Phi}\delta \end{aligned} \tag{2.3}$$

where the quadratic variables A, B, C, D and E are given by

$$\begin{aligned} A &= U\eta \\ B &= V\eta \\ C &= U\Phi \\ D &= V\Phi \\ E &= \frac{U^2 + V^2}{2(1-\mu^2)}. \end{aligned} \tag{2.4}$$

$\mu = \sin\theta$, Φ is the perturbation from a constant average geopotential $\bar{\Phi}$ (hence $\Phi^* = \Phi + \bar{\Phi}$), and ∇^2 is the Laplacian operator.

U and V can be represented in terms of η and δ through two auxiliary equations expressed in terms of a scalar stream function Ψ and a velocity potential χ:

$$\begin{aligned} U &= \frac{1}{a}\frac{\partial\chi}{\partial\lambda} - \frac{1-\mu^2}{a}\frac{\partial\Psi}{\partial\mu} \\ V &= \frac{1}{a}\frac{\partial\Psi}{\partial\lambda} + \frac{1-\mu^2}{a}\frac{\partial\chi}{\partial\mu} \end{aligned} \tag{2.5}$$

where

$$\begin{aligned} \eta &= \nabla^2\Psi + f \\ \delta &= \nabla^2\chi. \end{aligned} \tag{2.6}$$

3 The Spectral Transform Method

The global shallow water model, based on the Spectral Transform Method, is an implementation of the description given by Hack and Jakob [5]. A brief account of this description is given here, mainly to define the functions used within the code which shall be distributed in the functional parallelisation.

The spectral representation of a scalar field ξ is approximated by a truncated series summation of spherical harmonic functions $\{P_n^m(\mu)e^{im\lambda}\}$ given by

$$\xi(\lambda,\mu) \simeq \sum_{m=-M}^{m=N} \sum_{n=|m|}^{N(m)} \xi_n^m P_n^m(\mu) e^{im\lambda} \tag{3.1}$$

where $P_n^m(\mu)$ are Legendre functions and the imaginary number $i = \sqrt{-1}$. In particular, attention shall be restricted to the triangular truncations $N(m) = M$, for $M = 85$ and $M = 170$. From [5], the spectral coefficient ξ_m^m is therefore

$$\begin{aligned} \xi_n^m &= \int_{-1}^{1} \left[\frac{1}{2\pi}\int_0^{2\pi} \xi(\lambda,\mu)e^{-im\lambda}d\lambda\right] P_n^m(\mu)d\mu \\ &\equiv \int_{-1}^{1} \xi^m P_n^m(\mu)d\mu. \end{aligned} \tag{3.2}$$

The inner integral is a Fourier Transform

$$\xi^m = \frac{1}{2\pi}\int_0^{2\pi} \xi(\lambda,\mu)e^{-im\lambda}d\lambda \simeq \frac{1}{I}\sum_{k=1}^{I} \xi(\lambda_k,\mu)e^{-im\lambda_k} \tag{3.3}$$

where $\lambda_k = \frac{2\pi k}{I}$, for $I \geq 3M+1$; this condition allows exact unaliased Fourier Transforms of the quadratic terms [5].

The outer integral is a 'Legendre Transform'

$$\xi_n^m = \int_{-1}^{1} \xi^m(\mu)P_n^m(\mu)d\mu \simeq \sum_{j=1}^{J} \xi^m(\mu_j)P_n^m(\mu_j)w_j \tag{3.4}$$

where μ_j denotes Gaussian latitudes, w_j the Gaussian weight at latitude μ_j, and J the number of Gaussian latitudes from pole to pole, for $J \geq \frac{3M+1}{2}$; similarly, this condition allows exact unaliased Fourier Transforms of the quadratic terms [5].

In spectral space, the equations become

$$\begin{aligned}
\frac{\partial \eta_n^m}{\partial t} &= q_n^m \\
\frac{\partial \delta_n^m}{\partial t} &= d_n^m + \frac{n(n+1)}{a^2}\Phi_n^m \\
\frac{\partial \Phi_n^m}{\partial t} &= s_n^m - \bar{\Phi}\delta_n^m
\end{aligned} \tag{3.5}$$

where

$$\begin{aligned}
q_n^m &= -\sum_{j=1}^{J}\{imA^m(\mu_j)P_n^m(\mu_j) - B^m(\mu_j)H_n^m(\mu_j)\}\frac{w_j}{a(1-\mu_j^2)} \\
d_n^m &= -\sum_{j=1}^{J}\{imB^m(\mu_j)P_n^m(\mu_j) - A^m(\mu_j)H_n^m(\mu_j)\}\frac{w_j}{a(1-\mu_j^2)} \\
&\quad +\frac{n(n+1)}{a^2}\sum_{j=1}^{J}B^m(\mu_j)P_n^m(\mu_j)w_j \\
s_n^m &= -\sum_{j=1}^{J}\{imC^m(\mu_j)P_n^m(\mu_j) - D^m(\mu_j)H_n^m(\mu_j)\}\frac{w_j}{a(1-\mu_j^2)}
\end{aligned} \tag{3.6}$$

where $P_n^m(\mu)$, $H_n^m(\mu)$ are the normalised associated Legendre function and Legendre function derivative respectively. The semi-implicit time step procedure is applied to the equations (3.5), in which the linear terms are treated implicitly and the quadratic terms explicitly within a central difference scheme [5].

4 The spatial parallelisation

To test the hybrid method, a comparison is made with the 1-D decomposition by latitude, described in Foster [4]. The divisions of the horizontal space are along lines of latitude, divided as evenly as possible into $2p$ parts, see fig. 1.

The $2p$ parts are grouped into North/South paris for computational efficiency [4]; the spectral space is triangular (see fig. 2) and this pairing provides a good load balance for the computations performed in the spectral space.

This division implies the use of sequential Fast Fourier Transforms (FFT's)and parallel Legendre Transforms (LT's). (It is noted that this parallelisation has the same number of communications as in the transpose approach used in IFS [10]. Applying the transpose method to this problem, the decomposition is by latitude when performing Fourier Transforms/Inverse Transforms, but is transposed to give a decomposition by longitude when performing Legendre Transforms/Inverse Transforms; this means that both the FFT's and the LT's are now sequential, which is useful for the vector machine architecture presently at ECMWF. However, in the present paper only the 1-D decomposition by latitude is considered.)

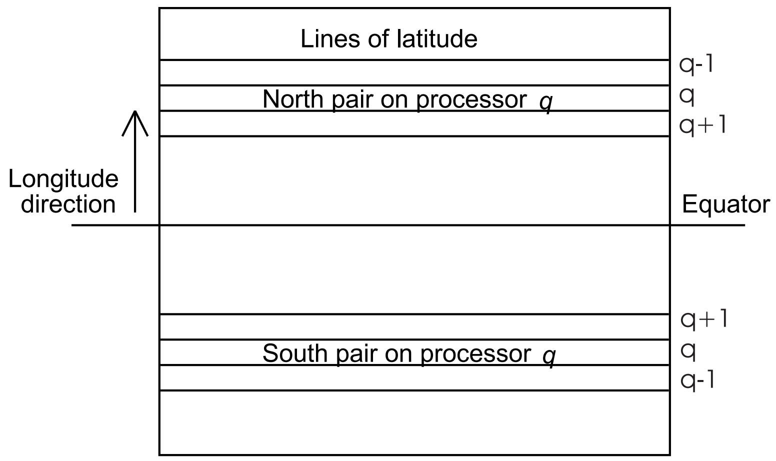

Figure 1: Parallelisation by spatial division into North/South pairs

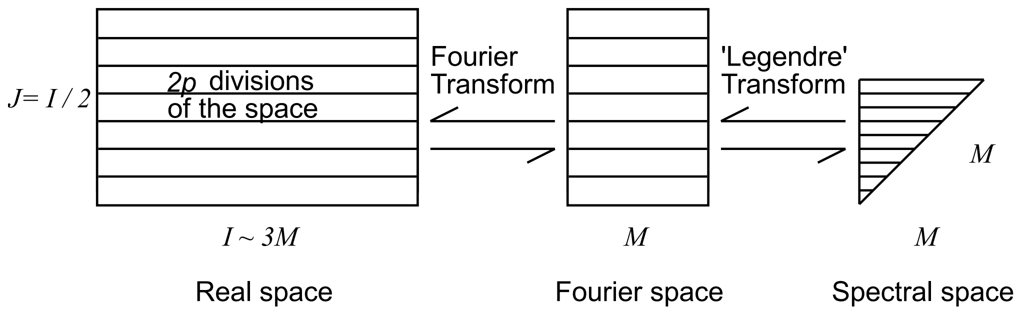

Figure 2: The 1-D decompositon of space by latitude

4.1 Memory allocation

A perfect scaling of the memory allocated to a vector occurs if the vector length divides evenly between the processors. However, because the spectral transforms are global, some of the vector functions must have their total length allocated on each processor. During computation of the transition from time step t to $t+1$, the variables in Fourier space are $q(t)$, $d(t)$, $s(t)$, $\eta(t)$, $\delta(t)$, $\Phi(t)$, and $\eta(t+1)$, $\delta(t+1)$, $\Phi(t+1)$ in spectral space. There is scope, though, for getting around this problem by using a buffer vector function whose total length is allocated on each processor, but having the vector length for all the variables divided evenly between the processors; the above variables are then copied into the buffer vector during the transform. It is noted that both the Legendre function and the Legendre function derivatives must also have there total length allocated to each processor.

4.2 Load balance

The load balance depends upon how evenly the vector space, dimension J in real and Fourier space, and dimension M in spectral space, is divided into $2p$ parts. This depends upon the ratio $J/2p$ and $M/2p$, since we assume the same work load for the calculation of each spatial pair of points (see fig. 1), and each spectral pair of points. If the ratios are integers, then there is an even distribution of the work load. However, if the ratios are non-integer, then some processors will have to be allocated a greater load than others, creating a load imbalance. In this implementation, we shall allocate the greater load to processors nearer to the poles (see fig. 1), although this is arbitrary.

5 Spatial parallelism versus functional parallelism

Spatial (or data) parallelism divides the domain over which the data is given, and assigns different divisions to different processors. These divisions may either change (for example when the transpose parallel algorithm [10] is used) or remain the same during the execution of the code. In contrast, functional parallelism divides the functions within the code between different processor levels. We note that the same variable but at two different time steps, or the same variable but in two different spaces (for example real space and transform space), is treated as two different functions. Since functional and data parallelisms are independent one from the other, the processor levels shall be represented as being orthogonal to the spatial domain (see fig. 5) and a hybrid parallelisation which uses both the data and functional parallelisation approaches shall be considered (see section 6).

Currently, Global Weather/Climate Models which use the Spectral Transform Method employ data parallelism. However, there is a high communication overhead for this parallelisation of the spectral transforms because they are global, which means that every processor must pass its transform data to every other processor. This overhead is a major factor contributing to poor parallel efficiency. The aim of the present paper is to show that by also considering functional parallelism a more optimised parallelisation (for the execution time) may be obtained. An optimal version usually has load balanced evenly between processors, and the number and volume of communications between processors minimised.

In particular, we shall compare the performance of the standard spatial parallelisation described in section 4 with the new hybrid parallelisation for a shallow water global model of the earth's atmosphere. For the spatial parallelisation, there are of order P communications on each processor. (We note that although the number of communications per processor for parallel FFT's is of order $\log_2 P$ and so is less, most parallel models use either parallel LT's, or transpose data between transforms like IFS [10] which, for this problem, results in the same communicaton overhead.)

This parallelisation shall be contrasted against the hybrid parallelisation in which there are m levels: the functions within the algorithm are distributed between the m levels (functional parallelism) and the horizontal space is decomposed between p processors (spatial parallelism) in the same way as described in section 4; hence a total of $P = mp$ processors are now used, and to move a function beween levels requred of order p communications. Hence, the communications overhead on each processor has been reduced by a factor m.

Also, we note that the hybrid method has the advantage of containing both the functional and spatial parallelisms, which are independent of each other, in two regards: firstly, changes needn't be made to the spatial parallelisation when adapting the code to incorporate the new approach; secondly, a greater maximum number of processor can be used, as well as a greater freedom of choice in the parallelisation. It is hoped that all these factors will make the hybrid method an important tool in the optimisation of parallel global models.

6 Hybrid method

The functions are evenly divided between m processor levels in such a way as to have similar amounts of computations (FLOP count) on each processor, and to minimise the time spent by any one processor waiting to either send or receive a message. Both two and four levels ($m = 2$ and $m = 4$ respectively) are considered; the figs. 3 and 4 show which functions have been allocated to which processor levels (each colour represents a different processor level). The arrows indicate the dependencies; an arrow pointing from function A to function B means that A is needed in the calculation of B. The figures represent one time step t through the code, the functions at the top of the figure being those given from the previous time step $t-1$, and the functions at the bottom of the figure being thouse evaluated within the time step t and passed on to the next time step $t+1$. We note that during the time step t, the semi-implicit time step requires $\eta(t-1)$, $\delta(t-1)$, and $\phi(t-1)$. However, from figs. 4 and 5 the semi-implicit time step algorithm recieves $\eta(t)$, $\delta(t)$ and $\phi(t)$ from the calculations in spectral space. Thus, these values are calculated and communicated one time step earlier than necessary, but this does not affect the procedure. We also note that the Asselin filter [5] is employed to prevent modal splitting, and this procedure does require $\eta(t)$, $\delta(t)$ and $\phi(t)$, as well as $\eta(t-1)$, $\delta(t-1)$ and $\phi(t-1)$, during the time step t.

6.1 Memory allocation

The communications incurred are from sends/receives of a function from one level to another. If the function A is required in the evaluation of the function B, and A and B are on different processor levels, then A must be sent to the processor level of B. This means memory must be allocated for the function A on both processor levels.

Hence, like the spatial parallelisation, there is not perfect scalability; the scalability of memory for the two parallelisation approaches is very similar. We note, however, that there is scope for the introduction of buffer functions which can be used to temporarily hold more than one function at different places during a time step, and which minimise the scalability problem.

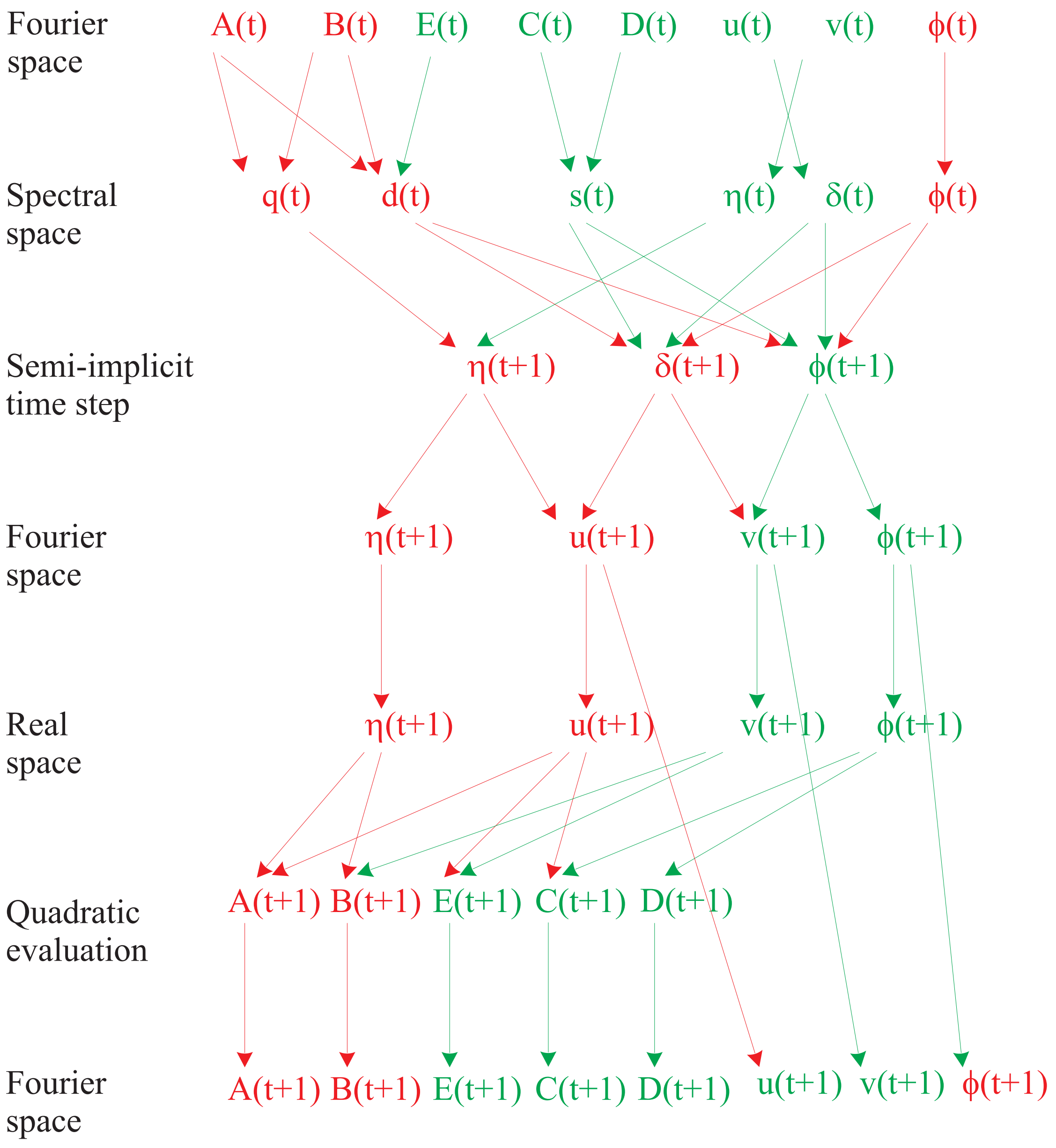

Figure 3: The two level functional parallelisation

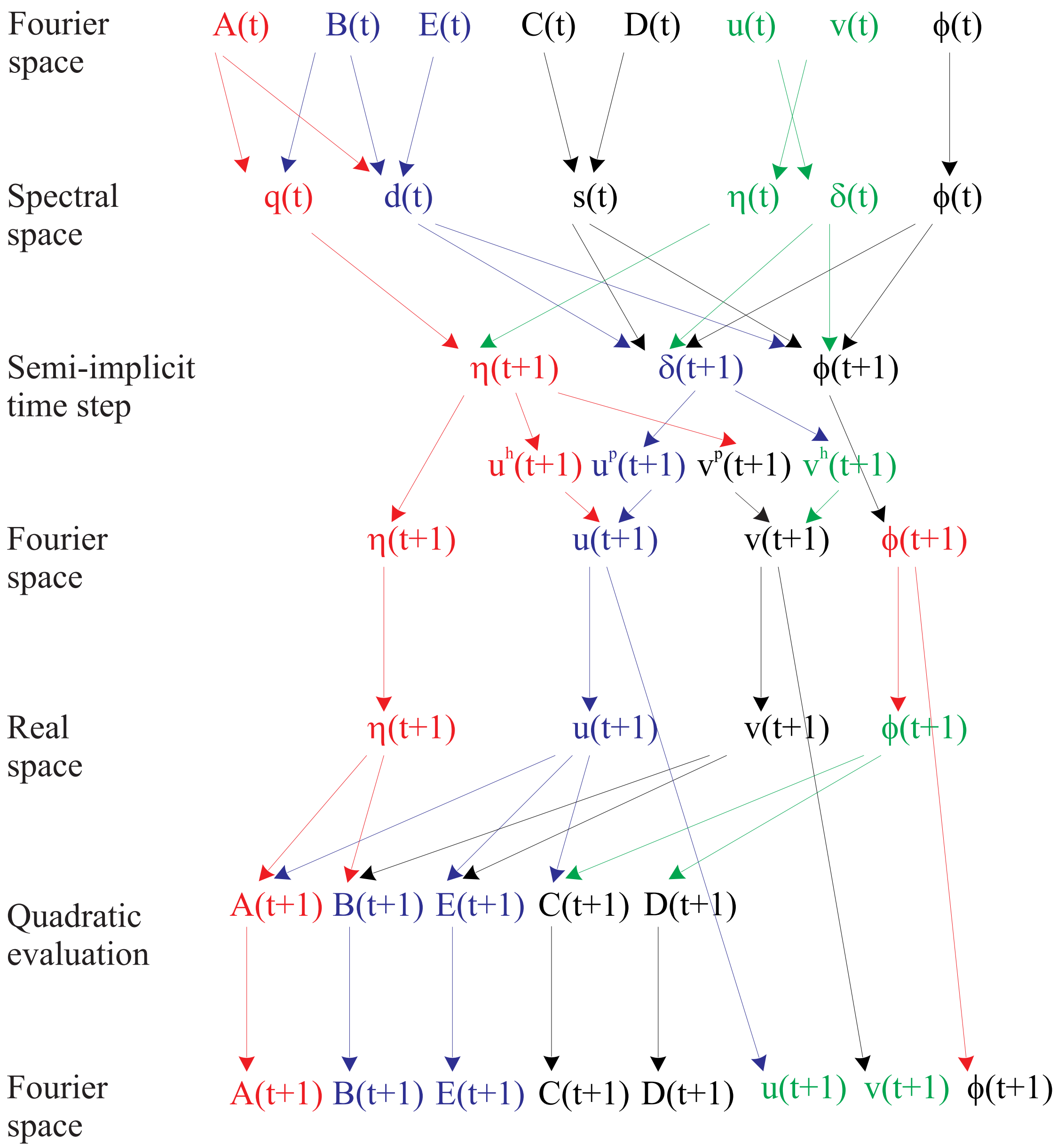

Figure 4: The four level functional parallelisation

6.2 Load balance

The success of this method depends upon achieving good load balance between processor levels. This is obtained by distributing the FLOP's evenly; for example, consider the routines to perform Legendre inverse transforms. Calculating the functions $\eta(t+1)$, $u(t+1)$, $v(t+1)$, and $\Phi(t+1)$, requires a FLOP count ratio of 2:6:6:2 respectively (see the Appendix). To provide a better load balance on the four processor levels, the functions $u(t+1)$ and $v(t+1)$ are split into two parts, $u(t+1) = u^h(t+1) + u^p(t+1)$ and $v(t+1) = v^h(t+1) + v^p(t+1)$ given by

$$\begin{aligned} u^h &= -\sum_{n=|m|\ n\neq 0}^{M} \frac{a}{n(n+1)}(\eta_n^m + f_n^m)H_n^m(\mu_j) \\ u^p &= -\sum_{n=|m|\ n\neq 0}^{M} \frac{a}{n(n+1)} im\delta_n^m P_n^m(\mu_j) \end{aligned} \tag{6.1}$$

and

$$\begin{aligned} v^h &= -\sum_{n=|m|\ n\neq 0}^{M} \frac{a}{n(n+1)}\delta_n^m H_n^m(\mu_j) \\ v^p &= -\sum_{n=|m|\ n\neq 0}^{M} \frac{a}{n(n+1)} im(\eta_n^m - f_n^m)P_n^m(\mu_j) \end{aligned} \tag{6.2}$$

where f_n^m are the spectral coefficients of the Coriolis force. The functional parallelism, given in fig. 4, is such that the FLOP count ratio for the Legendre Transforms/Inverse Transforms on the four processors is $0.87 : 0.81 : 0.81 : 1$ (see the Appendix). Also, the choice of the particular distribution of functions is determined by the need to reduce the possibility of synchronisation delays. Experimental results characterising load balance and idle time between processor levels are now given for the problem size T85. From the asymptotics, the Legendre Transforms/Inverse Transforms are seen to be the largest routines. These are plotted in figs. 5 and 6.

In fig. 5 we see that by using two processor levels excellent load balance is achieved with very little idle time.

In fig. 6, using four levels, we see that there is some load imbalance, and this shall be reflected in the results obtained when small numbers of processors are used. However, as the number of processors is increased the communication overheads play a more significant fole, and so using four levels becomes a more attractive option.

6.3 Combining the spatial and functional parallelisations

The two parallelisms, spatial and functional, are combined in a hybrid method: mp processors are used, p processors on each of m levels (we consider $m = 2$ and $m = 4$). The p processors on a level parallelise the space, and the functions are distributed between the m levels. Each level of processors is arranged in an MPI communicator group. Broadcast communications occur within the group (inter communications), and send/receive communications occur between groups (intra communications), see fig. 7.

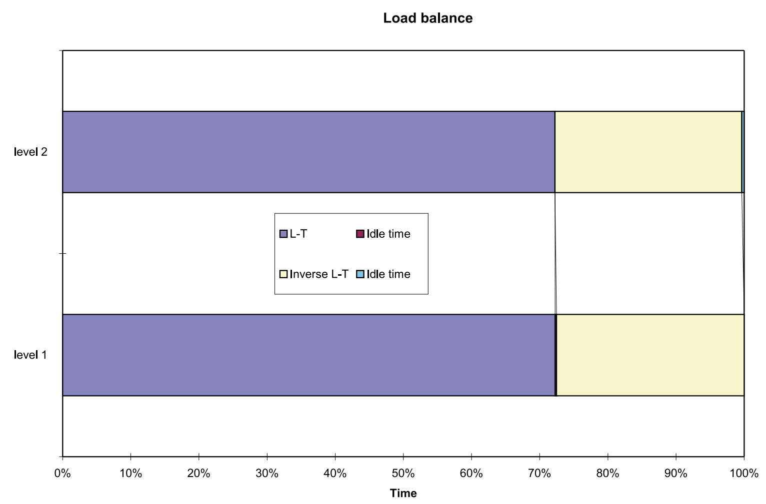

Figure 5: Load balance for two processor levels

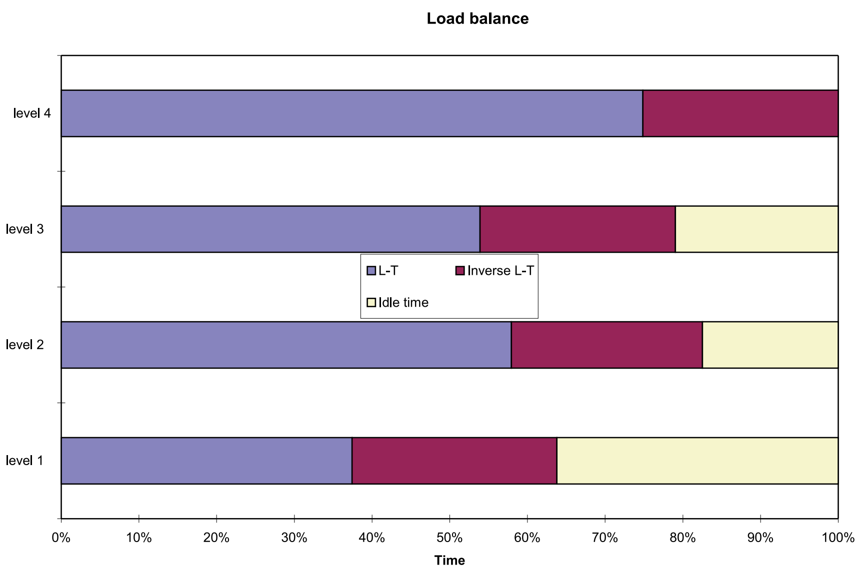

Figure 6: Load balance for four processor levels

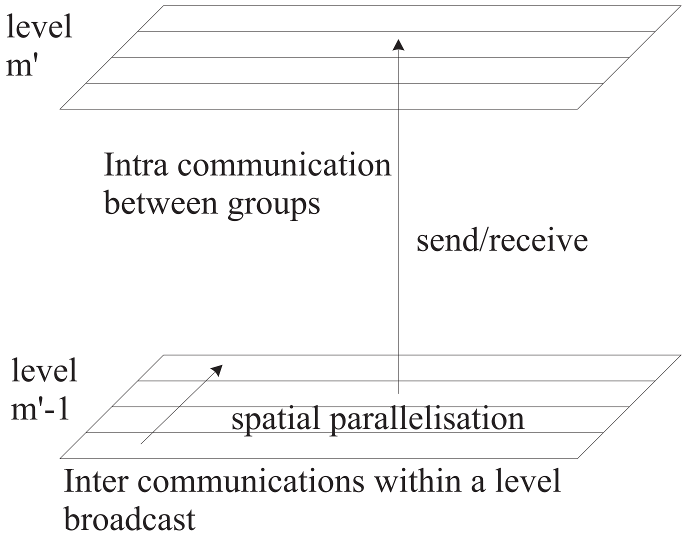

Figure 7: The hybrid parallelisation

	Number	**Length**	**Volume**
Spatial:			
Broadcast communication:	P	N/P	N
Hybrid:			
Broadcast communication:	$p = P/m$	$N/p = Nm/P$	N
Send/Receive communication:	$p = P/m$	$N/p = Nm/P$	N

Table 1: The asymptotic orders for the communications on each processor

6.4 Communication overhead of the hybrid method: asymptotic analysis

The asymptotics for the communication overhead on $P = mp$ processors is nowe given for the two parallelisations, spatial and hybrid. The hybrid method uses p processors in the spatial parallelisation of a level, and m processors for the functional parallelisation into m levels. Table 1 shows the order of the **number** of communications per processor, the vector message **length**, and the order of the **volume** of the communication, defined as the number of communications multiplied by the vector message length. The total vector length is given as N.

There is a reduction in the number of communications per processor of order m. The length of the message has increased by a factor m, and so the volume of messages per processor remains the same. However, there is also a reduction in the number of processors that are involved in this communication, from P to p, so there is a reduction in the total volume of messages of order m.

7 Results

The implementation described here was programmed in C and used the MPI parallel communication library, and the code was tested using some of the test set for shallow water equations given by Williamson et. al [9] (in particular, the advection and Rossby wave [1]), and run on the distributive RISC architecture of the Hitachi SR2201. Use was made of the MPI process groups facility for communications when implementing the hybrid method.

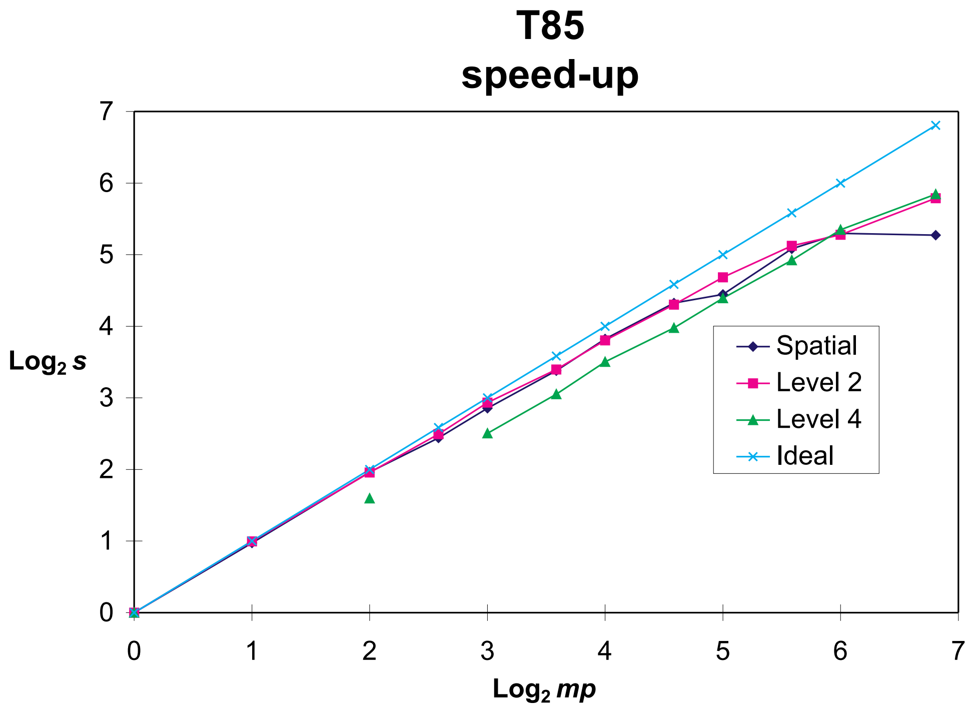

Figure 8: Speed-up for the problem size T85

Results are given for two problem sizes, the triangular truncations with $M = 85$ and $M = 170$, which are called T85 and T170 respectively, and are chosen to make full use of the FFT's: the FFT algorithm for vector lengths of powers of 2 is implemented. Up to 112 processors are used, and in figs. 8 and 9 we plot speed-up against the total number of processors $P = mp$.

The execution time relative to the sequential execution time is given by $T = t/t_s$, where t is the execution time and t_s is the sequential execution time. The speed-up s is defined as $s = 1/T$. Thus ideal scaling is along the line $\log_2 s = \log_2 mp$, and actual results lie below this line, see fig. 8.

We first discuss the results for problem size T85. Using four levels, there is load imbalance (see 6.2), and this is reflected in the results given by fig. 8. However, as the number of processors is increased, the communication overhead becomes more significant, and the level four hybrid parallelisation becomes more attractive. For 64 and 112 processors, the level 2 and level 4 hybrid parallelisations are more optimal, and we expect this trend to continue for higher numbers of processors. For 32 processors ($\log_2 P = 5$) within a level, there is poor load balance for the spatial parallelisation (see fig. 4.2). The improvement in the execution time by using the level 2 hybrid scheme for this number of processors is about 25 per cent. An additional reason for this large improvement is due to the independence between the spatial and functional parallelism.

We now discuss the results for the problem size T170. For this problem size, the level 2 parallelisation either gives similar or marginally better results than the spatial parallelism for up to 64 processors, and significantly better results for 64 processors. We also note how the level 4 hybrid parallelisation is approaching the spatial parallelisation for 64 and 112 nodes. We expect that for higher numbers of processors the level 4 hybrid parallelisation will be

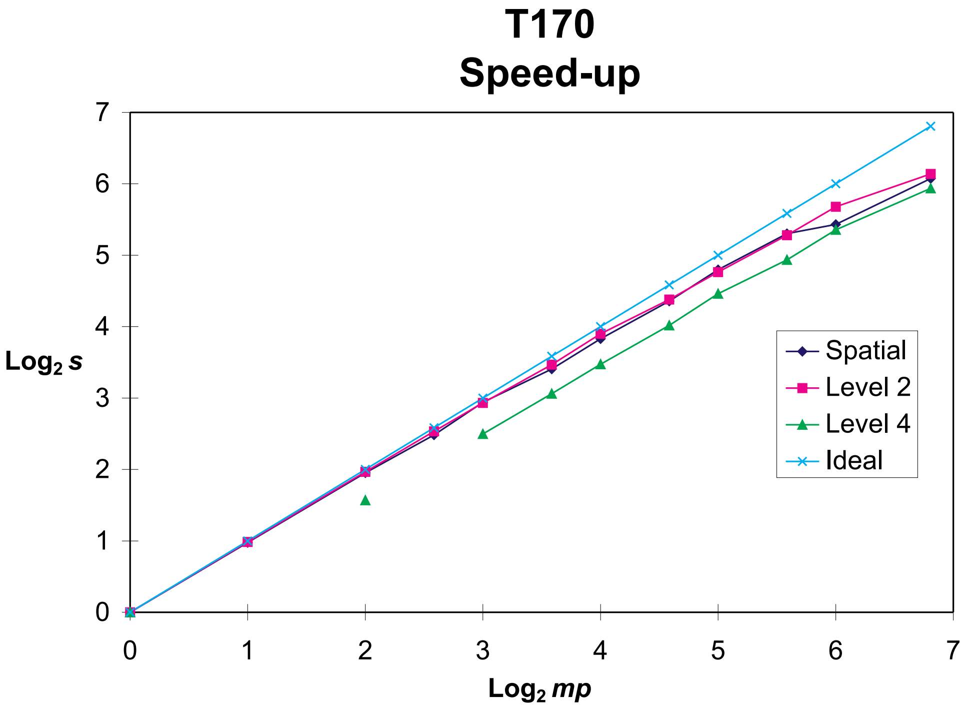

Figure 9: Speed-up for the problem size T170

better still.

8 Discussion

Asymptotically, the hybrid method has a better scaling, with a reduction in the number of communications per processor by m, and a reduction in the total volume of communications by m, compared to the 1-D horizontal discretisation by latitude. This advantage becomes more apparent for greater numbers of processors (see figs. 8 and 9). For processor numbers less than 50, the advantage is not so obvious: Foster et al's [4] predictions of poor scalability were made for problems of the order hundreds of nodes, and the results presented here indicate that the proportion of time spent passing messages was not significant enough on up to 50 processors to radically improve performance.

However, the results show that the hybrid method can be used as a way of optimising parallel code even for small numbers of processors; for example, the spatial parallelism T85 runs poorly on 32 processors, but better on 16 processors because of an improved load balance. Thus, an optimisation for 32 levels can be obtained by considering two levels of 16 processors each (see fig. 8). Thus utilises the property that the functional and spatial parallelisms within the hybrid method are independent, and so the extra functional parallelism can be used on top of an existing optimised spatial parallelism. It is therefore hoped that the hybrid method will be found to be a useful tool in optimising parallel implementations of Global Models which use the Spectral Transform Method.

A future goal is an implementation for a Global Climate/Weather Model. For this case, functional parallelism can be performed within the model physics as well as the dynamics, and the communication costs to redistribute should be the same. However, there is a much greater oppportunity for good load balance and for the use of greater numbers of processor levels within the functional parallelism because of the increased numbers of functions in the code. This was one of the drawbacks of using functional parallelism for the shallow water equations, in that only a small number of processor levels could be considered. Indeed, the more functions within the code the better the opportunities of a good functional parallelism. So we expect the hybrid method presented here to be more successful for a full Global Climate/Weather Model.

Acknowledgements
I would like to thank the General Purpose Computing Division (GPCD) of Hitachi Ltd for sponsoring the research, and Hitachi Dublin Laboratory (HDL). I would also like to thank my family, who are always there for me, and Louise, who has been my constant companion sice coming to Ireland.

References

[1] E. Chadwick. Sequential performance of the spectral transform method shallow water equation model and preparation for parallelisation. Technical Report HDL-TR-97-187, Hitachi Dublin Laboratory, Trinity College, University of Dublin, 1997.

[2] J. Drake and I. Foster. Special Issues on Applications: Climate and weather modeling. *Parallel Computing*, 21, 1995.

[3] G. Cats H. Middelkoop D.Streefland and D. Weierstra. *A meteorological model on a transputer network*, pages 47–77. In Hoffman and Maretis [7], 1990.

[4] I.T. Foster W. Gropp and R. Stevens. The parallel scalability of the spectral transform method. *Monthly Weather Review*, 120:835–850, 1992.

[5] J.J Hack and R. Jakob. Description of a global shallow water model based on the spectral transform method. Technical Report TN-343+STR, National Center for Atmospheric Research, Boulder, CO, USA, 1992.

[6] G.J. Haltiner and R.T. Williams, editors. *Numerical Prediction and Dynamic Meteorology*. John Wiley and Sons, New York, 1979.

[7] G.R. Hoffman and D.K. Maretis, editors. *The dawn of massively parallel processing in meteorology*. Springer-Verlag, New York, 1990.

[8] J.R. Holton. *An Introduction to Dynamic Meteorology, 3rd edition*. Academic Press, San Diego, 1992.

[9] 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 equatons in spherical geometry. *Journal of Computational Physics*, 102:221–224, 1992.

[10] S. Barros D. Dent L. Isaksen G. Robinson G. Mozdznski and F. Wollenweber. The IFS model: a parallel production weather code. *Parallel Computing*, 21:1621–1638, 1995.

[11] J. Pedlosky. *Geophysical Fluid Dynamics, 2nd edition*. Springer-Verlag, New York, 1987.

Appendix

The approximate asymptotic FLOP counts for the evaluation of the following functions are listed below:

Legendre Transforms:		*Inverse LT's:*	
q, s, p, η, δ:	$6n(M+1)(M+1)$	η, Φ:	$2n(M+1)(M+1)$
d:	9n(M+1)((M+1)	u, v:	6n(M+1)(M+1)
Φ:	$3n(M+1)(M+1)$	u^h, v^h, u^p, v^p:	$6n(M+1)(M+1)$

Fourier Transforms/Inverse Fourier Transforms: $nm\log_2 m$

Time step caculation:		*Quadratic evaluations:*	
η:	$(15/2)(M+1)(M+1)$	A, B, C, D:	$n(m+1)$
δ:	$(49/2)(M+1)(M+1)$	E:	$7n(m+1)$
Φ:	$(53/2)(M+1)(M+1)$		