A Massively Parallel Algorithm for the Spectral Method

Simon Tett
Departments of Physics and Meteorology,
University of Edinburgh *

Abstract

At the present time the Spectral method is widely used within the Meteorological community. If massively parallel computers are to be useful to this community then an efficient Algorithm for this method is required. The massively parallel computer that is considered is one in which each computational node has it's own private memory, CPU and has a fixed number of communication links (valency) rather than increasing the valency with the number of processors. Transputer based machines are an example of the former while hypercubes are an example of the later.

The major computational effort in the spectral method, at least for large truncations, are the legendre transforms and the fourier transforms. The natural method for both of these are pipelines. Each pipeline has, of course, different internal structures. The entire algorithm is a pipeline which carries out the spectral space computations, the Spectral transformation, grid point computations and computes the contribution to the Gaussian integration. The data element in this pipeline is all variables for each Gaussian Latitude.

Calculations will be given for the pipelines vector efficiency and the processor utilisation as the number of processors increase.

*Present Affiliation Hadley Centre, U.K. Meterological Office, London Road, Bracknell, Berkshire, RG12 2SZ

1 Introduction

At present, the Spectral Method is widely used by the Meterological community. Its use ranges from low resolution climate models (James and Hoskins, 1990; Bourke, 1988) to high resolution numerical weather prediction models (Girard and Jarraud, 1982). Some authors have suggested that the spectral method could be suitable for local area models if a suitable conformal transformation of the globe is chosen (Courtier and Geleyn, 1988). Their use is not likely to be discontinued in the near future. Therefore an efficient parallel algorithm for this method is required. This paper will outline such an algorithm.

The details of the spectral method are well know to the meteorological community (Machenhauer, 1979). The transforms in the method global transformations and require that data be communicated across the entire computational domain. Some models also require some local communications in parts of the method, an example of which is described in (Bourke, 1974). The model which was studied has been in use for some time and was first described in 1975 (Hoskins and Simmons, 1975) and will be referred to throughout this paper as the Reading Model. Most of the analysis in this paper is directed at hardware which is capable of overlapping communications and calculations.

The next section will provide a description of the method used to carry out a parallel implementation, that is a pipeline. The two following sections explain in detail how the two components of the spectral transformation, the Legendre transform and the fast Fourier transformation are carried out. Section 5 shows how the components are joined together and gives an analysis of the algorithms scaling behaviour. Suggestions on how an implementation could be done will be given. Implementations of the spectral method have been carried out by other authors on SIMD machines (Carver, 1988; Swarztrauber and Sato, 1990; Sato and Swarztrauber, 1988). This method is for a MIMD computer with fixed valency and which is configurable to any desired topology. As will later be shown the algorithm is only efficient for computers with large numbers of processors.

For future reference some hardware characteristics will now be given. These are r which measures the communications speed relative to the computations speed and v the valency of the processor. The valency of the processor is the number of external links the processor has while r is the number of floating point operations than can be done in the time to communicate one word to a neighbouring processor (along one link).

2 Pipelines

The method that was decided on for implementation of the spectral method was to build a functional pipeline. A pipeline can be thought of as algebraic decomposition in which a complex task is broken into several tasks (in this case each functional unit is the task). If a sequential operation consists of several functional units and the sequential operation requires to be done several times on different pieces of data then it may be efficient to split up the operation into its component functional units. Each unit will operate on a piece of data, pass this computed data out to the next part and take in another data element to act on. Figure 1 is a simple illustration of this where a brick wall is built using a pipeline.

In this example the task of building a wall is broken into four functional units. A task for this case is adding one brick to the wall. The four units are in order;

1. Prepare the brick.
2. Apply mortar to the brick.
3. Place brick on wall.
4. Tamp brick down.

Bricks (tasks) will be taken from a pile and prepared by the first "brickie" in stage 1. She will then pass this brick on to the next person in the pipeline. The process will continue. After, in this case, 4 time steps the first brick will be added to the wall. From this time on a brick will be added to the wall every timestep. If each stage in the pipeline takes different amounts of time then some "brickies" will spend some of their time doing nothing useful. [1] The time between bricks being added to the wall will be given by the time taken by the slowest "brickie".

[1]Nothing as far as we are concerned, they may find this time extremely useful !

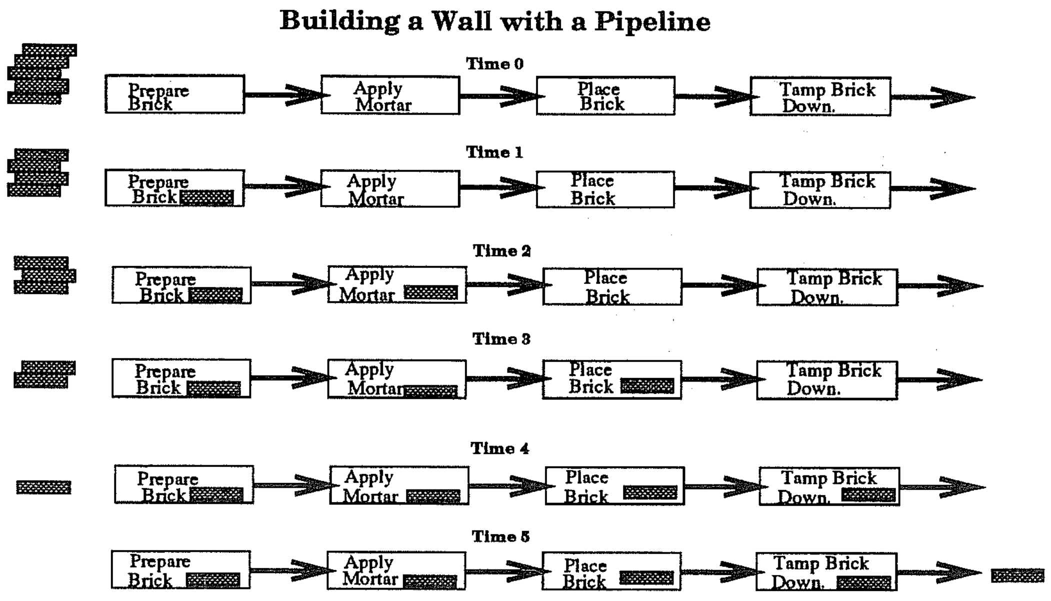

Figure 1: A simple Pipeline

A few preliminary definitions will now be given. The timestep, τ, is the time interval between between data elements at the output end of the pipe. The startup time, S, is the time it takes for the first data element to travel the length of the pipeline. This time will be less than or equal to the numbers of units in the pipe (or depth), U, times τ. If the processor can overlap communications and calculations then this startup time is less than the number of units in the pipe times 2τ. The factor of 2 in the startup time is because there can be no overlapping of communications and calculations as there is nothing to be overlapped with the calculations.

At this stage details of τ will be left to later, it depends on the underlying architecture and algorithms used. The efficiency, V_e, of a pipeline is given by;

$$V_e = \frac{\tau d}{\tau d + S} \tag{1}$$

d is the number of data elements that are to be computed on. A lower bound on efficiency is given from the relationship $S(2) \leq U\tau$ and in that case equation 1 can be rewritten as;

Table 1: Required Parallel Complexity

Stage	No. of Processors
Timestep Inc.	$O(\mathcal{T}^2)$
Forward Lgnd.	$O(\mathcal{T}^2)$
Inverse FFT	$O(\mathcal{T} \ln \mathcal{T})$
Non-lin Products	$O(\mathcal{T})$
Forward FFT	$O(\mathcal{T} \ln \mathcal{T})$
Gaussian Int	$O(\mathcal{T}^2)$

$\mathcal{T}$ is the truncation number (either Rhomboidal or Triangular).

$$V_e \geq \frac{d}{d + (2)U} \tag{2}$$

The pipeline will be running at 50% efficiency, or more, when d, the number of tasks, is equal to $(2)U$, (twice) the pipeline length. Bracketed cases apply when the hardware can overlap communications and calculations.

Now considering the timestep τ, it is clear that this is given by the maximum of all the times taken by each stage in the pipeline. This time may or may not include communications time depending on the characteristics of the underlying hardware on to which the the pipeline is mapped.

Next considering the spectral method, a flowchart for one iteration is given in figure 2. This flowchart is at the level of the different units within the spectral transformation. This flowchart can, at least conceptually, be turned into a pipeline by turning figure 2 on its side and choosing the transformation of the different Gaussian latitudes to be tasks. Figure 3 shows this and the computational complexity of each stage in this pipeline.

In order that the each stage be balanced, as the timestep is given by the slowest part within the pipeline, the number of processors in any stage should be proportional to the computational complexity. Table 1 shows the constraints that are required on the number of processors at each stage.

3 The Fast Fourier Transform

This section will detail the unit of the pipeline that carries out the fast Fourier transform (or FFT) component of the spectral transform. This

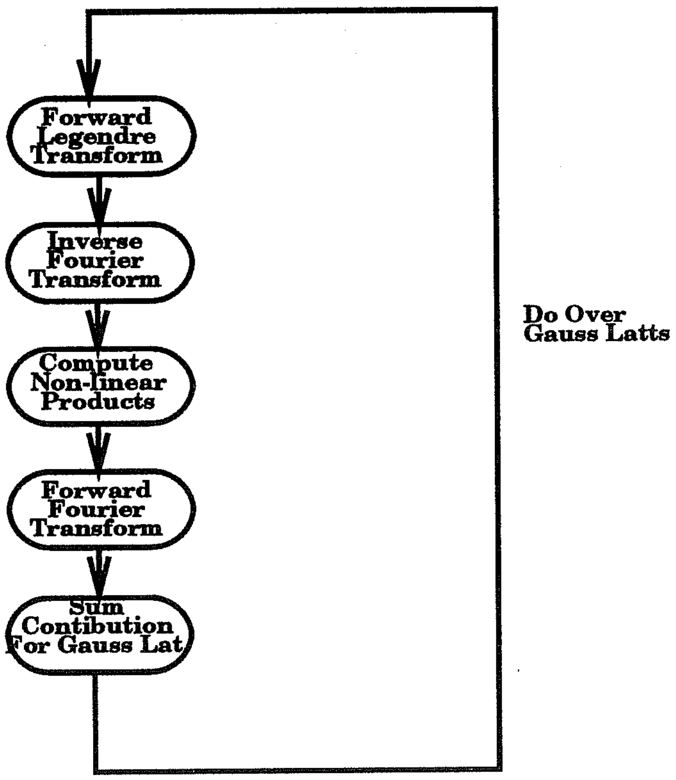
Forward Legendre Transform
Inverse Fourier Transform
Compute Non-linear Products
Forward Fourier Transform
Sum Contibution For Gauss Lat
Do Over Gauss Latts

Figure 2: Flowchart of spectral method

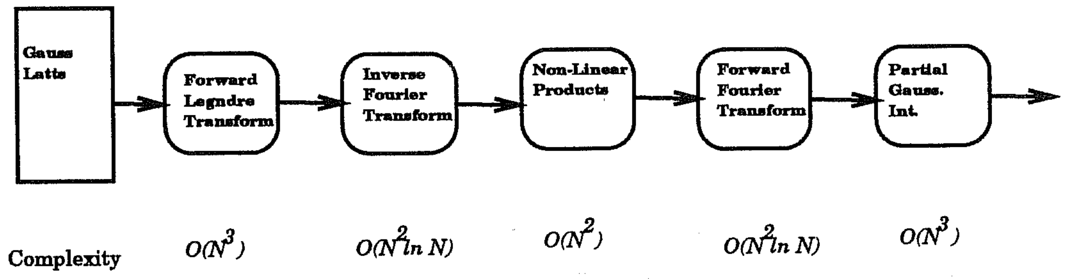

Figure 3: Pipeline and Computational complexity for spectral method

unit is itself broken down into a pipeline though there are only two types of units within this sub-pipeline. The numerical details of the FFT are well known (Press *et al*, 1986). The important equations will be repeated here as necessary.

As will be later shown this part of the algorithm is the bottle neck in the entire problem. The FFT requires very few operations per stage of the transform and thus if the communications speed, relative to the computations speed of the processor, are slow then this approach may not be cost effective and a smaller number of processors and/or a different decomposition strategy should be used.

The important parts of the FFT, in terms of it's implementation on a massively parallel computer, are the recurrence relationship which demands a particular kind of communications pattern and the fact that a binary representation of the harmonics that starts ordered in one space will be bit reversed in the other. If, as in the Reading model, no local communications are required in either gridpoint space or spectral space then it is not necessary to carry out a bit-reversal or reordering process. For future reference the bit reversing operator for λ bits will be denoted by $\overline{}^{\lambda}$ i.e $\overline{M}^{\lambda}$ is the number formed by reversing the the bits of a binary representation, length λ, of M. M must, of course, satisfy the constraints $M \geq 0$ and $M < 2^{\lambda} - 1$. If M is equal to $\sum_{i=0}^{\lambda-1} m_i 2^i$ then $\overline{M}^{\lambda}$ is $\sum_{i=0}^{\lambda-1} m^i_{\lambda-(i+1)}$ with $m_i = 0$ or 1.

The recurrence relationship for the FFT are;

$$\overline{F^k_i}^{(l+1)} = \overline{F^k_i}^{(l)} + \omega^k_{2^{l+1}} \overline{F^{k+2^l}_{i+\frac{N}{2^{l+1}}}}^{(l)} \qquad i \in \{0,\dots,2^l-1\}, \qquad j \in \{0,\dots,2^{q-l}-1\} \tag{3}$$

$$\overline{F^{k+2^l}_i}^{(l+1)} = \overline{F^k_i}^{(l)} - \omega^k_{2^{l+1}} \overline{F^{k+2^l}_{i+\frac{N}{2^{l+1}}}}^{(l)} \qquad i \in \{0,\dots,2^l-1\}, \qquad j \in \{0,\dots,2^{q-l}-1\} \tag{4}$$

For the spectral method the inverse transformation is carried out first followed by the forward transformation. At present no local horizontal communications are required in grid point space. The use of semi-Lagrangian methods to compute the advection step in grid point space would change this (Ritchie, 1987). This more complex case is not considered further.

The method used to compute the FFT is to form a pipeline of width N

(N is an integer power of 2) and length $\log_2 N + K$, K is a constant number of processors. The value of which depends on the size of "internal" FFT. Details are discussed later. The width is the number of processors which will act on one Gaussian latitude. Each stage of the pipeline will compute the recurrence relationships using which ever one of eqns 3, 4 is appropriate. The number of points on a processor, O with $O = 2^p$, is M/N (M being the total number of points for the fourier transform with $M = 2^q$, $q \geq p$)

The mapping of the $\overline{F_i^k}^l$ to the processors is now considered. This takes the form of a mapping, dependant on l, from the i, k indices to a single index j. The processor identifier is given by j/O (integer division) with labels for the processors reading left to right.

The following expression satisfies various requirements.

$$j(l,k,i) = 2^{q-l}\tilde{k}^l + i \tag{5}$$

Values of k are given by $\overline{\lfloor j/2^{q-l} \rfloor}^l$ and i is given by $j \bmod 2^{q-l}$. These are obvious from the restrictions on the domains of i and j.

This mapping strategy leads to the following communications pattern for equations 3 and 4 respectively.

$$\{j, j + 2^{q-(l+1)}\} \mapsto j \tag{6}$$
$$\{j, j + 2^{q-(l+1)}\} \mapsto j + 2^{q-(l+1)} \tag{7}$$

When $q - (l+1) \geq p$ communications between processors are required. The natural, and well known (Hockney and Jesshope, 1988), topology that has this required communications pattern is shown in figure 4.

When no communications are required between processors then an "internal" FFT can be done. This "internal" FFT is a purely serial operation and thus all the well known optimisations for serial computers can be used (Temperton, 1983). These serial parts can, and in order to load balance the pipeline should, be pipelined.

At this stage it should be recognised that the distributed FFT needs some extra operations (Fox *et al*, 1988).Their solution to this inefficiency is to use the richer topology of the Hypercube to make all stages of the FFT's internal and then carry out a shuffle after each stage. The computation of $\omega^k_{2^{l+1}} \overline{F^{k+2^l}_{i+\frac{N}{2^{l+1}}}}^{(l)}$ is done twice in the distributed FFT, once on each processor

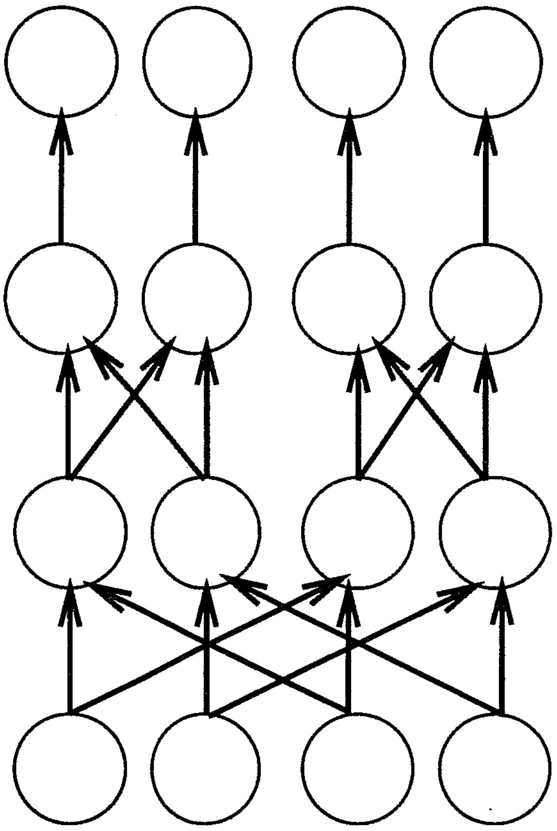

Figure 4: The processor topology for the fast Fourier transform

involved in computing the FFT using which ever one of equations 3 and 4 is relevant. Another inefficiency is that in order to keep the pipeline balanced it is unlikely that many traditional optimisation methods for the FFT will increase performance.

The balancing of the fourier pipeline is now considered. All the distributed parts of the transform take the same time. The time per site is fixed and cannot be varied. That is max$(r, 4)$ for the overlapping case. The only free parameter available is the number of processors that will compute the "internal" part of the FFT. This will require $(q-p)c$ operations per site, c is the number of operations per site for one stage. The length[2], l, of the "internal" pipeline in order that the FFT pipeline be balanced is constrained by;

$$l \leq max(r, 4)/c \tag{8}$$

The number of extra processors is the smallest integer K s.t. $K \geq (q-p)/l$.

The previous discussion has only considered one part of the transformation, the inverse FFT. The forward transformation is also required. The traditional, serial method, is to explicitly carry out the bit-reversal after the transformation, do computations with data ordered then do the other FFT using the same algorithm as the first. Finally the data is again bit-reversed to return it to an ordered state. The following quote (Press *et al*, 1986), with some text emboldened, by the present author, to show emphasis, illustrates this.

"You can use decimation–in–frequency algorithms (without its bit reversing) to get into the 'scrambled' Fourier domain, do your operations there, and then use an inverse algorithm (without *its* bit reversing) to get back to the time domain. While elegant in principle, the procedure does not in practice **save much computation time**, since the bit reversals represent only a small fraction of an FFT's operations count ...".

For the case, as is being considered, where the FFT is distributed over the processors the bit reversal operation will require approximately the same time as the rest of the Fast Fourier Transform spent on communication and it will not be possible to overlap this communications work with any calculations. Thus the time taken by the algorithm is significantly increased. For the model

[2]length here means the number of stages in the pipeline on one processor

being considered it is not necessary to carry out the bit reversal operation as there are no communications between adjacent points in grid point space.

The forward part of the transform (or inverse if we start ordered in grid-point space) is very similar to the inverse transform. The main difference is that the transform starts bit reversed and in this case a labeling j' is used. In this case k and i are given by the following expressions;

$$\begin{aligned} k &= (j'^{q} \bmod 2^{l}) \\ i &= j'^{l} \end{aligned} \tag{9}$$

The startup time, per point on each processor, for the Fourier transform stage of the pipeline is:

$$S_{\mathrm{FFT}} = (4 + r)\log_2 N + Kr + (q - p)c \tag{10}$$

while the time, per point on each processor, between results at the output end of the pipe is:

$$\tau_{\mathrm{FFT}} = \max(r, 4) \tag{11}$$

Equation 11 assumes that the constraint given by equation 8 is satisfied.

4 Legendre Transform

This section will detail how the Legendre Transformation is implemented, as well as the Gaussian integration (which is similar to the the Legendre transform). In order to keep the pipeline balanced both parts will require $O(\mathcal{T}^2)$ processors.

Schematically the Legendre transformation can be described as;

$$v_G = \sum_{m=lower}^{upper} P_{Gm} x_m \tag{12}$$

G is the index for Gaussian latitude, P_{Gm} is the value of Legendre polynomial at m and Gaussian latitude G while x_m is the value of the variable at wavenumber m.

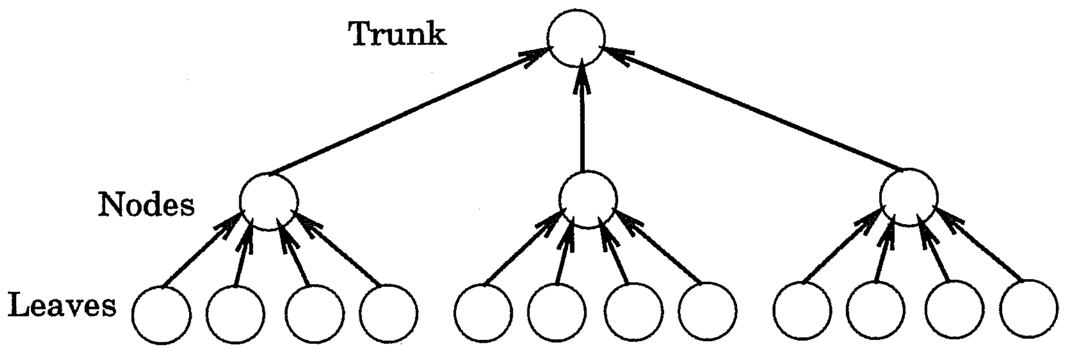

Figure 5: Processor topology for legendre transform

The strategy used is based on the fact that addition is associative [3]. This sum can be decomposed into several partial sums and the total sum can then be formed by adding the partial sums together. Note that in order to guarantee repeatability the partial sums should always be done in the same deterministic order.

The natural topology for this is to form a tree as shown in figure 5. The leaf processors should compute $P_{Gm}x_m \; \forall m$ within their domain and then compute the partial sum on their processor. Having formed the partial sums these should be passed up to the node processors "above" them in the tree. Each node processor would receive up to $v - 1$ partial sums.

These node processors would then sum these partial sums and pass the computed sums "up" the tree. The final sum will appear at the trunk processor.

At this stage the reader should be aware that only the leaf processors are doing any useful work. In this context useful means only that work that would be done on a serial computer. All the other processors are carrying out extra computational work as a result of using a parallel computer. The efficiency of this stage of the algorithm will now be computed.

Assume that the tree has d_L levels and each level is completely filled. That is a level n will have $(v-1)^n$ processors in it. Then the number of leaf processors is simply $(v-1)^{d_L}$ while the number of total processors required is $\sum_{n=0}^{d_L}(v-1)^n$. The efficiency, defined as the number of processors doing

[3]For the limited precision arithmetic done on a computer this is not strictly true. However a good algorithm should not be sensitive to the order of addition

useful work over the number of processors is then $\frac{1}{\sum_{n=0}^{d_L}(v-1)^{-n}}$ which is,

$$e(d_L) = \frac{v-2}{(v-1)-(v-1)^{-d_L}} \tag{13}$$

This expression drops, exponentially fast as the number of levels increase, to $\frac{v-2}{v-1}$. Assuming that the tree is always completely filled then the total number of processors P is given by:

$$P = (v-1)^{d_L}[\frac{(v-1)-(v-1)^{-d_L}}{v-2}] \tag{14}$$

The efficiency, in terms of P, after some easy algebraic manipulation is;

$$e(P) = \frac{v-2}{v-1} + \frac{1}{(1-v)P} \tag{15}$$

Next, having constructed the pipeline for the Legendre transform, its balance needs to be considered. Here the constraint is that the time taken by the leaf processors should be the same as that taken by the node processors. The number of computations that a leaf processor does is $2n-1$. For each site one multiplication and then $n-1$ adds to compute the sum, n being the sub-domain size. Each node processor will do $v-2$ operations, an add on each input. Giving the total time for each type of process;

$$\begin{aligned} \tau_{\text{leaf}} &= \max(2n-1, r) && (16) \\ \tau_{\text{node}} &= \max(v-2, r) \\ & && (17) \end{aligned}$$

The only parameter that can be controlled, once a choice of hardware has been made, is the computational load on the leaf processors. For hardware which can overlap communications and calculations then, in terms of reducing the wall clock time, nothing is gained by reducing $2n-1$ below r or below $v-2$. Formally then;

$$2n-1 \geq \max(r, v-2) \tag{18}$$

It will not be possible to indefinitely increase the number of processors on a fixed problem size. At some point there will be no gain in wall clock

N	S_{max}		
	r=8, v=4	r=4, v=4	r=4, v=8
21	4	8	6
63	14	25	18
213	43	85	60

Table 2: Maximum Speedup
Maximum speedup for various values of v (processor valency) and r (communications speed)

time. This value of P called $P_{\max}$ is given by the following equation. Note that this point may not be the maximum speedup.

$$P_{\max} = \frac{2N}{\max(v-1, r+1)} \tag{19}$$

N is the number of processors. Above this point increasing the number of processors will gain nothing and will probably cause a performance loss. The speedup is simply given by eP and will now be computed.

$$\begin{aligned} S &= e(P)P \\ \Rightarrow S &= \frac{v-2)}{(v-1)}P + \frac{1}{1-v} \end{aligned} \tag{20}$$

Take the derivative w.r.t. P of equation 20 to obtain.

$$\frac{\partial S}{\partial P} = \frac{v-2}{v-1} \tag{21}$$

This derivative is always positive and so maximum speedup will be obtained when P equals $P_{\max}$.

Table 2 shows the maximum speedup that can be obtained for various different parameter values.

This discussion has referred to the maximum speedup when the pipeline is running at full efficiency. That is the effect of start up time has been ignored. These calculated speedups can, of course, only be reached asymptotically as the number of tasks tend to infinity. For the hardware being considered the startup time is given by the following expression;

$$S_{tree} = 2(n-1) + d_L(r + (v-2)) \tag{22}$$

or expressed in terms of the number of processors P;

$$S_{tree} = 2(n-1) + (\log_{v-1}[P(v-2) + (v-1)])(r + v - 2) \tag{23}$$

As can be seen from this expression the startup time increases logarithmically with the number of processors.

Finally in this section the Gaussian integration is considered. This is schematically written as;

$$V_m = \Sigma_{G=0}^{G_{max}} P_{Gm} X_G \tag{24}$$

It is clear from this expression that the computation of any V_m is independent of the rest. Therefore decompose m over the processors and compute $V_m = \Sigma \forall m$ on that processor. As there are no constraints on how this distribution is carried it it would be sensible to use the same m distribution as was used to compute the Legendre transformation. The problem then is how to efficiently replicate the X_G over the processors. The best way is to build another processor tree just like in figure 5 all though the description of processes's on the various processors of the tree will be different from the Legendre tree.

The X_G will enter at the top of the tree and a copy of it will be sent to each processor below it in the tree. Each node will also have this behaviour. Each of the leaf processors will then increment it's contribution to V_m for the values of m which lie within it's domain.

Most of the earlier observations on processor efficiency and utilization are still true. The only differences are the number of operations that the leaf processors will do has increased from $2n-1$ to $2n$. Therefore a similar equation to 18 is obtained for a constraint on n.

$$2n \geq r \tag{25}$$

5 Joining Everything Together

This section will show how the remaining parts of the method are done. It will also explain how all the functional units are joined together and then derive some expressions for the speedup of the algorithm (although some simplifying assumptions will be made). Some details are implementation dependant and so will be outlined in the following section.

The two remaining parts of the method, which are both model dependant, are the non-linear/gridpoint computations [4] and the time update. This second part will be considered at the end of this section after the piplines' efficiency has been computed.

5.1 The Pipeline

The pipeline is build by starting with the FFT topology or butterfly topology which was shown in figure 4, whose width will be determined by the normal demands for the number of gridpoints in a longitudinal circle divided by the number of points per processor. At the base of the butterfly network each processor has attached a Legendre tree. It is not required that all the trees have the same number of processors. Connected to the top of the butterfly will be a number of processors equal to the width of the butterfly. These processors will compute the non-linear terms in grid point space.

The computation of the non-linear terms may cause considerable imbalance in the pipeline and thus a large loss of performance. The computations per site for the non-linear unit is given by $\mathcal{G}$. The exact details of $\mathcal{G}$ are dependant on the exact specification of the model and probably the processor hardware. For the Reading model the imbalance over the other parts of the model is about 4. In order to avoid this imbalance then for each processor at the top of the butterfly another tree should be build, It's depth is given by d_G. The number of leaf processors, $(v-1)^{d_G}$, should be equal or less than $\mathcal{G}/\tau$. For future reference this is denoted by $\mathcal{G}'$. There should be at least one point for each one of the leaf processors.

Figure 6 illustrates this entire forward spectral transformation (spectral space to gridpoint space)

An alternative approach would be to pipeline the computation of the non-linear terms. However analysis of the algebraic decomposition required here is difficult, very model and configuration specific. Thus balancing the pipeline would also be extremely difficult and very specific. If the grid point

[4]In a more complicated and realistic model these would include the parameterisation schemes.

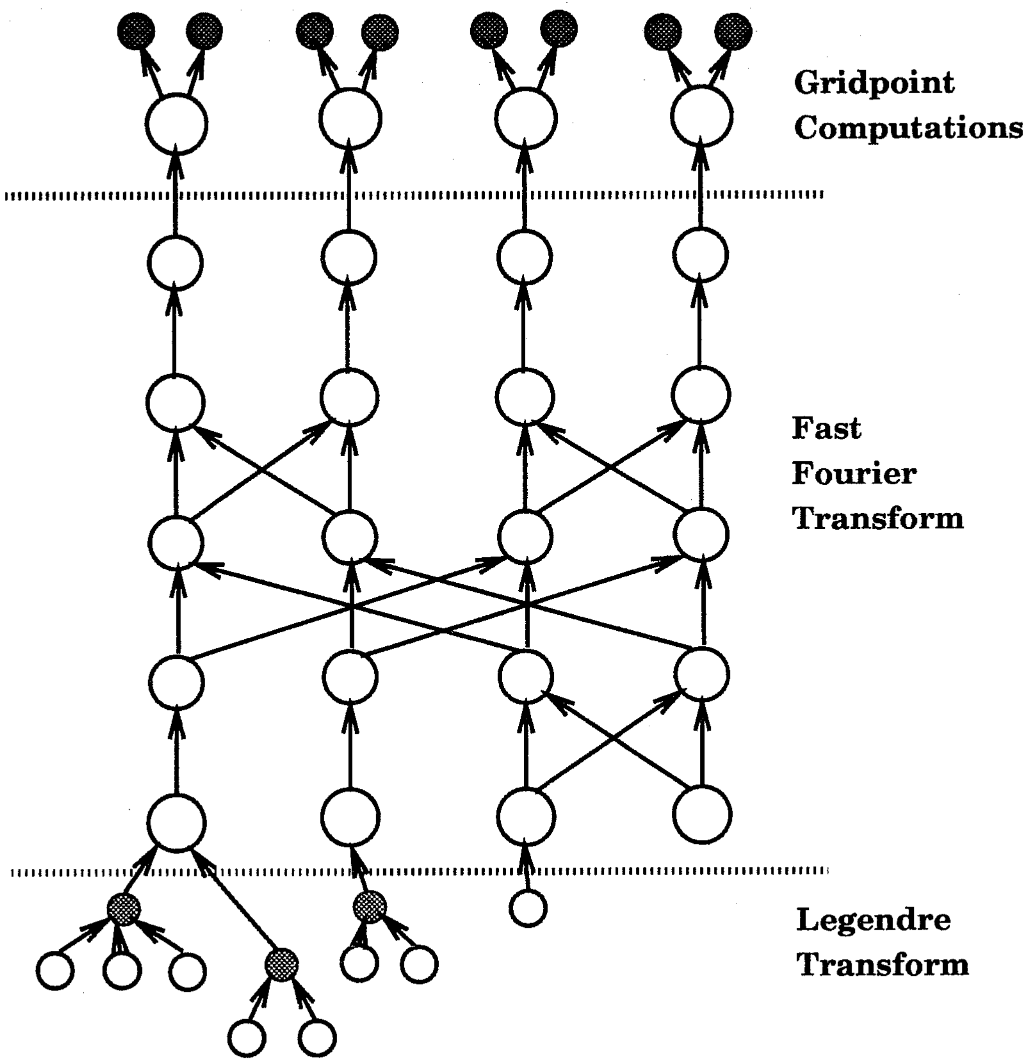

Figure 6: Processor topology for the forward transformation

computations were sufficiently time consuming then this approach would have to be used.

The inverse transformation (gridpoint to spectral) requires a fast Fourier transformation followed by a Gaussian integration. Previous sections (3, 4) have described the required topologies for these functional units. The network required is identical to that of figure 6 except for three things

1. The communications happen in the opposite direction (from top down rather than bottom up)
2. No computation of the non-linear terms is required, though if a tree is build for these terms that another will be needed to collect the data together.
3. There will also be some differences as a real FFT is used. These will not be further discussed in this paper

The reader should be aware that if the underlying hardware supports efficient bi-directional communications and rapid process swapping then some efficiency benefits will be gained if the total topology build is similar to figure 6. The tree processes will, in this case, run both a Legendre process and a Gaussian process. The butterfly processors will run both a forward and inverse FFT process. The node processors should, of course have both the copying and summing processes running. Doing this will balance out the pipeline as different numbers of variables could be transformed in the forward and inverse parts of the pipeline. The startup time will he halved. For models with a small truncation (T21, T42) this may be helpful. However only half the number of processors can be used in this case.

To complete the sub-section some calculations will be done to compute the efficiency of the pipeline. First the vector efficiency V_e of the pipeline will be computed. Table 3 summarizes the startup times and output times.

The reader should be aware that there are two contributions to efficiency. First the vector efficiency V_e defined in section 2 and then the utilization of each processor within the pipe. The utilization is the fraction of the time that the processor is doing useful work[5]. This utilization factor will stay constant as the problem scales (so long as the work per processor remains

[5]In this context useful means only that work that would be done on a serial computer

Table 3: Pipeline Times

Unit	S	τ
Legendre	$2n - 1 + d_L(r + v - 2)$	$\max(r, v - 2, 2n - 1)$
Gaussian	$2n + d_L(r + v - 2)$	$\max(r, v - 2, 2n)$
FFT	$(4 + r)\log_2 N + Kr + (q - p)c$	$\max(4, r)$
Non-lin Products	$\mathcal{G}/(v - 1)^{d_g} + rd_g$	$\max(\mathcal{G}/(v - 1)^{d_g}, r)$

constant). The factor which affects scaled speedup is the vector efficiency and this is controlled by the startup time.

$$S = \underbrace{4n - 1 + 2d_L(r + v - 2)}_{\text{Legendre/Gaussian}} + \overbrace{(4 + r)\log_2 N + Kr + (q - p)c}^{\text{Fourier Transform}} + \underbrace{\frac{\mathcal{G}}{(v - 1)^{d_g}} + rd_G}_{\text{Non-linear terms}} \tag{26}$$

From equation 14 d_L is $a\log_2 P_L + b$. P_L is the number of processors involved in computing the Legendre transform while a and b are constants. The contributions from the internal part of the FFT are also constant as is the contribution from the non-linear computations. Equation 26 can then be rewritten as;

$$S = A\log_2 P_L + B \tag{27}$$

The number of tasks is proportional to the truncation number, while from table 1 P_L is proportional to the square of the truncation. Therefore the number of tasks is given by $C\sqrt{P_L}$, C being another constant.

The vector efficiency is thus;

$$V_e = \frac{C\tau\sqrt{P_L}}{A\log_2 P_L + B + C\tau\sqrt{P_L}} \tag{28}$$

For $\sqrt{P_L}$ sufficiently large the number of tasks will dominate the startup time and V_e will tend to 1. In this limit the utilization will be $\frac{v-2}{v-1}$ and the asymptotic efficiency will be $\frac{v-2}{v-1}$. A cautionary note should be sounded, it may well turn out that sufficiently large truncation will be so large that the Spectral method is uncompetitive against a grid point method.

5.2 Timestep Increments

The final part of this section will consider the computation of the timestep increment. These, like the gridpoint computations, are highly model dependant.

The computation of the field values for a new timestep can be computed in either the Legendre processors or the Gaussian processors. The choice of which is preferred is dependant on the size of the data in the forward or inverse transforms. If the dataset is smaller during the inverse transform compared to the forward transformation then these computations should be done on the Legendre processors as communications cost is minimized. The opposite is true if the dataset is larger during the inverse transformation. The data is allready partitioned across the processors and so no extra data movement is required. Some models (Bourke, 1974) require nearest neighbour communications in which case leaf processors in neighbouring trees and/or the same tree may need connecting. Figure 7 shows this. The Reading model does not require any such connections.

Define the time taken by the increment, per discretised point in spectral space as $\mathcal{T}$. Let $\mathcal{T}'$ be the $\mathcal{T}/\tau$. $\mathcal{T}'$ is the time for the increment scaled by the characteristic time for the pipeline. S' is the scaled startup time and is defined similarly. The efficiency of the entire algorithm (pipeline plus timestep increment) will now be computed. Rather than computing the increment on just the Gaussian/Legendre processors an extra set of processors could be added in order to speed up the timestep computations. Assume that there are P_L processors involved in computing the Legendre or Gaussian transform and that an extra βP_L processors are added to compute the time step increment ($\beta \in [0, \ldots, \infty]$). These processors take no part in computing the Legendre or Gaussian transforms[6]. It can be shown that the efficiency of an entire algorithm is given by the time weighted efficiencies of all the units. That is;

$$e = \frac{\sum_i e_i \tau_i}{\sum_i \tau_i} \tag{29}$$

In this case there are two units. t refer to the timestep increment while p refer to the pipeline. P_p is the total number of processors in the pipeline. In order to simplify the analysis the effects of communications times are ignored.

[6]There would be no gain if they did as the pipe would then be unbalanced

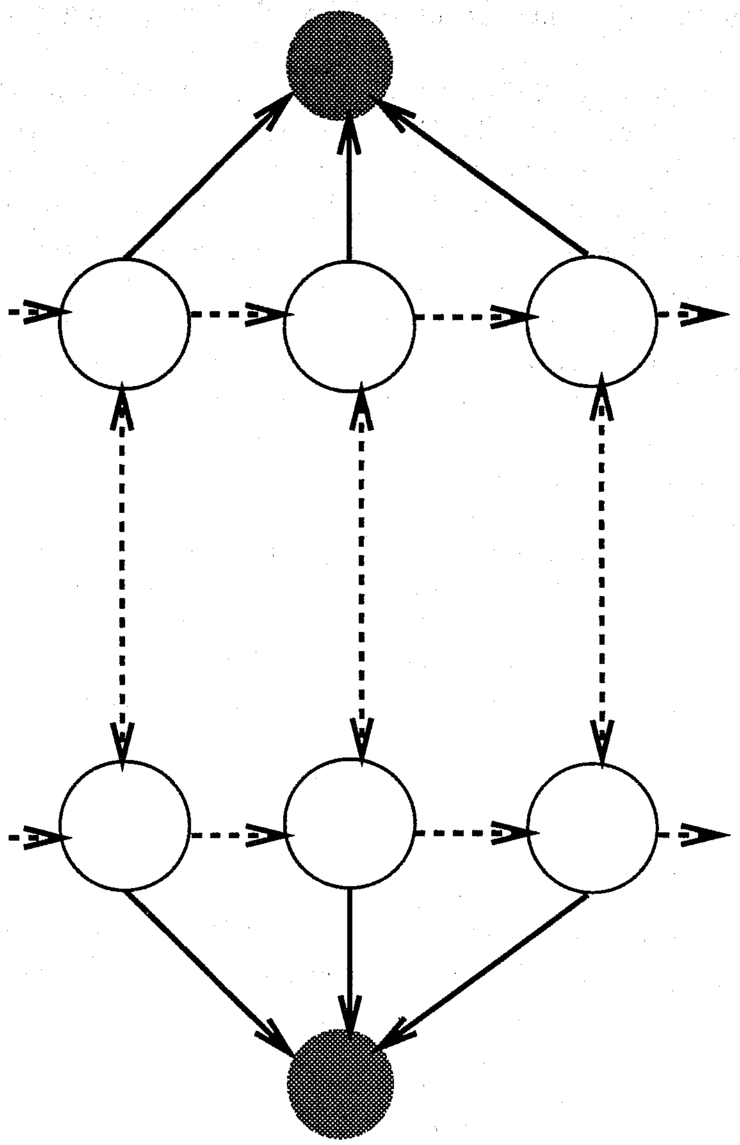
Figure 7: Possible communications between and within trees

$$\begin{aligned} e_t &= \frac{P_L(1+\beta)}{P_p+\beta P_L} \\ \tau_t &= \frac{\mathcal{T}'}{(1+\beta)} \\ e_p &= \frac{V_e P_p}{P_p+\beta P_L} \\ \tau_p &= N/V_e \end{aligned}$$

$$\Rightarrow e = \frac{1}{P_p+\beta P_L} \cdot \frac{\mathcal{T}'P_L + NP_p}{\mathcal{T}'/(1+\beta)+N/V_e} \tag{30}$$

It can be shown that $\frac{\partial e}{\partial \beta} \leq 0$ and therefore the efficiency will decrease with the maximum efficiency occurring when $\beta = 0$. This efficiency is,

$$e(0) = \frac{1}{P_p} \cdot \frac{\mathcal{T}'P_L + NP_p}{\mathcal{T}' + N/V_e} \tag{31}$$

When $N \gg \mathcal{T}'$, $P_l \sim P_p$ then $e(0) \sim V_e$. The efficiency in this case is dominated by that of the pipeline. The speedup is given by the efficiency multiplied by the number of processors, which gives,

$$S = eP = e(P_p + \beta P_L) = \frac{\mathcal{T}'P_L + NP_p}{\mathcal{T}'/(1+\beta)+N/V_e} \tag{32}$$

It can be shown that $\frac{\partial S}{\partial \beta} > 0$ and therefore the maximum speedup will occur at $\beta = \infty$. This is not particularly realistic and will be interpreted as $\beta \gg \mathcal{T}'$.

$$S(0) = \frac{\mathcal{T}'P_L + NP_p}{\mathcal{T}'/ + N/V_e} \tag{33}$$

Define the relative speedup gain as $\Delta S(\beta)$ given by;

$$\Delta S(\beta) \equiv \frac{S(\beta) - S(0)}{S(0)} \tag{34}$$

This relative speedup gain measures the speedup gain relative to using no extra processors by using βP_L extra processors to compute the timestep increment. It is,

$$\Delta S(\beta) = \frac{T' + +N/V_e}{T'/(1+\beta) + N/V_e} - 1 \tag{35}$$

The maximum gain is $\Delta S_{\max}$ and, as was allready shown, occurs at $\beta = \infty$.

$$\Delta S(\infty) = \frac{T'V_e}{N} = \Delta S_{\max} \tag{36}$$

In terms of $\Delta S_{\max}$ $\Delta S(\beta)$ can be rewritten as,

$$\Delta S(\beta) = \frac{\beta \Delta S_{\max}}{\Delta S_{\max} + (1+\beta)} \tag{37}$$

A graph of this is shown in figure 8 and is an example of Amdahl law behaviour. When N is small, relative to T', then quite large speedups may be obtained. This is because the computation of the timestep increment dominates the time taken by the Algorithm, significantly reducing the time taken by the timestep will significantly reduce the time taken by the entire Algorithm. In the case when N is large and $V_e \approx 1$ then the relative gain from increasing β is very small as the time taken for the entire Algorithm is dominated by the time taken by the pipeline.

Finally in this sub-section the value of β that gives half the maximum speedup will be computed.

$$\begin{aligned} \Delta S(\beta) &= \frac{1}{2}\Delta S_{\max} \\ \Rightarrow \tfrac{1}{2}\Delta S_{\max} &= \frac{\beta \Delta S_{\max}}{\Delta S_{\max} + (1+\beta)} \\ \Rightarrow \beta = \Delta S_{\max} + 1 & \end{aligned} \tag{38}$$

This Algorithm is unusual in that its *scaled* speedup is super-linear, i.e. $S(P+1) > \frac{(P+1)}{P} S(P)$. There are three reasons for this:

1. The number of tasks linearly with the system size while the startup time grows logarithmically. Therefore V_e will increase with increasing processor number.

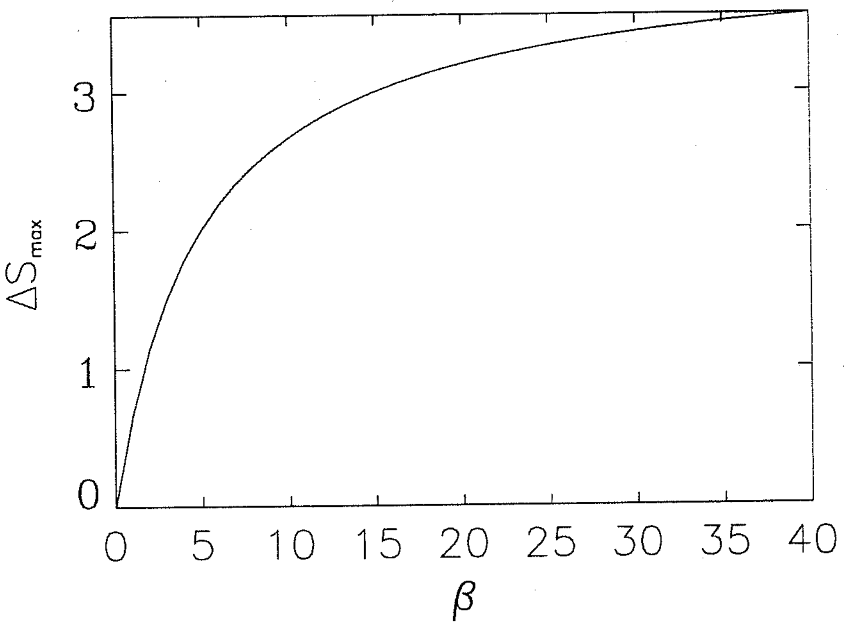

Figure 8: Relative speedup gain vs β

2. The utilization factor will be dominated by the leaf processors rather than the inefficient node and FFT processors.

3. The relative effect of the computation of the timestep increment on the speedup will fall as the number of processors increase.

The efficiency of this algorithm when using small number of processors is very poor, however if the hardware can support efficient multi-tasking then a possible solution may be possible. The partition into processes has divided the computations load equally among many processes – each process will require the same computational effort. The efficiency for small numbers of processors could be increased by mapping several processes to the same processor, this will reduce the startup time of the pipe. Further if the number of processors is the same as the number of Legendre leaf processes – then the inefficiency referred to in point 3 will not be present. The communications will be less efficient though for small numbers of processors this effect will be small.

6 Implementation Details

This section will detail a possible implementation of the Reading Model on the Transputer machine at Edinburgh.The machine at Edinburgh is build from 400 T800 transputers (Bowler *et al*, 1987) and has been in existence since 1987(Bowler *et al*, 1989). The aim of the implementation is to convert the existing FORTRAN code. This is done for three reasons:

1. The process of converting this relatively simple model may provide some guidance to the effort in converting a larger and more complex model.

2. It is hoped that this conversion could be quickly done.

3. Verification of each stage is possible by comparing the partially converted model against the serial model.

The communications between processes was provided by a set of Fortran library routines supplied by Meiko called CS-Tools(Mei). Meiko also provided a set of routines to build a loader/configurer to allow mapping of processes to processors. Separation of the two allows development of the separate processes to occur on a workstation. Much of what follows is specific to the Meiko system and the Reading Model – however the author hopes that this will encourage others to convert their spectral models and provide some aid in the process. The author estimates that approximately 6 months are required to convert the Reading Model.

The general approach to convert the model is, for each one of the component sub-routines, a "wrapper" module would be written which should first carry out data initialisation, then repeatedly carry out the following tasks: communications from the previous module in the pipeline; if necessary process the input data into a format suitable for the sub-routine; call the sub-routine; again if necessary do some processing on the data in order to put it in a form suitable for output; then transfer data to the next module in the pipeline.

Considering the grid-point computation, there are only two modifications that require to be done; the maximum length of the longitudinal sub-strip on the processor will be different from the serial model and will need to be computed by the loader program. The actual size of this strip, if the tree for the gird-point processes is build, may vary from process to process and require initialisation from the initialisation module. Apart from these changes the initial grid point subroutine may be used.

Due to the need to avoid bit-reversal the FFT required rewriting. Even if not avoiding bit-reversal was an option, the subroutine would still have needed extensive modifications as communications occur frequently and cannot be isolated outside the serial subroutine.

The next modules that are considered are the Legendre and Gaussian transforms. In both cases the serial sub-routines have a triangular data-structure. This large triangle can be completely covered by a set of smaller triangles, all these triangles are the same size as that used by a serial $T(2^n-1)$ model. For each of these triangles again the original subroutine can be used. This restriction on the size is to allow a radix 2 FFT to be used. Some points on some sub-triangles will lie outside the original triangle. At these points the value of the Legendre polynomials are set to zero in order these points make no contribution to the transform. Figure 9 shows this mapping. Some of the triangles are upside down and reversed, in this case some data-manipulation is required to convert the sub-triangles into the required form for the serial subroutines. After the sub-routine has processed the data some more data-

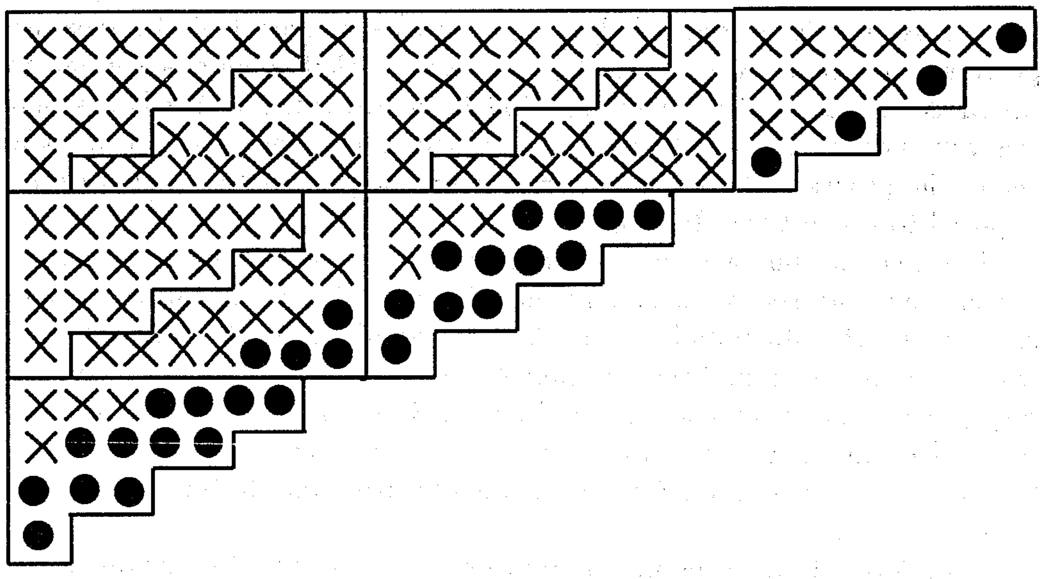

Figure 9: $T21$ model partitioned into 9 $T7$ modules
This figure shows how a t21 model could be split over 9 processors. The circles represent points where the Legendre polynomials are set to zero.

manipulation is required; the longitudinal rows will require reversing for the upside, reversed triangles in order to compute the FFT.

7 Concluding Remarks

As of June 1991 implementation of the algorithm is almost complete, the various processes are written and produce similar results to the serial version of the model. The only differences are due to the different order of addition on the parallel version of the model compared to the serial version. There are 2 questions that require answering before an implementation on several hundreds or thousands or processors could succeed.

- On these large parallel machines can an arbitrary topology be build and, in particular, can the effects of "long" wires be neglected.
- The model investigated is a pure dynamics model – the effects of "physics" have been ignored. Exactly how to implement these schemes

in the context of this algorithm is not clear at present.

8 Acknowledgements

This work was done as part of the Authors Ph.D work. This work was supported by the U.K. Meteorological Office and S.E.R.C. Use was made of the ECS which is supported by the DTI, Computer Board and S.E.R.C. . Substantial support for the ECS was also provided by University of Edinburgh and industrial affiliates.

Many thanks are due to the department of Meteorology at Reading University for providing the Spectral Model code used for this study. Thanks are due to Bob Davis at the UKMO for drawing figure 8.

References

Bourke, W., 1974. A multi-level spectral model. i. formulation and hemispheric integrations. *Monthly Weather Review*, **102** pp. 687–701.

——, 1988. Spectral methods in global climate and weather prediction models. In Schlesinger, M. E., editor, *Physically-Based Modelling and Simulation of Climate and Climate Change*. Kluwer Academic Publishers.

Bowler, K., R. Kenway, G. Pawley, and D.Roweth, 1987. *An Introduction to Occam 2 Programming*. Chartwell-Bratt, Sweden.

——, ——, and D. J. Wallace, 1989. The edinburgh concurrent supercomputer: project and applications. In C.R.Jesshope, and R. Reinartz, editors, *CONPAR 88*, pages 635 – 642. Cambridge University Press.

Carver, G., 1988. A spectral meteorological model on the ICL DAP. *Parallel Computing*, **8** pp. 121–126.

Courtier, P., and J.-F. Geleyn, 1988. A global numerical weathere prediction model with variable resolution: Application to the shallow water equations. *Quarterly Journal of the Royal Meteorological Society*, **114**(483) pp. 1321–1346.

Fox, G., M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, 1988. *Solving Problems on Concurrent Processors*, volume 1 General Techniques and Regular Problems. Prentice-Hall International, London.

Girard, C., and M. Jarraud, 1982. Short and medium range forecast differences between a spectral and grid point model. an extensive quasi-operational comparison. Technical Report 32, European Centre for Medium Range Weather Forecasting, ECMWF, Shinfield Park, Reading.

Hockney, R. W., and C. Jesshope, 1988. *Parallel Computers 2*. Adam Hilger, Bristol.

Hoskins, B. J., and A. J. Simmons, 1975. A multi-layer spectral model and the semi-implicit method. *Quart. J. Roy. Met. Soc.*, **101** pp. 637–655.

James, I. N., and B. J. Hoskins, 1990. An overview of the uk universities' atmospheric modelling project. Technical Report 12, UK Universities' Global Atmospheric Modelling Project, Dept. of Meteorolgoy, University of Reading, Reading RG6 2AU.

Machenhauer, B., 1979. The spectral method. In *Numerical methods used in atmospheric models*, volume II of *No. 17*, pages 121–275. GARP publication.

Meiko Scientific, 650 Aztec West, Bristol, BS12 4SD, UK. *CS-Tools for SunOS ver. 1.10 - 1.14*. Ref. No. 83–009A00–02.02.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, 1986. *Numerical Recipes: the Art of Scientific Computing*. Cambridge University Press.

Ritchie, H., 1987. Semi-lagrangian advection on a gaussian grid. *Monthly Weather Review*, **115** pp. 608–619.

Sato, R. K., and P. N. Swarztrauber, 1988. Benchmarking the connection machine. In *Proceedings of the Supercomputing Conference, Orlando Florida*, pages 304–308. IEEE Computer Society Press.

Swarztrauber, P. N., and R. Sato, 1990. Solving the shallow water equations on the Cray X-MP/48 and the Connection Machine 2. In Hoffman, G.-R., and D. K. Maretis, editors, *The Dawn of Massively Parallel Processing in Meteorology*, pages 260–276. Springer-Verlag.

Temperton, C., 1983. Self–sorting mixed–radix fast fourier transforms. *Journal of Computational Physics*, **52** pp. 1–23.