

The Implementation of a Large-Scale
Numerical Model of the Atmosphere
on a PC-Based Transputer Network

Author :

Paul A. Halton B.Sc.

Supervisor:

Michael O'hEigeartaigh

Submitted to:

Dublin City University


School of Computer Applications

For the Degree of Master of Science.

August 1992

Declaration

This study is based on the author's own investigative work.
It has not been previously submitted for a degree at any
academic institution.



Paul Halton B.Sc.

28th August 1992

Table of Contents

Chapter 1	1
Introduction	1
1.1 Motivation for this research work	3
1.2 Brief Description of the HIRLAM Model	4
1.3 Selection of a Test Case	5
1.3.1 Selection of "Control" Platforms	5
1.4 Overview of Thesis	6
Chapter 2	9
Model Description	9
2.1 Introduction	9
2.2 Weather Forecast Model	11
2.3 The HIRLAM Model	12
2.3.1 The Continuous Equations	14
2.3.2 Finite Difference Scheme	18
2.3.3 Arrangement of Variables on the Grid	21
2.4 Time Integration Scheme	23
2.4.1 Length of Time Step	24
2.4.2 Multi-Grid Solution Method	25
2.4.3 Size of Integration Area	26
2.5 Horizontal Diffusion Scheme	27
2.6 Boundary Relaxation Scheme	27
2.7 Physical Parameterisation	29
2.7.1 Radiation	30
2.7.2 Convection	30
2.7.3 Stratiform Condensation	31

2.7.4	Vertical Diffusion	31
2.8	Numerical Formulation of Semi- Lagrangian Methods	32
2.9	Computational Time Step Algorithm . .	37
2.10	Summary	40
Chapter 3	42
HARDWARE DESCRIPTION		42
3.1	The Transputer	42
3.1.1	Transputer Architecture	42
3.1.2	Transputer Communication Links	44
3.1.3	Transputer Memory	46
3.1.4	Transputer Networks	46
3.2	Control of Transputer Communication .	47
3.3	Maximising the benefits of on-chip RAM	47
3.4	Booting a Transputer Network	47
3.5	Host PC Configuration	48
3.6	Graphics Subsystem	49
3.7	Process Scheduling on the Transputer .	49
3.8	Channel Communication	51
3.8.1	Software Channel Types	52
3.8.2	Channel Mapping	52
3.9	Grid-Based Communication on a Transputer Network	53
3.10	Summary	54

Chapter 4	55
Programming Languages for Transputers	55
4.1 Introduction	55
4.2 OCCAM	55
4.3 PARALLEL "C" and PARALLEL PASCAL	57
4.4 PARALLEL FORTRAN	57
4.5 Channel Communication using PARALLEL FORTRAN	58
4.6 Tasks and Threads in Parallel Fortran	59
4.6.1 Tasks	61
4.6.2 Threads	62
4.7 Configuration Language for Task Placement	64
4.7.1 The Static Configurer	64
4.7.2 The Flood-Fill Configurer	66
4.7.3 Processor Farms	66
4.7.4 Disadvantage of Flood-Fill Configuration	67
4.7.5 Static Configuration for HIRLAM	67
4.8 Memory Management	70
4.8.1 Determination of Static Space Requirements for a Task	70
4.8.2 Determination of Heap and Stack Requirements for a Task	70
4.8.3 Placement of Critical Routines in fast RAM	71
4.9 Summary	72

Chapter 5	73
Algorithms for Parallel Environment	73
5.1 Introduction	73
5.2 Analysis of Sequential Program	74
5.3 Grid Point Algorithms	75
5.3.1 Arrangement of Variables in HIRLAM model	76
5.4 Computation of Workspace Parameters	78
5.5 Finite Difference Methods	79
5.6 Time Integration	80
5.7 Numerical Instability	81
5.8 Efficiency of Semi-Implicit Time Stepping	82
5.8.1 Disadvantages of FFT in Transputer Environment	82
5.8.2 Iterative Solution Method	83
5.8.3 Alternative Parallel Solution Methods	83
5.8.4 Alternative Iterative Methods	84
5.9 Degrees of parallelism in iterative solution methods	84
5.10 Speedup of Iterative Convergence	85
5.11 Grid Partitioning	86
5.12 Data Distribution by Grid Partitioning	87
5.13 Possible Ways of Partitioning	91
5.14 Summary	92

Chapter 6	93
Model Modifications Undertaken	93
6.1 Introduction	93
6.2 Modifications made in phase one.	93
6.3 Modifications made in phase two	94
6.4 Modifications made in phase three	94
6.5 Task Placement Strategy Devised	95
6.5.1 Using a Multiplexer Technique	96
6.6 Consequences of Task Redistribution	97
6.6.1 Advantages of Task Distribution	97
6.6.2 Disadvantages of Task Distribution	97
6.7 Problems reading binary files	98
6.8 Compiler Problems associated with Large programs	98
6.9 Using The Parallel Fortran Debugger, TBUG.	99
6.9.1 Extracting a Sample Data Set from a VAX-4200 program	99
6.9.2 Limitations of TBUG Debugger	100
6.10 A Sample Test Program	101
6.10.1 Code for the Master Task	103
6.10.2 Code for the SLAVE Task	104
6.10.3 The Flood-Fill Configuration File	105
6.10.4 The Normal Configuration File	106
6.10.5 Link Files for Master and Slave Tasks	108
6.10.6 Batch File to control compilation Master- Slave Tasks	109
6.11 Summary	110

Chapter 7	111
Program Performance Evaluation	111
7.1 Introduction	111
7.2 Performance of Sequential version on the ROOT Transputer	112
7.3 Comparison of Execution Times	113
7.4 Parallel Architecture Implications	115
7.4.1 Hardware Implications	115
7.4.2 Software and Algorithm Implications	116
7.4.3 Compiler Implications	117
7.5 Parallel Behaviour of the Atmosphere	118
7.6 Data Flow Computing	118
7.7 Speedup and Amdahl's Law	119
7.8 Scaled Speedup	120
7.9 Summary	122
 Chapter 8	 123
HIRLAM Program Execution on Transputers	123
8.1 Introduction	123
8.2 HIRLAM Code Characteristics	123
8.3 Memory Requirements	124
8.4 Program Initialisation Phase	125
8.5 Main Computational Phases	126
8.5.1 Main Integration Phase	132
8.5.2 Semi-Lagrangian Phase	132
8.5.3 Computation of the Physical Parameters	133
8.5.4 Explicit Leapfrog Time Stepping Phase	134

8.5.5	Resolution of Helmholtz Equations	
	Phase	135
8.6	Input Test data	136
8.6.1	Boundary Scheme Used	137
8.7	Summary	138
Chapter 9	140
	Using HIRLAM to Model a Severe Storm	140
9.1	Introduction	140
9.2	Development of Hurricane Charlie	142
9.3	Selection of Initial Data	142
9.4	The Storm Over Ireland	145
9.5	Using HIRLAM to Forecast the Storm	146
9.5.1	Grid Spacing	147
9.6	Using HIRLAM to Forecast Surface	
	Pressure	148
9.7	Using HIRLAM to Forecast Wind	152
9.8	Using HIRLAM to predict Geopotential	160
9.9	Using HIRLAM to predict Temperature	161
9.10	Using HIRLAM to predict Vertical	
	Velocity	180
9.11	Summary	180
CONCLUSIONS and FUTURE DIRECTIONS	181
References	186
Glossary	192
Appendices	A - D

ABSTRACT

This thesis is a report of the study of a large-scale Numerical Weather Prediction Model. The study investigated the feasibility of applying parallel algorithms to the HIRLAM Model so that it could be implemented on a Transputer Network.

A set of partial differential equations which describe the behaviour of the atmosphere are presented and numerical methods are explored. The investigations focused on time critical regions of the sequential programs. From these a criterion for task distribution was devised. Expressions for the computation of speedup and scaled speedup were derived.

A special set of test data, extracted from the Analysis of Hurricane Charlie (August 1986) was used as input data for a 24 hour forecast. We report on the Case Study and how the model predicted the storm over Ireland.

For comparison purposes the Model was also run on a VAX 4200 and on a Dell 386-SX PC, with the same data. The execution of the critical program modules was monitored throughout and a table of results is presented.

Acknowledgements

I wish to thank members of the Research Department of the Irish Meteorological Service, Peter Lynch, Aidan McDonald and Jim Hamilton for providing invaluable information including the Numerical Model used in this research project. Their help and guidance is gratefully appreciated. I also wish to thank my Tutor Michael O'hEigeartaigh for his help, encouragement and inspiration during the course of the work. Finally, but not least, I wish to thank Mary Anne for her continuous support.

Chapter 1

Introduction

During the last decade, computer research scientists, with the financial backing of major government departments and organisations in several countries, have been investigating and developing parallel processing architectures. Alongside this research, computer languages suitable for the application of parallel algorithms on concurrent computers have been designed, tested and implemented.

Complex computer programs which could only have been run on supercomputers, such as the CRAY series, can now be resolved using inexpensive concurrent platforms. The Transputer was launched in 1985 and was hailed by INMOS, as a most significant event in concurrent processing[19]. Despite the early criticisms of the transputer environment, transputers continue to grow in popularity. This is evident from the growing number of publications and research papers devoted to developing alternative algorithms and languages best suited to this environment.

The RISC architecture of the transputer is considered suitable for a wide variety of applications including image processing, fingerprint matching, graphics workstations, artificial intelligence and weather forecasting. From a

hardware point of view the building of transputer networks to perform tasks in parallel is no longer a serious challenge.

However, software development for the transputer has not advanced as quickly. In the past, a shortage of well proven parallel algorithms inhibited the growth of software development. This situation is gradually improving with many system designers and programmers taking up the challenge of exploiting the benefits offered by the transputer environment.

This dissertation reports on the research work carried out in taking a large-scale, semi-operational numerical model of the atmosphere from a sequential Von Neumann platform to a parallel platform offered by a transputer network. The work gives an insight into the difficulties of porting a large Fortran program onto a PC-based Transputer System.

The numerical weather prediction model, HIRLAM-2, was kindly made available for this research work. The HIRLAM, (HIgh Resolution Limited Area Model), system currently represents 50 man-years of research and development work, contributed jointly by the meteorological communities of six European countries including Ireland. The model is, therefore, based on the most modern numerical techniques and is coded in Fortran in accordance with clearly defined guidelines and consists of more than 20,000 lines of code.

1.1 Motivation for this research work

The motivation for undertaking the research work with a large Numerical Weather Prediction (NWP) model was:

(1) to build upon previous experience gained in the development and implementation of an atmospheric simulation model of the Shallow Water Equations on a T414 transputer using the OCCAM programming language;

(2) the availability of a Fortran compiler for the transputer environment reduced the risk of language and algorithm translation errors, during the porting procedure;

(3) the set-up costs for a 'supercomputer' such as a CRAY-YMP is prohibitive for an organisation like the Irish Meteorological Service with strict budgetary constraints. Therefore, a transputer solution, offering an implementation platform which could be added to as budgets allow, could be an attractive alternative;

(4) to gain an understanding of the HIRLAM weather forecast model together with the underlying equations and algorithms;

(5) to investigate modern methods of concurrency suitable for the parallel implementation of the model on a transputer network.

Many of the concepts of parallelism in Meteorological models have been explored, particularly at the European Centre for Medium-Range Weather Forecasts (ECMWF). It has to date hosted three Workshops on the use of parallel processing

in meteorology. The workshops began in 1986 and have been held every second year since then [25,26].

During the early stages of developing the HIRLAM system the research scientists used the facilities of the CRAY series of computers at ECMWF. The CRAY provided the only platform which was capable of executing the program with the resolution required. Therefore, the final semi-operational version of HIRLAM-2 is designed to be memory resident and uses long vector computational techniques. Thus, it is clear that at the moment the model requires a platform capable of offering the computational power of a supercomputer such as a CRAY.

The questions to be put are :-

- (a) Is there an alternative inexpensive implementation solution available?
- (b) Is a parallel solution capable of delivering a significant gain in speedup?
- (c) Does the existing program structure require radical modifications in order to run in a parallel environment?

1.2 Brief Description of the HIRLAM Model

The HIRLAM system was specifically designed to provide high resolution weather forecasts over a limited area represented by a grid-point formulation. Some Scandanavian countries are now running a version of the model four times every day. Extensive tests have been carried out to determine its ability to identify the rapid development of storms which

are not detected by existing medium- to long-range weather forecast models. Medium-range weather forecasts are currently only produced at ECMWF once per day.

In order to be sure of detecting small localised intense storms the Limited Area Model (LAM) must use a high resolution grid of about 0.5° in both the Longitude and Latitude directions. This ideally requires grid values of 110 and 100, respectively, with 16 levels or more in the vertical. The time to compute a 24 hour forecast on a CRAY-YMP, with this resolution, and using 6 minute time step intervals, is reported as 424 seconds in [4].

1.3 Selection of a Test Case

The 'Hurricane Charlie' of August 1986 was chosen as a suitable test case for this research work. The actual weather data leading up to the development of the storm was used as input test data. The program was set up to produce a 24 hour forecast based on this input data. The storm has been well documented. It was, therefore, an ideal test case for comparison of forecast predictions against the actual weather recorded during the storm.

1.3.1 Selection of "Control" Platforms

It was decided early in the study to set up a sequential version of the HIRLAM program on a Digital VAX-4200 computer. The VAX-4200 computer has a clock rate of 114 MHz, 64 bit architecture, 4 Gbytes of virtual address space and physical

memory space of 512 Mbytes. The sequential version of the HIRLAM model, which was run on the VAX-4200, was used as a "control" model against which results from the transputer environment could be compared.

For comparison purposes another sequential version of the HIRLAM model was run on a DELL-320SX PC. The DELL-320SX PC features a clock rate of 20 MHz, 16 bit architecture, base memory of 640 Kbytes and extended memory of 1 Mbytes.

A network of eleven T800 transputers was available. Each transputer provided a clock rate between 20 and 25 MHz, 32 bit architecture, and an on-chip memory of 2 Mbytes including 4 Kbytes of fast RAM. These transputers were hosted by an Olivetti 486i PC with a clock rate of 33 MHz, a 32 bit architecture, a base memory of 640 Kbytes and an extended memory of 8 Mbytes.

1.4 Overview of Thesis

In chapter 2 the methods used in the development of numerical models of the atmosphere are discussed. In particular the approach used in the development the Limited Area Model known as HIRLAM is described. This involves brief descriptions of the underlying model equations. The discussion focuses on the Semi-Lagrangian Scheme used in the model and describes the computation of air parcel trajectories.

Chapter 3 focuses on the Transputer and describes its architecture and working environment. In particular, the

concepts of interprocessor communication, over channels, are introduced. The discussion also focuses on the suitability of transputer networks for grid based communication. Chapter 4 deals with the programming aspects of application development in the transputer environment. The concepts of message passing between communicating sequential processes on a distributed architecture are introduced. Network configuration and transputer memory management principles are outlined.

The issues affecting the porting of sequential Fortran programs to a transputer environment are addressed in Chapter 5. Suitable algorithms are introduced and the significance of their adaptation is discussed. Chapter 6 described the steps taken to modify the actual HIRLAM program. A sample Parallel Fortran program is used to illustrate the development approach used in the modifications.

The consequences of moving to a parallel environment are outlined in Chapter 7. The timing performance of the various program modules implemented on two sequential computers are compared to the results obtained with the implementation on the transputer network. The issues of speedup and its computation are also addressed.

The actual program structure as implemented on the transputer network is presented in Chapter 8. The main phases of the program are described. The analysis and boundary data used are also described.

Chapter 9 begins with a description of hurricane Charlie. The HIRLAM model is used to model the behaviour of

the atmosphere based on the analysis data for midnight on 25 August 1986. It produces a 24 hour forecast by using lateral boundary data from the medium range weather forecast, for the same period, as issued by the ECMWF. A selection of forecast products are compared by using the output from the VAX-4200 as a 'control' for the quality of the corresponding products from the transputer implementation.

This is followed by a Concluding summary of the work undertaken and points to further research work.

Chapter 2

Model Description

2.1 Introduction

The results of study in the area of Dynamic Meteorology in the mid twentieth century have been consolidated in recent years. This is evident in the formulation of the current version of the HIRLAM model. The numerical formulation of the HIRLAM model is discussed in this Chapter together with the implementation schemes employed to produce a 24hr forecast. Dynamic Meteorology deals with the atmospheric motions associated with changes in weather patterns.

There are five stages involved in the HIRLAM system and they are: the analysis, the initialisation, the forecast, the post processing and the graphics presentation of results. A Data Flow Diagram for these stages is shown in Figure 2.0.

The HIRLAM system[16] is an up-to-date limited area weather forecasting system which can be used operationally for the production of short range weather forecasts by the weather services of the participating countries. The version of the model made available for this research work has been optimised for a vector processor such as the CRAY-YMP8 at ECMWF. Therefore, it was coded in FORTRAN and consisted of very long loops.

Data Flow Diagram for HIRLAM program suite

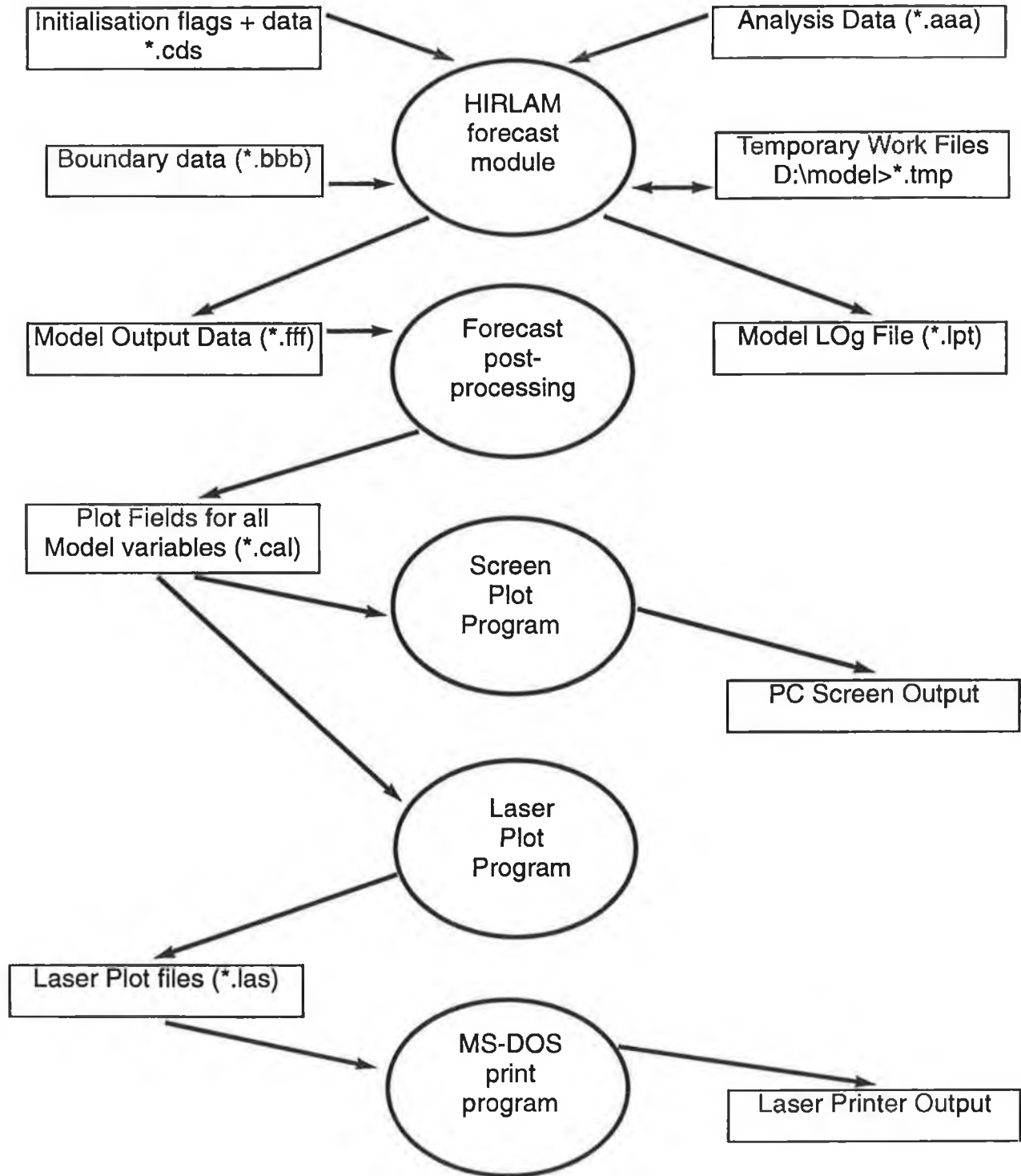


Figure 2..0 : Top Level Data Flow Diagram for HIRLAM programs

A data flow diagram for the HIRLAM system implemented during this research work is shown in Figure 2.0. The structure diagrams, for the process marked 'HIRLAM forecast module', are presented in Chapter 8. The research work undertaken focuses on the 'forecast module' of the HIRLAM program suite. The 'post-processing' and 'graphics modules' were also made available for this research project. The latter modules were used for the production of the final forecast products. During the early stages of the research work all of the modules were adapted for implementation on a DELL-320SX PC. This was achieved with the aid of Fortran Compiler from Salford University[20].

2.2 Weather Forecast Models

Atmospheric motion is governed by three fundamental physical laws: conservation of mass, conservation of momentum and conservation of energy[35]. An infinitesimal control volume in a fluid is used for the derivation of the Mathematical relations which express these laws. In fluid dynamics, two types of control volumes are generally used.

The first type is known as the Eulerian frame of reference[35]. This uses a control volume whose position is fixed, relative to the coordinate axes. The control volume consists of a parallelepiped of sides dx , dy , dz . The status of mass, momentum and energy, at any given time, depends on the fluxes due to the flow of fluid through the boundaries of the control volume. The Eulerian system is convenient for solving most problems, because the field variables are

related by a set of partial differential equations whose independent variables are the coordinates x , y , z and t .

The second type is known as the Lagrangian frame[35]. In this case, the control volume consists of an infinitesimal mass of "tagged" fluid particles. The position of the control volume is governed by the motion of the fluid within it, and it always contains the same fluid particles. The Lagrangian system is useful for the derivation of conservation laws because these may be expressed in terms of a particular mass element of the fluid. In this case, it is necessary to follow the time evolution of the fields of the various individual fluid parcels. Therefore, the independent variables are x_0 , y_0 , z_0 and t_0 , where x_0 , y_0 , z_0 , designate the position of a fluid parcel at a reference time t_0 .

2.3 The HIRLAM Model

The HIRLAM model is a primitive equation model which uses a three-dimensional, (3D), grid point formulation with second order difference approximations for the spatial derivatives. The main forecast variables are temperature, atmospheric pressure, air density, specific humidity, horizontal wind speed and direction, and vertical wind.

These physical quantities characterise the state of the atmosphere at any point in time. At each point in the atmospheric continuum they have unique values. The physical quantities are referred to as field variables. Laws governing atmospheric motion involve field variables being expressed in terms of partial differential equations (PDE).

Basic equations for the HIRLAM model include:

(a) Continuous Equation Formulation; (b) a Finite Difference Scheme; (c) a Time Scheme; (d) Horizontal Diffusion with a Non-Linear Second Order Scheme and a Linear Fourth Order Scheme; (e) a Boundary Relaxation Scheme.

The equations of motion are derived from Newton's Second Law of Motion. It stipulates that if there are several forces acting on a body we can form the resultant of these forces. The resultant force will produce an acceleration in the direction of the resultant[40]. The forces which act on a unit mass of atmospheric fluid are gravity, pressure-gradient force, the Coriolis force and friction.

The force of gravity is the resultant of two forces namely the gravitational pull of the earth and the centrifugal force due to the earth's rotation. The gravitational force decreases with height above the earth's surface.

When atmospheric pressure varies with distance, the air experiences a pressure-gradient force. This force tends to drive the air from an area of high pressure towards an area of low pressure. It is a three dimensional vector denoted by:

$$-\nabla_p = - \left(\frac{\partial p}{\partial x} i + \frac{\partial p}{\partial y} j + \frac{\partial p}{\partial z} k \right) \quad (2.1)$$

where i , j , k , are unit vectors in the x , y , z directions respectively. The pressure-gradient force is negative because it acts from high to low pressure values.

The behaviour of the atmosphere is subjected to a deflecting force known as the Coriolis force[35]. The motion

of a body on the rotating earth, when viewed by an observer who is also rotating with the earth, behaves as though there is such a deflecting force[40]. The Coriolis force is sometimes known as 'geostrophic acceleration'.

The general expression for Coriolis force, f , used in meteorological models is:

$$f = 2\Omega V \sin\theta \quad (2.2)$$

where V = the speed of the body
 Ω = the rate of rotation of the system
 θ = the latitude

In meteorology, the effects of friction are important for the flow of air over the earth's surface. The effect of friction, within a fluid, that arises from molecular collisions is termed Viscosity Force[40]. If the momentum of air is reduced by "eddy" exchange rather than molecular exchange then the retarding force of the air motion is called "eddy" friction. Friction is most important in the lowest 1000 metres of the atmosphere. This layer is often referred to as the friction layer[35].

2.3.1 The Continuous Equations

The HIRLAM model uses a horizontal grid which is a regular spatially staggered latitude-longitude grid known as the Arakawa C-grid[30]. This allows a spherical coordinate system (λ, θ) to map onto a projection such as the x, y axis

by using two metric coefficients (h_x, h_y).

For a general vertical coordinate $\eta(p, p_s)$ which is pressure based and terrain based we have:

$$\eta(0, p_s) = 0 \quad \text{and} \quad \eta(p_s, p_s) = 1$$

Therefore, for a distance $(\delta X, \delta Y)$ on the surface of the earth we have:

$$\delta X = ah_x \delta x \quad \text{and} \quad \delta Y = ah_y \delta y$$

and for the spherical rotated coordinates we have:

$$\delta X = a \cos\theta \delta\lambda \quad \text{and} \quad \delta Y = a \delta\theta$$

where a is the radius of the Earth.

The Momentum Equation for the Horizontal wind component, u , along the x -axis is given by:

$$\frac{\partial u}{\partial t} = (f+\xi) v - \eta \frac{\partial u}{\partial \eta} - \frac{R_d T_v}{ah_x} \frac{\partial \ln p}{\partial x} - \frac{1}{ah_x} \frac{\partial}{\partial x} (\phi+E) + P_u + K_u \quad (2.3)$$

and the Vertical Wind component, v , along the y -axis is:

$$\frac{\partial v}{\partial t} = -(f+\xi) u - \eta \frac{\partial v}{\partial \eta} - \frac{R_d T_v}{ah_y} \frac{\partial \ln p}{\partial y} - \frac{1}{ah_y} \frac{\partial}{\partial y} (\phi+E) + P_v + K_v \quad (2.4)$$

The Thermodynamic Equation expresses the law of conservation of energy. It is written in terms of the potential temperature, T , and includes terms which represent the effects of evaporation, condensation and radiation.

The Thermodynamic Equation conserves the sum of kinetic energy and the available potential energy and takes the form:

$$\frac{\partial T}{\partial t} = -\frac{u}{ah_x} \frac{\partial T}{\partial x} - \frac{v}{ah_y} \frac{\partial T}{\partial y} - \eta \frac{\partial T}{\partial \eta} + \frac{KT_v \omega}{(1 + (\delta - 1) q) p} + P_T + K_T \quad (2.5)$$

where

$$\xi = \frac{1}{ah_x h_y} \left(\frac{\delta}{\delta x} (h_y v) - \frac{\delta}{\delta y} (h_x u) \right) = \text{Vorticity}$$

$$E = \frac{1}{2} (u^2 + v^2) = \text{Kinetic Energy}$$

$$T_v = T \left(1 + \left(\frac{1}{\epsilon} - 1 \right) q \right) = \text{Virtual Temperature}$$

$$\delta = \frac{C_{pv}}{C_{pd}} = \text{Divergence}$$

$$\epsilon = R_d / R_v$$

R_d = Gas Constant of dry air

R_v = Gas Constant of water vapour

ϕ = Geopotential

ω = Pressure Vertical Velocity

K_i = Tendency of i due to horizontal diffusion

P_i = Tendency of i due to physical parameterisation

C_{pv} = Specific Heat of Moist air at constant pressure

C_{pd} = Specific Heat of Dry air at constant pressure

K = Horizontal diffusion coefficient

f = Coriolis Parameter, Equation(2.2)

p = pressure

q = specific humidity

The law of conservation of moisture is expressed by the Moisture equation as follows:

$$\frac{\partial q}{\partial t} = - \frac{u}{ah_x} \frac{\partial q}{\partial x} - \frac{v}{ah_y} \frac{\partial q}{\partial y} - \eta \frac{\partial q}{\partial \eta} + P_q + K_q \quad (2.6)$$

The Hydrostatic equation is used as a diagnostic equation in order to compute the geopotential, Φ . The variation of geopotential with respect to pressure depends only on temperature. This relationship is expressed by the Hydrostatic equation:

$$\frac{\partial \Phi}{\partial \eta} = - \frac{R_d T_v}{p} \frac{\partial p}{\partial \eta} \quad (2.7)$$

The Continuity equation expresses the law of conservation of mass. It states that a mass of air entering an elementary volume is equal to the increase of mass within it.

The continuity equation is given as:

$$\frac{\partial}{\partial \eta} \frac{\partial p}{\partial t} + \nabla \cdot (\vec{v}_h \frac{\partial p}{\partial \eta}) + \frac{\partial}{\partial \eta} (\eta \frac{\partial p}{\partial \eta}) = 0 \quad (2.8)$$

where the divergence operator is defined by:

$$\nabla \cdot \vec{v}_h = \frac{1}{ah_x ah_y} \left(\frac{\partial}{\partial x} (h_y u) + \frac{\partial}{\partial y} (h_x v) \right) \quad (2.9)$$

The divergence operator expresses the time rate of horizontal expansion of the air per unit area.

By integrating the continuity equation using the boundary conditions $\dot{\eta} = 0$ when $\eta = 0$ and $\eta = 1$ the equation for surface pressure tendency is obtained. It takes the form:

$$\frac{\partial p_s}{\partial t} = - \int_0^1 \nabla \cdot (\vec{v}_h \frac{\partial p}{\partial \eta}) d\eta \quad (2.10)$$

Vertical motion is related to horizontal wind divergence by means of the continuity equation. This relationship is expressed by the Pressure Vertical Velocity equation:

$$\omega = \frac{\partial p_s}{\partial t} + \int_{\eta}^1 \nabla \cdot (\vec{v}_h \frac{\partial p}{\partial \eta}) d\eta + \vec{v}_h \cdot \nabla p \quad (2.11)$$

and finally the equation for $\dot{\eta}$ is given by:

$$\dot{\eta} \frac{\partial p}{\partial \eta} = (1 - \frac{\partial p}{\partial p_s}) \frac{\partial p_s}{\partial t} + \int_{\eta}^1 \nabla \cdot (\vec{v}_h \frac{\partial p}{\partial \eta}) d\eta \quad (2.12)$$

2.3.2 Finite Difference Scheme

The design of finite difference methods in meteorological models, such as HIRLAM, follows well established principles and are covered in detail in [2] and [3].

In the application of finite difference methods the model equations are averaged over horizontal and vertical scales which are larger than the grid length to be used. The

effect of averaging is to make the solution smooth over the grid scale. Since the solution is smooth the derivatives in the equations can be approximated by finite differences. Therefore, the grid point values represent cell averages and the finite difference equations are resolved by integrating them over the cell.

The variables such as temperature, wind components and humidity are vertically staggered over 'full' levels while geopotential and vertical velocities are staggered at 'half' levels. By implementing the horizontal differencing scheme on an Arakawa C-grid [1] we can ensure that the conservation properties hold.

It is convenient to separate the vertical and horizontal discretisation into computations for pressure surfaces, hybrid surfaces and sigma surfaces. Any surface can be defined by the general expression:

$$P_{k+\frac{1}{2}} = A_{k+\frac{1}{2}} \eta + B_{k+\frac{1}{2}} \eta P_s(x, y); \quad k=0, \dots, KLEV \quad (2.13)$$

where

$A_{k+\frac{1}{2}}$ = constant defining a vertical coordinate

$B_{k+\frac{1}{2}}$ = constant defining a vertical coordinate

$KLEV$ = number of vertical levels in the model

The conservation properties of the horizontal scheme used with the Arakawa C-Grid are as follows:

$$U_k = \Delta A_k u_k \quad \text{for } k \leq K_1$$

$$U_k = \overline{\Delta p_k^x} u_k \quad \text{for } K_1 < k \leq K_2$$

$$U_k = \Delta B_k \overline{p_s^x} u_k \quad \text{for } k > K_2$$

$$V_k = \Delta A_k v_k \quad \text{for } k \leq K_1$$

$$V_k = \overline{\Delta p_k^y} v_k \quad \text{for } K_1 < k \leq K_2$$

$$V_k = \Delta B_k \overline{p_s^y} v_k \quad \text{for } k > K_2$$

The surface pressure tendency Equation(2.10) can be computed as follows:

$$\frac{\partial p_s}{\partial t} = - \sum_{j=1}^{klev} \frac{1}{ah_x h_y} (\delta_x(h_y U_j) + \delta_y(h_x V_j)) \quad (2.14)$$

Equation(2.14) states that the rate of increase of the surface pressure at a given point is equal to the mass convergence in a unit cross section column above that point.

We can now find the other tendencies, layer by layer, by integrating the hydrostatic and continuity equations vertically. The hydrostatic equation i.e. Equation(2.7) gives:

$$\phi_{k+\frac{1}{2}} - \phi_{k-\frac{1}{2}} = - R_d (T_v)_k \Delta \ln p_k \quad (2.15)$$

In order to obtain the geopotential values at full levels the computation is divided into two parts at each layer ,k, thus:

$$\phi_k = \phi_{k+\frac{1}{2}} + \alpha_k R_d (T_v)_k$$

and

$$\phi_{k-\frac{1}{2}} = \phi_k + \beta_k R_d (T_v)_k$$

where $\alpha_k = 1 - \frac{p_{k-\frac{1}{2}}}{\Delta p_k} \Delta \ln p_k$; $k=2, \dots, KLEV$

$$\alpha_1 = \ln 2$$

$$\beta_k = \Delta \ln p_k - \alpha_k$$
 ; $k=2, \dots, KLEV$

$$\phi_{KLEV+\frac{1}{2}} = \phi_s$$

We can express the equations for continuity and vertical advection in finite difference form using similar methods to those shown. Figures 2.1 and 2.2 show how the model variables are arranged on the horizontal and vertical grid structures.

2.3.3 Arrangement of Variables on the Grid

It is standard practice to divide the atmosphere into layers, and to treat the Earth's surface and the 'top' of the atmosphere as boundary layers. The horizontal velocity components are placed at mid-points of layers and the vertical velocity is placed on the layer boundaries.

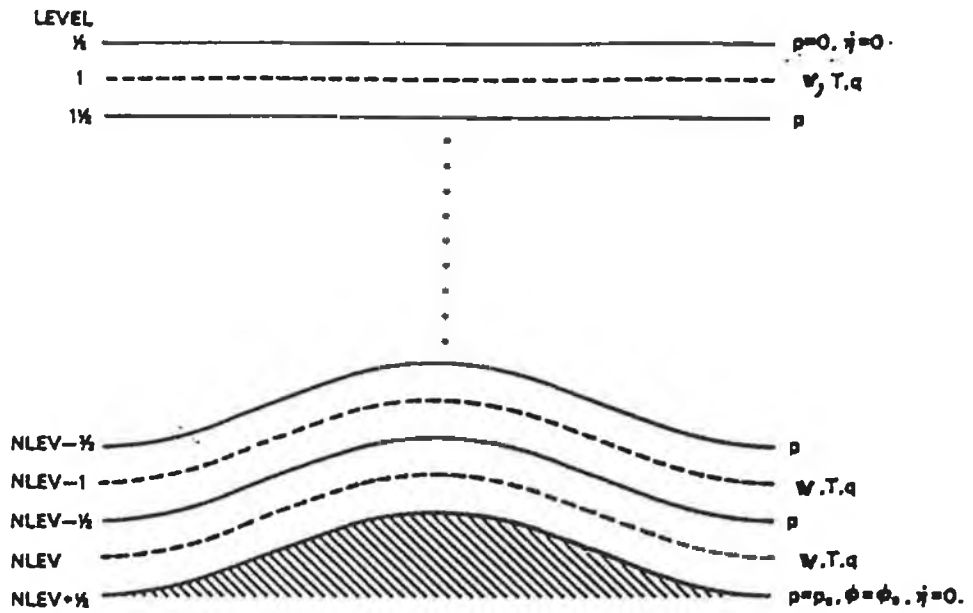


Figure 2.1 Vertical Structure of the HIRLAM Model

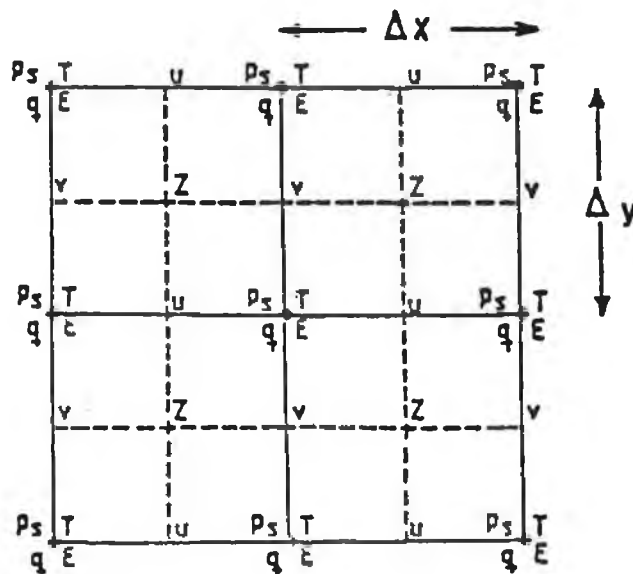


Figure 2.2 Horizontal Structure of the HIRLAM Model showing arrangement of variables on the Lat-Long Grid, where broken lines join mid-points and solid lines join full-points. [1].

To solve the equations for vertical and horizontal velocity the geopotential, ϕ , is computed at layer mid-points and this in turn requires the vertical integration of Equation (2.7).

There are two methods for computing ϕ . The Arakawa C-grid method[30] stipulates that if the potential temperature, T , is held at layer mid-points then the lower boundary condition $\phi = \phi_g$ is used at the earth's surface and Equation(2.7) is integrated over complete layers.

In the second method, the Charney Grid Method[33], the potential temperature is held at layer boundaries. After the first integration from the surface of the earth to the mid-point of the first layer, the integration proceeds directly from the mid-point of one layer to the next. The former method is used in the HIRLAM model.

2.4 Time Integration Scheme

The HIRLAM model uses a three time level Semi-Implicit Time Stepping Scheme[34]. These are accomplished by :-

- (1) Computing explicit new values at the time $t + \Delta t$ using a Leapfrog Method[1] for the Dynamic Terms and a Forward scheme for the remaining terms.
- (2) Adding in the Implicit Correction Terms to the explicitly computed values at the new time step. The new time step values are relaxed towards the prescribed boundary values.
- (3) These values at time $t + \Delta t$ are time-filtered, using the Asselin time filter scheme[1],[32].

In all semi-implicit and implicit solution methods a partially elliptic system in three-dimensions, (3D), must be solved during each time step. The semi-implicit formulation is based on linearisation around a constant basic temperature (300 degrees Kelvin) and a constant basic surface pressure (800hPa). The model also provides for integration to be performed using an explicit time scheme. The semi-implicit formulation takes the form of a 3D Helmholtz equation[1]:

$$\Delta_{tt}d - (\Delta t)^2 G \nabla^2 (\Delta_{tt}d) = \Delta_{tt}\hat{d} \quad (2.16)$$

where

$$\Delta_{tt}\hat{d} = \nabla \cdot (\vec{v}^{n+1} + \vec{v}^{n-1} - 2\vec{v}^n)$$

$$d = \nabla \cdot \vec{v}_k$$

$$G = \text{Combination of Vertical Discretisation}$$

Constraints

The solution of the Helmholtz equation (2.16) is computed separately for each of the KLEV vertical layers. These equations are then solved by use of Fourier sine-transform in the east-west direction and by Gaussian elimination in the north-south direction[1].

2.4.1 Length of Time Step

The model primitive equations are fundamentally hyperbolic in nature. They represent certain waves which must be resolved. Therefore, to capture their true nature, the time step should not exceed half the period of these waves. Otherwise, the results would produce different waves

depicting a longer period. Therefore, there is a meteorological upper limit to the length of time step which can be used. This limit is thought to be in the region of 3 hours.

2.4.2 Multi-Grid Solution Method

In the Multi-Grid Solution Method, an elliptic equation is solved on a sequence of grids, in a "defect correction" fashion. Recall that conventional point-wise iterative methods can quickly smooth out high-frequency errors, but they converge slowly when the errors have low-frequency components.

The idea in the multi-grid approach is that these errors should be removed before applying any iteration. These low-frequency errors could be removed on a coarser grid because in such cases the high-frequency errors cannot be represented at all [27].

The idea is to alternate the coarse grid correction procedure and the local smoothing iteration on a finer grid. In this way a very fast solver is achieved. This solver is applied recursively over a logarithmically growing number of grids. These are traversed back and forth serially until a solution is found. The seriality in traversing between levels is dictated by stability considerations.

Explicit methods are examples of solution methods that allow parallel processing. They do not require the gathering and/or dissemination of any global information. Therefore,

they can take advantage of a grid-like parallel architecture and can utilise any number of processors.

Semi-implicit schemes are different in that the parallel complexity of the elliptic solvers grows with accuracy.

2.4.3 Size of Integration Area

The size of the integration area must be chosen astutely so that the various integration routines can function properly. The nature of a limited area model, such as HIRLAM, is that forecast accuracy decreases with increased forecast length. Therefore, it is more suitable for short term 24 hour forecasts. Ideally, these forecasts should be run several times a day as new synoptic data becomes available.

The size of the integration area selected during this study was 38 longitude grid points and 34 latitude grid points. These grid points were placed at intervals of 1.5° along the earth's surface in each direction. The resultant coarse grid was centered around Ireland. A fine grid representation would require grid point resolution of 0.5° with a longitudinal value of 110 and latitudinal value of 100. The fine grid resolution would require computational resources far in excess of those available for this study.

2.5 Horizontal Diffusion Scheme

There are two optional Horizontal Diffusion schemes included in the HIRLAM model. The user can select one or other scheme at run time.

For a prognostic variable, F , the non-linear second order scheme is expressed as:

$$\frac{\partial F}{\partial t} = +K | \nabla^2 F | \nabla^2 F$$

where K = constant dependent on the grid resolution.

This scheme has a height dependent diffusion coefficient for specific humidity, with increased diffusion around the jet stream level[35].

The linear fourth order scheme is expressed as:

$$\frac{\partial F}{\partial t} = - K \nabla^4 F$$

This scheme includes a correction term for temperature to prevent unrealistic heating over mountain tops[16]. It is more sensitive to the development of changing weather patterns and it reduces unwanted short scale noise in the model[16].

2.6 Boundary Relaxation Scheme

The horizontal boundaries of the integration area are the specified latitudes and longitudes in the rotated coordinate system. A simplified relaxation technique proposed

by Kallberg and Gibson[37] is used. The technique relaxes the dependent variables, at time $(t+\Delta t)$, towards externally specified time dependent values. The relaxation is performed within a narrow boundary zone as shown in Figure 2.3.

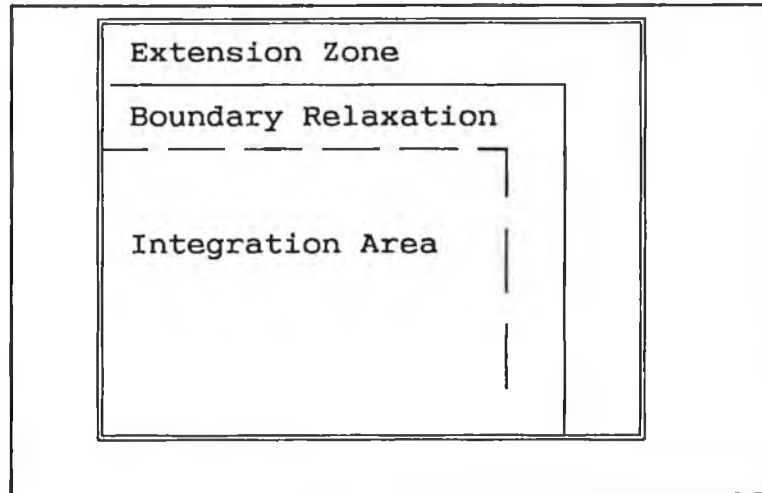


Figure 2.3 The inner integration area, boundary relaxation zone and extension zone as used by the HIRLAM Model.

The interior variables X_i are adjusted towards prescribed boundary values X_b after the semi-implicit corrections are performed. The technique uses the relationship:

$$X^{n+1} = (1-\alpha_b) X_i^{n+1} + \alpha_b X_b^{n+1} \quad (2.17)$$

where

X_b is linearly interpolated in time between boundary data sets in 6 hourly intervals

n = width of the boundary relaxation zone

$$\alpha_b = 1 - \tanh\left(\frac{2}{n-4} j\right)$$

j = number of grid points away from boundary point

Within the boundary relaxation zone the model forecast is mixed with the boundary field using a spatially varying weighting factor[16].

The quality of the boundary data used is very important for the production of a good forecast. The best source of boundary data has been found to be from the long range weather forecasts produced daily at ECMWF. The ECMWF forecasts are produced using a coarse mesh global model (more coarse than the resolution of limited area models), once each day. The HIRLAM model must be run on an integration area sufficiently large to avoid the resultant forecast being contaminated by errors associated with boundary zones. The ECMWF forecasts for the period 25th-26th August 1986 were used as boundary data for this case study.

2.7 Physical Parameterisation

The physical parameterisation schemes employed form a self contained portion of the overall HIRLAM model. These schemes involve the modelling of convection, radiation, condensation and vertical and horizontal diffusion. Simultaneous solution of all these equations keeps account of water, whether in ice, rain or vapour state, predicts atmospheric temperature distribution and continuous changing flow patterns. Such physical processes can occur on smaller scales and are incorporated by researchers as sub-models.

The actual mathematical formulations for the physical parameterisation scheme are described in [1] and only a verbal description of each will be given here.

2.7.1 Radiation

The parameterisation scheme for radiation distinguishes between direct and diffuse solar radiation. Scattering and absorption affects the transmission of shortwave radiation through the atmosphere. These atmospheric properties are expressed in terms of humidity and density. The longwave radiation is defined in terms of clear air cooling on temperature curvature. The contribution of cloud cover to the rate of heating/cooling and the net longwave radiation at the ground is defined by simple emissivity methods[4]. The emissivity of clouds depends on cloud type and depth which are defined in terms of the vertical hybrid coordinate.

2.7.2 Convection

The convection is applied using a Kuo-type scheme[44]. It stipulates that layers of the atmosphere involved in the convection are defined, so that an air parcel, which is lifted from one layer to the next, is positively buoyant and becomes saturated[4]. The upper limit of the convective layers is marked by the transition from positive to negative buoyancy.

Also, the vertically integrated moisture must be positive. The scheme permits more than one set of convective layers. The relative humidity in the convective layers determines the amount of moisture convergence remaining in the atmosphere. The remaining moisture is released as precipitation.

2.7.3 Stratiform Condensation

The condensation scheme stipulates that the specific humidity of a grid cell should not exceed a saturation value. The value is chosen so that it is less than the true saturation specific humidity[4]. A first order adjustment technique determines the amount of condensed moisture required to restore saturation. This amount of moisture is released as precipitation. The temperature is affected by the corresponding heat release.

2.7.4 Vertical Diffusion

Horizontal wind components, dry static energy and specific humidity are all affected by the vertical diffusion scheme. Surface fluxes affect the prognostic values in the lowest model layer and are determined by means of a drag coefficient formulation[4]. These drag coefficients are functions of static stability and of a roughness length.

A fraction of land and a fraction of sea ice is defined for each grid cell and from these a value for roughness length is determined. For each gridpoint the surface fluxes are computed separately over land plus sea ice and over open sea, including lakes.

Above the lowest model layer the computation of fluxes is based on a mixing length formulation. This formulation uses exchange coefficients which depend on static stability. A modified Richardson number[45] is defined to represent the effect of shallow convection[4].

2.8 Numerical Formulation of Semi-Lagrangian Methods

Semi-Lagrangian Methods work well for short term numerical models of the atmosphere such as HIRLAM. But, current formulations are not energy conserving and as such are not considered suitable for climate modelling.

The Semi-Lagrangian approach originated from the Lagrangian form of the evolution equation for a fluid variable[18]. The rate of change of a fluid variable, ψ , can be expressed as:

$$\frac{d\psi}{dt} = R$$

This has a solution of the form:

$$\psi(x_1, t) = \psi(x_0, t_0) + \int_T R dt$$

where (x_1, t) and (x_0, t_0) are two points which connect the trajectory, T , of a 'parcel' of fluid and R represents all the associated sources and forces.

If we assume that (x_1, t) represents a set of points on a regular grid, then we can compute, backwards-in-time, the trajectories which arrive at point (x_1, t) . We can find their departure points (x_0, t_0) . The departure points may not necessarily coincide with the exact points on the grid, and therefore, an interpolation technique must be employed to determine $\psi(x_0, t_0)$.

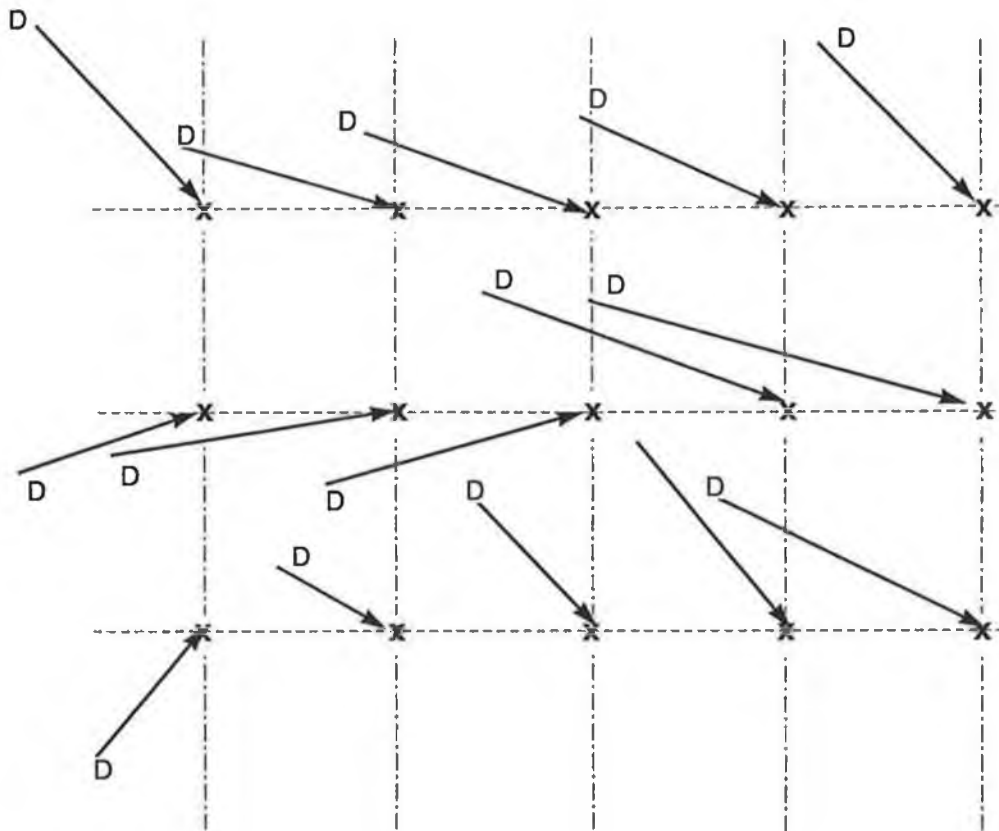


Figure 2.3 The Semi-Lagrangian Interpolation Scheme: Showing the Arrival Points, X, at exact grid points and their corresponding Departure Points, D, which are placed irregularly in space.

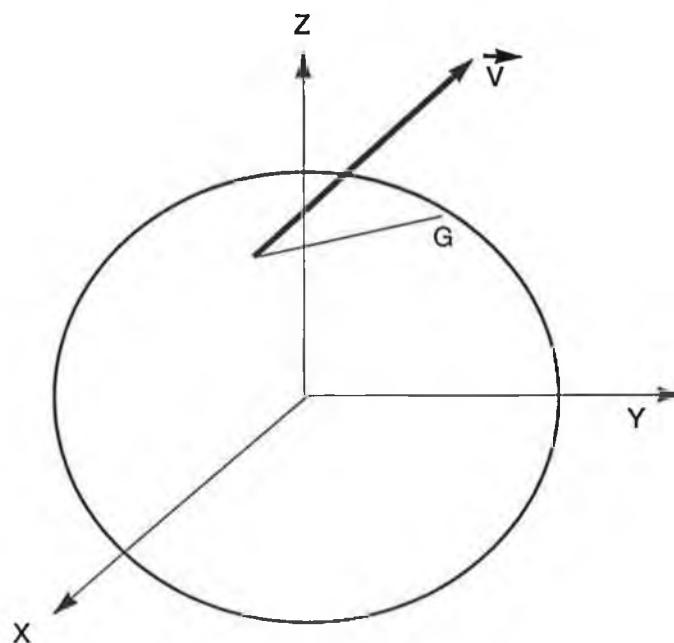


Figure 2.4 The Semi-Lagrangian Interpolation Scheme: Showing the computation of a trajectory on a sphere, representing the Earth. An auxiliary cartesian coordinate system, which is centered at the 'centre' of the Earth, is used .

Semi-Lagrangian formulations have been devised to exploit various interpolation techniques and various temporal discretisations. The formulations allow us to select contours whose elements coincide with the trajectories of the parcels.

In his paper to the Seminar on Numerical Methods in Atmospheric Models (Sept. 1991), Aidan McDonald[5] contrasts the Eulerian Method with the Semi-Lagrangian Method. He points out that the numerical integration of a linear advection equation such as:

$$\frac{\partial \psi(x, t)}{\partial t} + c \frac{\partial \psi(x, t)}{\partial x} = 0 \quad (2.18)$$

using an Eulerian scheme is restricted to the stability condition:

$$c \frac{\Delta t}{\Delta x} \leq 1$$

where Δt and Δx are the increments into which time and space, respectively, are divided for the purpose of integration, and where c is the constant advecting velocity.

He then applies a Semi-Lagrangian solution method and shows that stability is guaranteed for any c , provided the gridpoints used in the interpolation surround the departure point. He goes on to apply the Semi-Lagrangian technique to the barotropic vorticity equation given as:

$$\left[\frac{\partial}{\partial t} + v(r, t) \cdot \nabla \right] \eta(r, t) = 0 \quad (2.19)$$

where v = advecting velocity;

$$\eta = \xi + f$$

$$\xi = (k \cdot \nabla) v = \text{Vorticity}$$

$$f = \text{Coriolis parameter, Equation (2.2)}$$

$$d = 0 = \text{Divergence}$$

If we assume the divergence to be zero we can define the velocity in terms of the stream function, ψ , by:

$$v = k \cdot \nabla \psi \quad (2.20)$$

which gives the Poisson Equation:

$$\eta = \nabla^2 \psi + f \quad (2.21)$$

The solution of Equation(2.19) can be generalised as:

$$\eta [r(t+\Delta t), (t+\Delta t)] - \eta [r(t), t] = 0 \quad (2.22)$$

where $r(t)$ is solved by:

$$r(t+\Delta t) - r(t) = \int_t^{t+\Delta t} v[r(\tau), \tau] d\tau \quad (2.23)$$

In general the integration of Equation(2.19) using the Semi-Lagrangian method proceeds as follows :

Step 1 Construct y at time $t = 0$

Step 2 Integrate Equation(2.23) using a good approximation method and dividing Δt into M increments of $\Delta t / M$ each.

Step 3 Use Equation(2.22) to compute forecast value of $\eta[r(t+\Delta t), t+\Delta t]$

Step 4 Compute $v[r(t+\Delta t), t+\Delta t]$ for the next step.

This is done by solving the Poisson Equation (2.21) for $\psi[r(t+\Delta t), t+\Delta t]$ and then compute $\eta[r(t+\Delta t), t+\Delta t]$ from Equation(2.20)

Step 5 Repeat Steps 2-4 N times to get forecast at $N\Delta t$

Figures 2.4 and 2.5 illustrate the Semi-Lagrangian Scheme.

The detailed procedure for Steps 2 and 3 are as follows:

Let $t = n\Delta t$

Choose $r(t+\Delta t)$ to be of grid points $r_{i,j}$ then

Step 2 is:

Taking $M = 1$, solve Equ.(2.23) iteratively at each point

using $\hat{\varphi}^{(0)} = v_{i,j}^n$

so that

$$\hat{\varphi}^{(k+1)} = v[r_{i,j} - \hat{\varphi}^{(k)} \frac{\Delta t}{2}, (n + \frac{1}{2}) \Delta t] \quad (2.24)$$

The departure point position for the arrival point (i, j) is given by:

$$r(t) = r_{i,j} - \hat{\varphi}^{(k+1)} \Delta t$$

Step 3 is:

Since $r(t)$ is now known $\eta [r(t), t]$ can be computed using suitable interpolation of the known $\eta^n_{i,j}$ using, for example, a Lagrange Biquadratic interpolation[5],[17] and this in turn allows us to proceed to Steps 4 and 5.

The Notation used above, such as η^n_j , is defined as:

$$\eta (j\Delta x, n\Delta t)$$

2.9 Computational Time Step Algorithm

During each time step a sequence of computations is carried out by the model over the integration area. At time step, t , the field variables, horizontal and vertical velocity, temperature, specific humidity and pressure are evaluated at each grid point in the integration area. The algorithm to perform these computations proceeds as follows:

Do for each time step:

Step 1:

Perform Fourier transforms to calculate the explicit dynamic tendencies of the field variables at time t .

Step 2:

Calculate the rate of change associated with each of the field variables at time t .

Step 3:

Perform Fourier transforms to calculate the grid point field variables at time $t-\Delta t$

Step 4:

Calculate the rate of change associated with the physical parameters at $t-\Delta t$

Step 5:

Evaluate non-linear and linear terms along trajectories by interpolating between grid points using the Semi-Lagrangian scheme.

Step 6:

Calculate the diffusion of momentum, water vapour and temperature $t-\Delta t$

Step 7:

Unpack relevant boundary data at predetermined time steps.

Step 8:

Perform explicit adjustments in preparation for application of Helmholtz solver.

Step 9:

Perform boundary relaxation computations from the inner integration area to the extended fields of the lateral boundaries at time t and $t+\Delta t$.

Step 10:

Apply Helmholtz Solver.

Step 11:

Perform implicit adjustment on solution of the Helmholtz equation at time t and $t+\Delta t$.

Step 12:

Apply Asselin time filter[32] to obtain updated values at time t and $t+\Delta t$.

End Do.

2.10 Summary

In the HIRLAM Model implemented on the transputer network, the vertical discretisation of the Eulerian Hybrid-coordinate model was retained as far as possible. The Semi-Lagrangian approach used three time levels and an interpolating treatment of the 3-D advection. It was converted to tangential cartesian coordinates for the updating of the wind fields. The computation of the vertical velocities was similar to the conventional Eulerian Scheme. The continuity Equation(2.8) was computed using a Semi-Lagrangian Scheme which was formulated so as to allow $\ln(p_s)$ to be retained as a forecast variable.

The Semi-Lagrangian technique allowed the use of longer time steps but these had to be chosen so that it did not increase the damping effect in the meteorological modes. It also introduced additional smoothing into the model, associated with the interpolation of fields to departure points. In practice this was small and could be ignored for short-term forecasts.

The cost of the Semi-Lagrangian scheme was principally due to the interpolation stage. Therefore, this aspect of the model code had to be efficiently coded to retain accuracy at minimum cost in computation time. Linear interpolation was used in the computation of trajectories and in the evaluation of their mid-point terms. A mixed cubic/linear interpolation

method was used for the terms evaluated at the departure points. The associated interpolation routines were by far the most time-consuming routines in the entire model.

Experiments carried out at ECMWF on the efficiency of Eulerian versus Semi-Lagrangian Techniques indicate that the Semi-Lagrangian time step is about 20% more expensive than the original Eulerian Code[6]. This deficit can be virtually recovered if longer time steps are used with the Semi-Lagrangian technique.

Chapter 3

HARDWARE DESCRIPTION

3.1 The Transputer

The first transputer was launched by INMOS in 1985 and was considered a most significant event in concurrent computing[19]. A transputer is a special chip designed to support concurrency in operation. It was made possible by the major advances in CMOS(1.5 micron) integrated circuit technology during the 1980's. At present there are three different types of transputers namely the T212, T414 and T800. According to INMOS, the transputer is similar in principle to a transistor, in the sense that both are elementary building blocks for complex systems.

3.1.1 Transputer Architecture

It is not necessary to know much about the transputer hardware when writing programs in any of the languages mentioned throughout this report. However, a short description is included here for background information.

Transputers are delivered on a standard IBM PC-XT compatible expansion board known as a B008 which can carry up to ten Transputer Modules (TRAMS). These TRAMS are board level transputers which integrate processor, memory and peripheral functions[23].

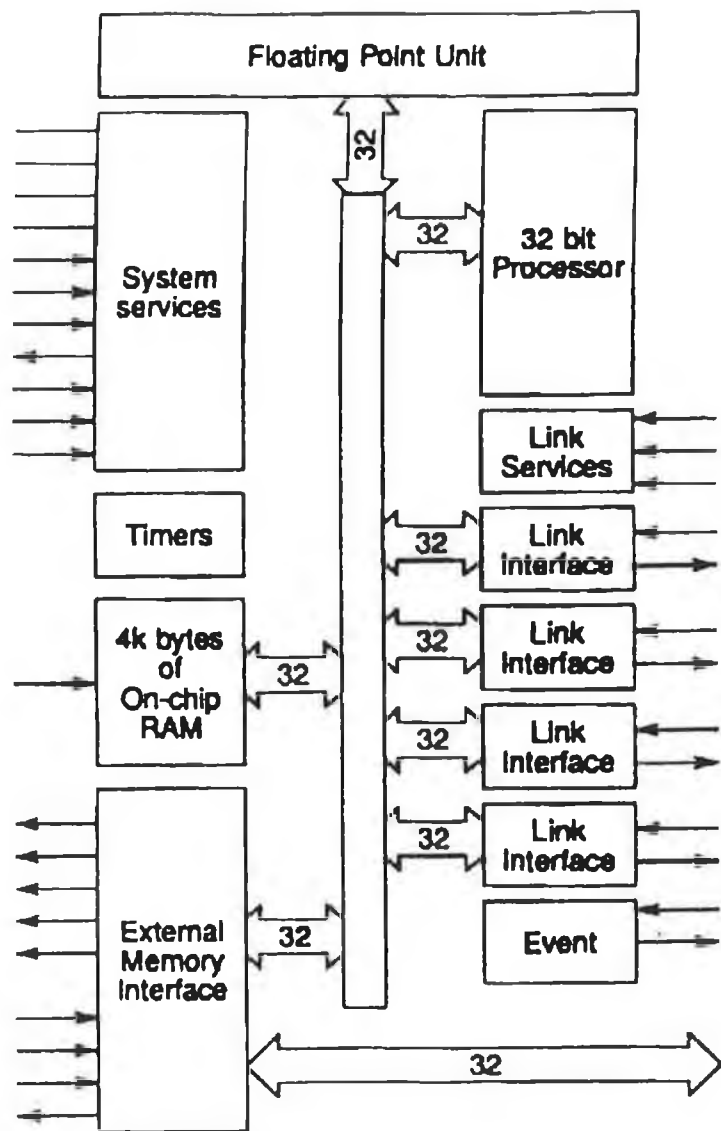


Figure 3.1: Block Diagram of the INMOS T800 Transputer

The Transputer has a RISC architecture with a performance capacity of 5 to 10 Mips with low power dissipation of less than 1 watt. The T800 is a complete computer on one chip. It is a 32-Bit processor roughly equivalent to the Intel 80386 processor. It has an IEEE floating Point Unit(FPU), (1.5 MFLOP for the 20 Mhz device), which is about twice as fast as an Intel 80387 Maths co-processor chip.

3.1.2 Transputer Communication Links

Each Transputer type has 4 communications links which are full duplex serial I/O links. They are driven by a dedicated eight channel direct-memory-access(DMA) engine. This allows high speed serial links to other transputer units.

The Transputer-to-transputer links provide a combined data communications capacity of 5 MBytes per second and can operate concurrently with internal processes. This is where the transputer departs from the shared Bus concept employed in other multi-processor architectures.

It allows the parallel connection without the overhead of having to provide complex communication control. The advantages over multi-processor buses are that, there is no contention for communication, there is no load capacity penalty as more transputers are added, and bandwidth does not become saturated as the system grows in size.

Configuration of T800 Transputers Used

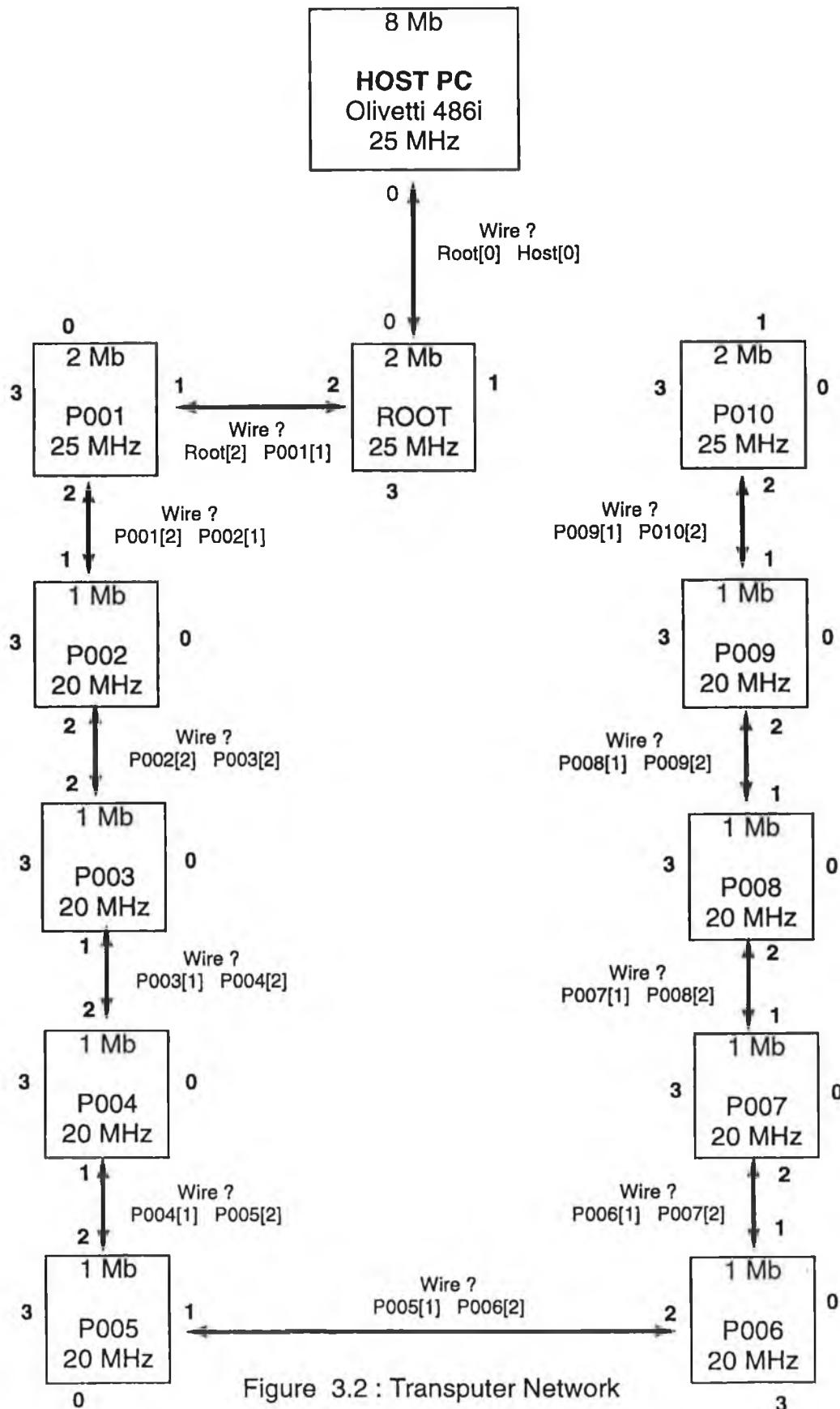


Figure 3.2 : Transputer Network

3.1.3 Transputer Memory

The T800 has 4 Kbytes of very fast (50 micro-second) static RAM, 2 timers and each channel can operate at up to 20MBit/sec. It has a memory interface for controlling up to 4 GBytes of external memory. The on-chip static RAM assists in the elimination of the processor to memory bottlenecks. The efficient use of each processor's time slices is controlled by a microcoded scheduler. A schematic view of a T800 Transputer is shown in Figure 3.1.

3.1.4 Transputer Networks

A system may comprise of one transputer or a network of transputers. These may function as a concurrent system according to a topological arrangement which can be software configured (provided the complete set of physical strapping is already in place). The transputer network used for this case study is shown in Figure 3.2.

Each Transputer in a transputer network can run a different task simultaneously. For instance, it can be used to perform different computations at grid boundaries. On vector machines, such as the **CRAY Series**, such exceptions are difficult to program and inefficient to implement. Furthermore, computations involving embedded conditions cause problems on vector machines but not on transputer networks.

For example, conditional expressions at different gridpoints possibly result in different truth evaluations. A vector computer has to perform both branches of the conditional statement for each gridpoint and only select the

desired one afterwards. A distributed system of transputers avoids this inefficiency.

3.2 Control of Transputer Communication

Communication via four integrated serial links is completely controlled by hardware. After a process has set up a link communication, that communicating process is suspended until the communication is complete. The Hardware takes care of reading/writing data from/to memory via the Direct Memory Access (DMA). The DMA has virtually no effect (<1%) on overall processing speed, even when all four links are operated simultaneously.

3.3 Maximising the benefits of on-chip RAM

The on-chip RAM is a valuable feature. To take full advantage of it we can use an options file, at link time, to specify which routines we wish to have placed in RAM at run time. This RAM is fast enough to keep up with the Processing Unit (PU). On the other hand for programs which rely on external memory references, the processing time is longer because the PU is idle for longer periods.

3.4 Booting a Transputer Network

A Transputer can boot, or start, either from an internal ROM chip or by another processor via a physical link. A typical boot instruction is implicit in the MS-DOS compatible command:

```
C:> AFSERVER -:b MODEL.B4
```

This boot method ensures that members of a Transputer network do not require a boot ROM. The desired boot method is decided at TRAM installation into the Host PC and is achieved by wiring a pin to logic "0" or "1".

When a transputer network is booted from a link, the ROOT transputer is reset. The first block of data sent to the ROOT, via the "afserver" program, is loaded into the internal RAM and executed. To start up an entire network, the ROOT Transputer is booted up via the instruction flag "-:b" sent from the host. After that the ROOT is instructed to send appropriate boot information to neighbouring nodes.

3.5 Host PC Configuration

As we have seen, inter-processor communication between transputers is straight forward, but the ROOT transputer must be able to communicate with the HOST PC. An OLIVETTI 486PC with an INTEL 80486 processor was used as the Host to the transputer network. The Host PC used MS-DOS 5.0 and had 8 MBytes of extended memory and a 300 MByte Hard Disk.

Each transputer mother board, TRAM, is shipped with a device called a LINK ADAPTER. On one side it physically resembles a normal parallel I/O port and on the other side it supports the INMOS link interface. Additional hardware is used to make the BYTE_WIDE side of the link adaptor look to the Host PC as a DMA accessible I/O port on the PC's Bus.

Finally, the link side connects to any of the transputer links. With such a hardware configuration, a simple protocol can be used to achieve communication with a Server Program,

such as "afserver", running on the Host PC. This is how the Transputer environment can use the Host's screen, keyboard and disk file resources [13].

3.6 Graphics Subsystem

A general purpose graphics processor has been developed for the transputer environment. It consists of a modified IMS T800 transputer and it provides up to 8 Mbytes of Direct RAM (DRAM), for program development and data storage. A full description of this device is given in the Device User Manual[22].

3.7 Process Scheduling on the Transputer

Each transputer has a built-in capacity to manipulate processes. It has access to instructions to start and stop processes. These instructions use a linked list which contain all the processes which are ready to execute. It uses a simple time-sharing algorithm for process management. This specifies that processes, which are running longer than some fixed time interval, are moved to the end of the list and the next process is picked from the beginning of the list, for execution.

If a process has to wait for a communication or for a timer, it is temporarily removed from the list of processes and the next one is started up. It takes less than 1 micro-second for a context-switch between two processes to take place. Therefore, the use of multiple processes on one processor does not lead to much of an overhead.

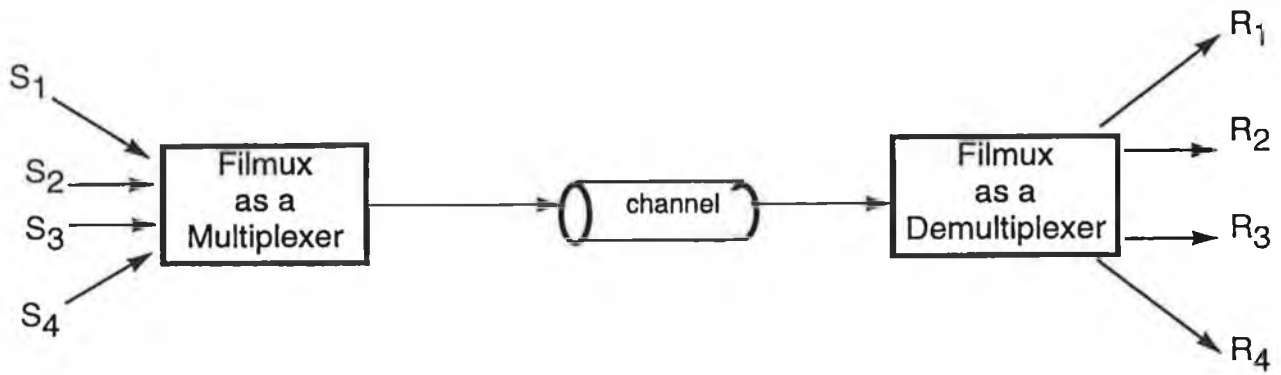


Figure 3.3 Use of Multiplexer / Demultiplexer technique to accomodate several virtual channels over one physical channel

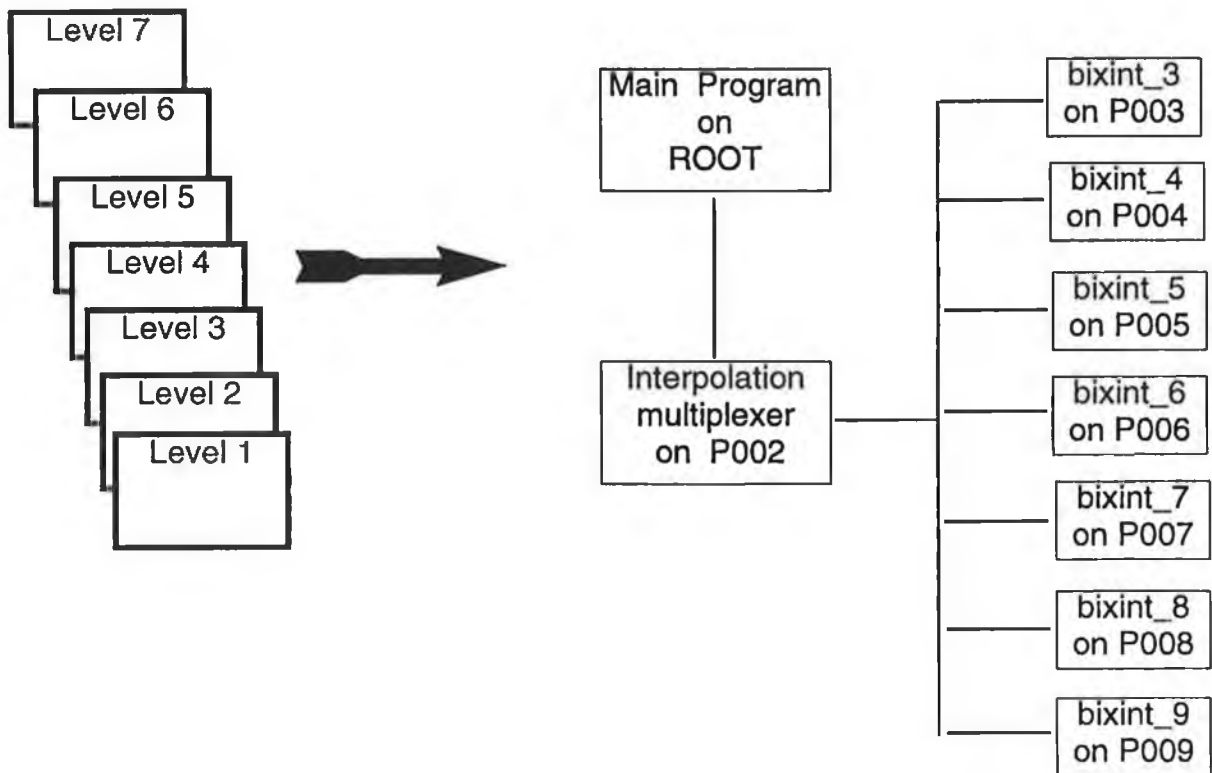


Figure 3.4 Using the Software Multiplexer technique to project the vertical interpolation onto 7 transputers.

3.8 Channel Communication

If parallel processes must communicate, they must do so through CHANNELS and not through shared variables or common blocks. Channel communication is quite simple and one can send or receive a message with a single instruction.

SENDER -----> CHANNEL ----->RECEIVER

Communication over a channel takes place in one direction only and is achieved according to a so called rendezvous principal which stipulates that the sender and receiver should simultaneously engage in the communication.

If one of them is not ready, then the other must wait, and a waiting process will be removed automatically from the list of ready processes. Communication is the only way to synchronise processes and if two processes fail to communicate correctly then the entire system will become DEADLOCKED. It is not possible for more than one process at either end to use a channel at the same time. [However by using the "ALT" functions and the 'fileMUX' program, from 3L, processes can alternate between channel I/O.]

The use of serial communication principles allows us to use synchronous communication which is data-driven. This is a very convenient programming method which allows the programmer to abstract from instruction timing. Therefore, although 3L-Fortran[14] provides a timing library it was not necessary to use it during this research project.

3.8.1 Software Channel Types

There are two types of software channels.

(1) Internal Channels for communication between processes on the same transputer; or in the case of the 3L-Fortran, for communication between threads of the same task;

(2) External Channels for communication between processes on different transputers.

3.8.2 Channel Mapping

External Channels are mapped onto a physical link while Internal Channels are mapped onto an arbitrary memory word. The same communication instructions can be used for both types of channel. Because of this it is transparent to the program or indeed the programmer. It is only at the configuration stage where decisions are made about the placement of processes or tasks and the designation of their channel parts.

As mentioned already, there is a Hardware restriction whereby, at most, one external channel in each direction can be mapped onto a physical link. To allow more "external" channels to share the same physical link it is necessary to implement a number of "virtual" channels to use that link. Virtual channels can be implemented by using "multiplexer" and "demultiplexer" processes on each side of the link as shown in Figure 3.3.

A multiplexer process communicates at one end with a number of processes via internal channels and on the other side with a demultiplexer on another transputer via one

physical link.

The multiplexer/demultiplexer communications can be done in two ways. One way is to send an identification of the virtual channel with each communication. The other way is much more rigorous whereby messages are always sent in a predetermined fixed order. This latter method proved much less useful for our application and was not used, while the former was chosen to handle the communication with the interpolation tasks on the a Sub-network of transputers.

3.9 Grid-Based Communication on a Transputer Network

The design of a network for parallel computation has two important objectives. One is to divide the computation in such a way that each processor has approximately the same amount of work to do. This is called load-balancing. The other is to control the interprocessor communication in such a way that wait times are minimised.

If we have a truly Grid-Based computation the first objective is easily achieved. The computation for each grid point is approximately the same and so the work can be evenly distributed over all of the processors available.

In the case of the second objective there are problems associated with algorithm modification in order to reduce communication overhead. It is likely that the communication will be replaced by extra local computations which of themselves could be at least as expensive as the original communication overhead.

So, to achieve maximum performance we must know what our communication costs are, what will be the cost of the extra computations and how they relate. It must be borne in mind that communication between processors, not directly connected, also affects the processors in between, which are involved in the message passing.

The choice of which dimensions to project onto a 2D grid depends on a number of program properties and on the number of processors available. We had 11 transputers, but, due to the size of the data areas required by the program, a model with 7 vertical levels was implemented. This enabled us to project the vertical interpolation onto 7 processors as depicted in Figure 3.4.

Unfortunately, we were unable to have the physical strappings put in place to enable a better software configuration so we opted for a multiplexer/demultiplexer configuration.

3.10 Summary

This chapter has given the reader background information on the architecture of the transputer and the working environment which it offers. The concept of interprocessor communication and transputer-to-transputer synchronisation techniques were introduced. These involved the principles of software channels and their mapping onto physical channels. The suitability of transputer networks for grid based communications was considered.

Chapter 4

Programming Languages for Transputers

4.1 Introduction

Currently there are several programming languages on the market all designed to apply parallel algorithms within the transputer environment. A language written in conjunction with the development of the transputer itself is OCCAM. Other languages emulate the techniques used in OCCAM. This 'family' of languages can be classified as message-oriented languages.

The basic problem, in any programming environment, is the management of complexity which envelopes the problem to be solved, the algorithms to solve it, the implementation language and the hardware upon which the eventual program will be executed. The programming task is fragmented into sub-disciplines, one for each architectural form. Chandy and Misra[24], attempted to address this difficulty and they showed how programs could be developed systematically for a variety of architectures.

4.2 OCCAM

As already discussed, transputer systems were designed with parallel processing in mind. They are based on the concept of communicating sequential processes (CSPs), as expounded by Hoare in his 1978 paper[12].

OCCAM is a machine level language and was the first language to be used for program development on transputers. It was specifically developed to efficiently exploit the parallel architecture offered by the transputer. In OCCAM communicating sequential processes executing in parallel, exchange data through CHANNELS which are one-way, point to point, pathways in memory. These are known as SOFT CHANNELS. For processes running on different transputers, a PLACE instruction can be issued. It allows the transformation from a SOFT CHANNEL to a HARD CHANNEL and therefore, allows data transfer to take place between the linked processes located on separate transputers.

A simple example illustrating the above method of channel communication in OCCAM[46,47] is:

```
CHAN OF INT Chan1 :
CHAN OF BYTE Chan2 :
PAR
  INT Item :
  SEQ
    Chan1 ? Item
  BYTE Item :
  SEQ
    Item := 250(BYTE)
    Chan2 ! Item
```

This example declares two 'software' channels'. One is an INTeger and the other is a BYTE channel. The PAR instruction stipulates that the program segments starting at SEQ should each be executed in parallel. In the first SEQ segment, 'Item' is read from channel Chan1.

The second SEQ executes two statements sequentially. The first of these statements assigns the ASCII value of 250 to 'Item'. The second statement sends 'Item' out along channel Chan2. Note that the variable 'Item' has a different meaning for each of the SEQ segments. Therefore, both SEQ segments can be executed in parallel because no data or channel dependencies exist.

4.3 PARALLEL "C" and PARALLEL PASCAL

Many software companies, particularly in Britain, are actively engaged in the development of high-level languages for the transputer. Parallel "C" and Parallel Pascal are now well established products from the 3L Company. The features they offer are in general similar to the features described for Parallel Fortran. During early investigations, some of the main HIRLAM program was re-coded using Parallel C. However, this was a confidence building exercise and was abandoned when the Parallel Fortran compiler was delivered.

4.4 PARALLEL FORTRAN

All of the existing languages, including Fortran, concentrate on explicit message passing as the chief operation of concurrency. The bulk of the software development done during this research work was implemented using 3L-Parallel Fortran Version 2.1 [14]. This product seemed to be the most established of the Fortran products on offer and comes from the same software house as Parallel C with which we had some experience.

Parallel Fortran follows closely the concepts used in OCCAM. Any computing system can be considered as a collection of concurrently active sequential processes which only communicate over channels. A channel, as we have already seen, can only carry messages in one direction. If two processes must exchange data, in both directions, then two channels must be used. A process can have any number of input and output channels but these are fixed at compile time. These can be completely described in an accompanying configuration file.

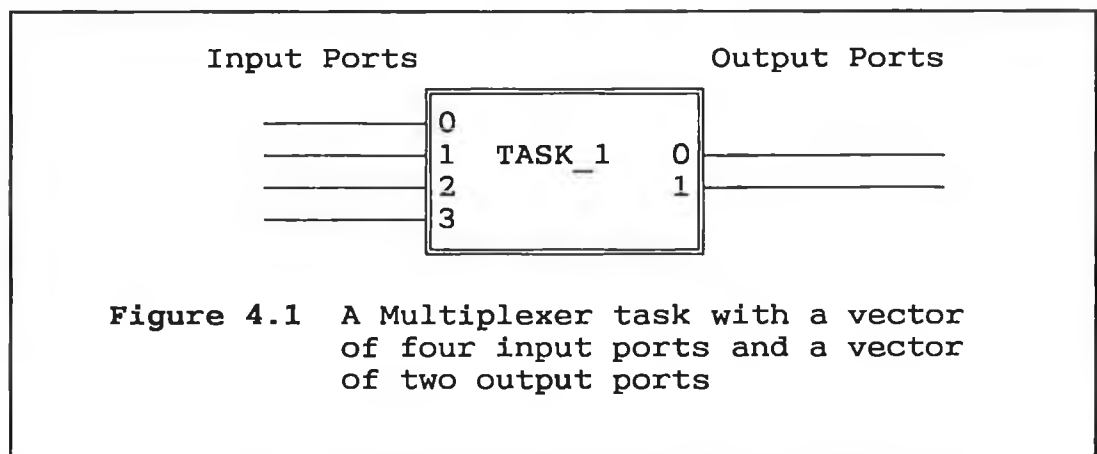
4.5 Channel Communication using PARALLEL FORTRAN

If a process wants to send a message over a channel it is always forced to wait until the receiving process reads the message. The actual implementation of any individual process could be a hardware device or a software procedure. Both can communicate via a shared channel at the speed of the slowest process. In short, we can say that hardware channels behave exactly the same as virtual channels since they provide synchronised unidirectional communication.

Programs written in Parallel Fortran can treat internal software channels and hardware link channels as identical. They can use the same send and receive instructions on both. This makes it possible to develop a parallel program on a single transputer, if space allows, and then later move some of the processes onto other transputers without having to recompile any code.

4.6 Tasks and Threads in Parallel Fortran

A complete application program can be described as a collection of one or more concurrently executing tasks. All tasks have their own memory regions for code and data storage. They also have a vector of INPUT PORTS and a vector of OUTPUT PORTS as depicted in Figure 4.1



The software concept is based on tasks and threads with message-passing over channels as the only means of communication between tasks. Message passing between threads is via internal channels or via shared common blocks. These common blocks should ideally be protected by a Semaphore scheme[53].

Tasks and Threads in Parallel Fortran-77

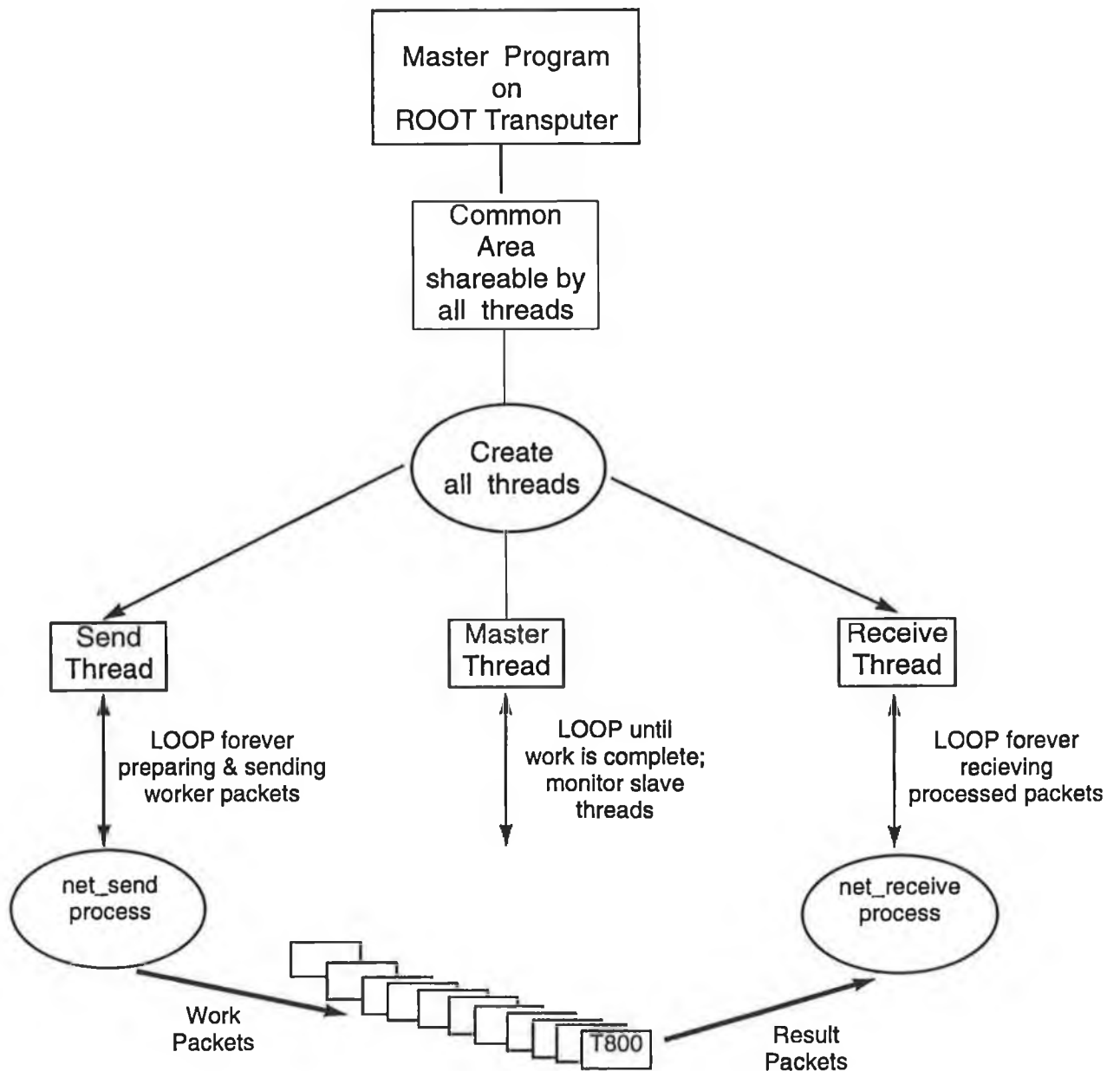


Figure 4.2 Master-Slave configuration with Threads created by the Master Task controlling the message handling with a farm of slave or worker tasks distributed over the Transputer Network. This uses the flood-fill configuration technique.

4.6.1 Tasks

The task concept is one which is characterised as follows:-

- * Tasks are autonomous program units which run in Parallel;
- * Tasks can terminate themselves and can create but cannot terminate other tasks;
- * Tasks communicate only by an exchange of messages over dedicated channels;
- * There is no concept of shared memory;
- * Applications are started by one initial or Master Task.

All user defined task systems should be homogenous and independent from the actual hardware configuration. This means that tasks are allocated to processors via a specific configuration file which can be altered without altering the compiled and linked tasks. The mapping of tasks to nodes is achieved by either the CONFIG or FCONFIG utility programs[14].

A task can be treated as an atomic building block for a parallel system. The Parallel Fortran Library is supplied with several building block tasks such as 'fileMUX'[14]. At the configuration stage a port will be BOUND to the address of a real channel. This binding phase is external to the task itself, which from the point of view of the configuration software can be treated as a "black-box". Thus, the bindings of a task can be changed in a configuration file without the necessity to recompile and link that task.

The port vectors associated with a task, are not normally accessible to the actual Fortran program. We can determine the values of the channel addresses to which they are bound by making calls to the library functions supplied for channel access. However, it is not essential that we should know the actual INTEGER value assigned to a logical channel. Tasks running on the same processor can have any number of interconnecting channels. Tasks on different processors can only be connected if there are physical wires to connect the physical links of the processors concerned.

If a task has to accept input from a large number of input channels, the run-time library provides functions to enable it to poll a group of channels. It continuously checks the group of channels individually to determine which one is ready to transmit messages. This "GUARDED INPUT" facility is equivalent to the ALT statement provided in OCCAM[46],[47].

4.6.2 Threads

Threads are light-weight processes which can be created by a Main task and are similar to "processes" of MODULA-2 and "coRoutines" of some other languages [14]. Threads must have their stack space allocated specifically for them by the TASK which creates them. However, the code space, heap, static and external memory are all shared by the threads of the same task. These threads can also have access to the same COMMON blocks. SEMAPHORE functions are provided to prevent threads which share data from interfering with each other. These functions must be incorporated into the specific threads[14].

The thread library provided with the 3L Parallel C compiler[15] allows the easy use of multiple threads for reading from a number of input channels simultaneously. However, to implement this facility in 3L Parallel Fortran it would have been necessary to introduce re-entrant subprograms to provide a type of recursion. This was not provided in the current version of the Fortran compiler. As a result a subprogram in Parallel Fortran may not be invoked through the thread mechanism more than once at the same time. However, if the problem to be solved can be divided into independent sections then a multi-threaded algorithm[14] may still be used effectively.

Threads may communicate by using channels in the same way as tasks or via shared memory. If we require two operations to be performed in parallel they can be executed by threads under the following conditions :

- * the operations will never need to run on distinct processors;
- * the operations are closely coupled; they share a lot of common code of the parent task;
- * the operations logically operate on shared data structures. In this way there is no need to copy the data for computation back and forth as messages.

One of the difficulties with the thread concept is that the Master Task must explicitly assign sufficient work space for each thread it creates, and therefore, there is a limit to the number of actual threads any task can create.

4.7 Configuration Language for Task Placement

The configuration language, supplied with the programming languages from 3L Ltd.[15], is a meta language. It is a syntactical programming language which must be written by the programmer according to a well defined programming grammar. The configuration program so written is then "compiled" using one of two special utility programs, namely "CONFIG" and "FCONFIG"[14], to produce a configured application.

4.7.1 The Static Configurer

The syntax for the static configurer, "CONFIG"[14], enables the programmer to give an easy description of both the physical processor networks and the user tasks, which have been written to run on them. A basic rule is that only one task can perform normal I/O with the MS-DOS environment of the HOST PC. This should be resident on the ROOT transputer and must be linked with the run-time library.

All of the tasks assigned to other transputers in the network must be linked with the stand-alone library. A typical task configuration file is shown in Figure 4.4. In this example the processors HOST and ROOT are declared and they are connected to each other via their respective port[0]. Then all the tasks are declared, together with their port vectors and data requirements. A 'place' statement specifies the tasks and the processors upon which they are intended to run. Finally, the two way connections between the tasks are specified according to predetermined requirements.

A 'WORM' utility is provided to check the current configuration status of the existing transputer network. The programmer must refer to the 'WORM' output when writing configuration programs.

```
!  
! reader.CFG      configuration file for a SINGLE reader task  
!  
! Hardware  
!  
processor host           ! THE HOST PC  
processor root          ! The transputer in the B004  
wire jumper -          ! connects  
    root[0] -          ! link 0 of the root  
    host[0]            ! to the PC bus  
  
!  
! Task declarations indicating channel I/O ports and memory  
! requirements  
!  
task afserver ins=1 outs=1  
task filter   ins=2 outs=2 data=10K  
task reader   ins=2 outs=2  
!  
!  
! Assign software tasks to physical processors  
!  
place afserver host           ! afserver runs on PC  
place reader root            ! everything else on root  
place filter root  
!  
! Set up the connections between the tasks.  
!  
connect ? afserver[0] filter[0]  
connect ? filter[0] afserver[0]  
connect ? filter[1] reader[1]  
connect ? reader[1] filter[1]
```

Figure 4.4 Typical Configuration file for a single user task, reader, which will run on the ROOT transputer.

4.7.2 The Flood-Fill Configurer

The flood-fill configurer, FCONFIG, only accepts a few essential statements. It then applies an algorithm which always places a MASTER task on the ROOT transputer and a copy of the WORKER task on ALL transputers including the ROOT. It uses a routing task, 'FROUTER', to manage the flow of work packets across the network of transputers. A typical flood configuration file would look like:

```
Task Master  File = Model  Data = 1000k
Task Worker  File = BIXINT  Data = 250k
```

The FCONFIG utility demands that the Worker task is linked to the stand-alone library while the Master task must be linked to the run-time library. This flood-fill configurer is designed for use in the development of a processor farm.

4.7.3 Processor Farms

Basically, a processor farm program consists of two tasks. The Master task is responsible for dividing the work to be done into small independent work packets. The Worker task is replicated throughout the network of transputers. Any Worker task which is idle within the network can receive an incoming message from the Master task. The Worker task then processes the data received and sends back "result" packets to the Master task.

Messages travelling in each direction must contain a packet identification field. This field enables the Master task to identify the "result" packets and check on the progress of the overall computation. It determines that the job is complete when the results of all sub-jobs have been received back from the farm of Worker tasks. A Master-Slave interpolation program for the HIRLAM model is included in Appendix B.

4.7.4 Disadvantage of Flood-Fill Configuration

A major disadvantage with the flood configurer is that it insists on placing a copy of the Worker task on the ROOT transputer. Since the HIRLAM program is already large it became impossible to use the flood configurer for this reason. The static configurer provides greater flexibility in specifying the allocation of tasks to processors. It was, therefore, used for the distribution of the tasks of the HIRLAM model on the transputer network.

4.7.5 Static Configuration for HIRLAM

A diagrammatic representation of the distribution of the HIRLAM tasks on the transputer network is show in Figure 4.3. The diagram depicts the processors in the network with "broken lines". The tasks in the network are shown with solid lines and each arrow indicates unidirectional channel

communication between the tasks. The "Main Program" task in the diagram has three input vector ports (0,1,2) and three output vector ports (0,1,2). Input vector port 2, for example, provides a communication link with the "Interpol Driver" task. The "Main Program" receives completed interpolation messages from the "Interpol Driver" which is a multiplexer task controlling communication with the interpolation routines placed on processors P003 to P009. It features eight input vector ports(0,...,7) and their corresponding eight output vector ports(0,...,7).

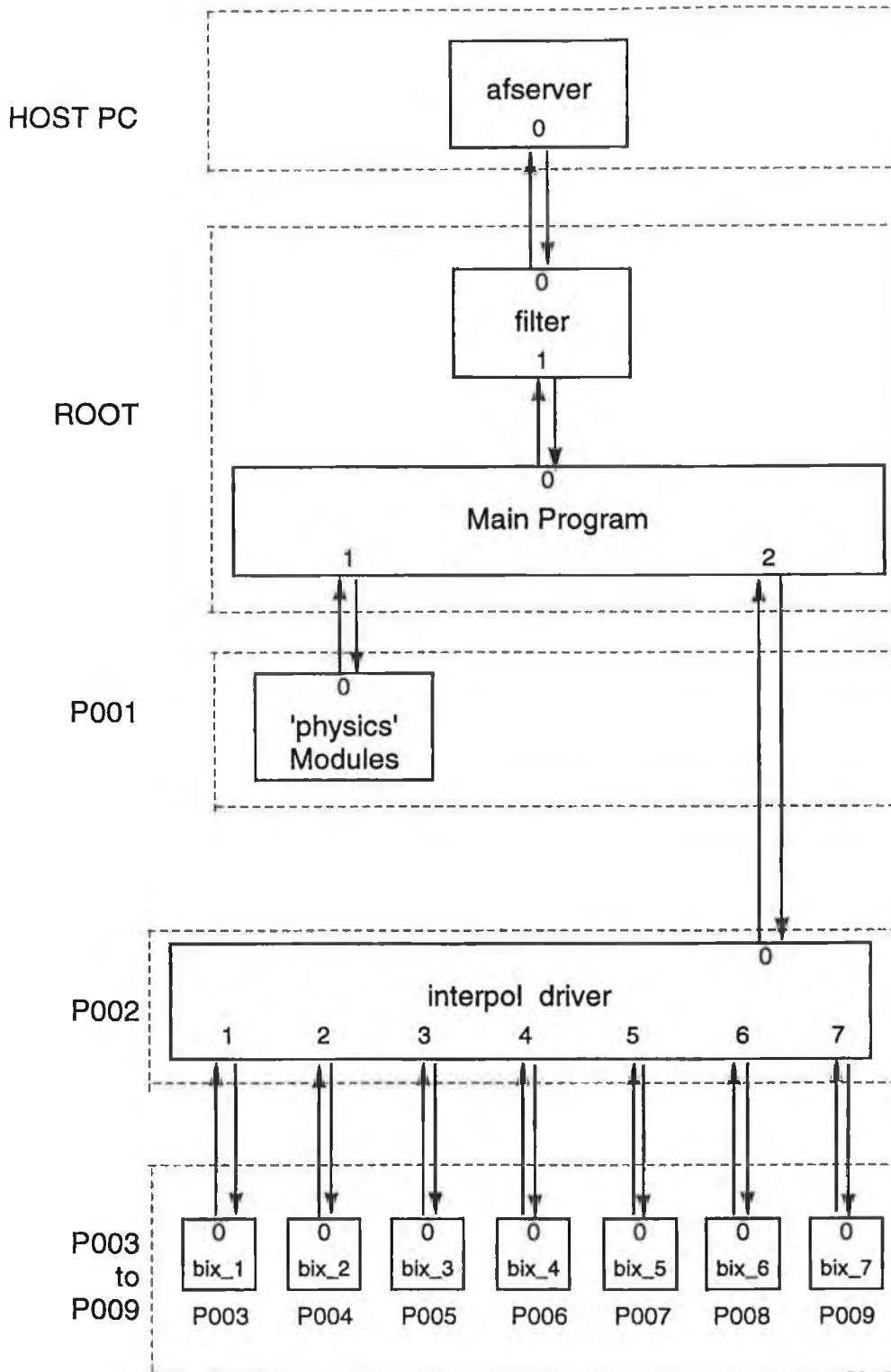


Figure 4.3 Mapping of Processes onto processors - using a multiplexer configuration file. The interpolation, driver distributes the interpolation jobs to the slave tasks on the transputer sub-network P003-P009. The work packets contain an identity field, so that results packets returning from the slave tasks can be identified. Tasks bix_1 . . . bix_7 are identical.

4.8 Memory Management

The current version of Parallel Fortran does not utilize a paging and swapping methodology involving the hard disk of the host PC. This represents a major problem for the programmer if the space required exceeds the entire on-chip transputer memory capacity available. When using the Configuration programs, the expected amount of memory required for some tasks must be specified. The sum of all the memory requirements of the tasks allocated to a particular transputer must not exceed the total on-chip memory capacity of that transputer.

4.8.1 Determination of Static Space Requirements for a Task

After a task is compiled and linked, we can avail of the DECODE utility, supplied with the compiler, to determine its static data requirements. The output from the decode utility is written to a file. The contents of this file can be examined to determine the number of words required by the user task. This value is displayed after the keyword **STATIC**. Since the T800 Transputer is a full 32-Bit chip we can determine the number of bytes required by multiplying the number of 'static' words by four.

4.8.2 Determination of Heap and Stack Requirements for a Task

Every task has a requirement for some heap and stack space. It is more difficult to estimate these requirements. For instance, each level of subroutine calling uses about five words of stack space as well as the space required for

variable data. In short, we must err on the side of over-estimation of the memory requirements. If a task exceeds its stated memory requirements the entire program suite will probably crash. This is usually indicated by the program going into a "Hung" state, with no diagnostic support.

4.8.3 Placement of Critical Routines in fast RAM

It is possible to ensure that critical routines or sub-tasks are placed, at boot time, on the 4K of fast on-chip RAM. This can give a dramatic improvement in speed of execution of the entire program. To achieve this the programmer must write an "options" file which contains the names of the tasks to be placed in the fast RAM. The linker program can be directed to read the options file and to extract the names of the critical tasks. The object code for these tasks is then removed, by the linker, from its current position in the main object code file and it is placed at the top of the executable image.

Then, at execution time, the code for the critical sub-tasks, which has been placed at the beginning of the executable image, is more likely to reside in fast RAM than the code towards the end.

4.9 Summary

The programming concepts of message oriented languages, such as OCCAM and Parallel Fortran have been discussed in this chapter. These languages provide facilities for communicating sequential processes (CSP) to be executed in parallel on one or more transputers. Communication is achieved over channels which are one-way point to point pathways in memory.

The concepts of tasks and threads, as used in Parallel Fortran have been introduced. Transputer network configuration techniques are described and the differences between static configuration and flood-fill configuration have been outlined. This chapter also discussed methods for the efficient management of transputer memory, within the context of the Parallel Fortran programming environment.

Chapter 5

Algorithms for Parallel Environment

5.1 Introduction

The most fundamental difficulty in porting large Fortran programs from a conventional Von Neumann environment, to a concurrent environment, is that they are written in a serial way. The HIRLAM model, for instance, has been developed to simulate the atmosphere. The behaviour of the atmosphere is inherently parallel in nature. In order to make use of the transputer environment the sequential application has to be converted back into a parallel application.

There are at least four methods of converting the system into a parallel application. The first is to start again from scratch and write the entire application in OCCAM, which has been designed to exploit the parallelism of the transputer. Considering the amount of effort already invested by researchers, in the sequential Fortran version, this method is not practical for such a large application. Even if this conversion method was successful, there is a reluctance among the scientific community to move away from Fortran to any "unknown" alternative programming language.

The second method is to use a suitable pre-compiler to translate the existing Fortran program into a form that can run on a transputer. This approach exploits the processing power of the transputer and is cheaper and faster than the first option. However, the development of suitable

translation toolkits is still at the research stage[8],[9]. Translation toolkits are likely to be an attractive option for many researchers in the future.

The third method is to design a hybrid system so that the Fortran programs could be run in an OCCAM harness or shell. It involves the identification of parts of the Fortran programs that can run in parallel. These parts could then be defined as subroutines that could be called via OCCAM. The hybrid approach has merits and has been successfully implemented on a Meiko Computing Surface of transputers[39].

The fourth method is to utilise an existing Parallel Fortran compiler, from 3L Ltd[14]. This programming language has been specially developed for the transputer environment. The implementation strategy devised during the study is discussed in the remainder of this Chapter.

5.2 Analysis of Sequential Program

In order to gain a full understanding of the operation of the HIRLAM model, a sequential version has been specifically implemented on a VAX-4200. The VAX-4200 implementation is used as a "control" model. This approach is useful when planning a porting strategy to the transputer environment. The input test data derived from the analysis of Hurricane Charlie[28] is used to produce a 24 hour forecast on the VAX-4200.

The next step involves the analysis of the program loops, to determine the most expensive segments of the underlying computations. This is achieved by measuring the

amount of CPU time used by all major subroutines in the HIRLAM model. The resultant log file is used to isolate the most critical program segments.

The two most expensive segments identified are (a) the routines associated with the computation of the physical parameters and (b) the interpolation routines associated with the Semi-Lagrangian scheme. These two areas have become the focus of further detailed analysis which involves the investigation of various algorithms used in the model.

5.3 Grid Point Algorithms

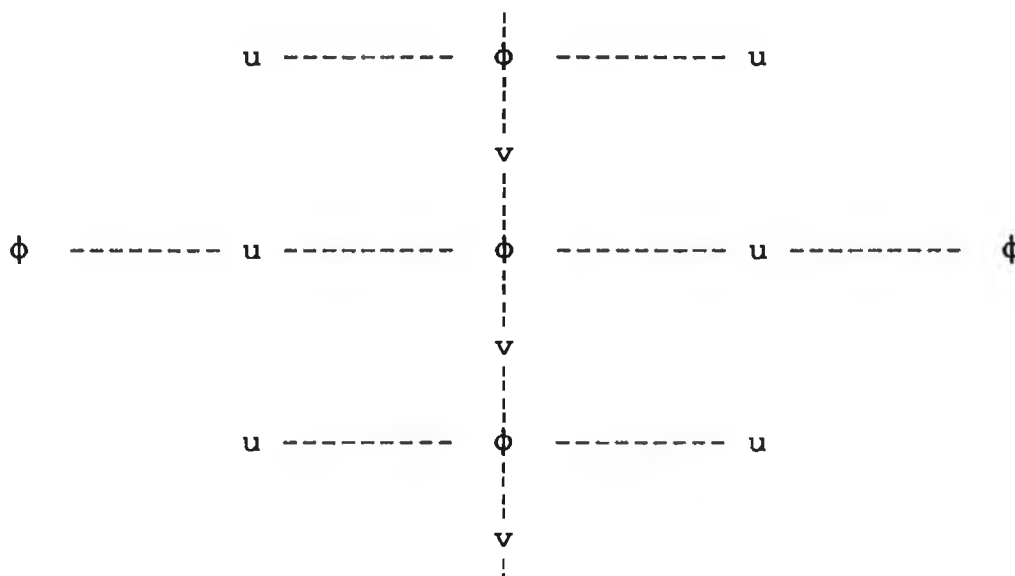
The concept of grid points is fundamental to almost every numerical atmospheric model, especially finite difference models. The grid points are used directly for computations of field variables in finite difference models. The principle underlying the use of grid points is that a continuous fluid (the atmosphere) may be represented by the values of the basic variables located at a finite number of points.

These grid points are arranged in a three-dimensional way. The size of the integration area used is 38 Longitude points by 34 Latitude points. The grid points are placed at intervals of 1.5° along the Earth's surface. Sixteen vertical levels are used for the VAX-4200 control model and only seven vertical levels are used for the transputer implementation.

This grid resolution is still not considered fine enough to model the local intensity of very severe storms. A resolution of 110 by 100 grid points over the same integration area would be required to produce more accurate short term forecasts. But, to model a fine grid such as this, with up to 20 vertical levels, would require roughly 12 MegaBytes of computer memory. The current program structure, if used to model such a fine resolution, could not be realistically accommodated on a distributed memory system such as the transputer network. It would have to be refined into a larger number of small communicating tasks with an emphasis on data independence.

5.3.1 Arrangement of Variables in HIRLAM model

In the HIRLAM model, the field variables, u, v, ϕ , are arranged on an Arakawa C-Grid[30] in the following advective form:



The accuracy of the finite difference approximation to the model equations described in Chapter 2 depends on how well the field variables are distributed over the grid points. The Arakawa C-Grid method is considered to be the most suitable grid method for the HIRLAM model[1].

The terms in the Helmholtz Adjustment equation(2.16) are approximated by central differencing, with averaging. This method is used to evaluate the field variables in the correct position on the grid. For example, the approximation to $\partial\phi / \partial x$ is:

$$\delta_x\phi = (\phi(x+\frac{1}{2}\delta x) - \phi(x-\frac{1}{2}\delta x)) / \delta x \quad (5.1)$$

To make a similar approximation to the ∇^2 term involves a horizontal approximation to the term $\nabla^2\omega$.

Therefore,

$$\nabla^2\omega = \delta_x\delta_x\omega + \delta_y\delta_y\omega \quad (5.2)$$

where

$$\omega = \frac{\Delta p}{\Delta t}$$

p = pressure

5.4 Computation of Workspace Parameters

To dimension Workspace Arrays in the HIRLAM model a parameter known as "MSLAB" is used[1]. Two areas are distinguished from each other in the implementation of HIRLAM-2. They are known as:-

- (a) 'Physics' Routines which deal with the physical parameterisation of the field variables;
- (b) 'Dynamic' Routines which deal with the resolution of the continuous equations.

For the remainder of this discussion these two areas will be referred to as "Physics Routines" and "Dynamic Routines" as is common practice in the study of Meteorological Models.

The following specifications are taken from [1] whereby MSLAB is computed from the formula:

$$\text{MSLAB} \geq (2 * \text{MLAT} - 1) \text{MLEV} / (34 + 27 * \text{MLEV}) \quad (5.3)$$

in the test case :-

$$\begin{aligned} \text{MSLAB} &\geq (2 * 34 - 1) * 7 / (34 + 27 * 7) \\ &\geq 67 * 7 / (34 + 189) \\ &\geq 469 / 223 > 2 \end{aligned}$$

Therefore we choose a value of 3 for MSLAB.

The Workspace required for the Dynamic Routines is :-

$$\begin{aligned} \text{WDYN} &= 2 * \text{MLEV} * \text{MLON} * \text{MLAT} && (5.4) \\ &= 2 (1292) * 7 \\ &= 2 * 9044 \\ &= 18088 * 4 \text{ Bytes} \end{aligned}$$

Workspace required for the Physics Routines :-

$$\text{WPHY} = \{ (34 + 27 * \text{MLEV}) * \text{MSLAB} + \text{MLEV} \} * \text{MLON} \quad (5.5)$$

in our case :-

$$\begin{aligned} \text{WPHY} &= \{ (34 + 27 * 7) * 3 + 7 \} * 38 \\ &= \{ 223 * 3 + 7 \} * 38 \\ &= 676 * 38 \\ &= 25688 * 4 \\ &= 102,752 \text{ Bytes} \end{aligned}$$

where

MSLAB = number of 'slab' slices used with Physics Routines

MLAT = number of grid points in latitude direction

MLON = number of grid points in longitude direction

MLEV = number of vertical levels

WDYN = work space required for Dynamic Routines

WPHY = work space required for Physics Routines

5.5 Finite Difference Methods

The governing equations, used with the HIRLAM model and described in Chapter 2, use spatial derivatives of the basic variables. In the finite difference method, each spatial derivative is replaced by a finite difference approximation[40]. The simplest finite difference approximation of the derivative $\partial p / \partial x$, at a grid point (i, j) , is:

$$\left(\frac{\partial p}{\partial x} \right)_{i,j} = \frac{P_{i+1,j} - P_{i-1,j}}{2a} \quad (5.6)$$

where, a is the grid length between points. The difference between the true value of the derivative and the finite difference approximation is known as the "truncation error". In the HIRLAM model, a staggered grid is used to make the application of finite difference approximations more accurate. For instance, a more accurate approximation at the point $(i+\frac{1}{2}, j+\frac{1}{2})$ is:

$$\left(\frac{\partial p}{\partial x}\right)_{i+\frac{1}{2}, j+\frac{1}{2}} = \frac{1}{a} \left(\frac{p_{i+1, j} + p_{i+1, j+1}}{2} - \frac{p_{i, j} + p_{i, j+1}}{2} \right) \quad (5.7)$$

In this case the finite difference approximation is computed over a single grid length, a . This method uses averaging to obtain the values of p , at the required position.

5.6 Time Integration

When the rate of change of each model variable is known at each grid point, a method of time integration is required so that the state of the model atmosphere may be advanced. The time integration scheme, used by the HIRLAM model, is outlined in Chapter 2. One of the methods used is the forward integration scheme known as Leapfrog.

Consider the variable q , which represents specific humidity in the model. Suppose the original input value of q is q_0 at some grid point, and the rate of change of q is $(\partial q / \partial t)_0$. The value of q_1 after one forward time step (δt) is:

$$q_1 = q_0 + \delta t \left(\frac{\partial q}{\partial t} \right)_0 \quad (5.8)$$

The state of each variable in the model atmosphere is computed at each time step. After n time steps

$$q_n = q_{n-1} + \delta t \left(\frac{\partial q}{\partial t} \right)_{n-1} \quad (5.9)$$

5.7 Numerical Instability

The simple step by step integration procedure is complicated by numerical instability. The application of a particular time integration scheme to a finite difference approximation may lead to numerical solutions where the numbers grow rapidly in size. In such circumstances, the computed result is often a long way from the true solution, and can eventually lead to a computer crash.

The leapfrog scheme avoids this instability. This is achieved by centering the computations in time. In this case, equation 5.9 is replaced by:

$$q_n = q_{n-2} + 2\delta t \left(\frac{\partial q}{\partial t} \right)_{n-1} \quad (5.10)$$

Therefore, for a given grid length, a , and a given maximum velocity, the stability criterion imposes a limit on the size of the time step, δt , which may be used [40].

Given the large number of grid points required to represent the area of integration, together with the model variables and the necessity for short time steps, it is easy to see how numerical modelling imposes a high computational demand on computer resources.

5.8 Efficiency of Semi-Implicit Time Stepping

The Arakawa C-Grid facilitates the resolution of the Helmholtz equations (2.16) on a rotated spherical grid. This in turn provides a fast solution method for the semi-implicit time stepping scheme. Therefore, the Helmholtz equations are resolved for a horizontal plane for each of the vertical levels of the model. Their solution consists of a Fast Fourier sine-Transform (FFT) in the east-west direction and Gaussian Elimination in the north-south direction.

An alternative to the FFT method is proposed by Hartley[41]. He called the new method the Fast Hartley Transform (FHT). This solution method maps a real function of time, $X(t)$, onto a real function of frequency, $H(f)$. The FFT method maps a real function of time, $X(t)$, onto a complex function of frequency, $F(f)$. Since the Hartley frequency function, $H(f)$, is real, only single arithmetic operations are required to compute it. It requires fewer computer storage resources. Therefore, it is capable of being twice as fast as FFT. It is proposed, that in future study of the HIRLAM model, the Fast Hartley Transform be substituted for the Fast Fourier Transform.

5.8.1 Disadvantages of FFT in Transputer Environment

The FFT method has a minor disadvantage in that the necessary factorisation limits the choice of the number of grid points in the east-west direction.

The FFT solution method requires global communication and this is inefficient for a distributed computer architecture such as a transputer network. Therefore, to avoid the high communication overhead due to the structure of the FFT algorithm it was decided to perform all of the associated computations on the ROOT transputer.

5.8.2 Iterative Solution Method

Gaussian elimination is used to solve a series of tri-diagonal systems[16] for the resolution of the Helmholtz equations(2.16). "It is possible to reduce the computation work necessary for solving, $Ax = b$, by taking into account special features of the coefficient matrix A , such as symmetry or sparseness" according to Conte and deBoor [21].

5.8.3 Alternative Parallel Solution Methods

For large sparse systems of equations, such as tri-diagonals, a distributed multi-frontal sparse matrix decomposition algorithm has been developed[42]. This algorithm is suitable for the message passing architecture of transputer networks and has been found to overcome the communication bottleneck associated with other sparse matrix solvers[42].

An alternative algorithm for the solution of triangular systems on a parallel processing system has been proposed by Montoye et al[43]. Using a data pipeline approach the algorithm requires $O(\log(N))$ fewer processing cycles than the Gaussian elimination algorithm, where N is the system size.

To implement either of the above algorithms on a transputer network it would be necessary to redesign the entire HIRLAM time integration procedure.

5.8.4 Alternative Iterative Methods

The Gaussian elimination algorithm can be replaced by less computationally expensive algorithms. Iterative methods which compute new values at any given grid point, by using the values at certain neighbouring grid points, are a very important class of grid algorithm.

For instance, the Jacobi method[21] and the Gaussian elimination method use old values of neighbouring grid points when computing new values. The Gauss-Seidel Method[21] uses new values from the neighbouring points if they are available. Explicit time-stepping techniques usually use the Jacobi method and this in turn requires two iterates to be kept in memory[21].

5.9 Degrees of parallelism in iterative solution methods

Consider a regular grid of N grid points. The degree of parallelism of a Jacobean method is proportional to N . However, Gauss-Seidel methods are preferred because of their convergence and smoothing properties. The Gauss-Seidel iteration only requires one vector in memory since each entry x^n is used in the calculation of all succeeding entries of x^n .

If we use a RED-BLACK ordering over the grid points, then, we can say that each step of a Gauss-Seidel method consists of two Jacobi steps, and therefore, has a degree of

Parallelism of $N/2$ [29]. If we have larger difference operators or larger neighbouring formulae, we can find an ordering of grid points by colouring with C colours. This will give a degree of parallelism of N/C .

The smoothing properties of any solution method, such as Gauss-Seidel, are important in a multi-grid context. They can be improved by using a multi-colour ordering. On the other hand, the asymptotic convergence properties of a solution method are usually independent of the colour ordering of the gridpoints.

5.10 Speedup of Iterative Convergence

For sparse matrices the number of arithmetic operations to be performed per step is small. However, iterative methods will not always converge to the required level of accuracy. Even when they converge a large number of iterations may be required. It is important to speed up the convergence of the iterative method.

The successive overrelaxation (SOR) technique is recognised as one of the best techniques available for speeding up the convergence of fixed-point iteration. The technique 'overshoots' the change from x^m to x^{m+1} . Therefore, for a given linear system $Ax = b$ the Gauss-Seidel iteration method can be expressed as follows:

$$x_i^{m+1} = x_i^m - \left(\sum_{j < i} a_{ij} x_j^{m+1} + \sum_{j > i} a_{ij} x_j^m - b_i \right) / a_{ii} \quad \forall i \in [1..n] \quad (5.11)$$

while the SOR method can be expressed as:

$$x_i^{m+1} = x_i^m - \omega \left(\sum_{j < i} a_{ij} x_j^{m+1} + \sum_{j > i} a_{ij} x_j^m - b_i \right) / a_{ii} \quad \forall i \in [1..n]$$

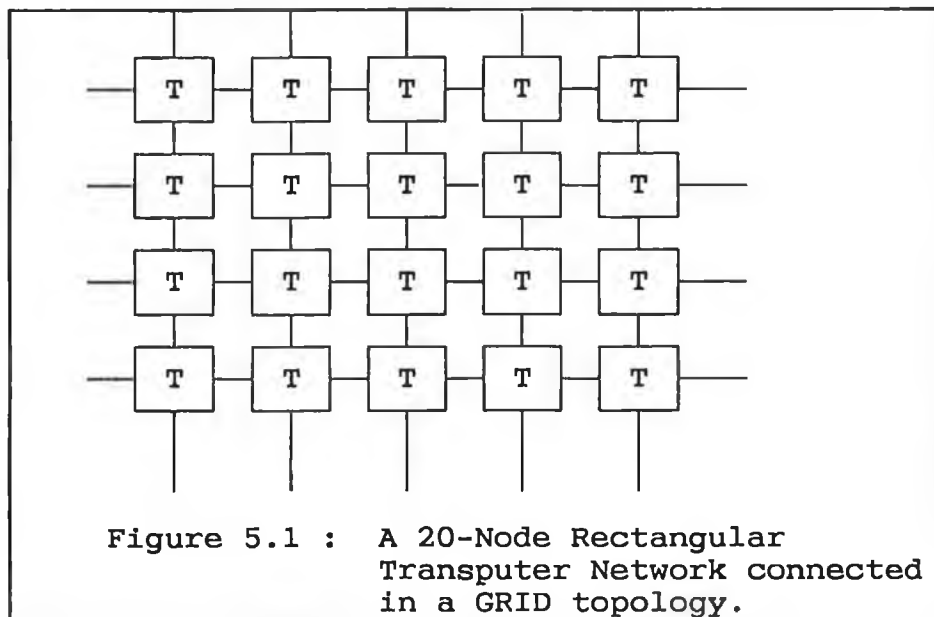
(5.12)

where $\omega (>1)$ is the overrelaxation parameter and typically has a value between 1.2 and 1.6 [21].

a is an element of matrix A

b is RHS row vector value

5.11 Grid Partitioning



A grid may be divided into rectangular sub-grids. This allows a typical parallel grid algorithm to be applied to it. Each sub-grid could be assigned to a transputer node in such a way that neighbouring sub-grids are mapped into neighbouring nodes.

To achieve this, practically, we would have to have the nodes in the Transputer network connected in a rectangular manner. Each of the north, south, east, and west channels should be connected to their counterparts on neighbouring nodes with wrap-around at the borders of the mesh as shown in Figure 5.1.

The variables associated with each grid-point in a sub-grid should be exclusively computed by their associated Node. If these variables were only transferred to the computing node just when they are needed, the process of transferring them would introduce a "blocking" which would typically introduce a communications bottleneck.

To prevent this, it is better to store such variables in "overlap areas". Then, after each iteration, the values in the overlap areas are exchanged and copied using a message-passing protocol to the appropriate Node, using channel communication. The use of the overlap areas does not alter the computation method, but ensures that all nodes are synchronised after each iteration.

5.12 Data Distribution by Grid Partitioning

FFT algorithms are inherently non-local and the coefficient of one specific wave number depends on all available input values. Part of the integration procedure uses a fast Poisson solver, which in turn uses one-dimensional FFT in each of the latitude and longitude directions.

To solve the Poisson Equation(2.21) on a distributed Memory architecture, such as a Transputer Network, it is much more appropriate to use a multi-grid solution method rather than an FFT Method. Typically, the number of actual computations for an FFT Method are similar to a multi-grid method but the communications overhead of the Multi-grid method is much lower than the FFT Method.

The Poisson equations(2.21), for Vorticity and Divergence take the form:

$$\nabla^2\psi = \xi \quad ; \quad \nabla^2\chi = \delta \quad (5.13)$$

After each forward step, the new values of horizontal and vertical velocity, u and v respectively, must be derived for the vorticity, ξ , and divergence, δ . If two Poisson equations are solved for the stream function, ψ , and the velocity potential, χ , then the velocities obtained by differentiation are:

$$V = k \cdot \nabla\psi + \nabla\chi \quad (5.14)$$

It is assumed that the dependent variables are independent of y, and are specified on a discrete grid:

$$\{X_0=0, x_1, x_2, \dots, x_n=L\}$$

where

L = total width of the integration area

n = number of grid points in horizontal direction

k = constant

It is also assumed that the dependent variables are periodic in x . Therefore, the two equations to be solved take the form:

$$d^2\phi / dx^2 = p \quad ; \quad \phi(0) = \phi(L) \quad (5.15)$$

The reference geopotential is arbitrary and so $\phi(0) = 0$ is chosen. Given $\delta x = 1$ the discrete equations are written as:

$$\begin{array}{rcl} -2\Phi_1 & +\Phi_2 & = P_1 \\ \Phi_1 & -2\Phi_2 & +\Phi_3 = P_2 \\ & \Phi_2 & -2\Phi_3 = P_3 \end{array}$$

$$\begin{array}{rcl} & & \Phi_{n-2} -2\Phi_{n-1} = P_{n-1} \\ \Phi_1 & & +\Phi_{n-1} = P_n \end{array}$$

The first equation above is multiplied by one, the second by two, and so on. The resultant equations add up to give:

$$N\Phi_1 = \sum_{n=1}^N nP_n \quad (5.16)$$

This gives us $\phi_1, \phi_2, \phi_3, \dots$ and so on until a solution is found. When the stream-function and the velocity potential have been derived, the velocities are then derived on the staggered grid using the equations:

$$U_n = (\chi_n - \chi_{n-1}) / \Delta x \quad (5.17)$$

$$V_n = (\psi_n - \psi_{n-1}) / \Delta x \quad (5.18)$$

Clearly, the solution method chosen to solve the Poisson equations(2.21) must be efficient so that maximum benefits from the parallel architecture are achieved. One solution method to fit on a 'star' of five transputers might be as follows:

Assuming the variables on the grid points are $u(i,j)$, we can say:

$$\nabla^2 u \rightarrow u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - 4u(i,j) \quad (5.19)$$

A relaxation algorithm for the solution rewrites the discretised equation as:

$$u(i,j) = [u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - f(i,j)] / 4 \quad (5.20)$$

where the RHS is evaluated and used to overwrite the element $u(i,j)$.

If any of the points in this expression are located on the boundary of the integration region then a constant boundary could be used. This is a simple version of the Gauss-Seidel algorithm which can be implemented in parallel on a transputer network. The efficiency of the implementation depends on the network topology and the communications bandwidth between the nodes.

This algorithm is particularly well suited to a fine-grained array of processors. However, the network of 11 transputers used with this study is a coarse-grained processor array which would require a high degree of data exchange between nodes. Furthermore, the nodes in the network were connected in a 'snake-like' manner, with just one physical link connecting each node to a neighbour on either side.

5.13 Possible Ways of Partitioning

(1) Partitioning in the Longitude Direction (x-axis):

If the FFT's are used for Fourier interpolation vertical layer boundaries, then there is a high overhead associated with a parallel implementation.

(2) Partitioning in the Latitude Direction (y-axis):

To partition the grid computation in this direction involves the addition of extra memory management and communication between processors.

(3) Partitioning in the Vertical Direction (z-axis):

Due to the current code arrangement in HIRLAM this partitioning approach is the most convenient option. There is no overhead in Memory Management among the processes. However, the boundary conditions at the upper layer boundary can cause some load imbalance. This can be partially resolved by assigning many grid-points to

the computation processes and only using a few processes altogether in the vertical direction. 7 processes for a 7 vertical level model were used in our research. The use of just 7 worker tasks for the interpolation task reduces the amount of additional workspace required for intermediate results in the integration.

5.14 Summary

The various aspects of porting a large Fortran program, such as HIRLAM, from a sequential environment to a concurrent environment, have been introduced. The availability of a suitable parallel Fortran compiler for the transputer environment meant that the porting of the program was feasible.

Algorithms suitable for application in a parallel environment have been explored in this Chapter. The algorithms are described from the point of view of their suitability for the HIRLAM model. The implementation of some the parallel algorithms outlined would require a significant amount of re-programming. Other algorithms could be incorporated quite easily but their subsequent verification would require exhaustive testing of the model.

Chapter 6

Model Modifications Undertaken

6.1 Introduction

An attempt was made to compile and link the original sequential version of the HIRLAM model using the Parallel Fortran compiler. The size of the resultant executable image was 4.5 Mbytes, but the on-chip memory of the ROOT transputer consisted of a mere 2 Mbytes. The shortage of memory became a major issue to be resolved. The approach to solving the memory shortage problem consisted of three distinct phases.

6.2 Modifications made in phase one.

Phase one involved the separation of the "physics" routines from the main model. This meant that the overall size of the program was reduced to approximately half its original size. Both parts of the program would eventually be placed on separate transputers with the Main part of the model on the ROOT transputer and the "physics" part on the P001 transputer. However, in order to get a working sequential program running on one transputer it was necessary to omit all references to the "physics" routines. The resultant scaled down program was expected to produce significantly worthwhile results for test purposes.

6.3 Modifications made in phase two

Phase two involved the removal of all references to a 'spare' variable which was eventually intended to be used for the computation of water vapour content. However, the actual governing equations for this variable are still only at the research stage and so it too was removed entirely from the program. This had the effect of freeing up approximately one sixth of the static space allocated for the program.

6.4 Modifications made in phase three

Phase three involved the evaluation of a wide range of suitable parameter combinations (in accordance with the schemes described in Chapter 5, Section 5.4). The aim of this phase was to arrive at a combination of program parameters which would reduce the space requirements for the workspace arrays and COMMON blocks. This in turn would allow a scaled down version of the main program to be compiled, linked and run on one transputer, initially. It was considered essential to retain the good test data provided for this case study and for the later comparison of results.

Therefore, the Longitude and Latitude grid points at values of 38 and 34 respectively were retained. The number of vertical layers was reduced in the analysis and boundary test data sets from the original 16 layers to 7 layers. This involved writing an interpolation program which would take as input a 16 level Boundary Data File, interpolate the data to 7 layers and output the new data set to a new file. Since, the lowest 1000 kilometres of the atmosphere is the most

important, in the HIRLAM model, a weighting factor was applied in the special interpolation program to preserve the consistency of the data in the first four layers above the surface of the earth.

The interpolation program used for this purpose is included in Appendix A. Both the input file and output file were in accordance with the data description format defined for the original HIRLAM program. The format stipulated that the data should be written to binary files in slab format[16].

6.5 Task Placement Strategy Devised

The space requirements for the 'Physics' task, which models the routines associated with the physical parameters, were such that it required the entire memory resources offered by the P001 transputer. The placement of such a specific task on one transputer precluded the use the processor farm technique. This was because, as we have already seen, the flood configurer expects that all transputers throughout the Network, including the ROOT, should run a copy of the Worker Task.

Therefore, it was decided that the network of transputers would have to be explicitly configured in order to make maximum use of available resources. Figure 6.1 shows how the network was configured eventually.

6.5.1 Using a Multiplexer Technique

A 7 to 1 Port Multiplexer Routine was written to send out work packets for the interpolation of each layer as quickly as possible. This ensured that for some of the time at least all 7 processors were computing different layers of the atmosphere. As each worker task completed a computation it returned its message out along the output channel from which it received the input data originally.

This allowed the integration to proceed along much faster than if there was only one processor to do the work. It also meant that there was full control over which task ran on which Processor and the available computer resources could be exploited with the least number of modifications to the model and with the greatest gain in speed.

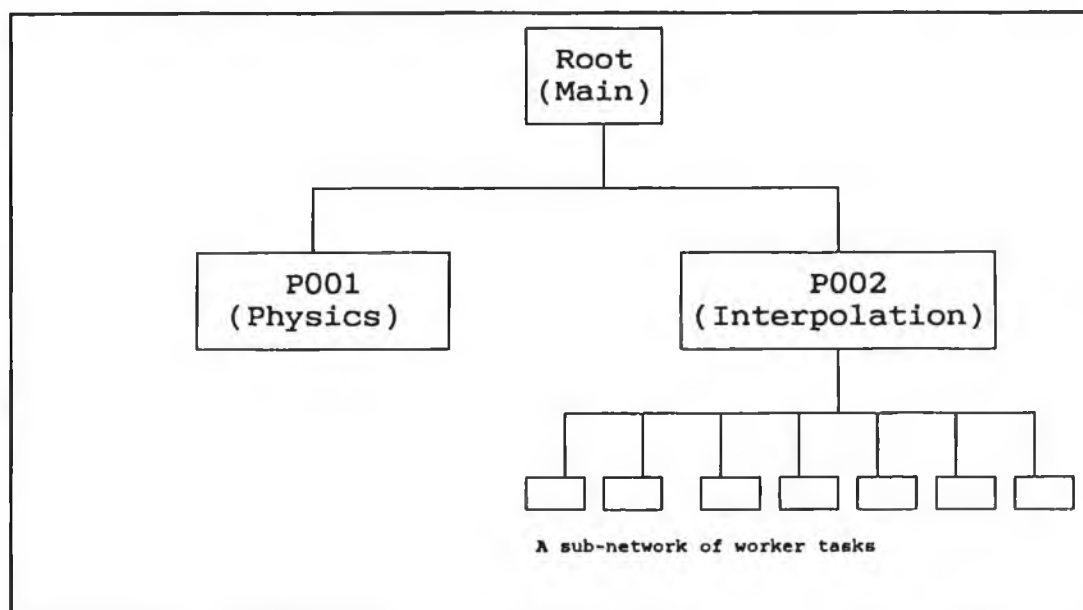


Figure 6.1 Schematic representation of 7 Interpolation Worker Tasks placed on 7 transputers and controlled by a multiplexer driver on P002. The ROOT transputer runs the Main program while P001 transputer performs the computations for the physical parameters.

6.6 Consequences of Task Redistribution

The removal of the 'Physics' routines from the main program and the updating of the common data structures accessed by both program segments became a major problem to be resolved. The consequences of these modifications are summarised below.

6.6.1 Advantages of Task Distribution

Initialisation of the main task on the ROOT transputer and the remaining tasks allocated to the P001 transputer could be executed in parallel. There was spare memory capacity on each of the respective processors which could be utilised, eventually, to execute a version of the model with finer granularity.

6.6.2 Disadvantages of Task Distribution

All of the 'Write' statements had to be removed from the routines on the P001 transputer. These tasks had to be compiled using the standalone library and could not be linked with the run-time library. All communication of the data to/from the task on transputer P001 was via message passing over channels. A "Driver" or Main Program Module was required to receive the Work packets and pass them on to the routines on the P001 transputer.

Some of the utility routines, devised for diagnostic purposes, in the original program, such as, "OUTPHY", were dispensed with. This had the advantage of saving space on the P001 transputer but the disadvantage of removing diagnostic

printouts for the 'Physics' routines on that transputer.

6.7 Problems reading binary files

The set of test data used for the case study was intended to be read from binary files. It was discovered that only files with a small record size could be read in this manner. In the case of files with larger records, the equivalent in ASCII of 4000 Bytes, the Parallel Fortran program would crash.

Eventually, it was discovered that the program would read the first 4 bytes as integer values, assume that this huge number was the first record length and begin to read it. Then, 32Kb later, it reported the record was 'too large'. This difficulty with binary files was not documented in the User Manual and was only resolved with some help from 3L Limited.

The solution was to write a conversion program, (see Appendix A), to convert all five test data files to ASCII. This in turn meant that all the "read statements" throughout the programs were modified to read in ASCII format. The binary to ASCII conversion program was written and executed using, an 80386 PC compatible, Fortran Compiler from Salford University[20].

6.8 Compiler Problems associated with Large programs

Large programs, which compiled and linked successfully, failed to run when executed via the 'afserver'. The exit status reported by the 'afserver' was that static space was

too small. This difficulty was overcome by instructing the 'afserver' utility program to treat all of the memory, including the on-chip RAM as external memory only. Although the program could now be executed on the transputer it suffered the penalty of not being able to benefit from the fast on-chip RAM. Programs executed in this way were relatively slow and there was no opportunity to use the debugger due to the depletion of processor memory. Therefore, the only debugging technique at our disposal was the 'old fashioned' write statements and these did not always isolate the instruction where the program failed.

6.9 Using The Parallel Fortran Debugger, TBUG.

To utilise the debugger, TBUG, effectively with any program which was written in a Master/Slave Configuration, it was necessary to write a smaller version of the program initially. The scaled down program used a typical data set but had smaller Memory requirements.

6.9.1 Extracting a Sample Data Set from a VAX-4200 program

The sequential version of the HIRLAM program on the VAX-4200 was modified so that a typical data set passing to the interpolation routines could be 'trapped' and written to a special data file. This file was then used for test purposes during the program modifications on the transputer. These modifications involved writing a scaled down version on the interpolation routines used with the Semi-Lagrangian Scheme discussed in Chapter 2.

An interpolation program was designed and developed to run on the transputer network in a Master-Slave configuration using the trapped data obtained from the VAX-4200. The TBUG utility program was used to debug the various Task and Thread routines and message passing techniques offered by the 3L Fortran-77 Programming language. The code used in the Master-Slave configuration is shown in Appendix B.

The initial test program used the Flood-fill configurer which insisted on placing one Worker Task on the ROOT Transputer. However, when we came to the main HIRLAM program, the placement of the extra worker task on the ROOT depleted the amount of memory available for use by the Master module and by TBUG itself. Although TBUG worked well with this test program and was invaluable as a development tool, it could not be used alongside the main program on the ROOT transputer.

6.9.2 Limitations of TBUG Debugger

TBUG was an essential tool in the early program development. Small modules were designed and tested, but, when they were incorporated into the larger program, TBUG failed to run. The main limitations of TBUG were found to be:

- (1) It did all its simulation of Master /Slave configurations on one processor.
- (2) It also ran out of memory and some of its error reporting was poor.

- (3) If there were extra processors available, and the configurer specified that they should be used, then TBUG should have actually placed TASKS on the processors which were intended to be used by the programmer.
- (4) In order to debug the test programs the TBUG parameters had to be adjusted, via menu options, so as to increase the workspace to the full transputer memory capacity. This meant that fewer Worker Tasks could be simulated.
- (5) We include a simpler example, in Figure 6.2, which enables TBUG to simulate up to 4 Worker Tasks working in Parallel.

6.10 A Sample Test Program

A Full working example program is now described together with all its accompanying configuration and link files. The program uses a master module and a slave module. Two configuration files are presented, the first is a flood-fill configuration and the second is a multiplexer configuration. Finally, a Batch file to control the program compilation, linking, configuration and running is presented.

The Master Task, shown in Figure 6.2(a), begins by including the declarations for the special Parallel Fortran library routines which support channel communication. Local program variables and buffers are then declared.

Program execution begins by assigning the value of the task's output port(0), returned from the function f77_chan_out_port(0), to the variable 'outport'. Function f77_chan_out_ports() is used to determine the total number of output ports assigned, at configuration time, to the Master Task. Likewise, function f77_chan_in_ports() determines the number of input ports assigned to the Master Task. The actual values of each of the input ports are then determined individually and their respective values are stored in vector 'invec'.

The main program segment then begins. It loops, receiving and processing messages from all input channels, until the 'complete' flag is set to 'TRUE'. This flag is set to 'TRUE' by the slave task, in this instance, when the work is complete. The program uses the function 'f77_alt_wait_vec()' to scan the vector of input ports. If there is a message waiting to be received from a particular port then the identity of that port is returned. The incoming message can now be read from the identified input port. Otherwise, function 'f77_alt_wait_vec()' waits and scans all of the input ports until a new message arrives.

All incoming messages have an 'agreed' format which is known to the 'Master' and 'Slave' tasks. In this example, the message consists of three distinct segments. The first segment is read by procedure 'f77_chan_in_word' and contains the boolean value for the 'complete' flag. The second segment is read by another call to procedure 'f77_chan_in_word' and it contains the 'length' of the message which follows.

Using the 'length' value procedure 'f77_chan_in_message' reads in a message of 'length' bytes into the buffer 'buff', from input port 'inport'. Finally, the new message just received is formatted and written to the host screen.

6.10.1 Code for the Master Task

```

        program mast
c
c      Master Module
c
c      Parallel Fortran to demonstrate the message passing
c      techniques on a configurable Transputer Network.
c
        include 'alt.inc'
        include 'chan.inc'
        integer buff(10000)
        integer invec(128), inport, length, outport, inports
        logical complete

        outport = f77_chan_out_port(0)
c
c      put addresses of all input ports into invec
c
        inports = f77_chan_in_ports()
        complete = .false.
        do 100 i = 1, inports
            invec(i) = f77_chan_in_port(i-1)
            write(6,*)'Master: channel input ports:', invec(i)
100      continue

        do while( .not. complete)
c      Wait for a channel to be ready...
            write(6,*)' complete = ', complete
            inport = f77_alt_wait_vec(inports, invec)
            inport = invec(inport)

c      read completion flag, message length, message body

            call f77_chan_in_word(complete, inport)
            call f77_chan_in_word(length, inport)
            call f77_chan_in_message( length, buff, inport)

c      write the message to port 0 which is connected to
c      filter which in turn is connected to MS_DOS

            write(6,*)' length = ', length, ' from port ', inport
            do k = 1, length/4, 5
                write(6,*)' k = ', k, '|', (buff(i), i = k, k+4)
            end do
        end do
300      stop
        end

```

Figure 6.2 (a) : A sample Master Task in Parallel Fortran

6.10.2 Code for the SLAVE Task

```
program slave

c      Slave Module
c
c      A sample WORKER task which sends a series of 10
c      messages to a Master task via channel 0
c
      include 'chan.inc'

      integer outpost,length, inport, buff(1000)
      logical complete

      outpost = f77_chan_out_port(0)

      kkkk = 0
      inport = f77_chan_in_port(0)
      complete = .false.

200    continue          ! Loop back to here until
                          ! finished
          length = 100
          kkkk = kkkk + 1
          do 100 k = 1, length
            buff(k) = kkkk
100    continue

c
c      Now write buff message to port 0
c
          length = length * 4 ! Length of message in Bytes
          if(kkkk .ge. 10)complete = .true.
          call f77_chan_out_word(complete,outport)
          call f77_chan_out_word(length, outport)
          call f77_chan_out_message(length, buff, outport)
          if(kkkk .lt. 10)goto 200
      end
```

Figure 6.2 (b) : A sample Slave Task in Parallel Fortran

The Parallel Fortran code for the Slave Task shown in Figure 6.2(b) uses the same functions described for the Master Task to determine its port allocation.

The main loop in the Slave Task consists of a 'computation' loop which fills up the output buffer 'buff', with data. It then computes the length of each output message in bytes. It checks if all the computations have been completed.

It begins communication with the Master Task by attempting to send the value of the 'complete' flag by calling procedure 'f77-chan-out-word'. If the Master is 'listening' it will detect the incoming message and read it. Otherwise, the slave task must wait until the Master responds.

After the first word is read by the Master, the Slave task can then send the 'length' of the main message, out along its 'outport' channel. Finally, the message proper is sent by calling procedure 'f77_chan_out_message'. If the job is not complete the Slave task loops again, until it is complete.

6.10.3 The Flood-Fill Configuration File

The instructions expected by the flood-fill configuration program, FCONFIG, are quite simple but they deprive the programmer of a means of controlling how the tasks should be placed on the network. This control is completely handled by the FROUTER task to which FCONFIG links.

Therefore, the contents of file F_EXAMP.CFG are as follows:

```
! F_EXAMP.CFG
!  
Task Master  File = Master  Data = 280k  
Task Worker  File = Slave   Data = 170k
```

Figure 6.2 (c) : A sample Flood Configuration file

6.10.4 The Normal Configuration File

By using the configuration file, shown in Figure 6.2(d), the programmer has much more control over what task is placed where on the transputer network. If the size of memory space required by a task can be accurately determined, then, there is the opportunity to place as many tasks as will fit, onto one transputer. This assumes that the user has used the WORM utility program to determine the space available on each transputer.

The Configuration declares three processors, namely, HOST, ROOT and P001. Then the interconnecting wires are declared between the three processors. This is followed by declarations for each of the tasks. Each task destined for a transputer has its space requirements specifically declared. For example, the space to be allocated for the Master task on the ROOT transputer should be 540 Kbytes. The slave task on P001 requires 280 Kbytes. The number of input and output ports are specifically allocated to each task.

The next statements in the configuration file set up the connections between the various tasks. Each task, together with all its input and output ports, must be connected up explicitly with the ports of the tasks it must communicate with. This is a critical stage in the configuration and will result in deadlock in the transputer network if the task connections are not declared correctly.

The final statements in the configuration file assign the software tasks to the physical processors.

```

! EXAMP.CFG
!
! Configuration file for Master - Slave Example Program
! This configuration uses the task distribution Technique
!
Processor Host          ! 486  PC
Processor Root         ! Transputer interfaced to Host PC
Processor P001        ! Select Transputer P001 for Slave

Wire ? Host[0] Root[0] ! Connect PC to Root Transputer
Wire ? Root[2] P001[1] ! Connect Root Transputer to P001

! Task Declarations
Task AfsERVER Ins=1 Outs=1
Task Filter  Ins=2 Outs=2 data = 10K
Task master  Ins=2 Outs=2 data = 540K
Task slave   Ins=1 Outs=1 data = 280K

! Set up the connections between the tasks
Connect ? AfsERVER[0] Filter[0]
Connect ? Filter[0]  AfsERVER[0]
Connect ? Filter[1]  master[1]
Connect ? master[1]  Filter[1]
Connect ? master[0]  slave[0]
Connect ? slave[0]  master[0]

! Assign software tasks to physical processors
Place AfsERVER Host
Place Filter  Root
Place master  Root
Place slave   P001

```

Figure 6.2 (d) : Normal configuration file format

6.10.5 Link Files for Master and Slave Tasks

```

master.bin          ! Contents of Master.lnk
\tf2v1\frtlt8.bin  ! Link Run-Time-library
\tf2v1\taskharn.t8 ! Link T800 General
                   ! Library

slave.bin           ! Contents of Slave.lnk
\tf2v1\safrtlt8.bin ! Link Standalone library
\tf2v1\taskharn.t8 ! Link T800 General
                   ! Library

```

6.10.6 Batch File to control compilation Master-Slave Tasks

The Batch file shown in Figure 6.2 (e) includes switches at the compilation and linking stages to enable TBUG to access the Symbol Tables of the respective Tasks. Obviously, these switches increase the overall size of the program to be debugged. They should be removed from the compilation and linking stage when the program is fully debugged.

```
REM ! compile.bat
REM ! Phalton
REM ! 1/7/92
REM !
t8f master.f77/Zi/Zd          ! Include Tables for
t8f slave.f77/Zi/Zd          ! debugging with TBUG
rem
linkt/I/G @master.lnk, master.b4 ! Include the Global
linkt/I/G @slave.lnk, slave.b4   ! symbols for TBUG
rem
REM configure `STATIC' version of application
config EXAMP.CFG normal.app

REM configure 'FLOOD-FILL' version of application
Fconfig F_EXAMP.CFG flood.app

REM Submit the static program to the configured network
afserver -:b normal.app

REM Submit the FLOOD-FILL program to the entire network
afserver -:b flood.app
```

Figure 6.2 (e) : Batch file to control the sequence of compilation, linking and configuration stages

6.11 Summary

This chapter has described the modifications phases involved in porting the HIRLAM model to a concurrent version on a transputer network. The difficulties associated with shortage of memory on the transputer have been highlighted. These difficulties were overcome by devising a method of task distribution across the network of transputers available.

The techniques for transputer network configuration have been described. An example Parallel Fortran program, included in the chapter, illustrates the message passing techniques and channel communication provided by the Parallel Fortran programming environment.

Chapter 7

Program Performance Evaluation

7.1 Introduction

The performance of the HIRLAM Model can be evaluated under several headings. Ultimately, the most important performance metric is the accuracy and dependability of the weather forecast produced by the model. This topic is discussed in Chapter 9.

The next most important performance metric is the speedup gained by various versions of the program. The base time to improve upon was considered to be the time taken for a sequential version of the program to run on the ROOT transputer. Obviously, the I/O associated with accessing files on the hard-disk via the MS-DOS interface was an important factor since this interface is still a communications bottleneck. Therefore, to achieve good and accurate results it was essential to minimise such disk access. This was achieved by reducing the amount statistical diagnostic information written to disk during model run time.

This was balanced with the transputer memory limitations and the need to introduce temporary files which could be placed in the MS-DOS environment. In order to improve this aspect of performance it was decided to use a virtual disk, D: in the memory of the HOST PC. It had a capacity of 7 Mbytes which were virtually unused, previously, while the transputer environment was being used.

Another interesting aspect of performance was the numerical stability exhibited by the program running under various regimes. Results indicated that numerical stability remained within tolerable limits and compared favourably with the 'control' model which was run on the VAX-4200.

7.2 Performance of Sequential version on the ROOT Transputer

As already discussed, the size of the model and the amount of Workspace which could be loaded onto one Processor were significant limiting factors. To load the Main Sequential program into the ROOT transputer memory, it was necessary to treat all of the on-chip memory as 'external memory'. This meant that the very fast 4k of on-chip RAM could not be utilised for the purposes for which it was designed.

Obviously, this produced a very "slow" configuration but nevertheless it was still 100 times faster than the same size of program running on a 320SX PC without a Maths Co-processor. The first successful run, using this slow configuration, achieved an execution time of 19 minutes for a 24 hour forecast.

The next step in the program distribution procedure was to remove the Interpolation routines from the Main program and to compile them separately. Then at link time, an options file was used to specify that the Interpolation should be loaded first. This guaranteed that the specified interpolation routines would be loaded into fast RAM at run

time, ahead of the slower Main program which would get loaded first by default. This also produced a good improvement in execution time. There was a saving of 2 minutes on the first execution time. It was an encouraging result and showed that the focus of attention on the Interpolation routines was justified.

7.3 Comparison of Execution Times

The performance of the HIRLAM model executed on the transputer network was compared to the performance of a sequential version on a VAX-4200. For illustration purpose, the model performance statistics from the DELL 320SX PC were retained.

The execution times, of the major program routines, obtained on all three platforms is presented in Table 7.1. The program on the DELL-320SX PC took 25 hours to produce a 24 hour forecast. This is clearly of no use to the forecaster whose job is to warn the public of impending severe weather conditions. On the other hand, the VAX-4200 version was produced in 736 seconds. This is just over 12 minutes.

The transputer network execution time for a 24 hour forecast was 844 seconds, 14 minutes approximately. Clearly, it is possible to implement a large scale numerical model on a PC based transputer network. Execution time, with further refinement, could be improved.

TIMING RESULTS FOR FORECAST PROGRAM MODULES

as executed on three different Platforms

Seconds/time step

Forecast Program Modules	VAX-4200	DELL-320SX	Transputers
Dynamic Module	4.121	363.30	3.354
Forward Time Step	0.297	34.90	0.209
SL Non-Linear Terms	1.184	170.60	1.539
Interpolation to Full Levels	0.246	43.84	0.372
Interpolation to Mass Points	0.172	32.04	0.319
Centring	13.720	2044.00	21.250
Computation of Interpolation Fields	1.910	187.00	1.500
Interpolation	13.880	1680.00	18.840
Semi-Lagrangian Module	31.220	4106.00	43.830
Explicit Adjustment	0.938	94.30	0.769
Helmholtz Solver	4.102	385.00	5.144
Implicit Adjustment	0.719	107.00	1.147
TOTAL TIME FOR 24hr Forecast	736 secs	25 hrs.	844 secs.

Table 7.1 Computation Time for various Forecast Modules tested on a VAX-4200, DELL-320SX IBM-Compatible PC and a Transputer Network. Each cell entry in the table is the value obtained for one Time Step Loop. The Total Time to produce a 24hr Forecast is given at the end.

7.4 Parallel Architecture Implications

The programmer in a Parallel Environment has to be aware of the Hardware, Software, Programming Language constructs and Algorithms which are at his/her disposal. Old habits which survive adequately in a sequential environment can be severely punished, performance wise, if carried over to the new parallel environment. Careful consideration must always be given to the various parameters. Adequate programming tools must be developed to ensure accurate programming and an efficient speed of delivery on the part of the Software Engineer[8].

7.4.1 Hardware Implications

The solution of primitive equations of the Model is achieved with a non-linear system of time-dependent partial differential equations. When using the Parallel architecture of the transputer network to implement the solution method, a number of architecture dependent features have to be considered. These are principally:

- * Number of Processors available;
- * Processor Speed;
- * Communication Startup time;
- * Communication Bandwidth;
- * Memory distribution among the processors;
- * Processor Network Topology.

7.4.2 Software and Algorithm Implications

Many factors influence the design of algorithms for execution on Parallel architectures. The factors to be considered in the case of the transputer environment are:

- * the number of task and/or thread creations and synchronisations;
- * the amount of sequential code in each task/thread;
- * the contention for shared common blocks or data structures;
- * the distribution of the workload;
- * the efficiency of the underlying parallel algorithm.

The addition of extra processors into the network offers increased computational capability in a shorter time-frame. We are tempted, indeed encouraged, to increase spatial accuracy by increasing the number of grid points. For example, it is tempting to change from a coarse distribution of one grid point at every 1.5 degrees interval to a fine resolution of 0.5 degrees in both longitude and latitude dimensions. Also increasing the number of vertical levels is desirable.

Improving the resolution implies that many of the computational algorithms have to be broken up and resolved explicitly. It is essential at all times to preserve the concept of energy conservation and the consistency of the system of equations. These principles apply, in particular, to gravity wave drag, convection and turbulence modelling[35].

However, processes which compute radiation and precipitation, for instance, could remain as before. These considerations in themselves impose a limit on how fine a resolution can ultimately be used in forecast models. It also has implications for the maximum number of processors ultimately required to compute such forecasts for dependable accuracy.

7.4.3 Compiler Implications

At the moment there are many compilers on the market offering automatic parallelization of Fortran Code. All of these have limitations because Fortran has an excessive generality as a formal language compared to Pascal or C. Parallelism is not naturally expressible in standard Fortran-77 or earlier versions. However, program statements can quite often be executed in parallel if there is total data independence between the statements. The detection of such independence by compilers means that these compilers themselves must use very complex reasoning algorithms. This in turn makes such compilers difficult to write and expensive to run[9].

OCCAM, on the other hand, is founded on the principles of communicating sequential processes[12] and as such is designed to make parallelism in programs as explicit as possible during the loading stage. However, as experienced during the development of the 'Dynamo Simulation Model' in OCCAM2, on a single T414 transputer, such facilities of themselves are not sufficient. We cannot come from a

"Sequential" background and hope that OCCAM will naturally allow us to express mathematical models of the atmosphere in Parallel. The Algorithms themselves must be re-drafted, a task which leads to a rather involved computer programming effort.

7.5 Parallel Behaviour of the Atmosphere

As we have already seen, the atmosphere itself behaves in a parallel manner, at the physical level. Therefore, in most cases the underlying physical quantities at a molecular level can be expressed using parallel algorithms.

It is from the study of such molecular level activity that the Mathematical structure of these models is initially derived using differential operators. Therefore, on this physical and mathematical level, the parallelism of the atmosphere is "natural". It is only in the coding that they are moulded into a sequential representation.

7.6 Data Flow Computing

Data flow computing is one of the most radical approaches to multiprocessing since there is no concept of serial program structure as defined by Von Neumann[24]. Machine instructions are activated on the arrival of data and its operand. This principal is employed in the OCCAM language. Problems to be resolved can be represented by data flow diagrams(DFDs).

A data flow algorithm is defined by O'Leary and Stewart[49] as a collection of 'instructions' in a directed graph that represents the flow of data between the instructions. The instructions only execute when the data they require has arrived. One of the advantages of the data flow approach is that computational tasks are independent of the relation between the size of the problem and the number of transputers. Another advantage is that tasks are independent of the physical communication structures between transputers.

DFDs are said to be in parallel but at the final stage they are subjected to an unnatural sequential flow for implementation in a conventional sequential language for a conventional Von Neumann Architecture. Therefore, the development of Parallel programs should concentrate on the concept of preserving the "Natural" parallelism of the physical models. Programmers must "think Parallel".

7.7 Speedup and Amdahl's Law

Amdahl's law is one of the most frequently used theorems in parallel and vector computing and it states that any fixed serial component in a code, sets an absolute upper limit to the obtainable speedup[7],[10].

The law states:-

$$\text{Maximum Speedup} = \text{Total CPU Time} / \text{time consumed in the Serial component}$$

and the gain from added processors shows a hyperbolic diminishing returns pattern expressed as:-

$$Speedup = \frac{1}{SEQ + \frac{PAR}{NODES}}$$

where SEQ = the Serial Fraction of the code
 PAR = the Parallelizable Fraction of the code
 NODES = the number of processor Nodes available
 SEQ + PAR = Total Code

We can assume that the sum of the SEQ part and the PAR part represents all of the code.

$$\therefore SEQ + PAR = 1$$

The overall speedup obtainable is also affected by other factors such as:-

- * Overhead in task distribution over the Processor Network at startup time;
- * Fixed synchronisation overhead;
- * Fixed bandwidth;
- * Inherent Sequential nature of elliptic problem solvers which are used in the implicit and semi-implicit time-stepping schemes of meteorological models.

Amdahl's Law assumes that we are always solving the same problem and are not striving for increased precision on a finer grid to use the total computer power available.

7.8 Scaled Speedup

Gustafsson et al. in 1988[11] introduced a better notion which they called Scaled Speedup. It is expressed as follows:-

$$\text{Scaled Speedup} = SEQ + (1 - SEQ) \text{ Nodes}$$

This represents the time a parallel computer uses to solve a problem and divided into the time for one processor to solve it. This figure will grow linearly with the addition of extra

processors to the topology, but, it ignores the likely difficulties of fitting an entire problem onto a single node initially.

In general, the deterioration of speedup due to fixed overhead such as those mentioned earlier, can be overcome by keeping the relative granularity of the algorithm high. However, the overhead of distributing work to a large number of processors is significant. For instance, during the implementation of the HIRLAM model on the transputer network, a large amount of information was gathered globally from transputers and redistributed to them again. This introduced a logarithmic serial complexity which ironically would grow with the number of processors used. Therefore, the scaled speedup for elliptic problems is controlled and expressed by:-

$$\text{Scaled Speedup} = \text{SEQ} + (1 - \text{SEQ})\text{NODES}/\text{Log}(\text{NODES})$$

The obtainable speedup rate can deviate from the ideal achievable as the number of transputers introduced increases.

This is attributable to:-

- (a) inter-processor communication for data swapping;
- (b) uneven load balancing of the data among the processors;
- (c) host interface overhead.

In the specific implementation carried out for this research project, on the network of eleven transputers, there was a limit imposed on the number of vertical levels which could be modelled. The addition of extra processors would allow for a corresponding increase in the vertical resolution. Since the interpolation algorithms are the most compute intensive program segments, further additional processors would contribute significantly to the overall speedup of forecast execution time.

7.9 Summary

The hardware and software implications for the implementation of applications, such as the HIRLAM model, in a parallel environment have been considered in this chapter. The performance timings of the HIRLAM model using three platforms, VAX-4200, Transputers and DELL 320SX PC, are presented in tabular form for ease of comparison. The timing results for the various critical program modules show that the performance of the transputer version is on a par with that of the VAX-4200.

The performance measurement of a parallel distributed architecture, offered by the transputer network, involves the evaluation of hardware, software and algorithms used. The efficiency of the compiler is an important consideration. The move away from sequential algorithms to parallel algorithms demands a new approach to the development of algorithms e.g. data flow computing.

Finally, the speedup of any program in a parallel environment is dependent upon the speed of performance of its slowest sequential part as outlined in Amdahl's Law and Scaled Speedup. In this case study the addition of extra processors would contribute to the overall accuracy and speedup of the HIRLAM model.

Chapter 8

HIRLAM Program Execution on Transputers

8.1 Introduction

This chapter describes the HIRLAM program which was implemented on the transputer network. The general flow of the computation algorithms remain the same as those used with the sequential program on the VAX-4200. The main exception is in the implementation of the Interpolation procedures which are now designed to run in parallel. Although the routines which compute the physical parameterisations have been moved to a specific transputer their computation algorithms still remain sequential.

A series of structure diagrams which are intended to pictorially describe the system, as implemented on the transputers, are included. The size of the program is outlined and this is followed by a discussion on its implications for memory management.

8.2 HIRLAM Code Characteristics

The Fortran code size for the main program modules consists of about:

10,000 lines for the "Model" Module;

4,500 lines for the "Update" Module;

1,000 lines for the "FFT" Modules;

500 lines for the "CRAY- type Function Calls" and

4,500 lines for the "Physics" Module

==> 20,500 lines of code in total

8.3 Memory Requirements

The code is highly vectorised with a typical vector length of 38 longitude and 34 latitude grid points. The total data area used by the COMMON blocks requires a memory capacity of 1 Mbyte for the main program on the ROOT transputer and a capacity of 1.5 MBytes for the 'physics' on the P001 transputer.

This means that a typical problem of $38 * 34 = 1292$ grid cells, with say, 16 vertical levels, would require about 8 MBytes of memory if the program and its data are to remain memory resident throughout. This space requirement cannot be accommodated by the memory of a single Transputer. Therefore, a number of temporary files are created during each run. These are placed on a virtual disk, D:, on the Olivetti 486 PC, in order to make use of the HOST memory instead of reading/writing to the hard disk.

So, typically the 3D fields are split into 2D slices known as slabs, with respect to the x-axis of the domain under computation. Using a system of subroutine calls the program reads the slabs into the common workspace as required.

8.4 Program Initialisation Phase

The Main Program controls the flow of the model and it begins by executing several forecast initialisation stages, such as reading the forecast control variables and the analysis fields to which the forecast is applied. The analysis for 0000z on 25th August 1986 is used as an input data set. This data is used as a case study for 'Hurricane Charlie' and is read from a specially prepared ASCII file.

The analysis file contains values for surface geopotential, dynamic prognostic variables and different surface fields which are used by the physical parameterisation routines.

The size of the integration area is decided at Compile time. It is determined by the parameter file which specifies the number of the Latitude and Longitude gridpoints and the number of vertical levels. The actual geographical position of the integration area is described in the data description record of the analysis file which is read during the model initialisation phase.

The initialisation phase is completed by defining constants in the semi-implicit time stepping scheme. The final initialisation task is to compute the actual time stepping increments required to produce at a 24 hour forecast.

The initialisation is followed by the integration which is performed using a loop over all the time steps as selected during the initialisation. During the Model run the statistics routine is called at certain time steps to analyse the current solution and to print samples of the solution to an output file. All of the above program steps are executed on the basis of 2D slices or slabs. Each 2D slice contains extra dummy space to hold the boundary conditions at each boundary. The dummy space holds copies of the appropriate values at the appropriate boundary.

8.5 Main Computational Phases

The HIRLAM Model comprises two main computational parts as discussed in Chapter 2. The Main part uses approximately 25% CPU time and remaining routines on the transputer network use 60% CPU time. A Structure Diagram for the Main model is shown in Figure 8.1. This is followed by a Structure Diagram for the 'physics' modules in Figure 8.2. The Semi-Lagrangian Scheme is illustrated in Figure 8.3.

In the Main part the computations are performed layer by layer, by scanning through all the horizontal points in the innermost loops, as shown in Figure 8.4. However, in the "Physics" subroutines, so called slab areas are used and these include a few, (3 in this case), latitude lines each. This means that the "Physics" subroutines are called several times during one time-step to complete the computations for all slab-areas, as shown in Figure 8.5.

Structure Diagram for HIRLAM Model Implemented on Transputer Network

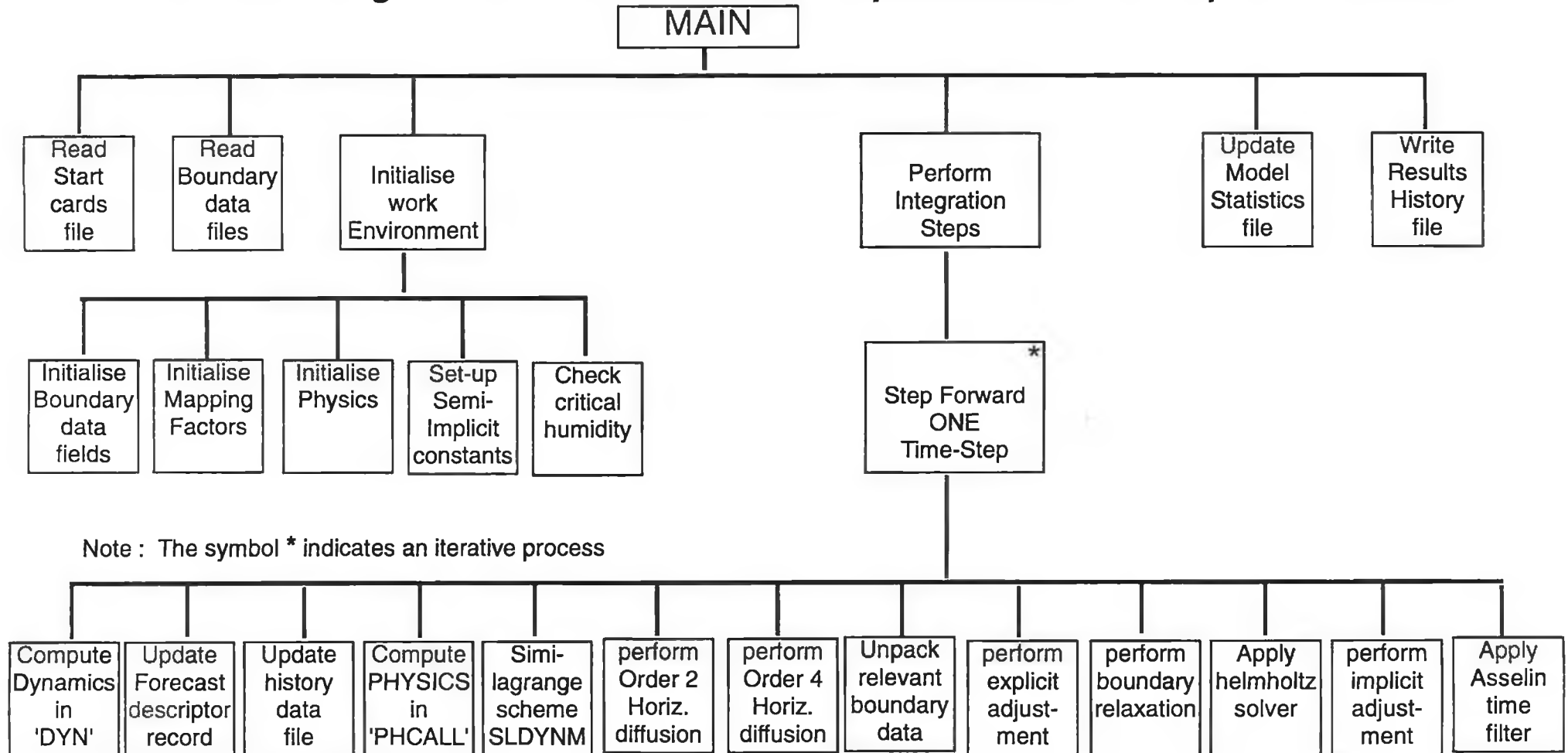


Figure 8.1 : Main Structure Diagram for Hirlam Model

Structure Diagram for "PHYSICS " portion of the HIRLAM Model

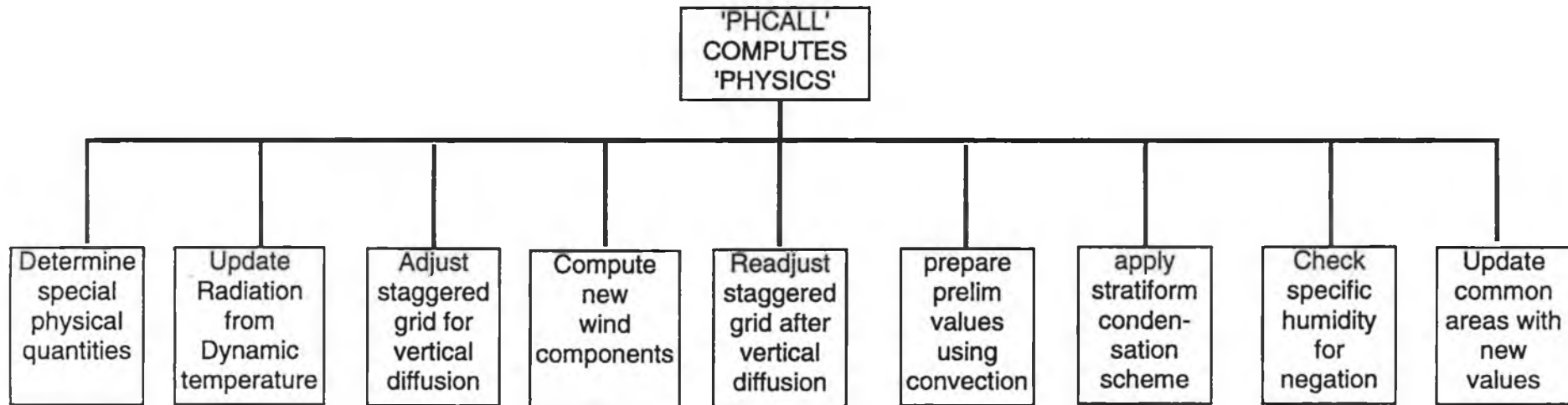


Figure 8.2 : Structure Diagram for 'Physics Modules' which were PLACED on the P001 Transputer.

**Top Level
Structure Diagram for Semi-Lagrangian Scheme**

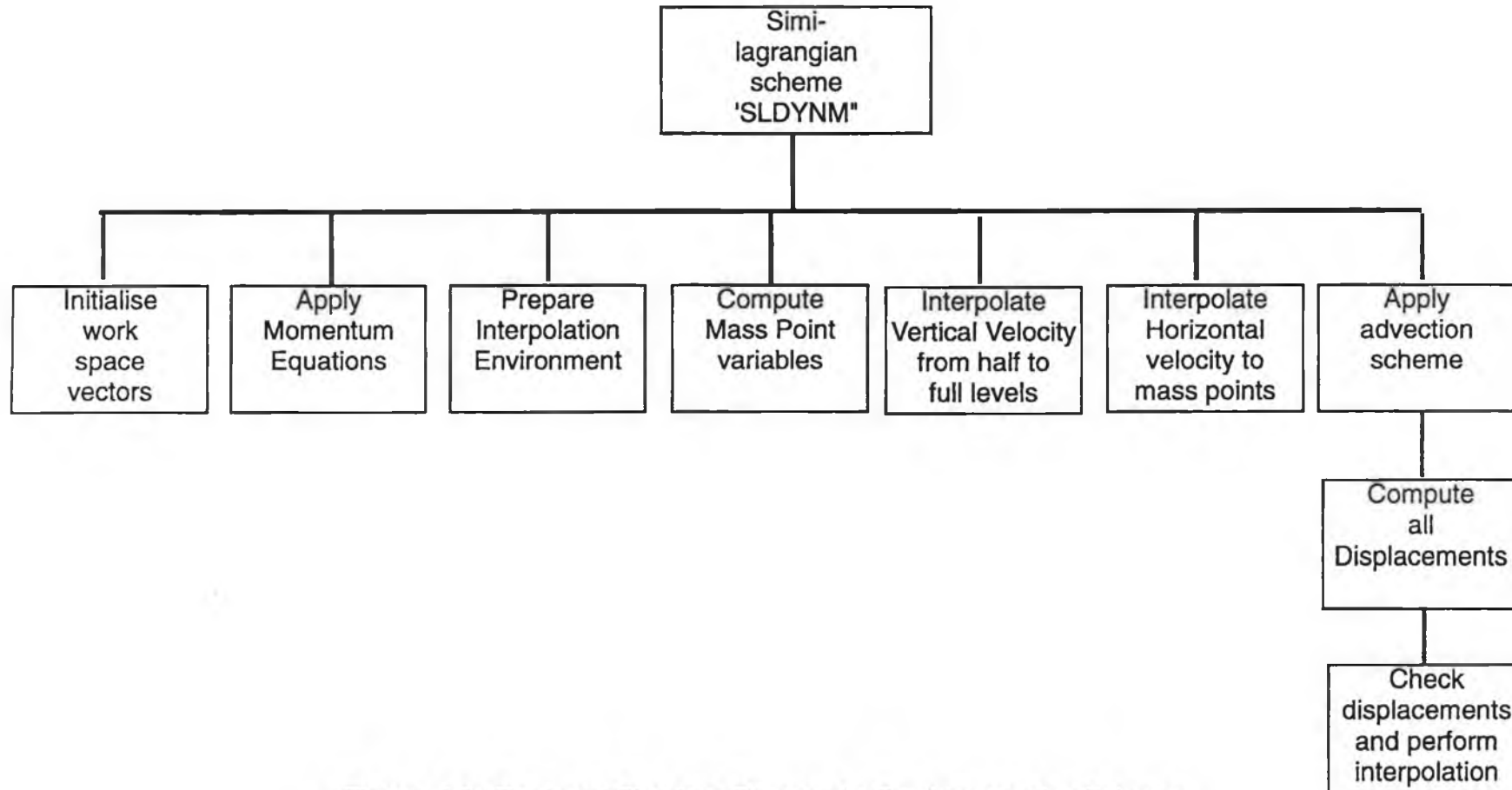


Figure 8.3 : Top Level Structure Diagram for Semi-Lagrangian Scheme.

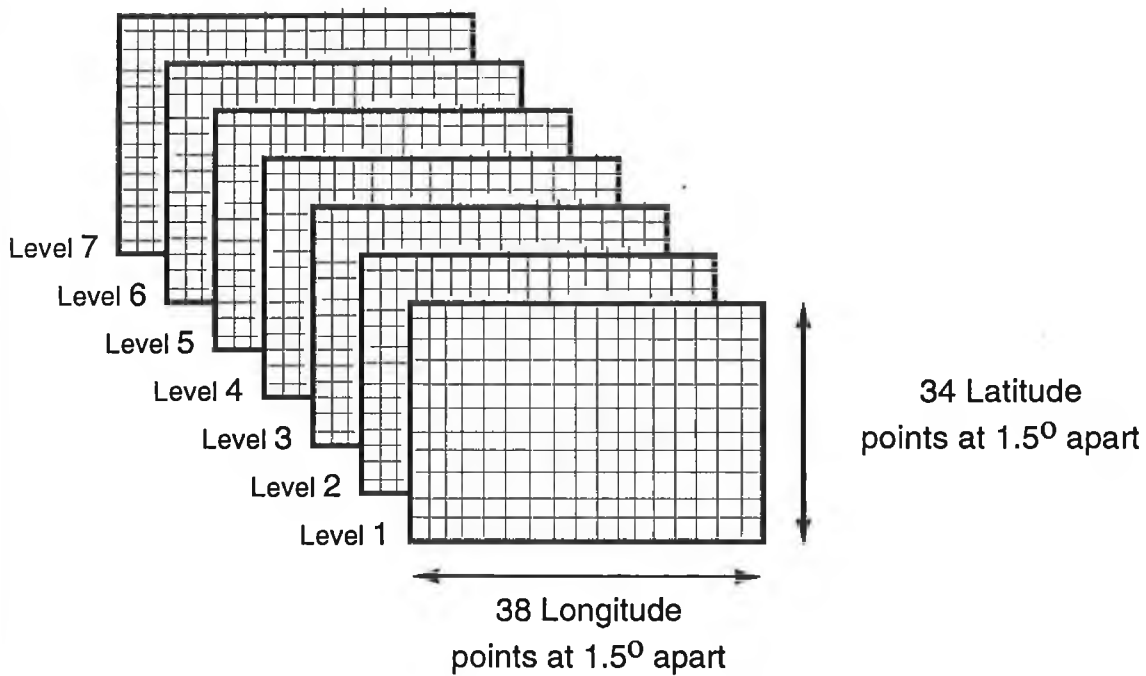


Figure 8.4 7-Layer Formation used by the 'Dynamics' part of the Main Program running on the ROOT Transputer.

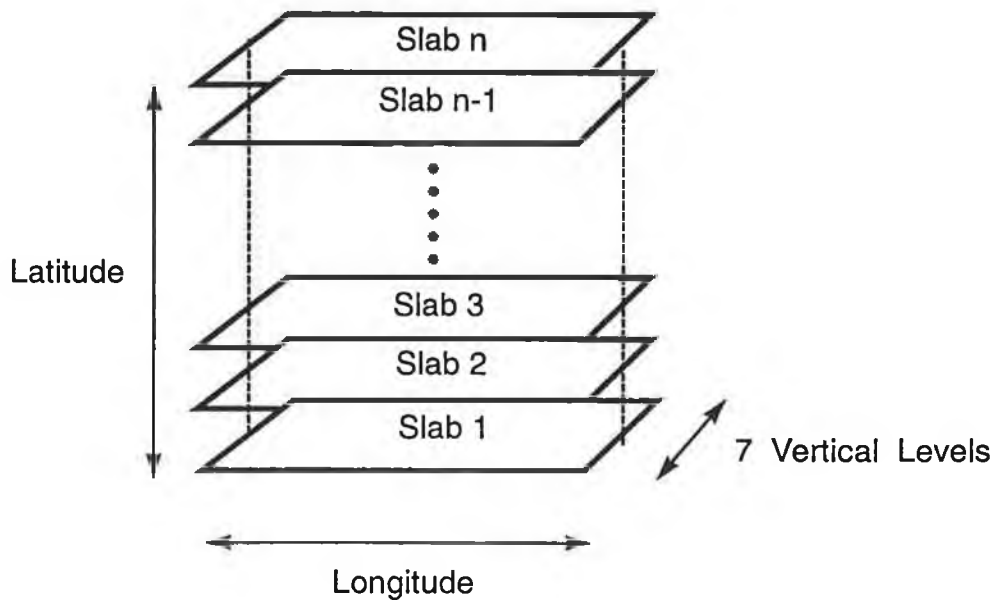


Figure 8.5 Slab Formation used by the 'Physics' routines on a two-dimensional Lat-Long mesh.

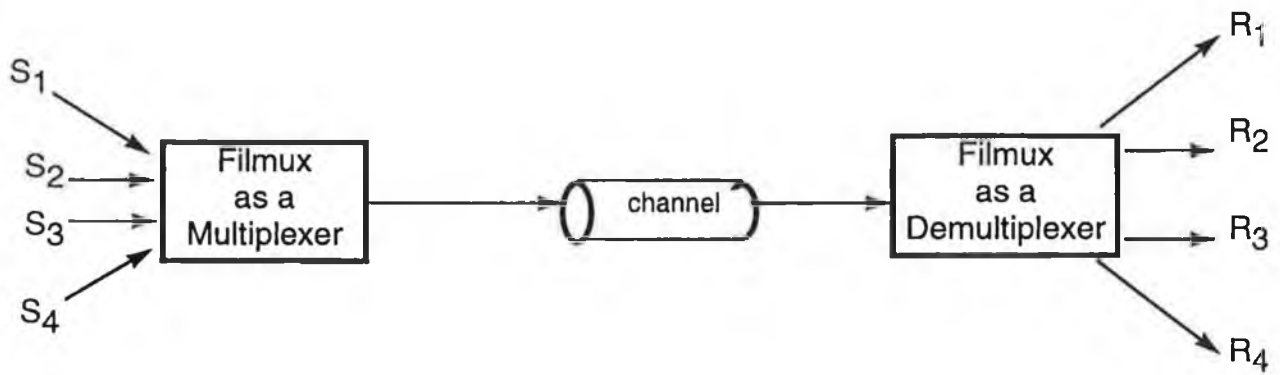


Figure 8.6 Use of Multiplexer / Demultiplexer technique to accomodate several virtual channels over one physical channel

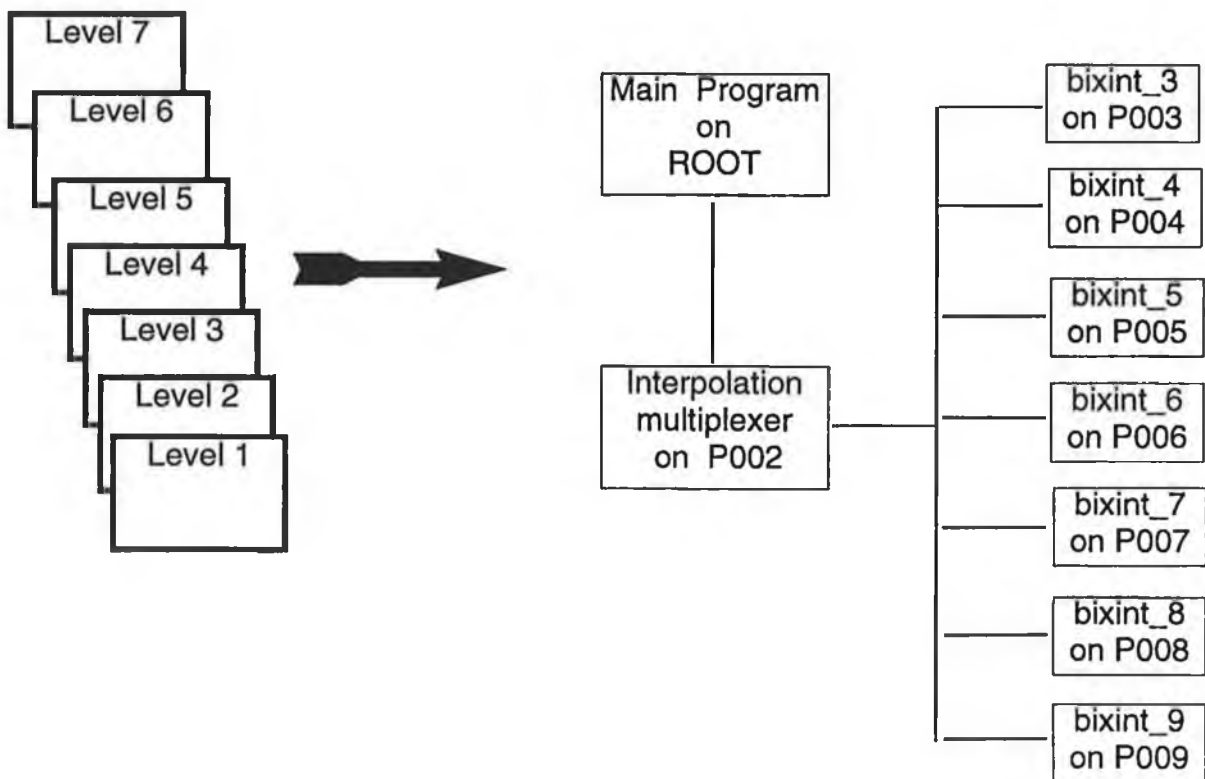


Figure 8.7 Using the Software Multiplexer technique to project the vertical interpolation onto 7 transputers.

8.5.1 Main Integration Phase

The dynamic and explicit tendencies are computed in subroutine 'DYN' which is called several times during each time step. It is called once to compute the time tendency of the surface pressure and 'KLEV' times to compute the time tendencies for the wind components (u,v), temperature and specific humidity. The Vertical integration of the Continuity Equation(2.8) and the Hydrostatic Equation(2.7), is done in a stepwise manner from level to level.

8.5.2 Semi-Lagrangian Phase

The subroutine 'SLDYNM' performs the interpolation of its arguments along their trajectories. It evaluates the non-linear and linear terms, in accordance with Robert's scheme[31],[54], by interpolating between gridpoints. Non-linear terms are arranged between departure and arrival points and are computed at central time. The interpolation computation is now performed by a special task, "Interpolation multiplexer", as shown in Figure 8.7. This module, located on P002 transputer, controls the message passing to and from a sub-network of 7 transputers, each supporting an identical copy of the Interpolation modules.

A structure diagram for this semi-Lagrangian phase is shown in Figure 8.4. This suite of routines is designed to cope with two and three time-level schemes. Trajectories are computed by the sub-network of slave interpolation tasks, for all of the 7 layers of the model atmosphere at the same time.

Another algorithm was developed to perform the interpolation in parallel. This algorithm used the 'Farm of Worker Tasks' technique and is depicted in Figure 4.2. But, as discussed in earlier chapters, it was not a suitable algorithm due to the memory space limitations imposed by the flood-fill configuration utility program.

8.5.3 Computation of the Physical Parameters

The computations associated with the time tendencies due to the physical parameterisation are now performed on P001 transputer by a routine called 'PHCALL'. This involves quite a lot of data exchange between the ROOT transputer and the P001 transputer. However, due to the high communications bandwidth provided by the inter processor links, the high volume of message passing does not appear to affect overall performance.

The vertical diffusion equations are solved implicitly. As we have already seen, it is convenient to do part of the time stepping in 'PHCALL' routine. On completion of 'PHCALL' all the model variables are updated to the value of the next time step in the LEAPFROG scheme.

When working on the physical parameterisation the integration area is divided into a number of sub-areas called "slab-areas". Each slab-area consists of all the gridpoints within a certain number of neighbouring latitude lines. The formula for the computation of the slab parameter, 'MSLAB' is presented in Chapter 5, Section 5.4.

All the physical processes within one slab-area are completed before moving on to the next slab-area. In this way the computations of neighbouring slab-areas are independent, except for, the vertical diffusion of the North-South wind component. The vertical diffusion equations are treated in MASS POINTS and an interpolation between mass points is done in the routine 'VDIFFX', which is called before and after the vertical diffusion, which in turn is performed by the routine 'VDIFF'.

The inherent dependence associated with the vertical diffusion algorithm, mentioned above, makes it difficult to separate out the "Physics" routines from the Main program entirely. This is because the program was originally designed so that both processes use a 'shared common block'. It is necessary to update the contents of this 'shared common block' by use of a special message passing technique, before and after calls to 'PHCALL'.

8.5.4 Explicit Leapfrog Time Stepping Phase

The Explicit Leapfrog Time Stepping phase performs the same type of computations as described in (8.5.2). This scheme is more suitable to the distributed memory configuration of transputer networks. Fortunately, the original algorithm, for the HIRLAM program, allowed for the Explicit Leapfrog Time Stepping to be selected by the programmer at compile time.

The computations are organised layer by layer starting from the lowest model layer. The changes due to Horizontal diffusion are computed and added in by the subroutine 'HDIFF'. The first part of the semi-implicit adjustment scheme is done in subroutine 'EXPADJ'. This also computes the right hand side (RHS) of the Helmholtz Equations(2.16), which require a set of Boundary values for the computation of the divergence. The necessary Boundary values are retrieved from the temporary work files, located during the initialisation phase, on the MS-DOS virtual disk D:, before subroutine 'EXPADJ' is called. After the explicit adjustment phase, all the dynamic variables are Boundary relaxed. This is done in subroutine 'BDMAST'.

8.5.5 Resolution of Helmholtz Equations Phase

After the completion of explicit adjustment for all levels, the Helmholtz Equations(2.16) are finally solved by calling the subroutine 'HHSOLV'. In the next stage of the integration procedure, the remaining implicit adjustment terms are computed by the subroutine 'IMPADJ' and this completes the corrections to the variables. The final part of each time step is to apply the ASSELIN time filters[16],[32] and this is done by routine 'TIMFIL'.

8.6 Input Test data

In chapter 6 the interpolation methods used in reducing the original test data set from 16 vertical levels to 7 vertical levels were described. The interpolation program used for this purpose is included in Appendix A. After this modification phase the files were reduced in the amount of data they contained but were physically larger because they had to be read in ASCII format by the current version of the Parallel Fortran compiler.

The data consisted of one analysis file valid for 0000z on 25th August 1986. There were five boundary files also. These were valid for the following times: 25/0000z, 25/0600z, 25/1200z, 25/1800z and 26/0000z.

The analysis and boundary data were extracted from archive records, at ECMWF, and written in a format which could be read by the HIRLAM program. The basic principle is that each file represents one 'model state' in time. The files are laid out in a succession of 'vertical slabs', normally arranged in North-South order. Each 'vertical slab' record contains all the model parameters at all levels for one line of latitude as shown in Figure 8.5. The grid points in a slab are organised in rows going from west to east, with one gridpoint for each parameter at each level.

The data in each file is preceded by a header record known as a Data Description Record (DDR). The relative position of each row of data or each field is completely determined by a pair of pointers in the header record.

The analysis file is read during the initialisation phase and the values for all of the field variables are extracted and used to set up the 'initial state' of the model atmosphere.

The forecast file produced from each model run uses an identical file format. This enables the standard post-processing module, shown in Figure 2.0, to read the file directly without modification.

8.6.1 Boundary Scheme Used

The HIRLAM model requires lateral boundary fields. These fields are extracted from the relevant ECMWF forecast files. In the boundary relaxation zone, as shown in Figure 2.3, which surrounds the integration area, all the prediction variables are relaxed towards their prescribed boundary values. The boundary values themselves are interpolated from the Analysis data or from the ECMWF forecast data. The model forecast values are mixed with the boundary field values within the boundary relaxation zone. This is carried out by using a boundary weighting factor[16].

Boundary fields are interpolated horizontally and vertically during each time step, by using a bilinear scheme[1]. The time interpolation of the boundary fields is linear between boundary write-up times which were at 6 hourly intervals. The specification of good lateral boundary data is critical to the eventual forecast produced. This is also true for the quality of the analysis data used to set up the initial state of the model.

If the integration area is small the central area will become 'contaminated' by the lateral boundary data very quickly. Therefore, if the quality of the boundary data is poor, (i.e. analysis and boundary data times do not coincide), an integration area must be chosen large enough to ensure that relaxation errors do not influence the central forecast area of interest.

The data used for this case study was 'good' data which had already been used for other research experiments. Therefore, it was expected that the resultant forecast obtained, by using the 'good' data, would closely resemble the actual synoptic weather conditions prevailing at the time for which the forecast was valid.

8.7 Summary

The version of the HIRLAM model implemented on the transputer network has been described, at a global level, in this chapter. The accompanying structure diagrams are intended to give the reader an overall view of the main computation phases in the implementation.

The memory requirements of the program are very substantial. The on-chip memory on each transputer is STATIC and once it is allocated it cannot be deallocated in Parallel Fortran. To overcome this difficulty it was necessary to utilise a virtual disk in the extended memory of the HOST PC. Without this approach it would not have been possible to accommodate the program in its present form on the transputer network.

The chapter continues with a discussion on the main phases of the program and concludes with a description of the analysis and lateral boundary data files used with this case study.

Chapter 9

Using HIRLAM to Model a Severe Storm

9.1 Introduction

A severe storm known as Hurricane Charlie, occurred over Ireland on Monday 25th August 1986. The storm developed as an offshoot of a hurricane which originated off South Carolina in mid August 1986. It was one of the worst storms in living memory [28].

In Ireland a number of people were killed in accidents related to the storm. Widespread flooding occurred in many areas of Ireland resulting in considerable water damage to property. Insurance claims, resulting from the storm damage, were quite substantial. The Irish Government introduced a rescue package for the most severely hit areas.

Hurricane Charlie's documented weather conditions were selected as the Test Case for running the HIRLAM model on a Transputer Network. The purpose of the case study was to explore the ability of the HIRLAM model running on a Transputer Network to predict the behaviour of an intense storm. The documented data was used as input test data.

In order to highlight the path taken by Hurricane Charlie an outline map of the Northern Hemisphere was used to plot its journey. Its position was plotted on the map at noon each day from the 16th to 27th August 1986 as illustrated in Figure 9.1.

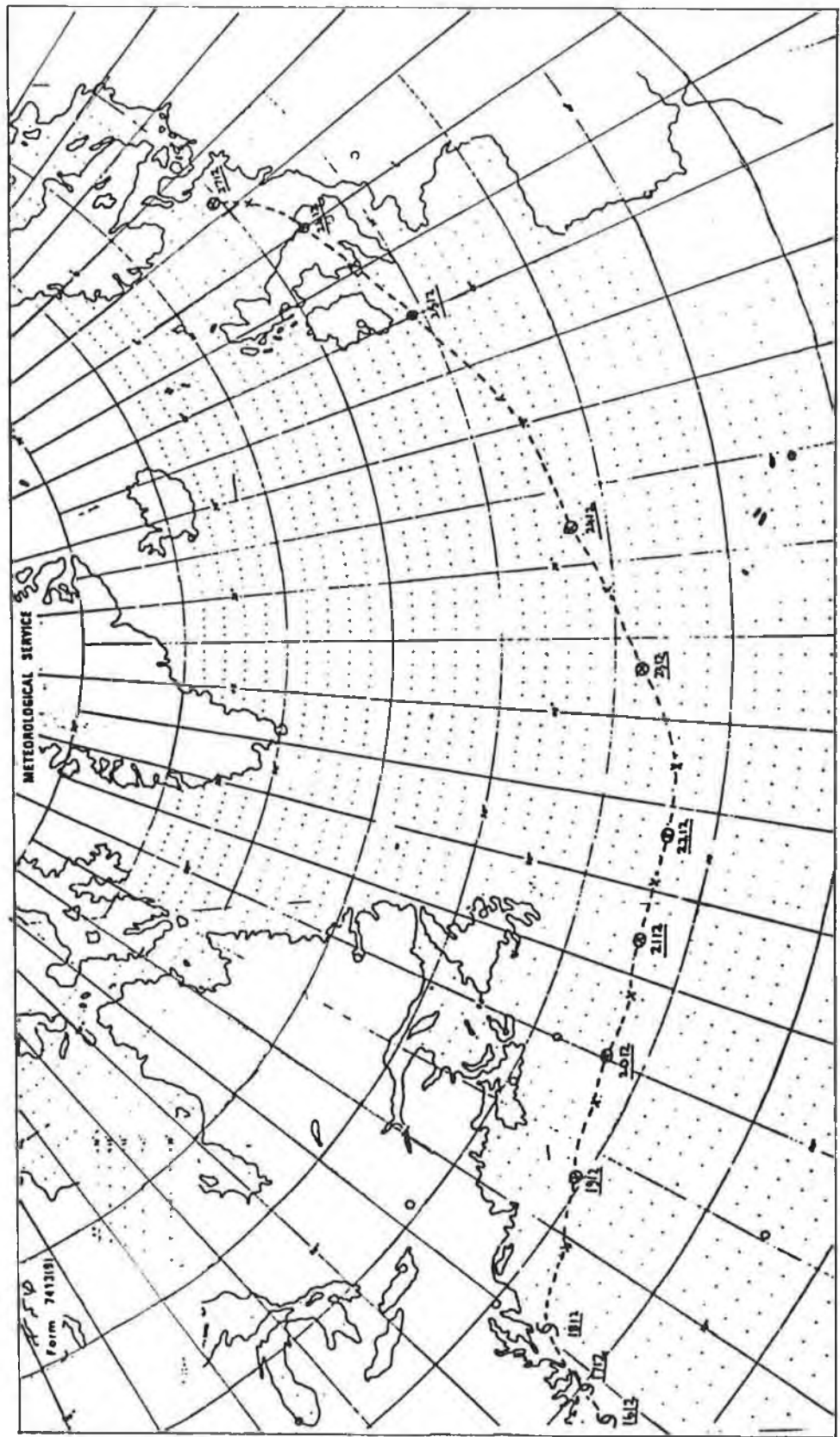


Figure 9.1 : Path travelled by 'Hurricane Charlie' from its inception to the point where it filled over the North Sea.

9.2 Development of Hurricane Charlie

Hurricane Charlie began off the eastern seaboard of the United States of America on the 15th August 1986. During the following days it initially moved north and then northeast. It was characterised by torrential rain and storm force winds. Figure 9.1 shows it positioned directly south of Newfoundland at 12 noon, on the 20th August. This depression crossed the Atlantic between the 20th and 26th August as shown in figure 9.1.

On 23rd August, the original depression split in two parts. One of these parts deepened rapidly and began to move northeastwards. On Sunday, 24th August, at 12 noon it was about 1200Km west-southwest of the Kerry coast. All over Ireland Sunday was a clear sunny day and people were unaware of the rapidly approaching storm.

9.3 Selection of Initial Data

The analysis of the actual weather conditions for 0000z on Monday, 25th August, was selected as the initial data for input to the HIRLAM Model. A plot of the analyzed surface pressure for that time is shown in Figure 9.2. At that stage the storm was 600 Km southwest of Ireland. It had a central pressure of 1003 hecto-Pascals (hPa). Ireland and the United Kingdom are shown with a slack south westerly airflow, with high pressure areas of 1021 hPa to the north of the area and of 1022 hPa over the Bay of Biscay.

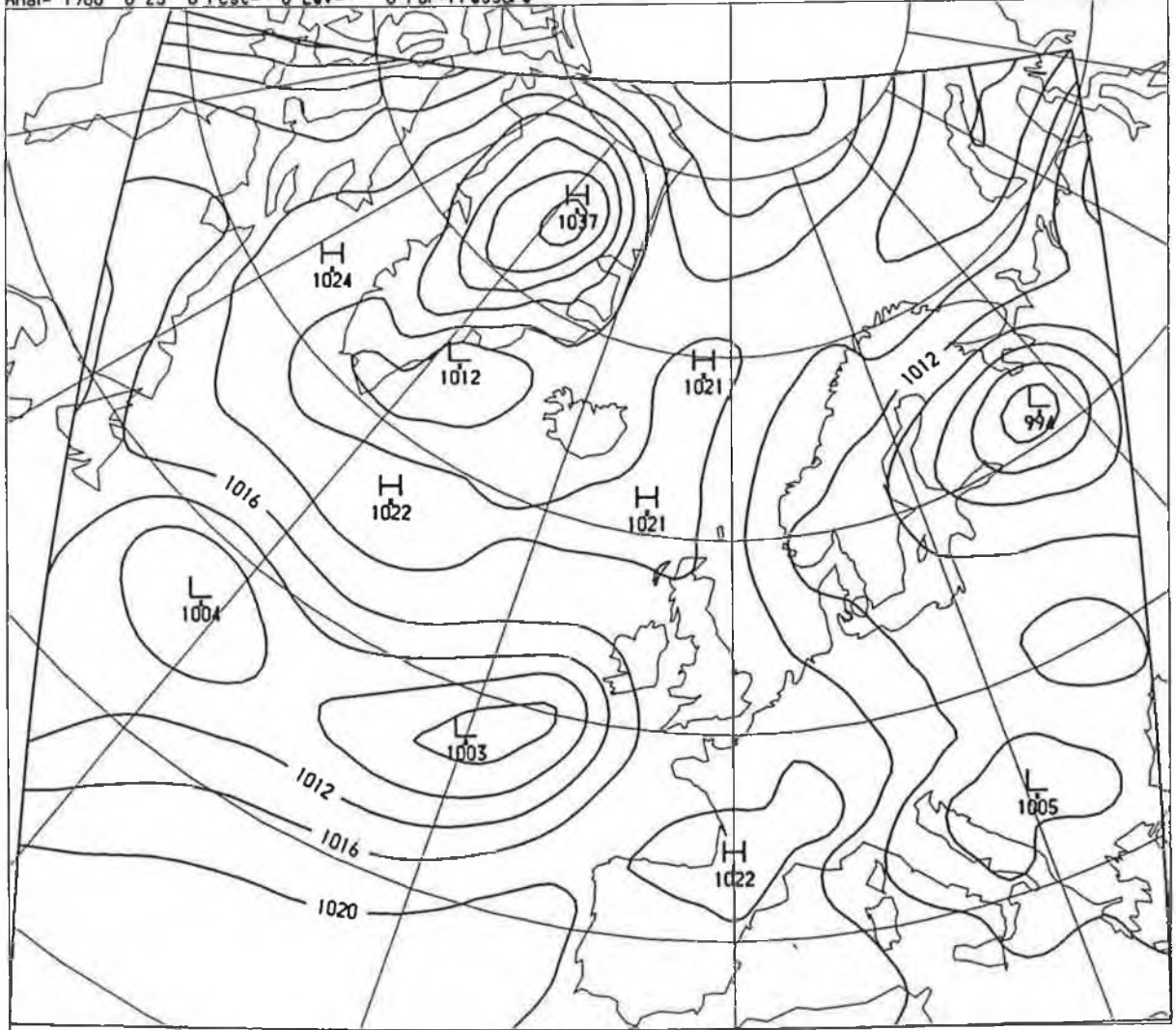


Figure 9.2 : Analysis of Surface Pressure valid at 0000 hours on Monday 25th August 1986.

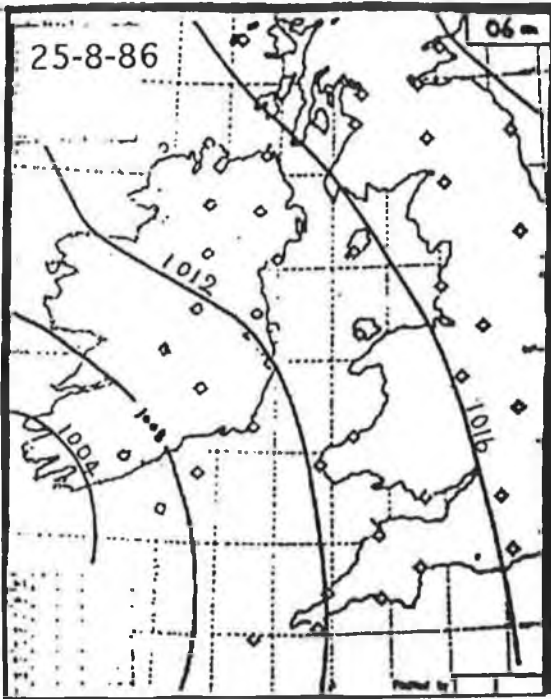


Fig 9.3 (a) : Surface pressure pattern at 0600 hours on 25-8-86

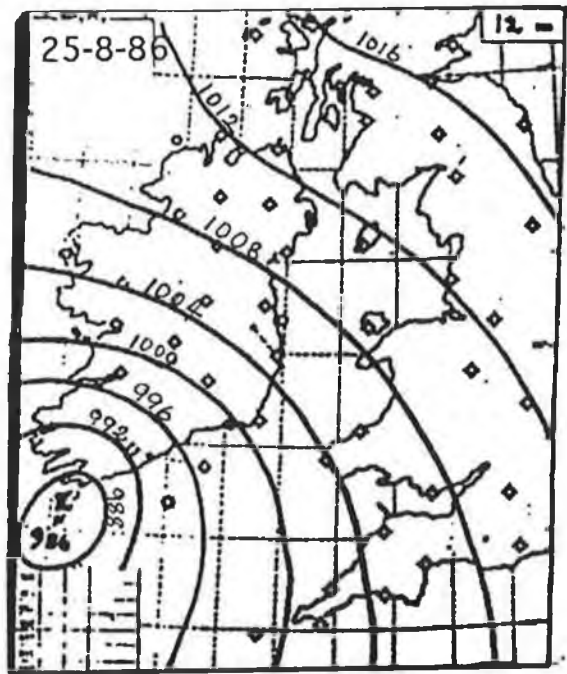


Fig 9.3 (b) : Surface pressure pattern at 1200 hours on 25-8-86

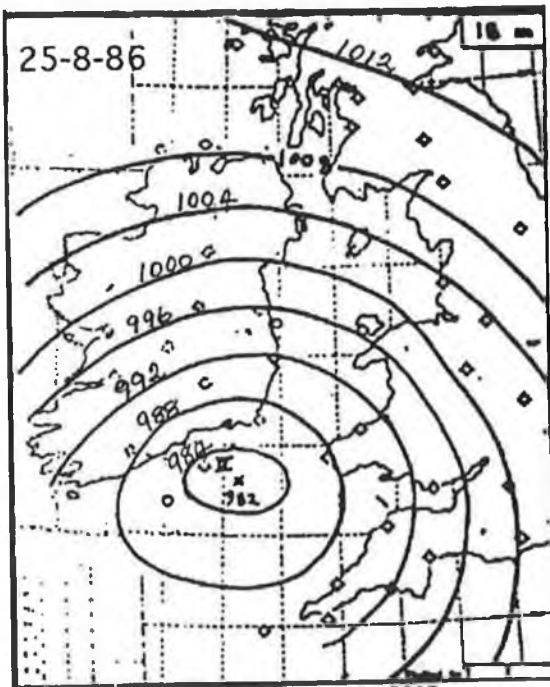


Fig 9.3 (c) : Pressure pattern at 1800Z

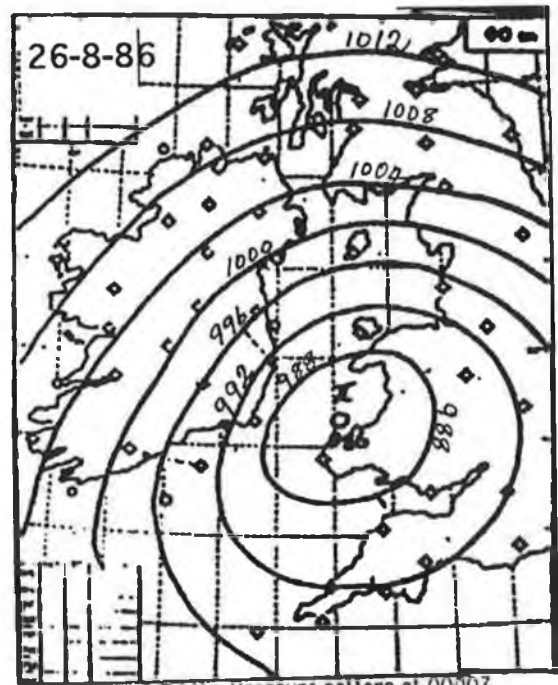


Fig 9.3 (d) : Pressure pattern at 0000Z on 26-Aug-1986

Figure 9.3 : Surface Pressure Pattern over Ireland taken at six hourly intervals on 25th August 1986.

9.4 The Storm Over Ireland

During Monday, 25th August, the storm moved very rapidly in an east-north-east direction. The centre of depression passed the south of Ireland to a position just southwest of Wales. Plots of the surface analysis of the atmospheric pressure for hours 0600z, 1200z, 1800z, on Monday 25th and 0000z on Tuesday 26th are shown in Figure 9.3.

These plots show that the depression deepened rapidly over Ireland. For example, at 0600z the pressure over south west Kerry was 1004 hPa and near the Isle of Man the pressure reading was 1016 hPa. By 1200z the pressure has decreased to 988 hPa over the south west of Kerry while at the Isle of Man pressure had decreased to 1012 hPa. This rapid change in atmospheric pressure over the south west of Ireland during that six hour period is characteristic of a very strong storm gradient.

There was heavy rain over the southwest of Ireland during the morning with 84mm recorded at Roches Point in County Cork. The wind direction over the east coast was particularly changeable during the day. The plot for 1800z shows that the centre of the depression had moved to a position just south of Waterford with a pressure centre reading of 982 hPa. At that stage the pressure gradient had become steeper with a difference of 24 hPa between the north east and south east costal areas.

During the following six hours the depression had moved in to the south west of Wales with a low pressure reading of

986 hPa at the centre. The winds were still storm force in nature at 0000z on the 26th August.

The wind backed from a southeast direction at 0600z , to the east in the afternoon and the northeast by evening. Dublin Airport reported gale force winds for several hours. During the afternoon and evening there were outbreaks of torrential rain. Over the 24 hour period from 0900z on 25th August to 0900z on 26th August 1986 many parts of the east and southeast had over 100mm of rain.

Indeed, over the Wicklow Mountains the rainfall ranged between 150mm and 200mm. This was caused by an east to northeast airflow which strongly interacted with the Wicklow Mountains. A pronounced orographic enhancement[52] of rainfall in the catchments of south Dublin resulted [28].

During Tuesday, 26th August, the storm continued to track across England. On 27th August, it had moved out over the North Sea as shown in Figure 9.1. The centre of the depression remained over the North Seas for several days and gradually abated.

9.5 Using HIRLAM to Forecast the Storm

The HIRLAM model was used to carry out two 24-hour forecasts on the analysis data of 0000z Monday 25th August. The first forecast was run on a VAX-4200. The second forecast was run on the Transputer Network. This approach provided the opportunity to make a range of comparisons between the respective implementations.

9.5.1 Grid Spacing

Both implementations used a grid spacing of 1.5 degrees. This provided a coarse grid resolution of 150Km between gridpoints. The sequential version of the model, which was run on the VAX-4200, used 16 vertical levels. The parallel version, run on the Transputer Network, used 7 vertical levels.

In order to refine the grid to a resolution of 0.25 degrees or 25 Km between gridpoints, it would have been necessary to increase the transputer computing capacity by an order of six. This could be achieved with the addition of many more transputers to the Network. Experiments carried out by the Irish Meteorological Service have shown that a fine grid is most desirable in order to accurately represent the orography [28].

In the case of the "Charlie" storm, the presence of the Wicklow Mountains played a crucial role in the enhancement of the precipitation over the Dublin Region. This orographic effect, characterised by the forced uplifting of moist air, resulted in larger rainfall amounts to the windward side of the high ground of the Wicklow mountains. The severity of the storm within such a confined local area can only be detected and predicted by high resolution Limited Area Models using a fine grid resolution.

9.6 Using HIRLAM to Forecast Surface Pressure

The plots for predicted surface pressure from the VAX-4200 are shown in Figure 9.4(a). Plots of the surface pressure, from the Transputer, are shown in Figure 9.4(b). In each case the plotted charts are valid for 0000z on the 26th August 1986. The output from the VAX-4200 throughout this case study is treated as the control forecast against which the accuracy of the output from the transputer is discussed. The contour interval used with each of the pressure plots is 4 hPa.

In each of these plots the atmospheric pressure over Iceland is predicted as 1018 hPa. The European land mass surface pressure is similar in both cases with a high of 1019 hPa forecast for the area north of Italy. However, in the case of the VAX-4200 output the eye of the depression is shown over Lands End with a value of 988 hPa while the transputer returns a value of 994 hPa. While their depression values differed they predicted the eye of the depression for the same location. The intensity of the depression can be clearly seen by the closeness of the isobars[52] marking the surface pressure gradient over Ireland and the United Kingdom.

In reality the eye of the storm was observed over south west Wales with a central depression value of 986 hPa as shown in Figure 9.3(d). Clearly, the VAX-4200 output is closer to the recorded values and differs by 2 hPa. The transputer output differs by 8 hPa. This difference is partly attributable to coarseness of the grid in the vertical direction. Only 7 vertical levels were used in the transputer version while the VAX-4200 used 16 levels.

The centre of the storm in each case could have been more accurately predicted if a finer horizontal grid mesh were used with the model (i.e. 110 by 100 grid points instead of 38 by 34).

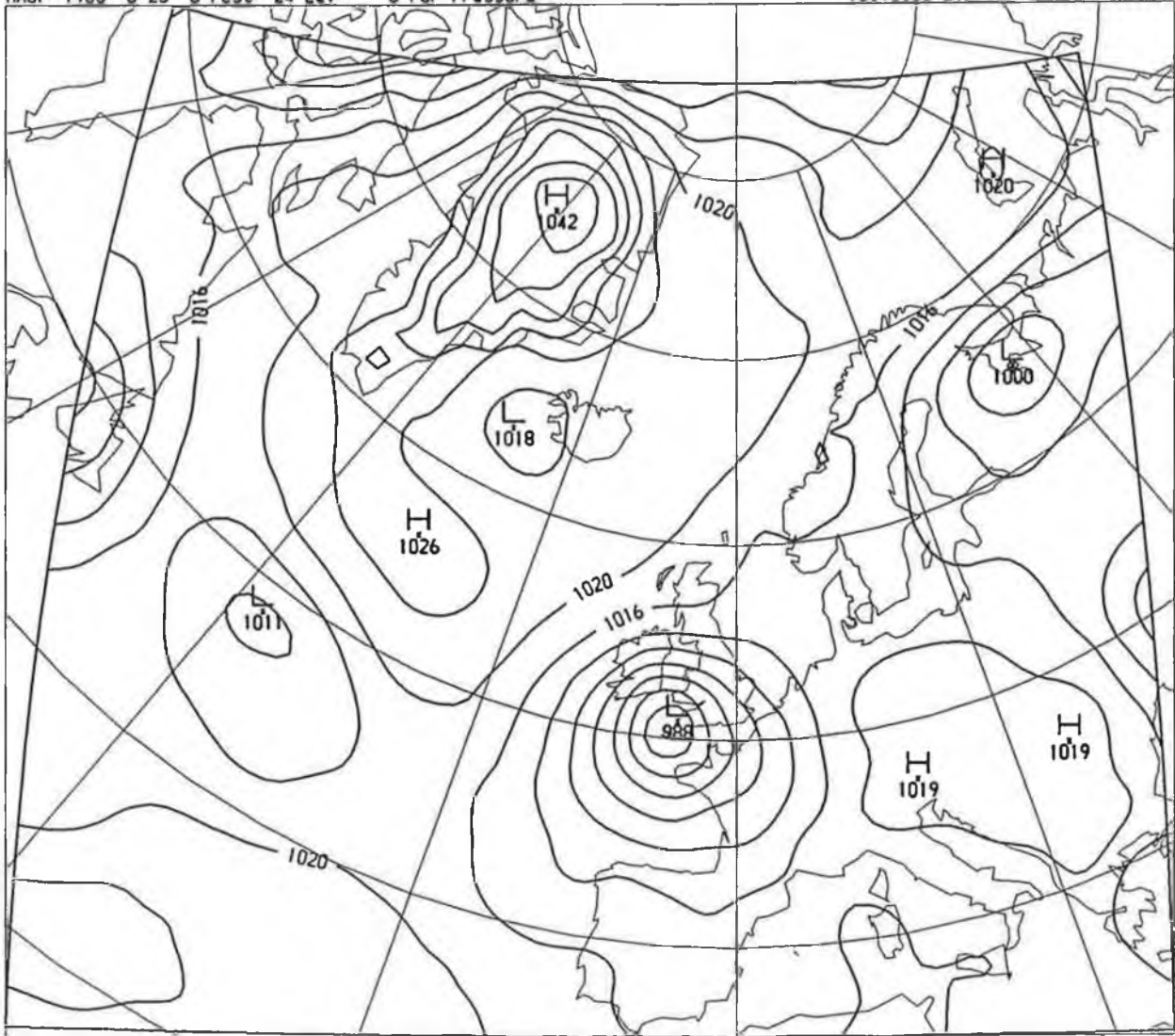


Figure 9.4(a) : Forecast of Surface Pressure valid at 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

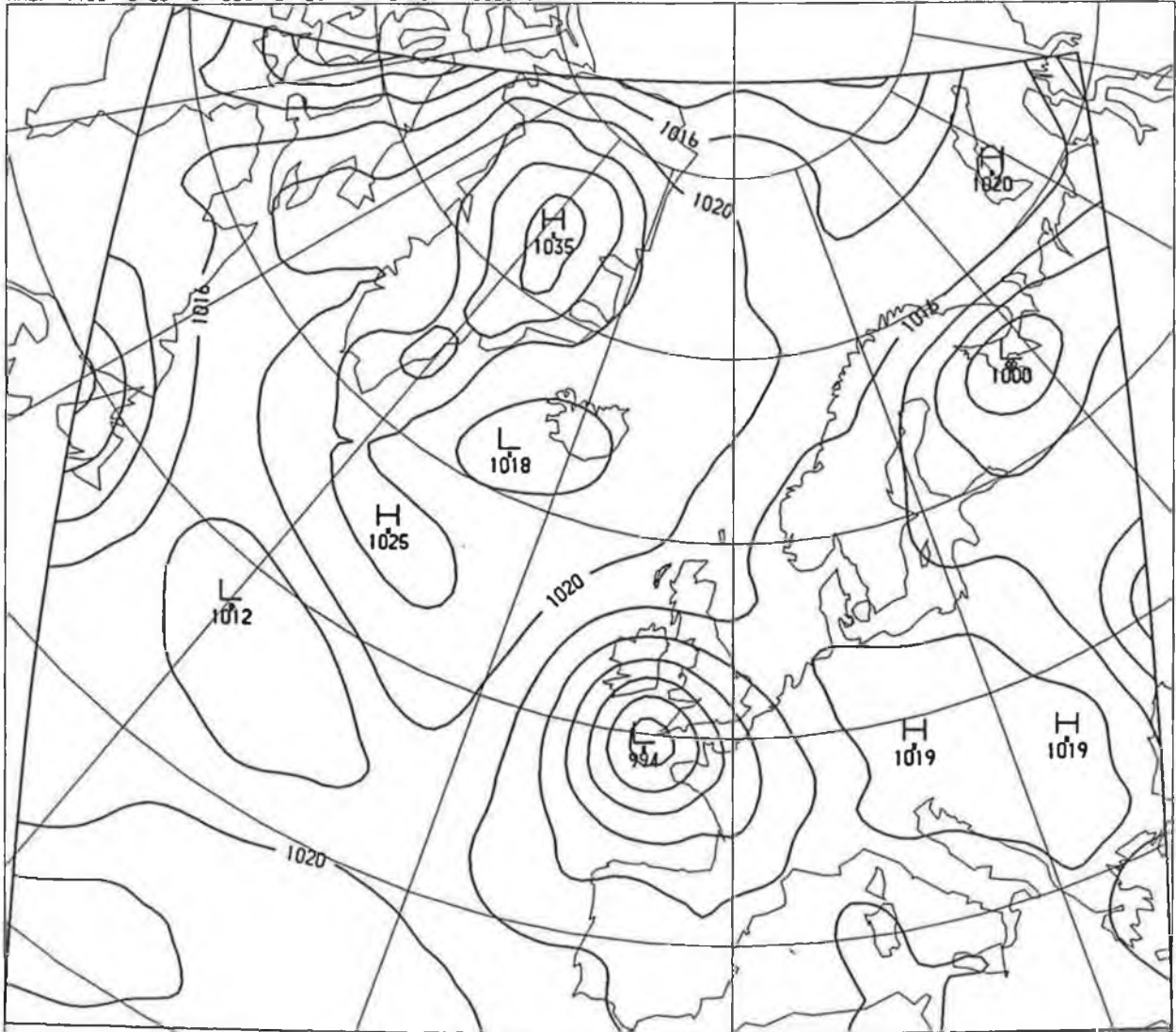


Figure 9.4(b) : Forecast of Surface Pressure valid at 0000 hours on Tuesday 26th August 1986, produced by the transputer program

9.7 Using HIRLAM to Forecast Wind

Wind prediction plots for each of the implementations are now discussed for the 1000 hPa, 500 hPa and 300 hPa levels. These are the standard atmospheric levels analysed by meteorologists when preparing forecasts. The 1000 hPa level approximates to 10 meters above the surface level of the earth while the 300 hPa level is at the upper atmosphere. All of these plots are valid for 0000z on 26th August 1986.

The wind arrows drawn in each of the wind plots indicate the wind speed and direction. For example the 1000 hPa level plot in Figure 9.5(a) shows wind arrows around the east coast of Iceland. These arrows predict a light northerly wind of 5 knots approximately. Over the northern tip of Scotland the arrows indicate a west-south-westerly wind of 15 knots. The wind speed is marked by the number of right angular lines protruding from the direction arrow. Each full line represents 10 knots, a half line 5 knots and an enclosed triangle 50 knots.

Wind fields predicted for the 1000 hPa level in Figures 9.5(a), (VAX-4200), indicate south westerly wind over the north west of Ireland of 25 to 30 knots. At the south east corner of Ireland winds of 45 knots are predicted. The corresponding transputer version also indicates similar prediction values for the north west coast.

However, the transputer returns 30 to 35 knots for the south east corner of the country. Near the eye of the storm the transputer prediction is slightly less turbulent and this discrepancy is attributable to the local intensity of the storm and the low resolution of the vertical levels used.

Wind fields, predicted for the 500 hPa level, by both model runs, are shown in Figures 9.6(a) and 9.6(b). All over Ireland the winds are predicted to be light and variable in both implementations with less differences noticeable than in the 1000 hPa level plots. Recall that in reducing the model input for the transputer from 16 levels to 7 levels the weighting factor applied in the interpolation program (Appendix A-3) resulted in virtually no interpolation between the vertical levels near the 500 hPa level.

Wind fields at the 300 (hPa) level in Figures 9.7(a) highlight wind speeds ranging from 45 to 75 knots over the mid Scottish region . In the transputer plot, Figure 9.7(b) for the same location wind speeds range from 20 to 35 knots. The wind direction in both plots is the same but winds are stronger near the path of the intense storm in the VAX-4200 plot.

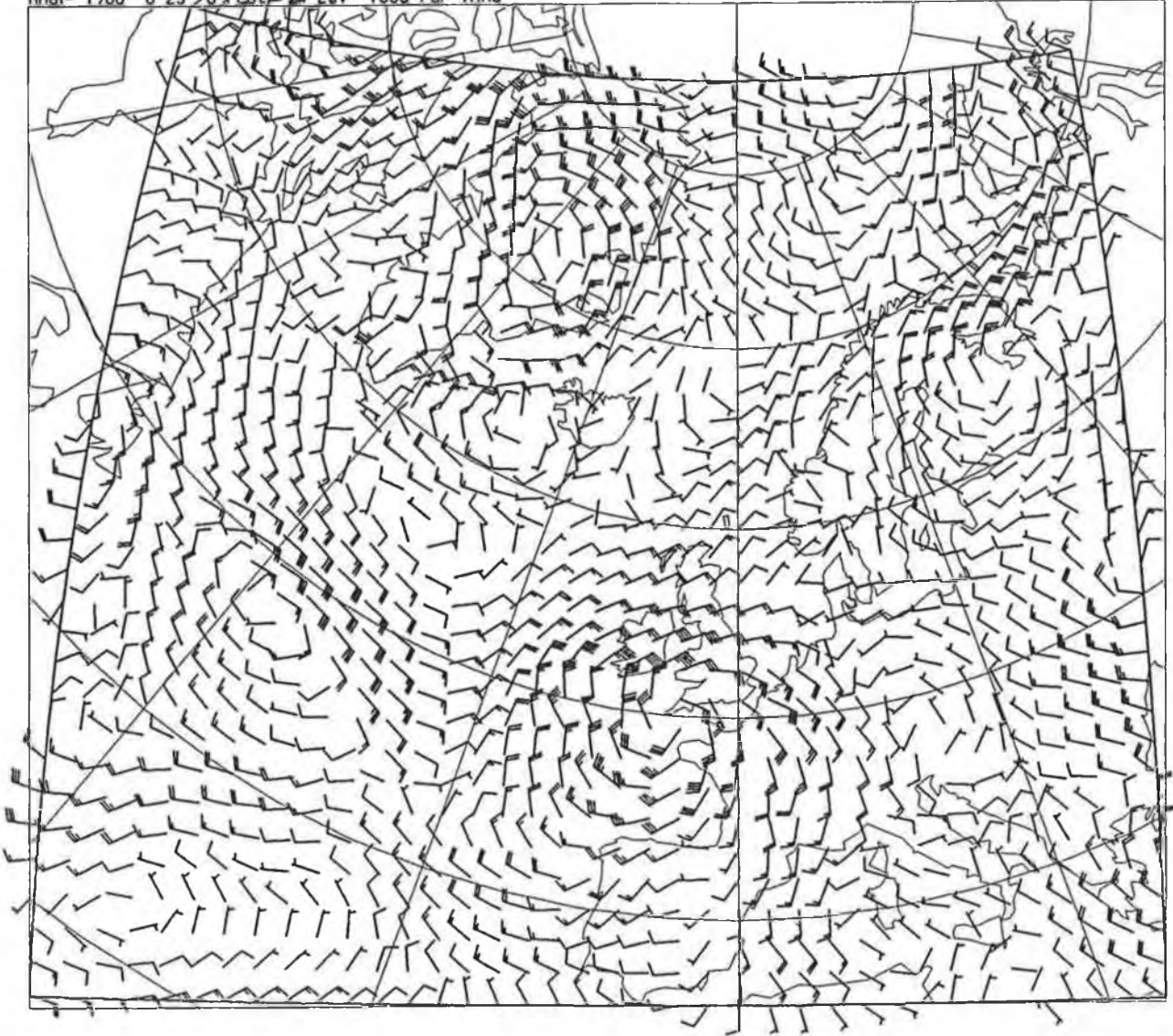


Figure 9.5(a): Forecast of 10 meter wind (1000 hPa Level), valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

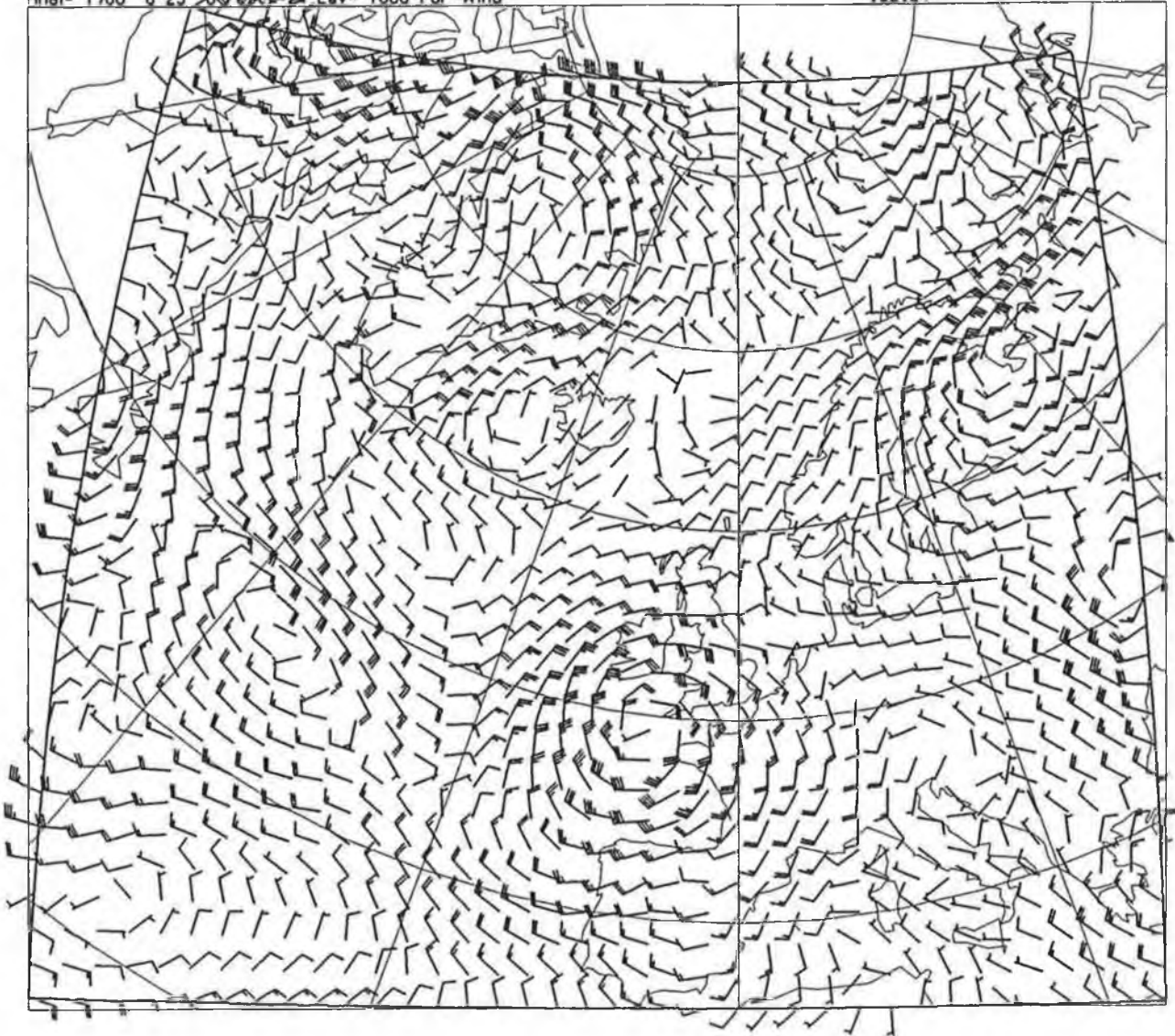


Figure 9.5(b): Forecast of 10 meter wind (1000 hPa Level), valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

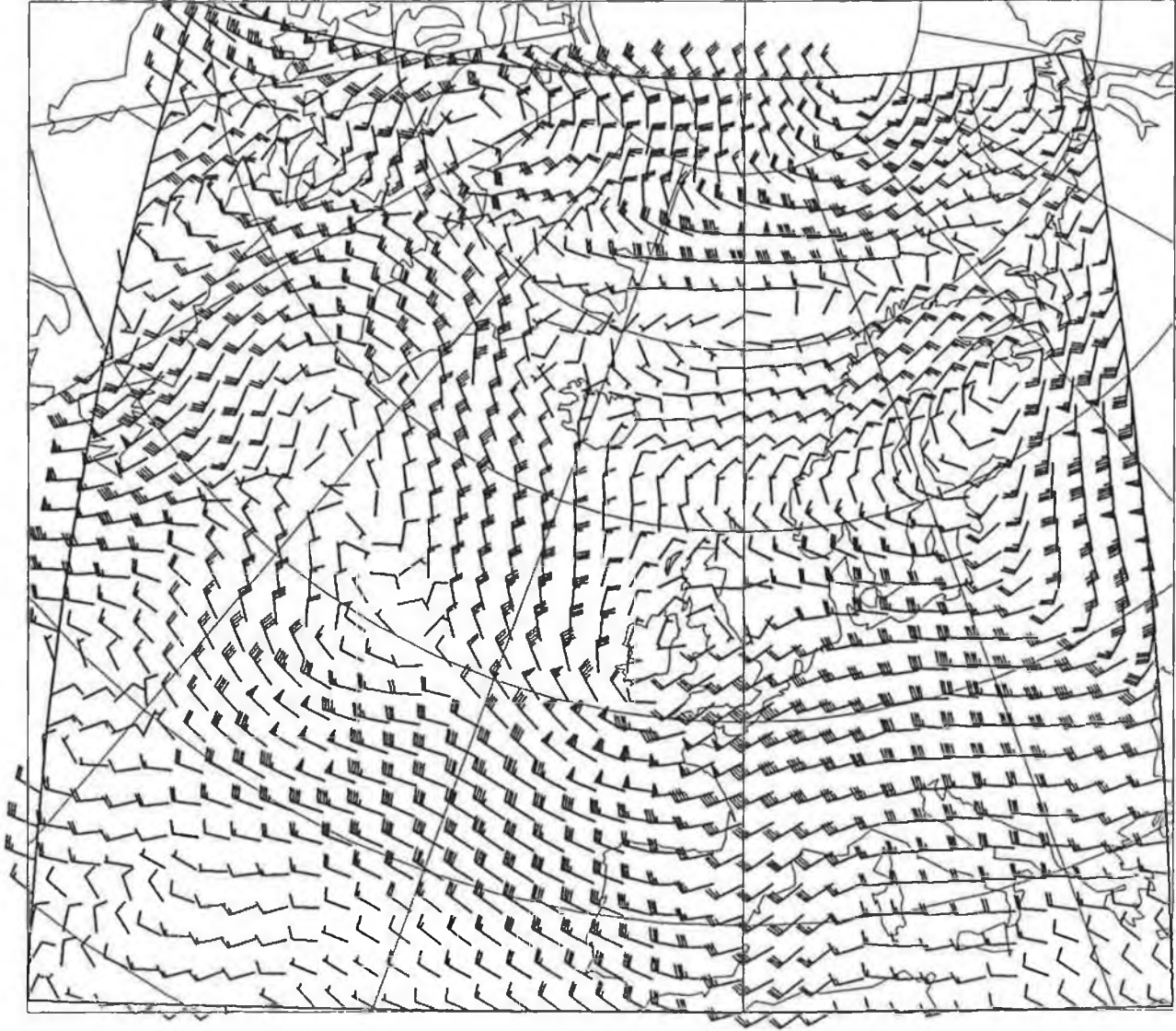


Figure 9.6(a): Forecast of wind at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

Anel= 1986 8 25 0/Fcst= 24 Lev= 500 Par=Wind

11:52:16 21/AUG/92 86082600.tff

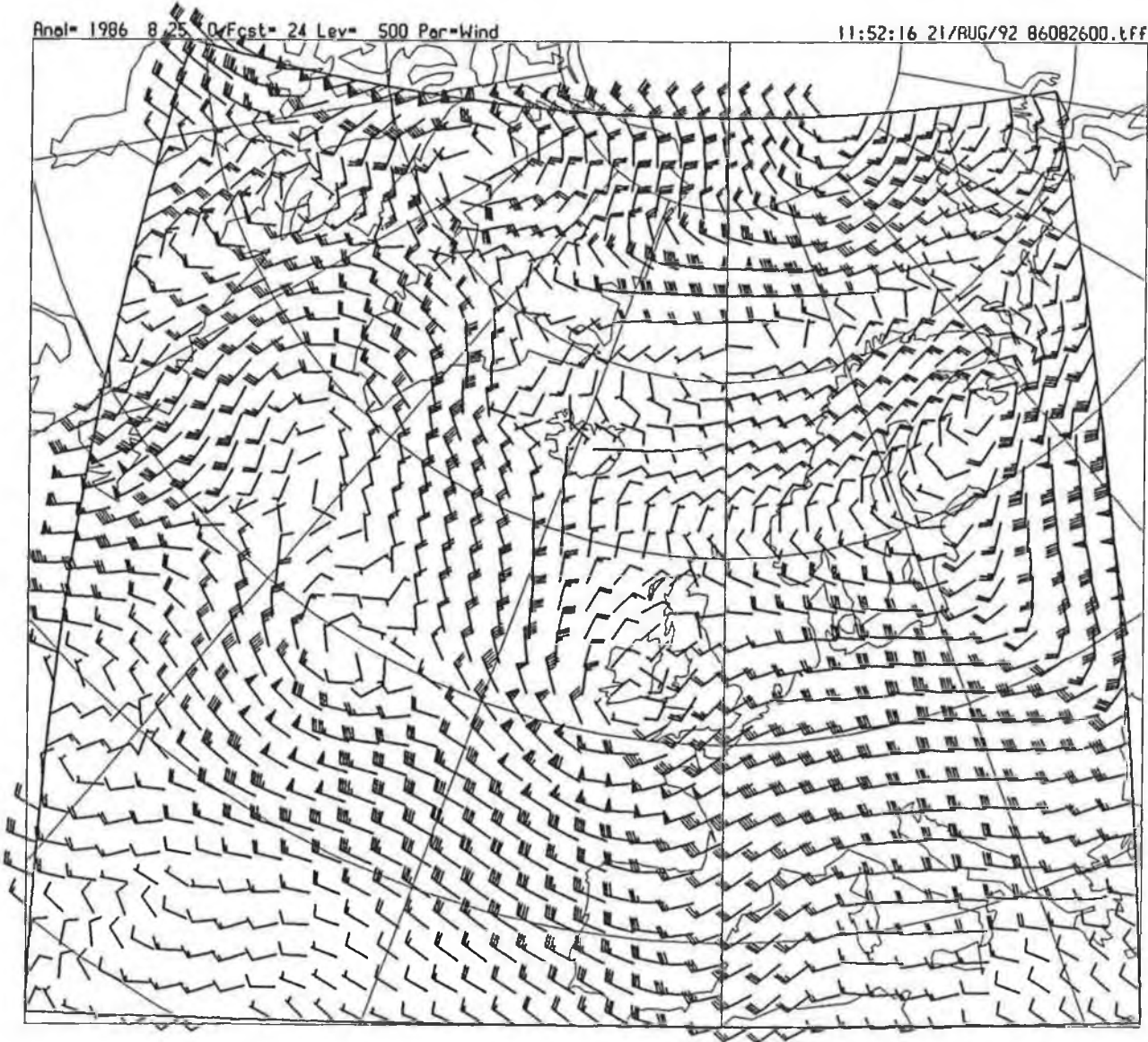


Figure 9.6(b): Forecast of wind at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

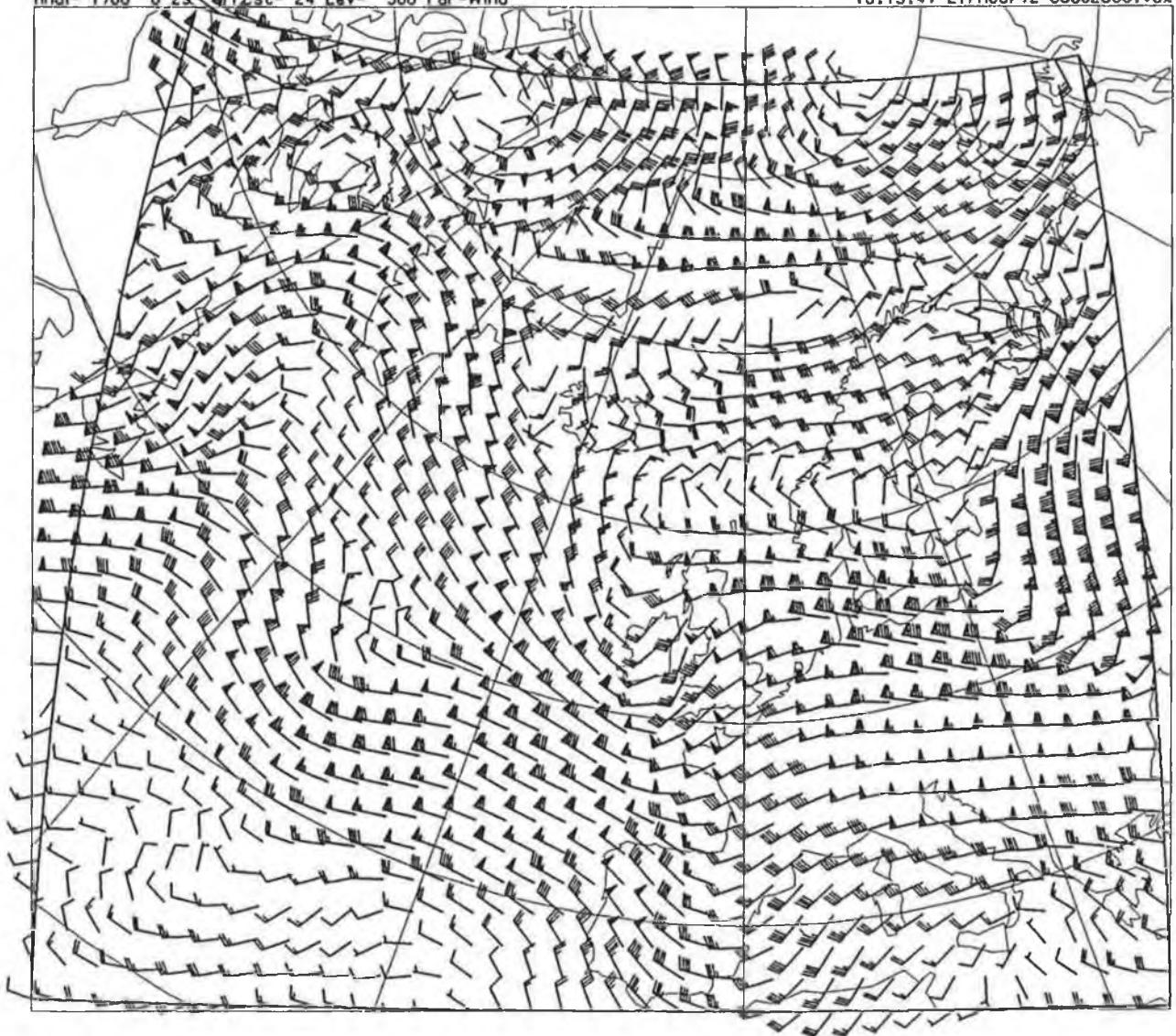


Figure 9.7(a): Forecast of wind at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

Anal- 1986 8 25 0 Fcst- 24 Lev- 300 Per-Wind

11:52:07 21/AUG/92 86082600.tff

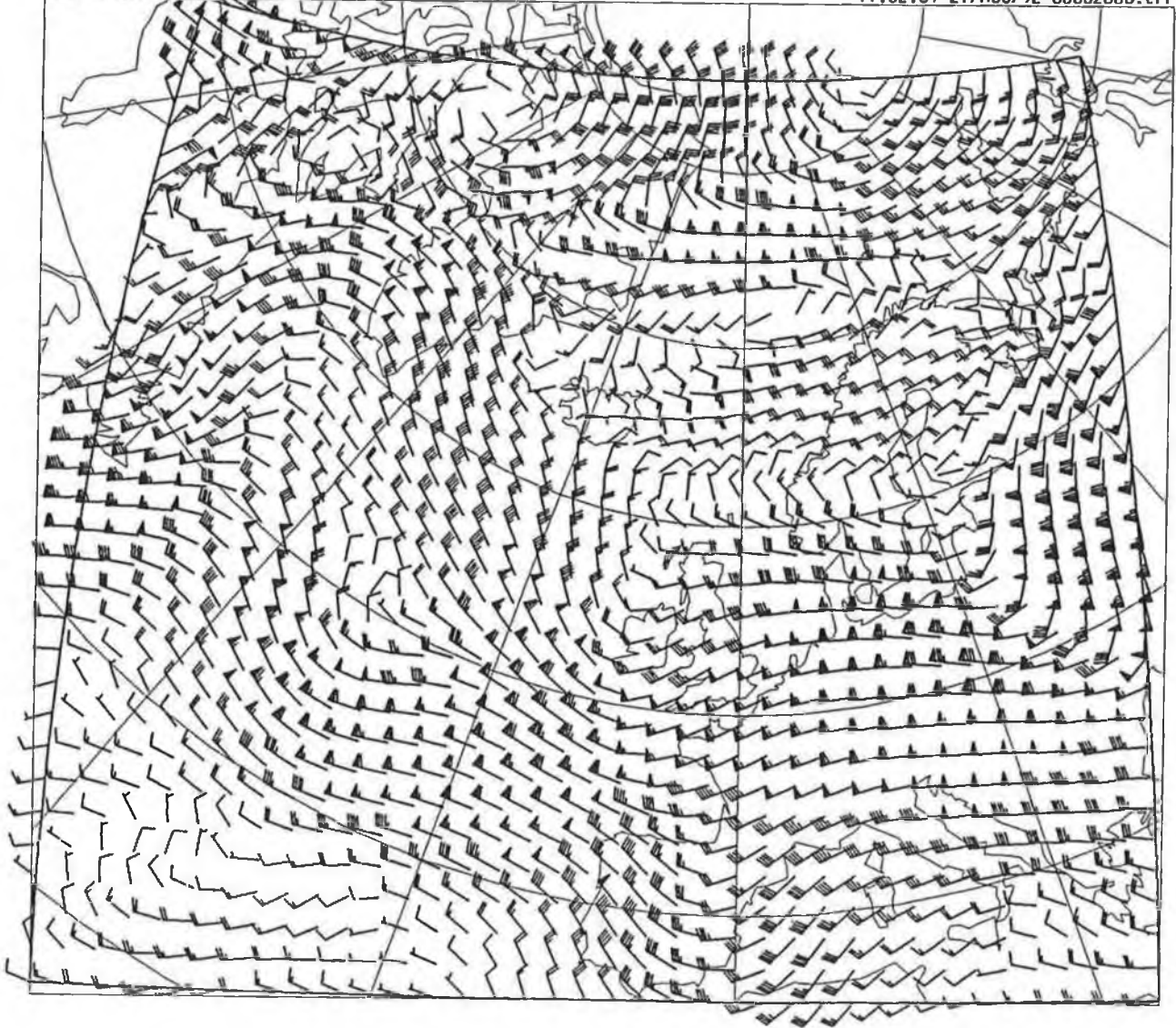


Figure 9.7(b): Forecast of wind at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

9.8 Using HIRLAM to predict Geopotential

Geopotential is defined as the potential energy acquired by a unit mass of atmospheric fluid on being raised through unit distance in a field of gravitational force[52]. In meteorology, geopotential is considered a better measure of height in the atmosphere than conventional geometric height. In the discussion on the geopotential plots that follow a contour interval of 60 meters is used throughout.

The three standard levels, 1000 hPa, 500 hPa, and 300 hPa, are used for the prediction of geopotential. Taking the 1000 hPa level, Figure 9.8(a) the plots at first glance look very similar. However closer observations reveal that over Greenland a high of 338 geopotential meters (gm) is shown on the VAX-4200 but on the transputer, Figure 9.8(b), it has a value of 288 gm. Similarly just west of Lands End at the centre of the storm the VAX-4200 shows a low of -103 gm and the transputer shows a low of -53 gm. In these extreme high and low values the differences in each case is 50 gm. All other values of the plots are more closely related where gradients are less steep.

The geopotential plots at the 500 hPa level are shown in Figures 9.9(a) and 9.9(b). In the VAX-4200 plot a low of 5520 gm is just forming to the north east of Ireland. The transputer plot, Figure 9.9(b) shows the low firmly established over Ireland with a centre of 5495 gm. In this particular case we can see that the transputer plot more accurately represents the true situation than the plot produced for the VAX-4200 forecast.

At the 300 hPa level the geopotential plots in Figures 9.10(a) and 9.10(b). The VAX-4200 version shows the contours 9360, 9300 and 9240 gm being 'pulled' towards the centre of the storm activity. The corresponding contours from the transputer output are more smooth and do not appear to be influenced to the same extent by the storm intensity.

9.9 Using HIRLAM to predict Temperature

The temperature plots, Figures 9.11, 9.12 and 9.13 are presented for the 1000 hPa, 500 hPa and 300 hPa levels respectively. The plot contours are known as isotherms and they join points of constant temperature. The symbol W, indicates a Warm sector and the symbol K denotes a Cold sector. The temperature over Ireland for the 1000 hPa and 300 hPa levels in Figures 9.11 and 9.13 coincides in the VAX-4200 and transputer plots. However, for the 500 hPa level the outputs differ as shown in Figures 9.12(a) and (b). The contour -20° Celsius is located to the north coast of Ireland in the VAX-4200 output and to the south coast of Ireland in the transputer plot. Close to the storm the temperature variation appears to be influenced by the granularity of the grid resolution used for the respective plots.

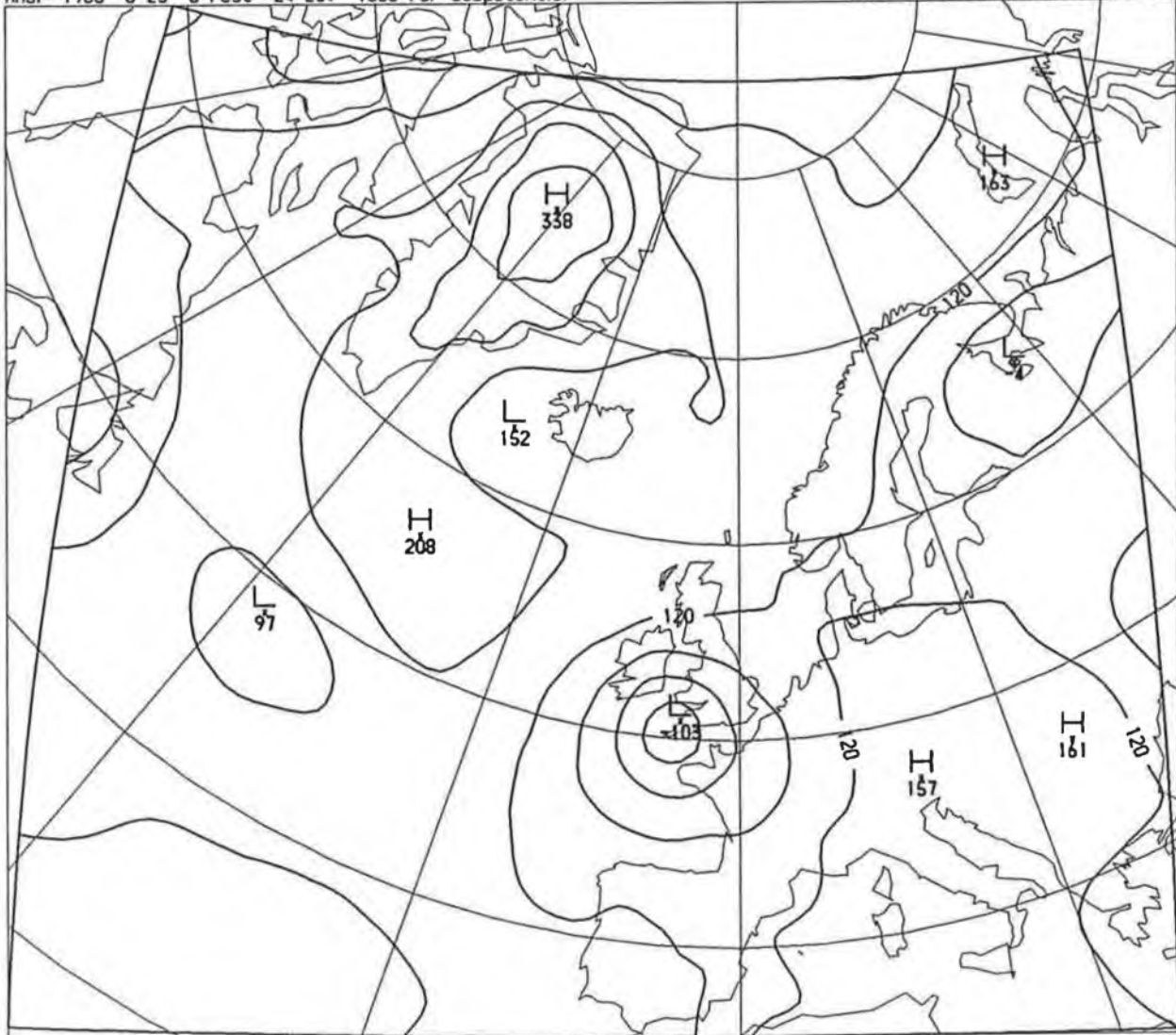


Figure 9.8(a): Forecast of Geopotential at 1000 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

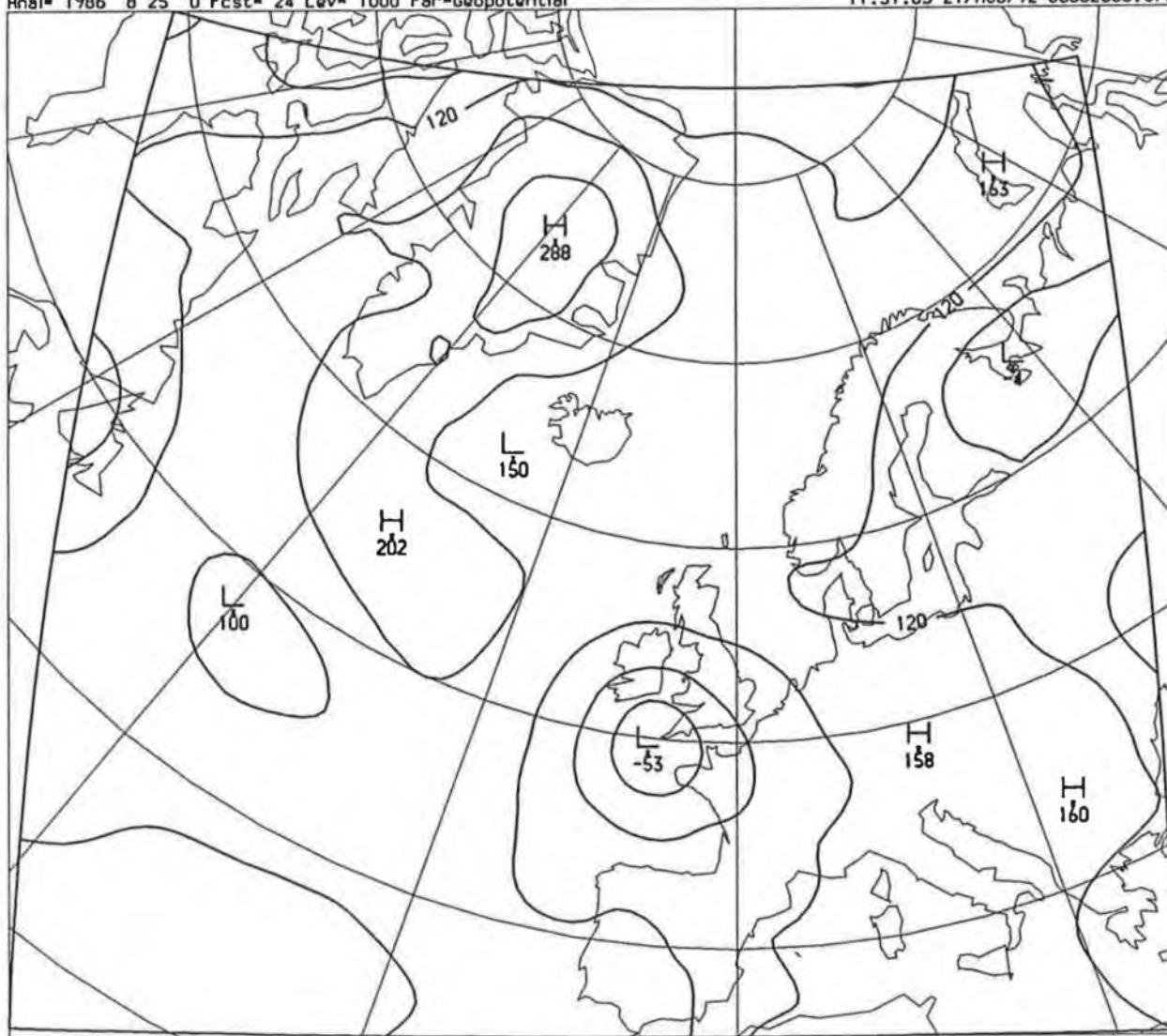


Figure 9.8(b): Forecast of Geopotential at 1000 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

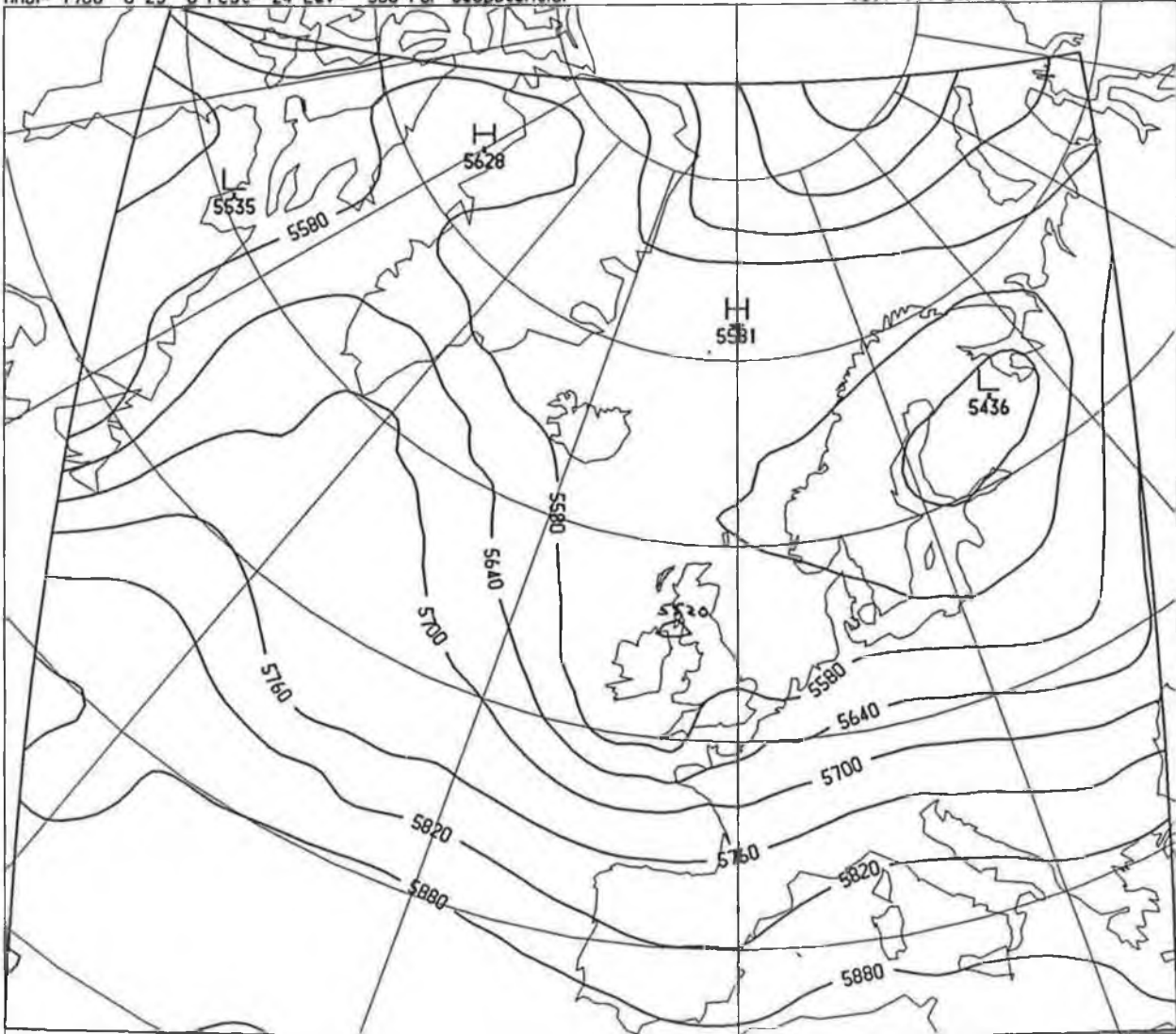


Figure 9.9(a): Forecast of Geopotential at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

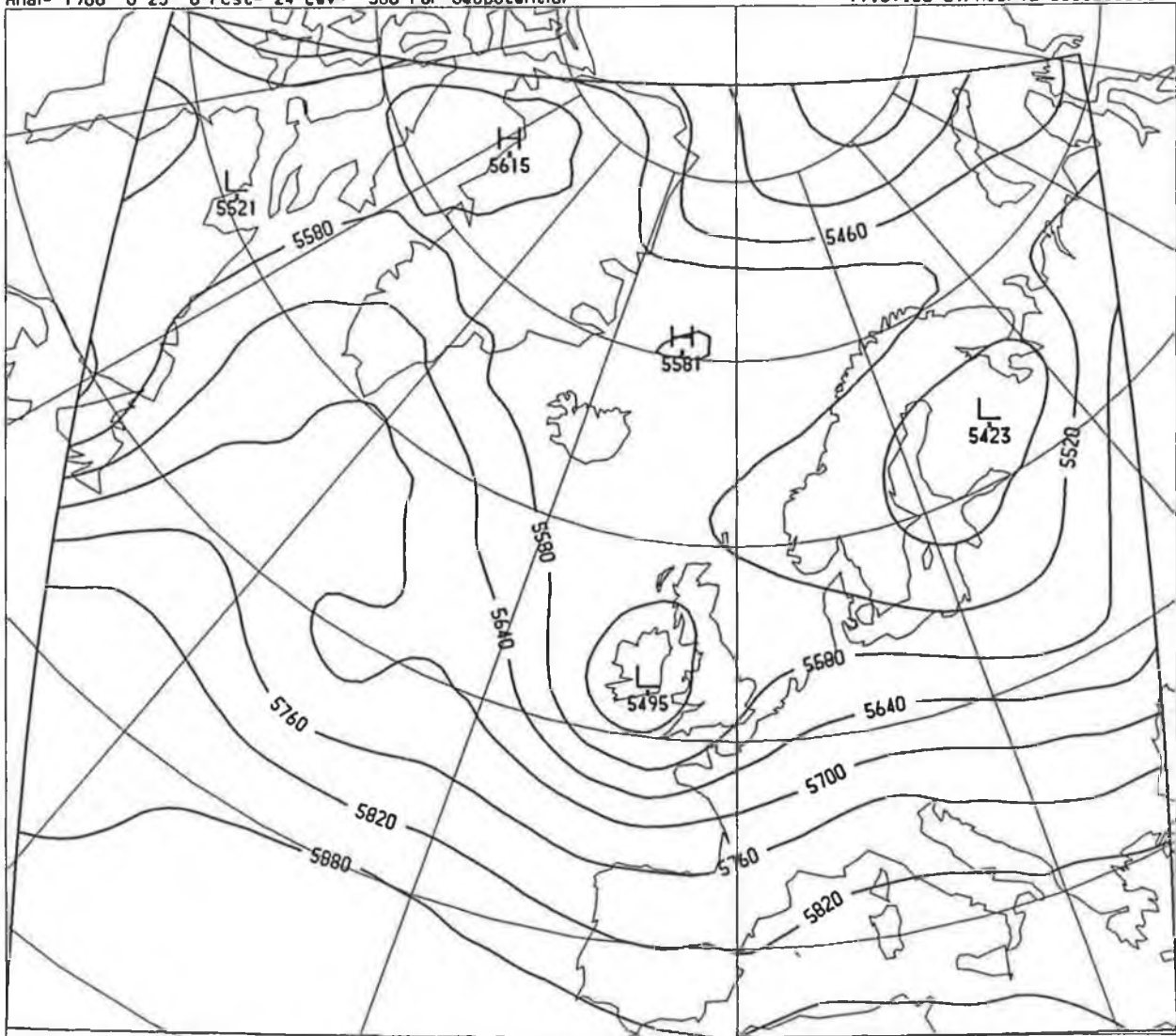


Figure 9.9(b): Forecast of Geopotential at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

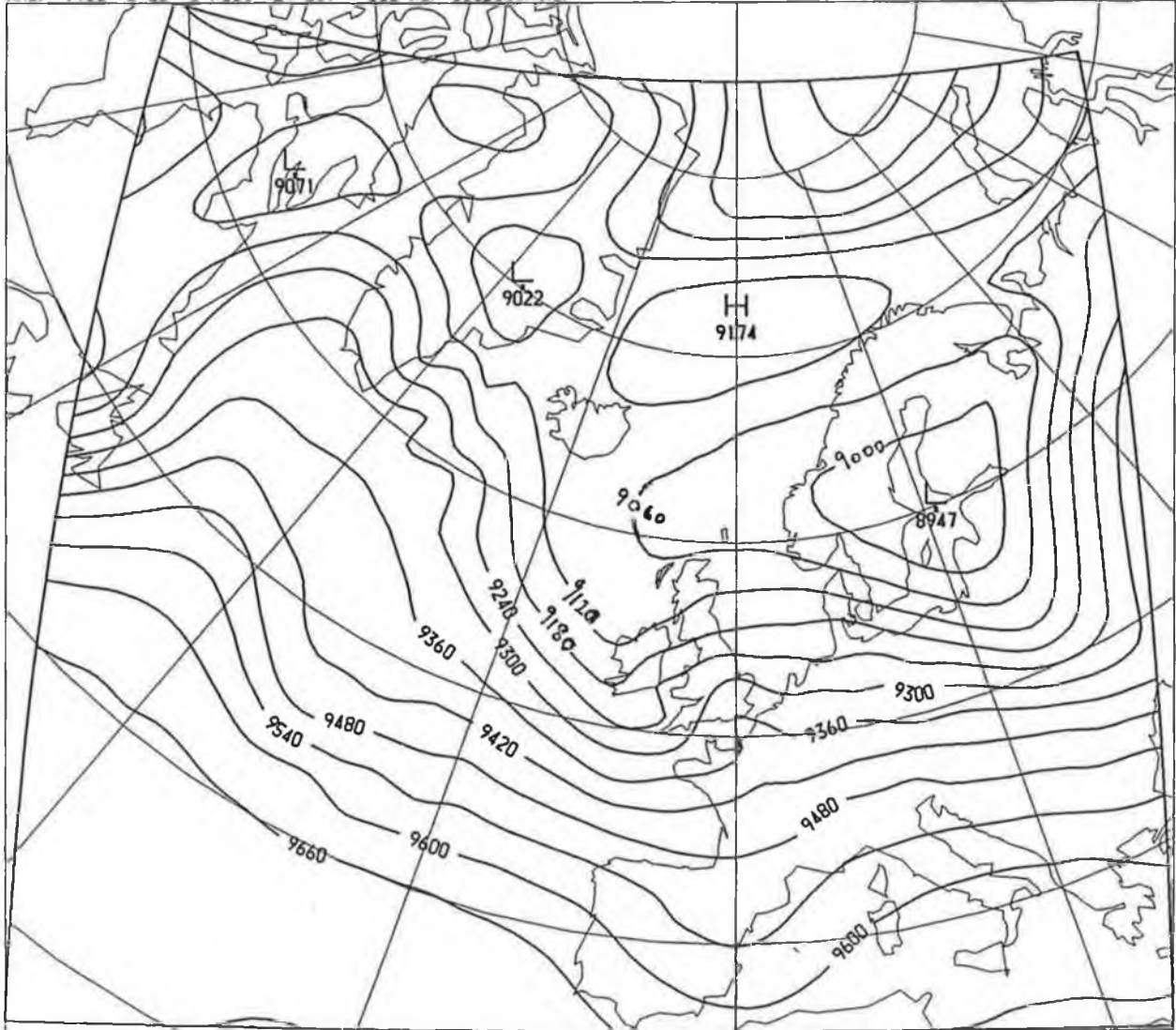


Figure 9.10(a): Forecast of Geopotential at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

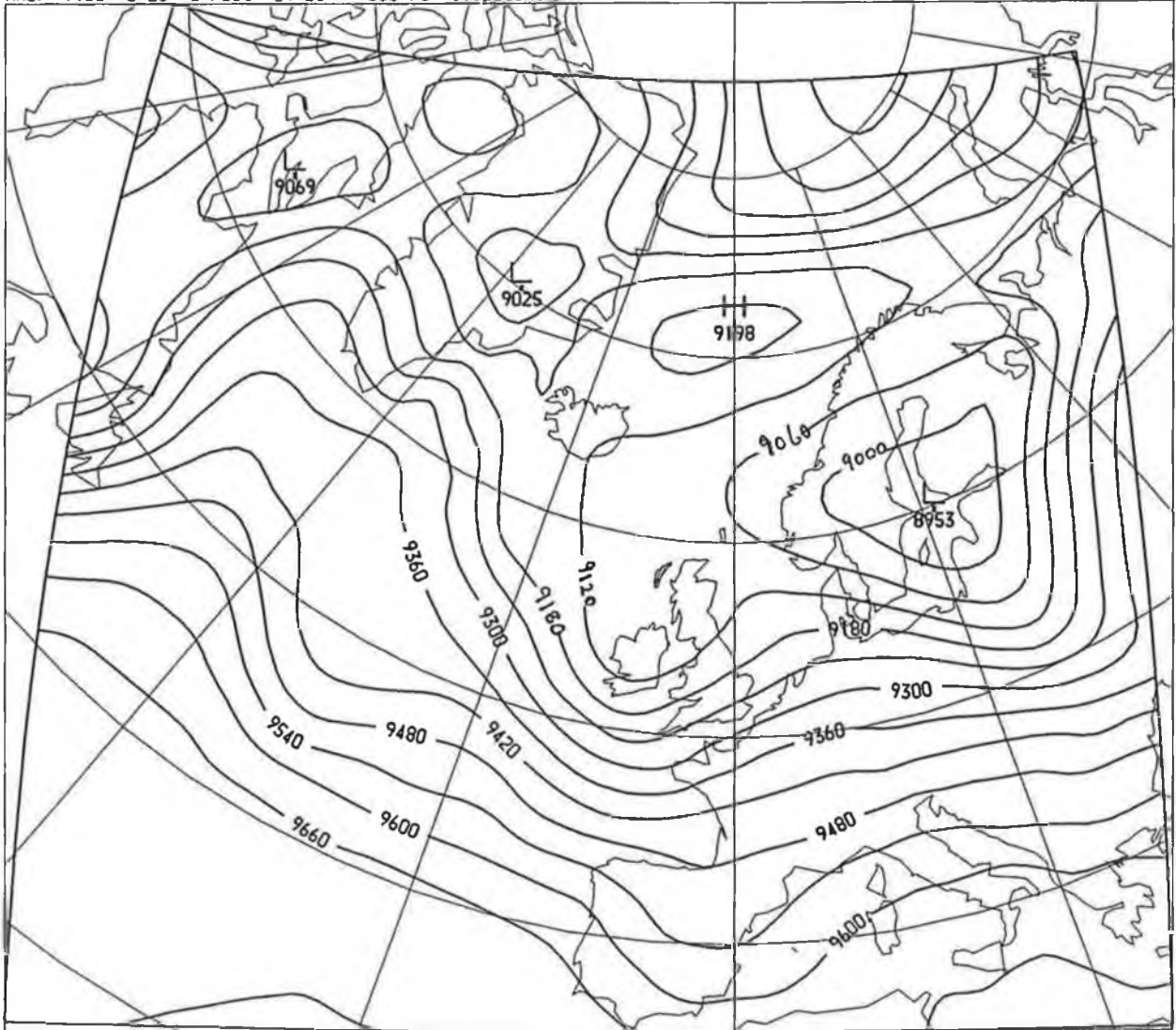


Figure 9.10(b): Forecast of Geopotential at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

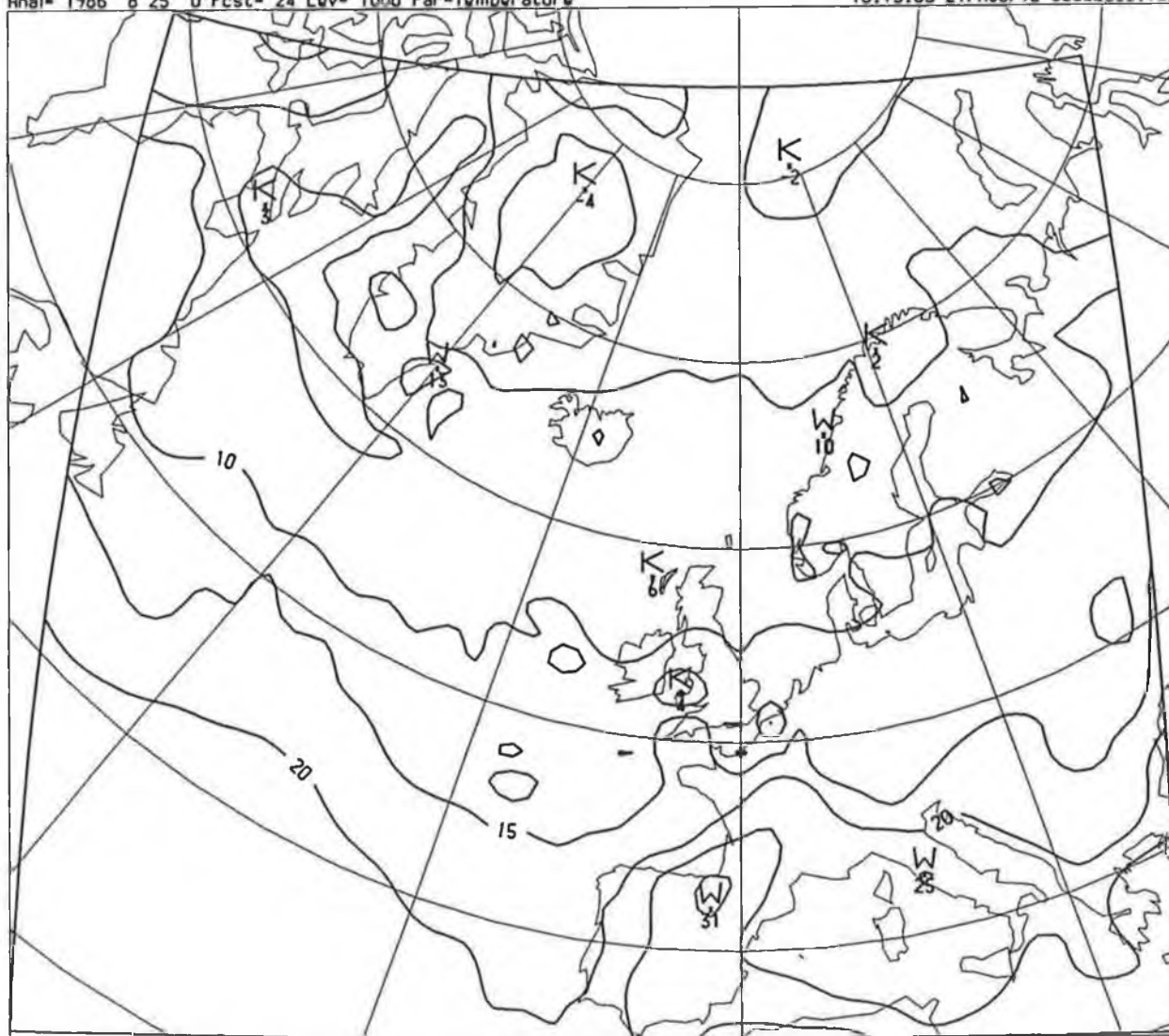


Figure 9.11(a): Forecast of Temperature at 1000 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

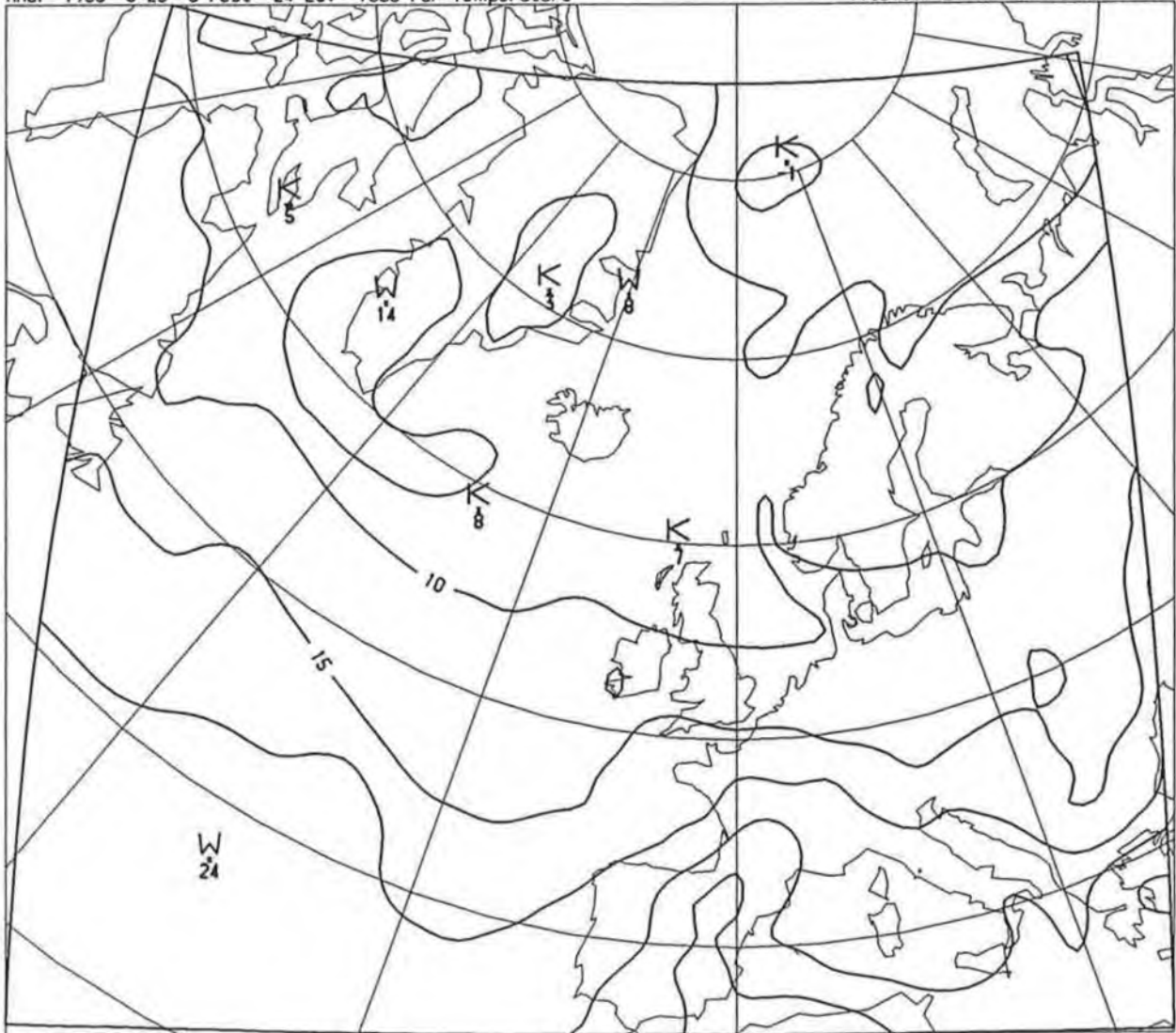


Figure 9.11(b): Forecast of Temperature at 1000 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

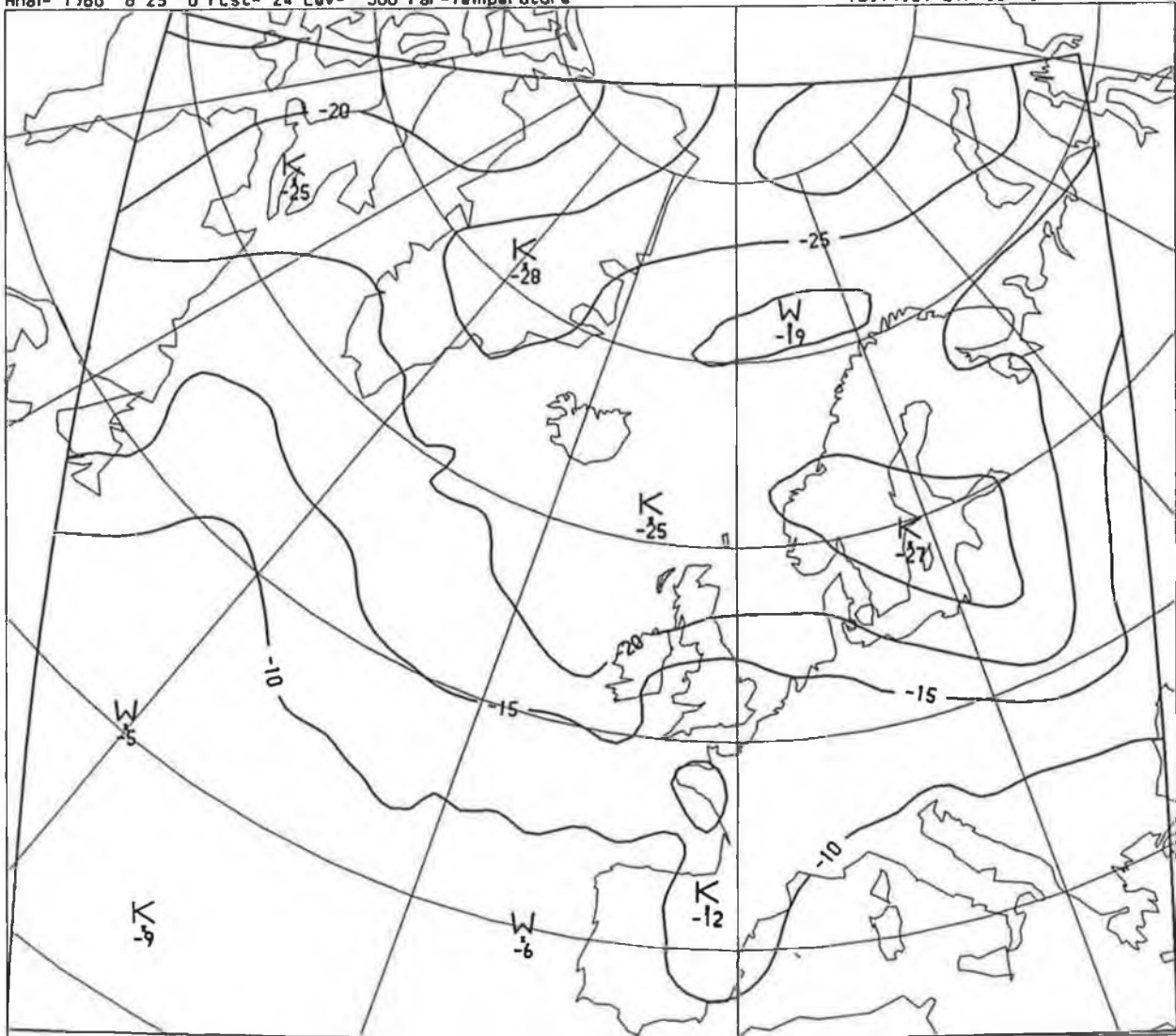


Figure 9.12(a): Forecast of Temperature at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

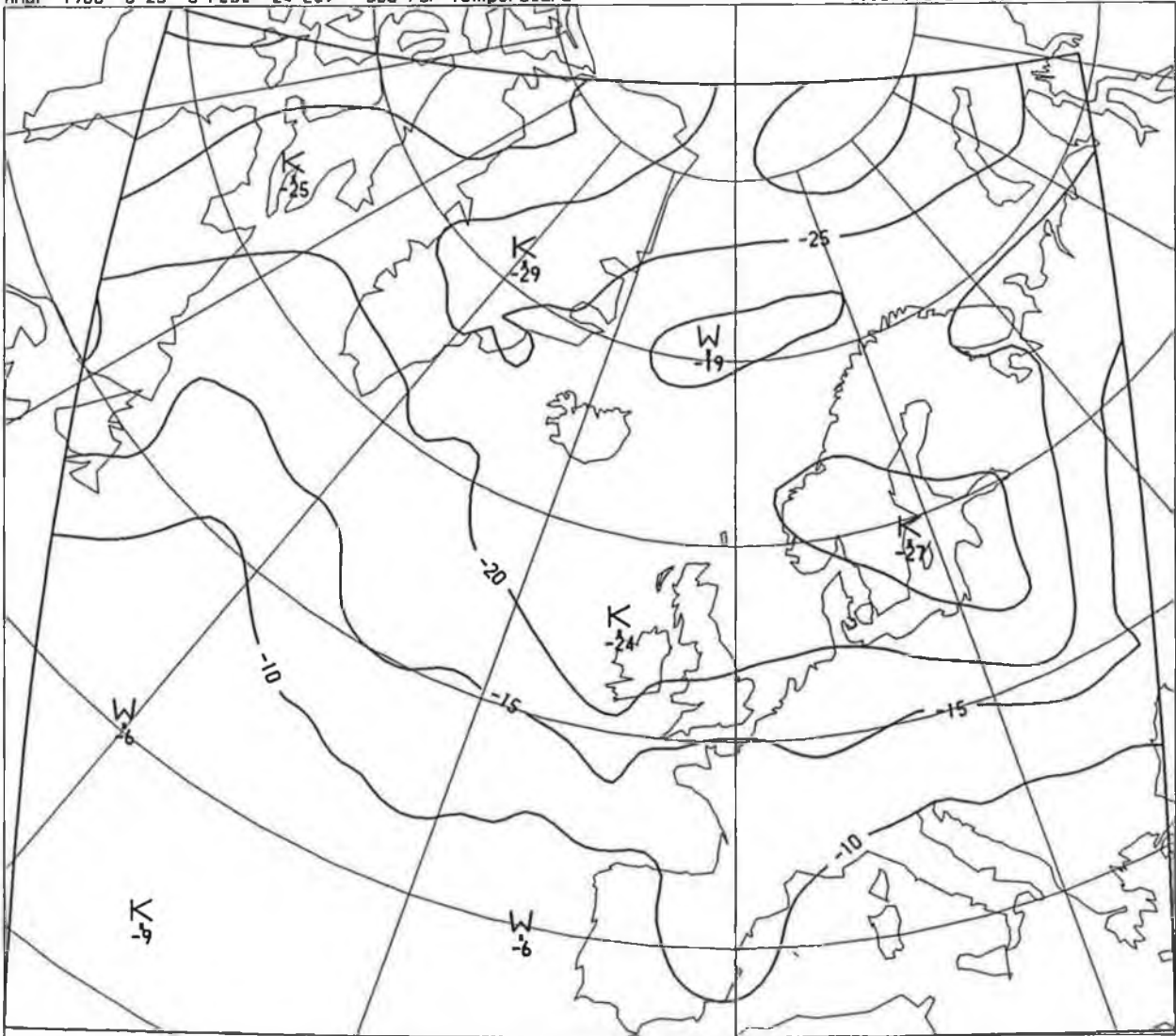


Figure 9.12(b): Forecast of Temperature at 500 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

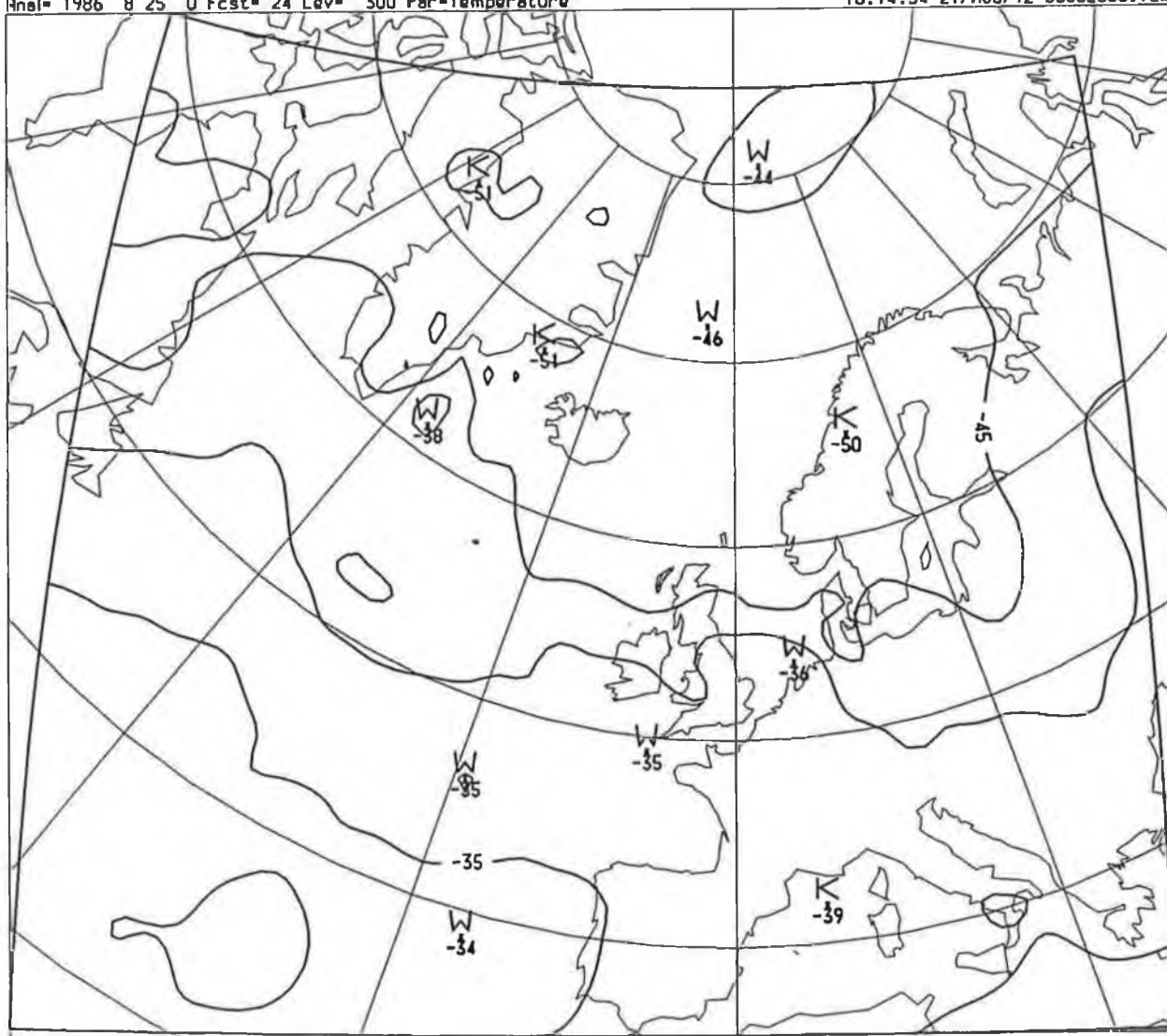


Figure 9.13(a): Forecast of Temperature at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

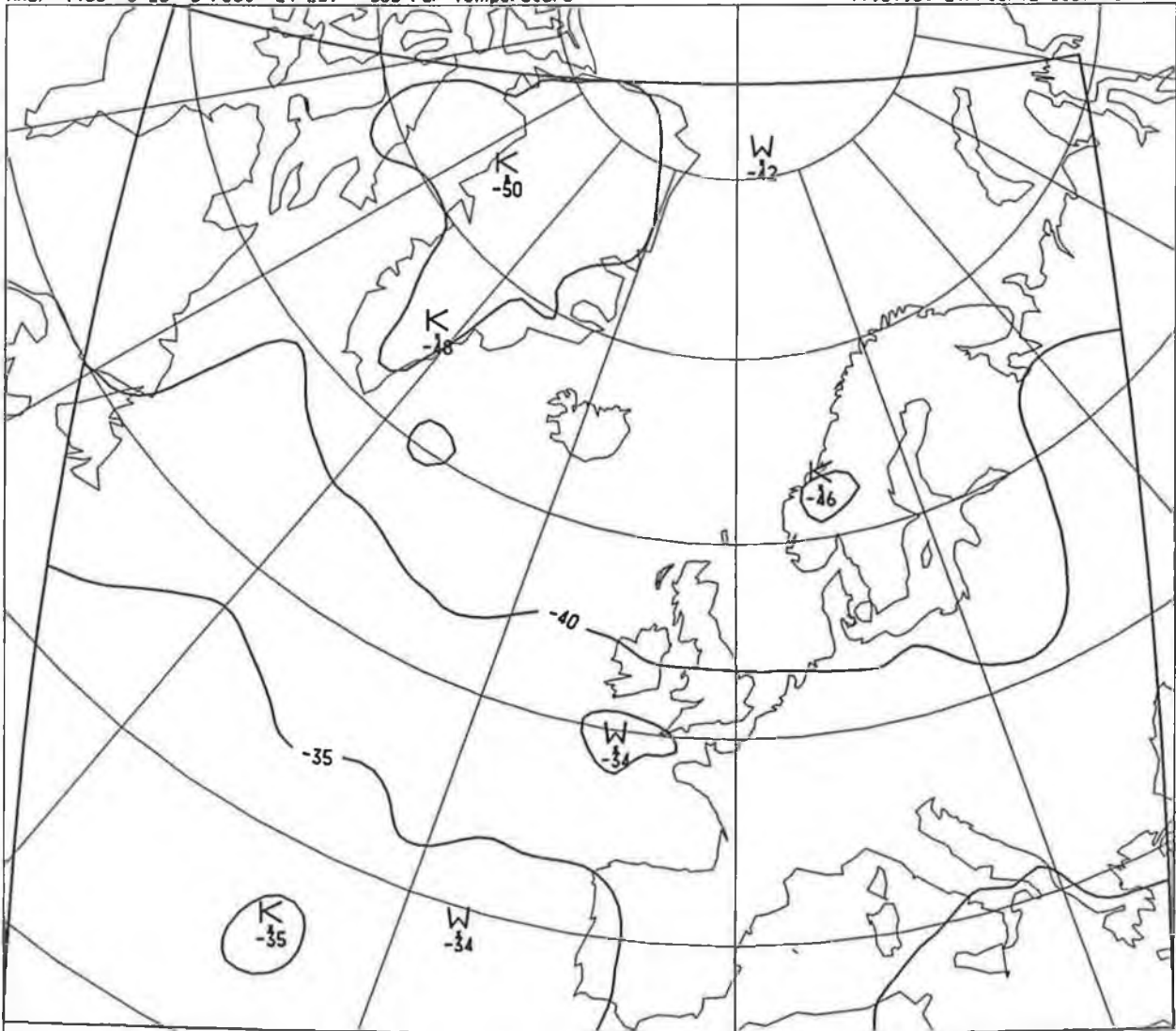


Figure 9.13(b): Forecast of Temperature at 300 hPa Level, valid 0000 hours on Tuesday 26th August 1986, produced by the transputer program

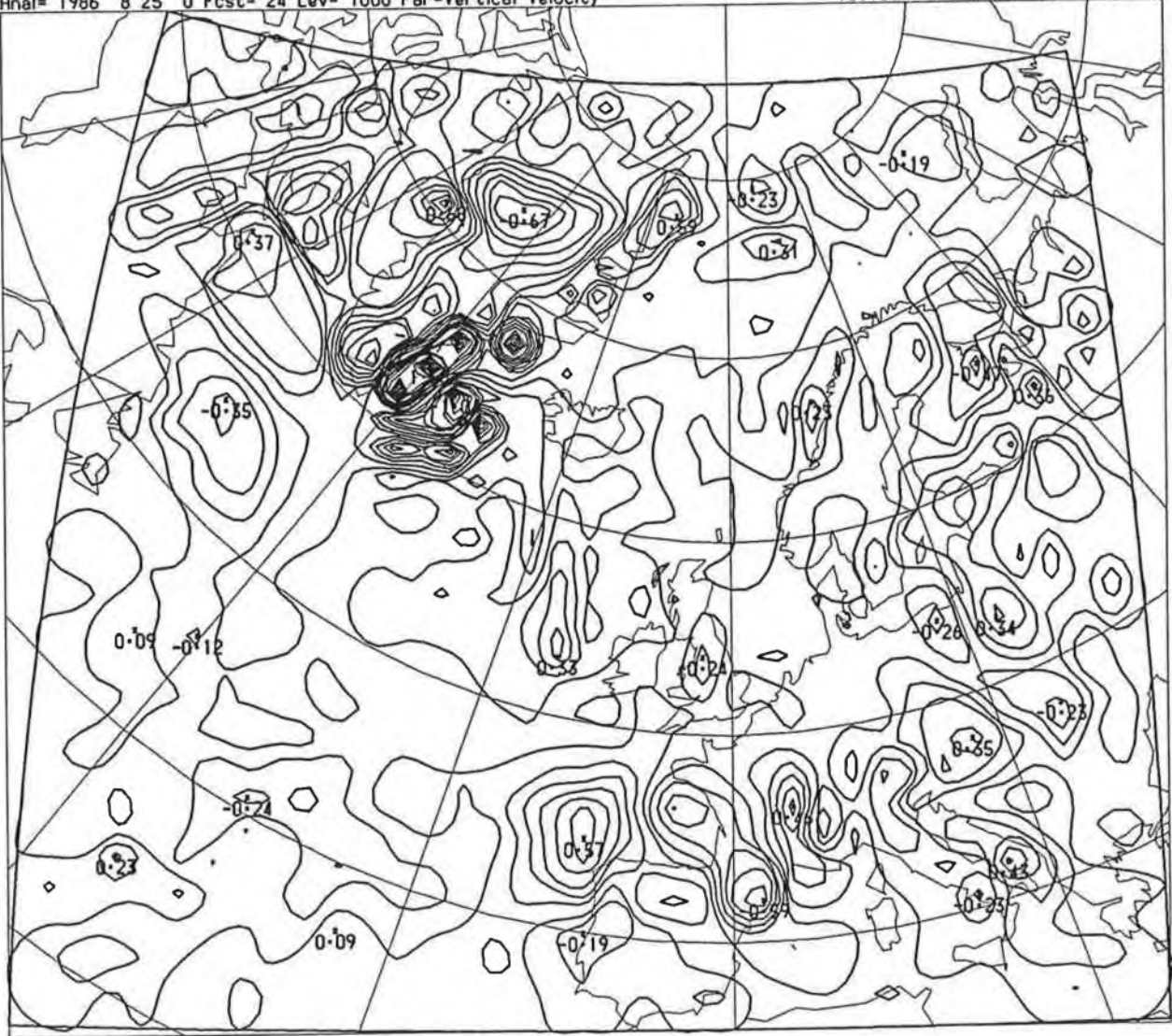


Figure 9.14(a) : Forecast of Vertical Velocity at 1000 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

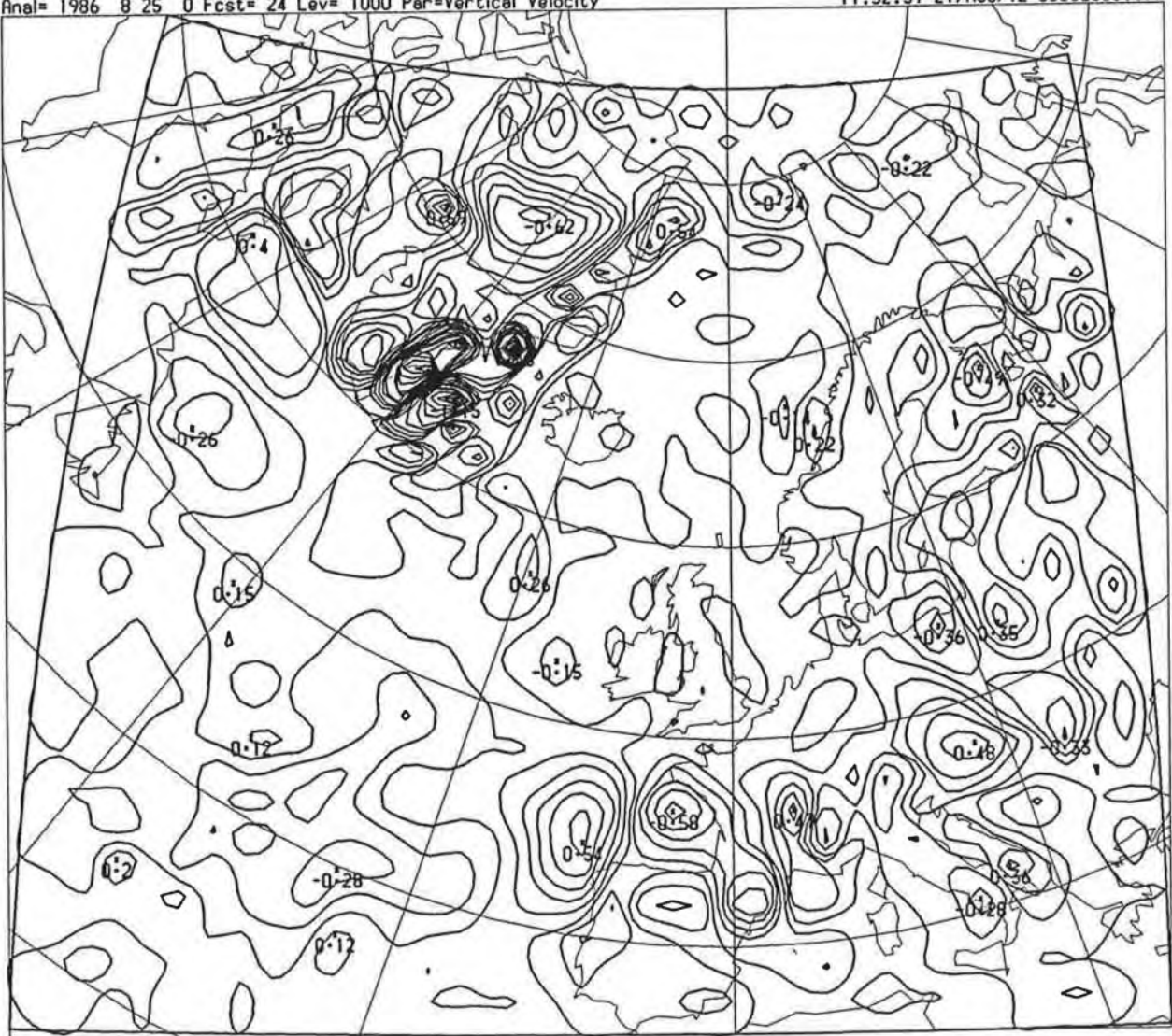


Figure 9.14(b) : Forecast of Vertical Velocity at 1000 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the transputer program

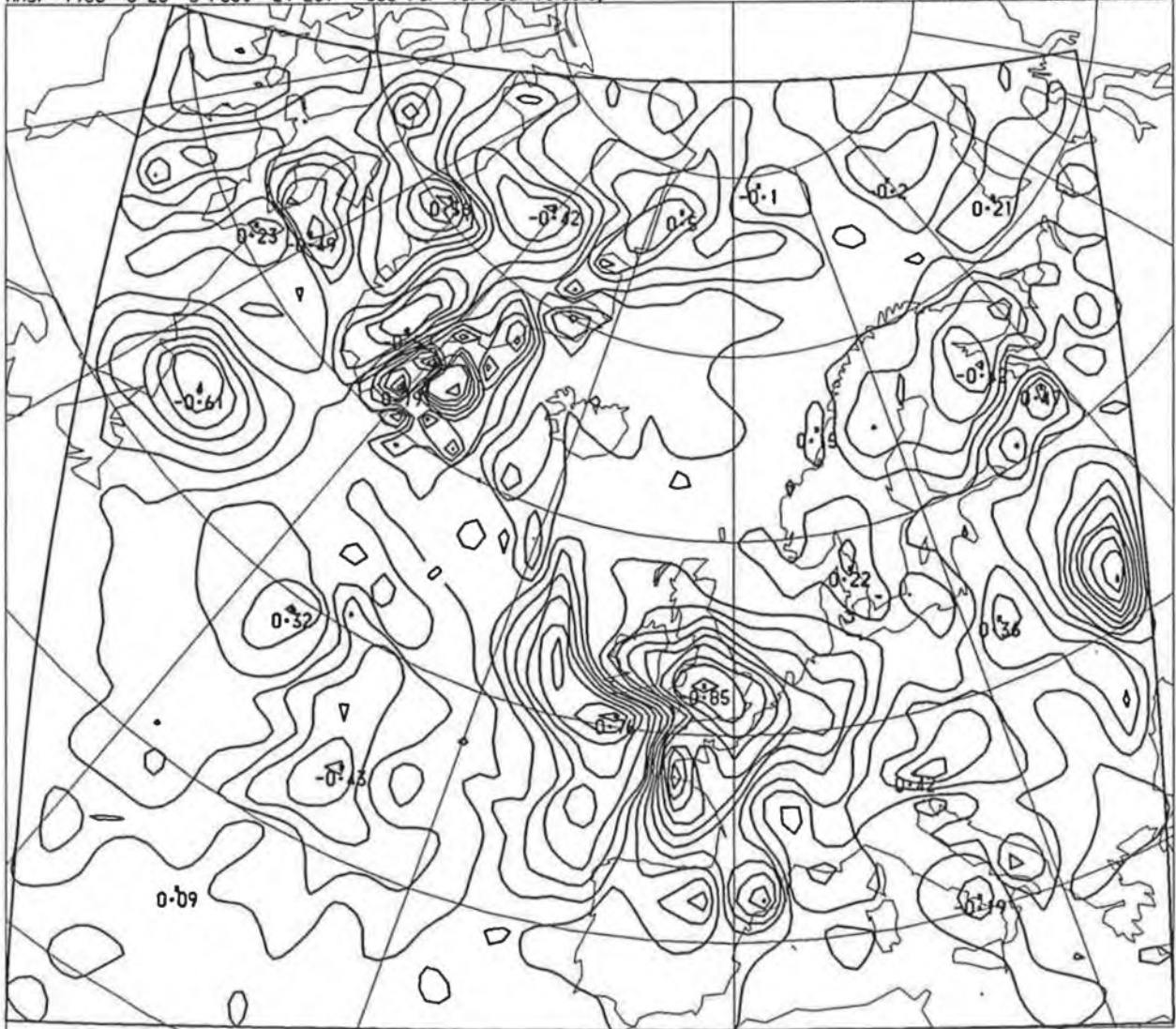


Figure 9.15(a) : Forecast of Vertical Velocity at 500 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

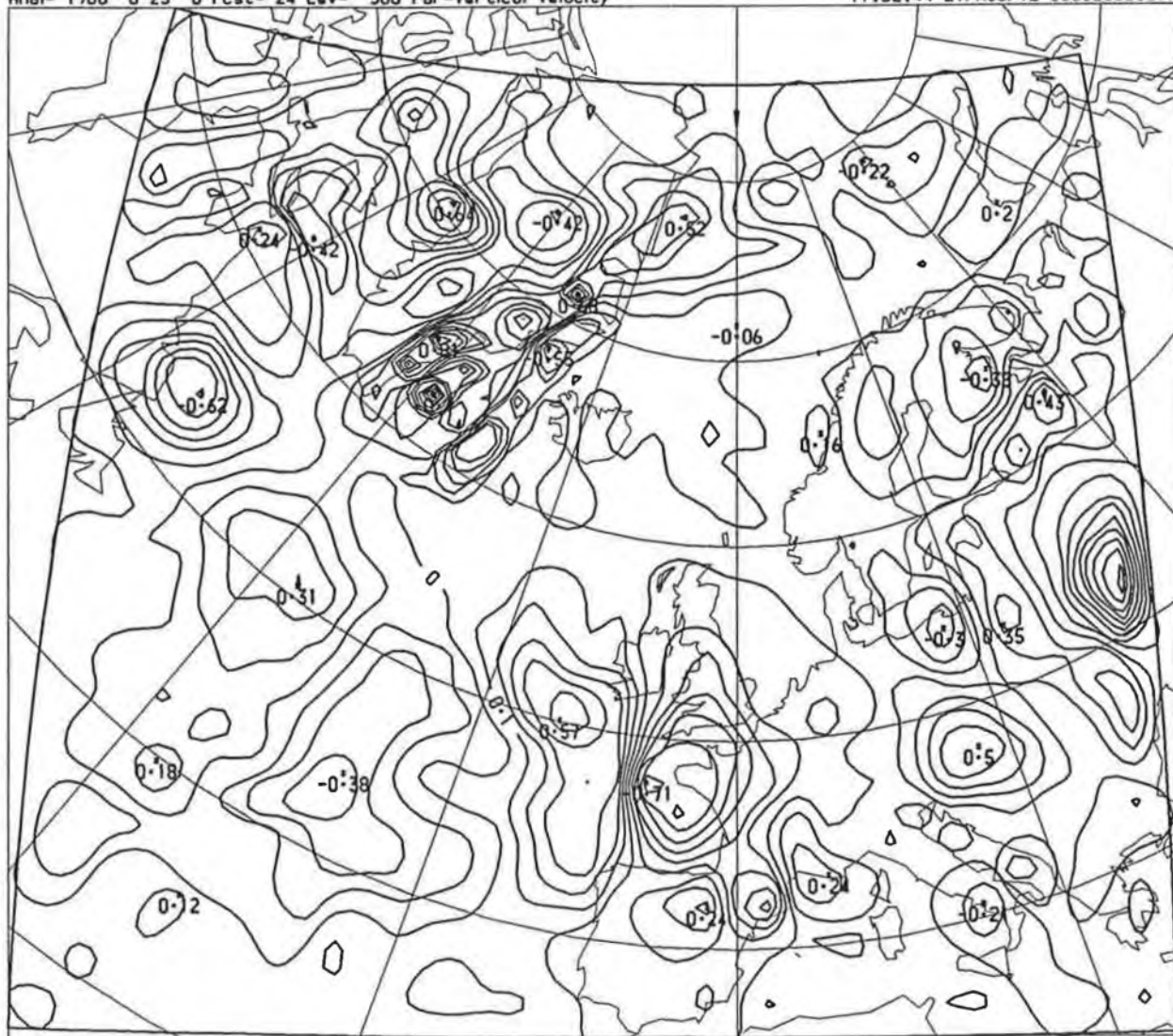


Figure 9.15(b) : Forecast of Vertical Velocity at 500 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the transputer program

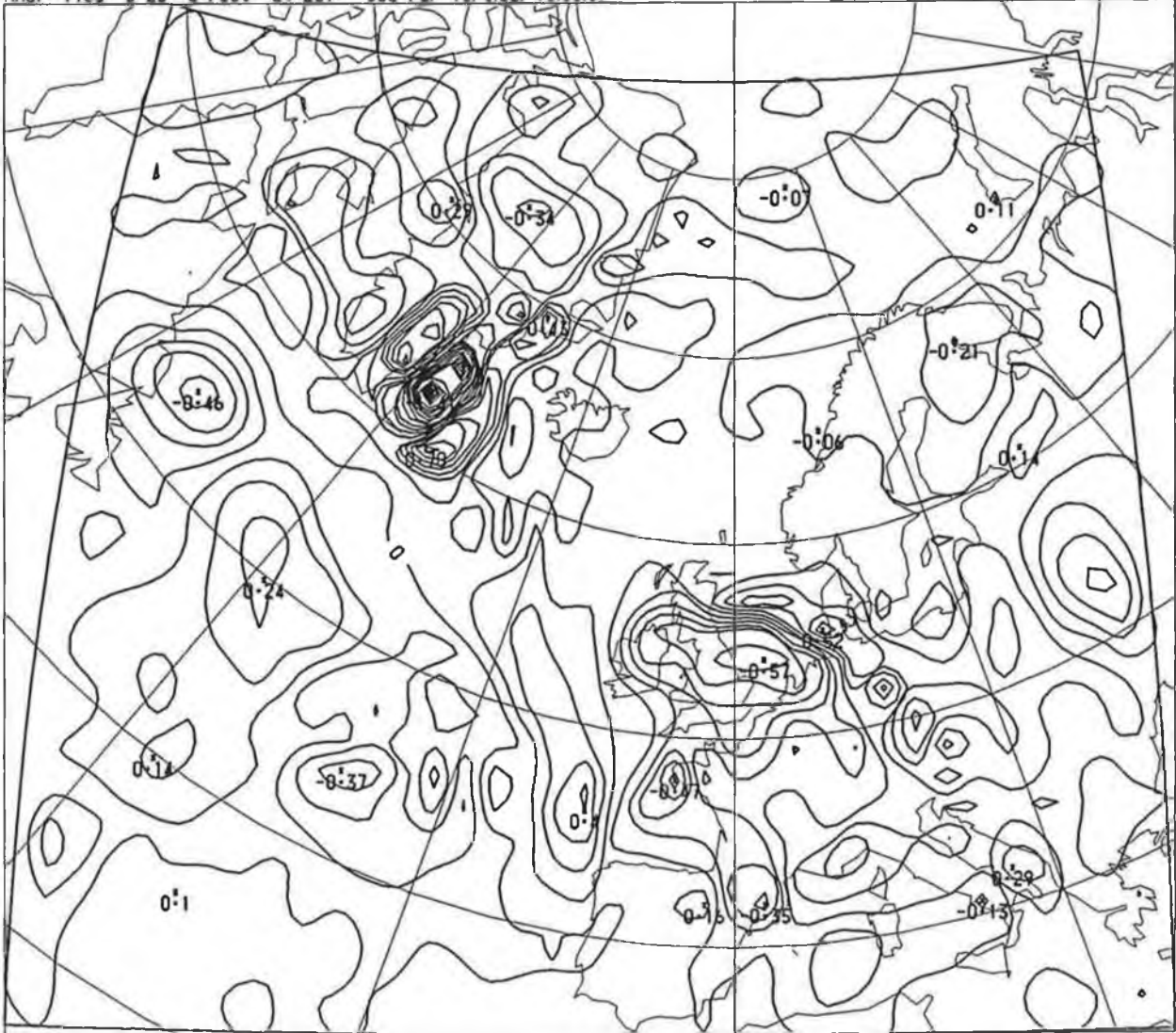


Figure 9.16(a) : Forecast of Vertical Velocity at 300 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the VAX-4200 'control' program

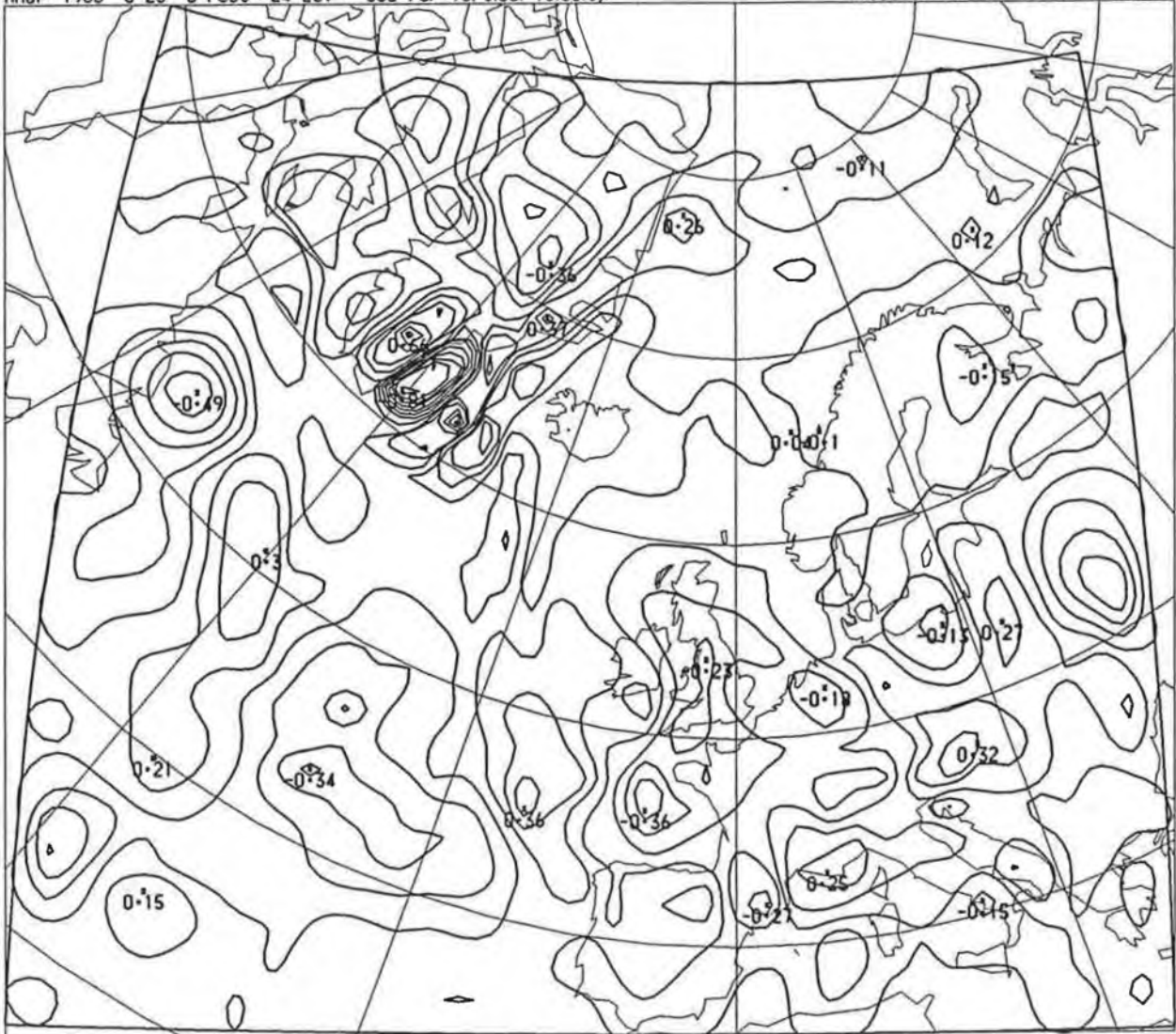


Figure 9.16(b) : Forecast of Vertical Velocity at 300 hPa Level, valid for 0000 hours on Tuesday 26th August 1986, produced by the transputer program

9.10 Using HIRLAM to predict Vertical Velocity

It was expected that the plots for the vertical velocity would exhibit a wide variation between the VAX-4200 and transputer implementations. This is because the number of vertical levels used were different in each case i.e. 16 and 7 respectively. The contour patterns show general similarities but their actual values differ as we move from the boundary areas towards the area over Ireland.

The difference in values at the lower atmospheric level, 1000 hPa, (Figure 9.14(a) and (b)), are less pronounced than at the higher altitudes as shown in Figures 9.15 and 9.16.

9.11 Summary

The concurrent version of the HIRLAM model, implemented on the Transputer Network, using parallel programming techniques, has been shown to be capable of producing forecasts which are comparable with those produced by a sequential version on a VAX-4200.

Further refinement of the grid distribution techniques, within the implementation, would provide a more detailed forecast of intense storms at local levels. The refinement in the vertical levels from 7 to 16 in the transputer implementation would clearly provide more accurate resolution to the forecast products. The refinement would result in plots from the transputer coming closer to the plots from the VAX implementations. This refinement could be easily made possible by the addition of an extra nine transputers to the current available network of 11 transputers.

CONCLUSIONS and FUTURE DIRECTIONS

This study has shown that it was possible to port a large scale Numerical Weather Prediction Model, written in Fortran, onto a Transputer Network. In the context of overall program size it was essential to devise a method of determining the most time critical regions of the program. This was achieved by placing a time-check before and after each module call. The value of the elapsed time was written to the model log file.

The segment of the log file produced during the first time step of the Model on the Transputer Network is presented in the Appendix. Execution times for all of the critical program modules were analysed. In particular, it was evident that the interpolation routines were the most expensive in terms of elapsed time for one loop execution, and in terms of how frequently they were called. These routines became the focus of detailed algorithm analysis and alternative concurrent algorithms were investigated.

Initially, it was essential to reduce the overall size of the program parameters in order to get the program up and running in a sequential manner on one Transputer. This still resulted in a problem with Memory capacity. So, the program was further reduced in size. This was achieved by the removal of a "spare" element known as "Water Vapour Content". The "spare" element was not serving any useful purpose and merely inherited the values of relative humidity.

This feasibility study was only possible because we were able to use a Parallel Fortran Compiler, developed for the Transputer Environment. It would not have been possible or indeed practical to translate a large program suite, such as HIRLAM, from Fortran to OCCAM, for example.

The availability of a Parallel C compiler, which produces binary object code, compatible with that of the Parallel Fortran Compiler, is an advantage. Program developers can write Message-passing interfaces in Parallel C and link them with the main Fortran Modules.

However, Parallel Fortran still has several limitations. These have already been described in earlier chapters. Such limitations deprive the Fortran programmer of the freedom to simply concentrate on the Algorithm to be programmed. When programming for a Parallel environment it is essential to be mindful of the architecture, the algorithm, the programming language and limitations imposed by data dependencies.

The notion of Task Placement is a key step in moving from a serial architecture to a distributed architecture. The concept of a "processor farm" is a simple method of building applications for a Transputer. This method restricts the programmer to a special type of algorithm. It stipulates that one Master task must be placed on the ROOT Transputer and a copy of the Worker task must be placed on every Transputer in the Network.

However, the concept of a "processor farm" for the case study proved unsuitable. A Multiplexer/Demultiplexer Configuration technique was used because it was much more versatile.

To perform the detailed calculations, required for complex flows and geometries in Numerical Weather Prediction Models, only supercomputers have been shown to provide the speed and memory requirements. Fine Grid Analysis can involve many thousands of calculations. A supercomputer can reduce computation time significantly in the case of the HIRLAM Model. Networks of Transputers can be configured to achieve the computational capacity of a supercomputer such as a CRAY-YMP.

Numerical Weather Forecasting is one area where much progress is being made in the development of parallel algorithms, particularly for vector processors with centralised memory. However, to successfully implement parallel processing algorithms, on transputer networks with distributed memory, different algorithms are required in the formulation of solutions to problems.

Researchers are developing systems to convert sequential scientific programs into multi-task programs suitable for execution on parallel computer architectures. One approach is to follow a 'manual' procedure for the detection of parallelism in the program code. A method of program transformation would then have to be manually undertaken. The transformation method used during this research work could be

classified as a 'manual' approach. It is time consuming and many algorithms have to be refined or replaced.

Another approach is to use a suitable compiler or pre-compiler to handle the transformation task. Some compilers are capable of the automatic detection of parallelism in scientific programs[50]. However, they are still not sufficiently robust to be able to provide global optimisation of the transformed program.

A third approach has been proposed by Saltz et al.[51], who claim that the compile-time information is sometimes inadequate. The proposal is to perform preprocessing on program loops at execution-time. To achieve this, a framework for performing a 'loop dependency analysis' has to be set up at compile-time[51]. Then at run-time, 'wavefronts of concurrently executable loop iterations', can be identified. The 'wavefront' information can then be used during execution-time to rearrange loop iterations so that they can be executed in parallel. It is evident from this approach that it is difficult to write pre-compilers and compilers which can be depended upon to transform sequential programs into parallel programs. A compiler offering such sophisticated functionality would require a computing platform with substantial performance capacity to run itself, not to mention an application program, such as HIRLAM.

Specialist software such as OCCAM for the transputer is gradually being replaced by standalone languages like the Parallel Fortran-77 language from 3L Limited. The standalone languages provide a mature programming environment. Programs written in Fortran-77 for sequential computer architectures can be ported with relative ease to transputer networks hosted by PCs. The porting procedure should be much more straight forward in the future as more pre-compilers, debuggers and implementation procedures are standardised using the topical 'open systems' philosophy.

The parallel execution architecture offered by transputer networks is destined to play a major role in the transition of computer program development, on a single processor, to new development techniques in multi-processor environments. Scientists recognise the need for increased processing power and are demanding it. The hardware platforms are delivering a greater price/performance ratio than ever before. The software environments are improving but at a much slower pace.

In the case of Local Area Weather Forecasting, meteorologists are, after all, demanding the maximum processing capabilities in order to achieve accurate simulations of the behaviour of the atmosphere in the shortest possible time. The milestones on the road to meeting this demand are clearly pointing in the direction of parallel solution methods.

References

- [1] Machenhauer B., 'HIRLAM Documentation Manual', HDM/ORIG, June, 1990
- [2] Mesinger, F. and Arakawa, A., 'Numerical Methods used in Atmospheric Models', Garp Publication Series No.14, WMO/ICSU Joint Organising Committee, 1976
- [3] Haltiner, G.J., and Williams, R.T., 'Numerical Prediction and Dynamic Meteorology', John Wiley and Sons, 1980
- [4] Gustafsson, N., 'The HIRLAM Model', from 'Numerical Methods in Atmospheric Models', VOL.II, pp115-145, 9th-13th September, 1991
- [5] Mc Donald, A, 'Semi-Lagrangian Methods', from 'Numerical Methods in Atmospheric Models', VOL.I, pp257-271, 9th-13th September, 1991
- [6] Simmons, A.J., 'Development of a High Resolution, Semi-Lagrangian Version of the ECMWF Forecast Model', from 'Numerical Methods in Atmospheric Models', VOL.II, pp281-324, 9th-13th September, 1991
- [7] Hoffmann, G.R. & Maretis D.K., 'The Dawn of Massively Parallel Processing in Meteorology', Springer-Verlag, 1990
- [8] Chandy, K.M. & Kesselman C., 'Parallel Programming in 2001', IEEE Software, 0740-7459/91/1100, pp11-20, November 1991
- [9] Applebe, B., Smith, K., & McDowell, C., 'Start/Pat: A Parallel Programming Toolkit', IEEE Software, 0740-7459/89/0700, pp29-38, July 1989
- [10] Kauranne, T., 'An Introduction to Parallel Processing in Meteorology' from 'The Dawn of Massively Parallel Processing in Meteorology', pp12-17, 1990

- [11] Gustafsson, J.L., Montry, G.R., & Benner, R.E.,
'Development of Parallel Methods for a 1024-
Processor Hypercube', SIAM Journal Scientific ~~Stat~~
Comput. 9(1988)4, pp609-638, 1988

- [12] Hoare, C.A.R., 'Communicating Sequential Processes',
from 'Communications of the ACM', 21, pp666-667,
1978

- [13] Wilson, P., 'Parallel processing Comes to PC's'
BYTE, pp213-218, November, 1988

- [14] 'Parallel Fortran Version 2.1.3 User Guide'
3L Limited, November, 1990

- [15] 'Parallel C Version 2.2.2 User Guide'
3L limited, July, 1991

- [16] Machenhauer, B., 'HIRLAM Final Report',
Technical Report No 5, Copenhagen, Dec. 1988

- [17] Bates, J.R., McDonald, A., 'Multiply-upstream,
Semi-Lagrangian advective schemes: analysis and
application to a multi-level primitive equation
model', Monthly Weather Review, Number 110,
pp1831-1842, 1982

- [18] Smolarkiewicz, P.K., ' Nonoscillatory Advection
Schemes', from 'Numerical Methods in Atmospheric
Models',
VOL.I, pp235-256, 9th-13th September, 1991

- [19] INMOS, 'The Transputer development Manual',
Inmos, 1988

- [20] Bailey, Vallance, Vapenikova & Pulford,
'The University of Salford FTN77/386 Reference
Manual', Revision C, August, 1989

- [21] Conte, S.D., deBoor, C., 'Elementary Numerical
Analysis - An Algorithmic Approach', McGraw-Hill,
pp223-233, 1981

- [22] Transtech Graphics Server (TTGS) User Manual,
Ref: TTGMAN0190, Transtech, 1990

- [23] INMOS, 'Getting Started on the Inmos Transputer Development System', Inmos, 1985
- [24] Chandy,K.M., Misra,J., 'Parallel Program Design - A Foundation', Addison-Wesley Publishing Co.,1988
- [25] Hoffmann,G.R., Snelling,D.F., 'Multiprocessing in Meteorological Models', Springer-Verlag, 1988
- [26] Cats,G., Middelkoop,H., Streefland.D., Swierstra,D.,
'A Meteorological Model on a Transputer Network', from 'The Dawn of Massively Parallel Processing in Meteorology', pp47-75, 1990
- [27] Lynch,P., 'Filtered Equations and Filtering Integration Schemes', from
'Numerical Methods in Atmospheric Models', VOL.I, pp119-159, 9th-13th September, 1991
- [28] Lynch,P., McGrath,R., 'Modelling a Severe Storm with the HIRLAM System', 'Technical Report', Irish Meteorological Service, 1989
- [29] Solchenbach,K., Thole,C., 'The Supernum Architecture and its Application to Computational Fluid Dynamics', from 'The Dawn of Massively Parallel Processing in Meteorology', pp214-230, 1990
- [30] Arakawa,A., 'Computational Design for Long-term Numerical Integration of the Equations of Fluid Motion:Two dimensional Incompressible Flow', from 'Journal of Computational Physics', Part 1, pp119-143, 1966
- [31] Robert,A., 'A Semi-Lagrangian and Semi-implicit Numerical Integration Scheme for the Primitive Meteorological Equations', Journal Metero. Soc., Japan, 60, pp319-325, 1982
- [32] Asselin,R.,A., 'Frequency Filter for Time Integration', Monthly Weather Review, 100, pp487-490, 1972

- [33] Charney, J., G., 'Integration of the Primitive and Balance Equations', from 'Proceedings on International Symposium on NWP', Japanese Met. Agency, pp131-152, 1962
- [34] Simmons, A., J., Burridge, D.M., 'An Energy and Angular Momentum Conserving Vertical Finite Difference Scheme and Hybrid Vertical Coordinates', Monthly Weather Review, 109, pp758-766, 1981
- [35] Holton, J.R., 'An Introduction to Dynamic Meteorology', Academic Press, New York and London, 1979
- [36] Willianson, D.L., Swarztrauber, P.N., 'A Numerical Weather Prediction Model - Computational Aspects on the CRAY-1', from Proceedings of the IEEE, Vol.72, No.1, pp56-67, 1984
- [37] Kallberg, P., Gibson, R., 'Lateral Boundary Conditions for a Limited Area Version of the ECMWF Model', WGNE Progress Report, No.14, WMO, Geneva, 1977
- [38] Bengtsson, L., 'Computer Requirements for Atmospheric Modelling', from 'Multiprocessing in Meteorological Models', pp109-116, 1988
- [39] Mill, J., 'How to Tap into Cheap Processing Power', New Scientist, pp50-55, 12th November 1988
- [40] Atkinson, B.W., 'Dynamical Meteorology - An Introductory Selection', Methuen, London and New York, 1981
- [41] O'Neill, M.A., 'Faster than Fast Fourier', BYTE, pp293-300, April 1988
- [42] Lucas, R.F., Blank, T., Tiemann, J.J., 'A Parallel Solution Method for Large Sparse Systems of Equations', from IEEE Transactions on CAD, VOL CAD-6, No 6, pp981-991, 1987

- [43] Montoye,R.K., Lawrie,D.H., 'A Practical Algorithm for the Solution of Triangular Systems on a Parallel Processing System', from IEEE Transactions on Computers, Vol C-31, No 11, pp1076-1082, 1982
- [44] Kuo,H.L., 'Further Studies of the Influence of Cumulus Convection on large-scale flow', Journal of Atmos. Sci., No. 31, pp1232-1240, 1974
- [45] Richardson,L.F.,(1922), 'Weather Prediction by Numerical Process', Cambridge University Press, reprinted by Denver Publications, New York, pp200-240, 1965
- [46] Jones,G.,Goldsmith,M., 'Programming in OCCAM', Prentice Hall International (UK) Ltd., 1988
- [47] Galletly,J., 'OCCAM2', Pitman Publishing, 1990
- [48] O'Leary,D.P., Stewart,G.W., 'Assignment and Scheduling in Parallel Matrix Factorization', Linear Algebra and its Applications No 77, pp275-299, 1986
- [49] O'Leary,D.P., Stewart,G.W., 'Data-Flow Algorithms for Parallel Matrix Computations', Communications on the ACM, Vol 28, No 8, pp840-853, August, 1985
- [50] Lichnewsky,A., Thomasset,F., Eisenbeis,C., 'Automatic Detection of Parallelism in Scientific Programs - Application to Array-Processors', presented at 'Seminar on Parallel Computing', at IBM-Europe Institute, Oberlech, Austria, pp1-19, 1986
- [51] Saltz,J.H., Mirchandaney,R., Crowley,K., 'Run-Time Parallelization and Scheduling of Loops', ICASE, NASA Langley Research Center, March 28, 1990
- [52] Meteorological Office, 'Meteorological Glossary', Sixth Edition, London, HMSO,1991
- [53] Ben-Ari,M., 'Principles of Concurrent and Distributed Programming', published by Prentice Hall International (UK), pp47-73, 1990

- [54] Robert, A., 'A stable numerical integration scheme for the primitive meteorological equations', from Atmos. Ocean, No 19, pp35-46, 1981

GLOSSARY

3L	Computer language software company, Scotland
ADIABATIC	An adiabatic process (thermodynamic) is one in which heat does not enter or leave the system
ADVECTION	Usually refers to the horizontal transfer of an air-mass property by virtue of motion
BAROTROPIC	A hypothetical state of the atmosphere in which surfaces of pressure and density coincide at all levels
BIND	Tasks ports in a transputer environment are bound by a 'CONNECT' statement in the Network configuration language from 3L Ltd.
CHANNEL	One way point to point pathway in transputer memory used by communicating processes
CONFIG/FCONFIG	Utility programs which configure a transputer in accordance with the instruction statements read from a user written configuration file
CORIOLIS FORCE	The apparent acceleration of air particles by virtue of the earth's rotation, with respect to fixed axes on the earth.
CRAY	A supercomputer from the Cray Corporation of America
CSP	Communicating Sequential Process
DELL	PC computer manufacturer, Limerick
DIVERGENCE	Expresses the time rate of depletion of a quantity of air particles per unit volume
DMA	Direct Memory Access
ECMWF	European Centre for Medium Range Weather Forecasts Reading, England.
EMISSIVITY	The ratio of radiation intensity emitted from a surface, to the radiation intensity, at the same wavelength, emitted from a black body at the same temperature[52]
FFT	Fast Fourier Transform

FROUTER	Special task used by the flood-fill configurer to manage the flow of work packets through the transputer network
Geopotential	The potential energy per unit mass of a body due to the gravitational pull of the earth
hPa	Hecto Pascal, a measurement of atmospheric pressure
HIRLAM	High Resolution Limited Area Model
IEEE	Institute of Electrical and Electronics Engineers - Develops standards in the areas of Electrical Engineering and Computing
IMS	INMOS trade name
INMOS	Transputer chip manufacturer
Intel	Computer chip manufacturer
Jet Stream	A fast narrow air current with characteristic vertical and lateral wind shears. A jet stream is usually many thousands of kilometres in width and length. It is located near the Tropopause.
KLON	The HIRLAM program parameter which defines the number of longitude grid points
KLAT	The HIRLAM program parameter which defines the number of latitude grid points
KLEV	The HIRLAM program parameter which specifies the number of vertical levels
LAM	Limited Area weather forecast Model
MEIKO	Computer company set up to exploit the power of the transputer for compute intensive applications[39]
MHz	Mega Hertz - a measure of computer clock speed
MODULA-2	Computer programming language
MSLAB	Parameter defining the width of the slab-area columns used with the computation of the physical parameters of the HIRLAM model
NWP	Numerical Weather Prediction
OCCAM	Parallel programming language specifically designed for transputer environment

Olivetti	Personal Computer (PC) Manufacturer
PDE	Partial Differential Equation
RAM	Random Access Memory
RHS	Right Hand Side
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
ROOT	This terms refers to the main transputer in a network of transputers. It is interfaced with the HOST PC
SLAB-area	The size of the base of a vertical column of atmosphere. Typically, the slab-area includes a few neighbouring latitude grid lines. During each time step the physical parameters are computed for all the slab-areas, by taking one slab-area at a time.
STRATIFORM	Cloud formation in a horizontal sheet or layer
TDS	Transputer Development System
TRAM	TRAnsputer Module
VAX-4200	A computer from Digital Corporation
Von Neumann	Type of computer architecture defined by John Von Neumann in 1944. The design is very much sequentially orientated and requires that each machine instruction is complete before the next may begin
VORTICITY	The three-dimensional property of the field of motion of a fluid
Wind Shear	The rate of change of the wind vector, V , with distance, d , in a specified direction perpendicular to the predominant wind direction.
WORM	A utility program supplied by 3L Limited for exploring transputer networks to determine the type of each transputer and how many there are

APPENDICES

Appendix A : Programs BA.FOR and SLABWR.FOR

Program BA.FOR	A-1
Program SLABWR.FOR	A-3
Subroutine SLABWRITE	A-9

Appendix B : Master/Slave Interpolation Programs

Program BIXM.F77	B-1
Subroutine Send_Bix	B-5
Subroutine Recv_Bix	B-8
Subroutine Bix_Mast	B-11
Program BIXW.F77	B-15
Subroutine BIXINT	B-17
Subroutine HORINT	B-22
Subroutine VERINT	B-26

Appendix C : Sample Master/Slave Configuration Files

BIX.CFG	C-1
FBIX.CFG	C-1
BIX.BAT [Batch Configuration File] ..	C-2
BIXM.LNK	C-2
BIXW.LNK	C-2

Appendix D : Sample Output for One Time Step D-1

APPENDIX A

PROGRAMS BA.FOR and SLABWR.FOR

```

program ba
C-----
C
C   P HALTON, MAY 1992
C
C   PURPOSE
C   -----
C   TO CONVERT BINARY BOUNDARY FIELDS (E.G. 86082500.AAA) TO ASCII
C
C   PARAMETER ( MLON = 38, MLAT = 34, MLEV = 16 )
C   PARAMETER ( MSLAB = 3 , MBDPTS = 8 )
C   PARAMETER ( MLNLT = MLON*MLAT )
C   PARAMETER ( MBDPT2 = 2*MBDPTS + 1 )
C   PARAMETER ( MLNLTB = MLON*MLAT - (MLON-MBDPT2)*(MLAT-MBDPT2) )
C-----
C   PARAMETER ( MBUFIO = 16*MLNLT + 8*MLNLTB )
C   CHARACTER*12 NAM,NAMO
C   COMMON / COMBUF / B(MBUFIO)
C-----
C   COMMON /COMDDR/
C-----
C           INTEGER SECTION           -----
C           ----- FILE SECTION
C   1 NCOOHL,NLDRHL,MLNXHL,MRCLHL,MDRLHL,NTYPHL,NEXPHL,NMDOIHL
C   1,NIDFHL(12)
C           ----- TIME SECTION
C   2,NDTVHL,NSCVHL,NDTBHL,NSCBHL,NFLHHL,NFLSHL,NDORHL
C   2,WIDTHL(13)
C           ----- GRID SECTION
C   3,NPRJHL,NPRCHL,NLOWHL,NLATHL
C           ----- LEVEL SECTION
C   4,NLTPHL,NLEVHL,NPPLHL,NRFLHL
C   4,NLPTHL(40)
C           ----- MULTI LEVEL FIELDS
C   5,NMLFHL,NWMMHL(40),NMPHL(40)
C           ----- SINGLE LEVEL FIELDS
C   6,NSLFHL,NWMSHL(40),NSPTRL(40),NSLTHL(40)
C           ----- REAL SECTION           -----
C           ----- GRID SECTION
C   C,APLOHL,APLAHL,AWESHL,AEASHL,ALALHL,ALAFHL,DLOWHL,DLATHL
C   C,GRIDHL(28)
C           ----- LEVEL SECTION
C   D,ALEVHL(40,4),RLEVHL(4)
C           ----- MULTI LEVEL FIELDS
C   E,STMXHL(40),STMYHL(40),STMZHL(40)
C           ----- SINGLE LEVEL FIELDS
C   F,STSXHL(40),STSYHL(40),SLEVHL(40,4)
C           ----- RESERVE
C   G,RSRVHL(150)
C
C   DIMENSION DDRHL(1000),NDDRHL(1000)
C   EQUIVALENCE (DDRHL,NDDRHL,NCOOHL)
C   NLON=MLON
C   NLAT=MLAT
C***** BINARY FILE
C   WRITE(5,91)
91  FORMAT(' INPUT BINARY FILE NAME: ',5)
C   READ(5,93)NAM
C   WRITE(5,92)
92  FORMAT(' OUTPUT ASCII FILE NAME: ',5)
C   READ(5,93)NAMO

```

```

93  FORMAT(A12)
    OPEN(UNIT=10,FILE=NAM,FORM='UNFORMATTED',ACCESS='SEQUENTIAL')
    OPEN(UNIT=11,FILE=NAMO,FORM='FORMATTED',ACCESS='SEQUENTIAL')
C*****
    READ (UNIT=10,ERR=810) NCODHL,NLDRHL,(NDDRHL(J),J=3,NLDRHL)
    WRITE(11,11)(NDDRHL(J),J=1,290)
    WRITE(11,10)(DDRHL(J),J=291,NLDRHL)
10  FORMAT(1X,5E18.12)
11  FORMAT(1X,10I9)
C-----
    ILAT1 = NLAT
    ILAT2 = 1
    ILATI = -1
    DO 2000 JY=ILAT1,ILAT2,ILATI
      print*,' converting for latitude ...',JY
      READ (UNIT=10,ERR=810) (B(J),J=1,MDRLHL)
      WRITE(11,10)(B(J),J=1,MDRLHL)
2000 CONTINUE
C-----
    STOP
810  WRITE(5,88)
88  FORMAT(' ERROR IN READING 10')
    STOP
    END

```



```

      program slabrw
c
c      P. Halton
c
c      April 1992
c
c      Read in a slab file. Interpolate horizontally and or vertically,
c      from 16 layers to 7 layers, and output the new slab file.
c
c-----
c
c      PARAMETERS TO DIMENSION INPUT FIELDS.
c
c      PARAMETER(jplon=38,jplat=34,JPELEV=16)
c      PARAMETER (JPMLF=4,JPSLF=25)
c      PARAMETER (JPFLDS=JPMLF*JPELEV+JPSLF)
c
c-----
c
c      PARAMETERS TO DIMENSION OUTPUT FIELDS.
c
c      PARAMETER(kplon=38,kplat=34,KPELEV=7)
c      PARAMETER (KPMLF=4,KPSLF=25)
c      PARAMETER (KPFLDS=KPMLF*KPELEV+KPSLF)
c
c-----
c
c      Names of files to be read in and written out.
c
c      character*30 filin,filout
c
c-----
c
c      Declare Data Description Record, DDR, OF INPUT SLAB FILE
c
c      COMMON /COMDDR/
c      -----          INTEGER SECTION          -----          FILE SECTION
c      1 NCOOHL,NLDRHL,NLNXHL,MRCLHL,MDRLHL,NTYPHL,NEXPHL,NMDOIHL
c      1,NIDFHL(12)
c      -----          -----          TIME SECTION
c      2,NDTVHL,NSCVHL,NDTBHL,NSCBHL,NFLHHL,NFLSHL,NDORHL
c      2,NIDTHL(13)
c      -----          -----          GRID SECTION
c      3,NPRJHL,NPRCHL,NLONHL,NLATHL
c      -----          -----          LEVEL SECTION
c      4,NLTPHL,NLEVHL,NPPLHL,NRFLHL
c      4,NLPTHL(40)
c      -----          -----          MULTI LEVEL FIELDS
c      5,NMLFHL,NWMMHL(40),NMPTHL(40)
c      -----          -----          SINGLE LEVEL FIELDS
c      6,NSLFHL,NWMSHL(40),NSPTHL(40),NSLTHL(40)
c      -----          -----          REAL SECTION
c      -----          -----          GRID SECTION
c      C,APLOHL,APLAHL,AWECHL,AEASHL,ALALHL,ALAFHL,DLONHL,DLATHL
c      C,GRIDHL(28)
c      -----          -----          LEVEL SECTION
c      D,ALEVHL(40,4),RLEVHL(4)
c      -----          -----          MULTI LEVEL FIELDS

```

```

E,STMXHL(40),STMYHL(40),STMZHL(40)
C ----- SINGLE LEVEL FIELDS
F,STSXHL(40),STSYHL(40),SLEVHL(40,4)
C ----- RESERVE
G,RSRVHL(150)
DIMENSION DDRHL(1000),NDDRHL(1000)
EQUIVALENCE (DDRHL,NDDRHL,NCODHL),(NMDIHL,AMD IHL)
C
C
C -----
C
C MAIN BUFFERS.
C
COMMON /cmbf/
1 BUFIN (JPLON,JPLAT,JPFLDS), BUFOUT(KPLON,KPLAT,KPFLDS)
C
REAL XBUFOUT(KPLON*KPFLDS)
C
C -----
C
logical lvert, lbuf
dimension ahals(jpelev+1),bhals(jpelev+1)
dimension ahalf(kpelev+1),bhalf(kpelev+1)
C
data iluin /11/
data iluout /12/
C
data nlev /7/
data lvert/.true./
data lbuf/.true./

data ahalf
1 /0.0000000000,
1 0.0000000000, 0.0000000000, 0.0000000000,
1 0.0000000000, 0.0000000000, 0.0000000000,
6 0.0000000000/

C
data bhalf
1 /0.0000000000,
2 0.1431978281,
3 0.2865701778, 0.4280178285, 0.5714721581,
4 0.7148338594, 0.8572660866, 1.0000000000/

C
data ahals
1 /0.0000000000, 0.0000000000, 0.0000000000,
2 0.0000000000, 0.0000000000, 0.0000000000,
3 0.0000000000, 0.0000000000, 0.0000000000,
4 0.0000000000, 0.0000000000, 0.0000000000,
5 0.0000000000, 0.0000000000, 0.0000000000,
6 0.0000000000, 0.0000000000/

data bhals
1 /0.0000000000, 0.0472790301, 0.0968019664,
2 0.1503064334, 0.2089847624, 0.2734752297,
3 0.3438725471, 0.4197154641, 0.5000003576,
4 0.5831676722, 0.6671144962, 0.7491835356,
5 0.8261721134, 0.8943251371, 0.9493408203,
6 0.9863661528, 1.0000000000/
C
C -----
C

```

```

C get name of the input file.
  PRINT 112
112  format(' Name of forecast file: ', $)
     read(5,223) filin
223  format(a)
-----
C get name of the output file
c
  PRINT 102
102  format(' Name of conversion file: ', $)
     read(5,203) filout
203  format(a)
c
c open the input and output files.
c
c
  OPEN(UNIT=ILUIN,FILE=filin,
+   FORM='UNFORMATTED',status='old')
C
c open the output file.
c
  OPEN(UNIT=ILUOUT,FILE=filout,
+   FORM='UNFORMATTED',status='unknown')
C
C -----
C
C* 2.0 READ SLAB FILE DDR
C
  READ(ILUIN) NCODHL,NLDRHL,(DDRHL(J),J=3,NLDRHL)

  if(abs(aplohl).le.1.0e-4) then
    write(6,*) 'old aplohl ',aplohl
    write(6,*) 'changing aplohl'
    aplohl = 180.0 ljh
    write(6,*) 'new aplohl ',aplohl
  endif
C
C -----
C
C FIND OUT TOTAL NUMBER OF FIELDS IN THE SLAB FILE
C
  NUMFLD = NMLFHL*NLEVHL + NSLFHL
C
C -----
c Check on dimensions.
c
  if( (jplon .ne.nlonhl) .or.
1  (jplat .ne.nlathl) .or.
2  (jpelev.ne.nlevhl) ) then
    print*, ' jplon, nlonhl ',jplon, nlonhl
    print*, ' jplat, nlathl ',jplat, nlathl
    print*, ' jpelev, nlevhl ',jpelev, nlevhl
    stop 'stop: dimensions incorrect in slabrw'
  endif
c
  if( (jplon .ne.nlonhl) .or.
1  (jplat .ne.nlathl) .or.
2  (kpelev.ne.nlev ) ) then
    print*, ' jplon, nlonhl ',jplon, nlonhl
    print*, ' jplat, nlathl ',jplat, nlathl

```

```

        print*, ' kpelev, nlev ', kpelev, nlev

        stop 'stop: dimensions incorrect in slabrw'
    endif

c
c -----
c
c* 4.0 IF NECESSARY, CHANGE SOUTH-NORTH LOOP ORDER TO
c*      NORTH-SOUTH LOOP ORDER
c*      IN THE READING OF THE SLAB FILE
c
    write(*,*) ' dlathl --- > ', dlathl
    IF( DLATHL.LT.0. ) THEN
    write(*,*) ' first dlathl option - no flip '
        JLAT1 = 1
        JLAT2 = NLATHL
        JLOOP = 1
    ELSE
    write(*,*) ' second dlathl option - flip '
        JLAT1 = NLATHL
        JLAT2 = 1
        JLOOP = -1
        DLATHL = - DLATHL
        ZSAVE = ALAFHL
        ALAFHL = ALALHL
        ALALHL = ZSAVE
    ENDIF

c
c -----
c
c 5.0 READ THE SLAB FILE DATA TO BUFFER
c
    DO 520 JLAT=JLAT1,JLAT2,JLOOP
    READ(ILUIN)((BUFIN(JLON,JLAT,JFLD),JLON=1,NLONHL),
    X                      JFLD=1,NUMFLD)
    520 CONTINUE

c
c -----
c
    IF( (.not.lbuf).and.(.not.lvert) ) THEN

c
        DO 110 JLAT=JLAT1,JLAT2,JLOOP
            DO JFLD=1,NUMFLD
                DO JLON=1,NLONHL
                    BUFOUT(JLON,JLAT,JFLD) = BUFIN (JLON,JLAT,JFLD)
                END DO
            END DO
        110 CONTINUE

c
    ENDIF

c
c -----
c
    IF( lbuf.or.lvert) THEN

c
c* 6.0 Modify the pointers of the ddr for sorting
c*      the input data to fit vertical interpolation
c*      extract A:S and B:S of ETA full levels
c
        DO 610 JMLF=1,NMLFHL
        610 NMPTHL(JMLF) = NMPTHL(JMLF)/NLONHL
    ENDIF

```

```

DO 620 JLEV=1,NLEVHL
620   NLPTHL(JLEV) = NLPTHL(JLEV)/NLOWHL
DO 630 JSLF=1,NSLFHL
630   NSPTHL(JSLF) = NSPTHL(JSLF)/NLOWHL
C
c     Vertical interpolation.
c     The output is in bufout.
c     Important: The DDR is changed in vert.
c
      if(lvert) then .
          call vert(lvert,nlev,ahalf,bhalf)
          NUMFLD = NMLFHL*NLEVHL + NSLFHL
      endif
C
C-----
C
C*   7.0   Modify the DDR
C
      NTYPHL = 0
      DO 710 JMLF=1,NMLFHL
710         NMPTHL(JMLF) = NMPTHL(JMLF)*NLOWHL
      DO 720 JLEV=1,NLEVHL
720         NLPTHL(JLEV) = NLPTHL(JLEV)*NLOWHL
      DO 730 JSLF=1,NSLFHL
730         NSPTHL(JSLF) = NSPTHL(JSLF)*NLOWHL
      MDRLHL = ( NMLFHL*NLEVHL + NSLFHL ) * NLOWHL
      MRCLHL = MAX(NLDRHL,MDRLHL)
C
C-----
      endif
C
C-----
C
C*   8.0   Write the model slab file DDR
C
      WRITE(ILUOUT) NCOOHL,NLDRHL,(DDRHL(J),J=3,NLDRHL)
C
c     WRITE(6,*) ' TEST-PRINT DDR'
c     WRITE(6,*)(MDDRHL(JL),JL=1,200)
C-----
C
C*   10.0   WRITE THE MODEL SLAB RECORDS
C
      write(6,*) ' jplon jplat jplon*jplat ',jplon,jplat,jplon*jplat
      irecl=NUMFLD*NLOWHL
      DO 1000 JLAT=JLAT1,JLAT2,JLOOP
          k=0
          DO JFLD=1,NUMFLD
              DO JLON=1,NLOWHL
                  k=k+1
                  XBFOUT(k)=BUFOUT(JLON,JLAT,JFLD)
              END DO
          END DO
          call slabwrite(ILUOUT,xbfout,irecl)
1000 CONTINUE
C
C-----
C
C*   11.0   CLOSE THE SLAB FILE - FINISHED
C

```

ENDFILE ILUOUT

C

C

C

C

END

```

      subroutine slabwrite(ILUOUT,xbfout,irecl)
      real xbfout(irecl)
      write (ILUOUT) xbfout
      end

c
      subroutine vert(lvert,mlev,a,b)
c
c-----
c interpolate vertically.
c
c vertical coordinate:  p =  a + b*ps
c kfield = 1 for a T-field.
c kfield = 2 for a U-field.
c kfield = 3 for a V-field.
c kfield = 4 for a q-field.
c
c-----
c
c PARAMETERS TO DIMENSION INPUT FIELDS.
c
      PARAMETER(JPlon=38,JPlat=34,JPELEV=16)
      PARAMETER (JPMLF=4,JPSLF=25)
      PARAMETER (JPFLDS=JPMLF*JPELEV+JPSLF)
c
c-----
c
c PARAMETERS TO DIMENSION OUTPUT FIELDS.
c
      PARAMETER(KPlon=38,KPlat=34,KPELEV=7)
      PARAMETER (KPMLF=4,KPSLF=25)
      PARAMETER (KPFLDS=KPMLF*KPELEV+KPSLF)
c
c-----
c
c MAIN BUFFERS.
c
      COMMON /cmbf/
      1 BUFIN (JPLOM,JPLAT,JPFLDS), BUFOUT(KPLOM,KPLAT,KPFLDS)
c
c-----
c
c Data Description Record, DDR OF INPUT SLAB FILE
c
c
c
      COMMON /COMDDR/
c
c-----          INTEGER SECTION          -----
c-----          FILE SECTION
      1 NCOOHL,NLDRHL,NLNXHL,MRCLHL,MDRLHL,NTYPHL,NEXPHL,MNDIHL
      1,NIDFHL(12)
c-----          TIME SECTION
      2,NDTVHL,NSCVHL,NOTBHL,NSCBHL,NFLHHL,NFLSHL,NDORHL
      2,NIDTHL(13)
c-----          GRID SECTION
      3,NPRJHL,NPRCHL,NLOWHL,NLATHL
c-----          LEVEL SECTION
      4,NLTPHL,NLEVHL,NPPLHL,MRFLHL
      4,NLPTHL(40)
c-----          MULTI LEVEL FIELDS
      5,NMLFHL,NMMHHL(40),NMPHHL(40)
c-----          SINGLE LEVEL FIELDS

```

```

6,NSLFHL,NWMSHL(40),NSPRTL(40),NSLTHL(40)
C ----- REAL SECTION -----
C ----- GRID SECTION -----
C,APLOHL,APLAHL,AWESHL,AEASHL,ALALHL,ALAFHL,DLOWHL,DLATHL
C,GRIDHL(28)
C ----- LEVEL SECTION -----
D,ALEVHL(40,4),RLEVHL(4)
C ----- MULTI LEVEL FIELDS -----
E,STMXHL(40),STMYHL(40),STMZHL(40)
C ----- SINGLE LEVEL FIELDS -----
F,STXHL(40),STSYHL(40),SLEVHL(40,4)
C ----- RESERVE -----
G,RSRVHL(150)
DIMENSION DDRHL(1000),NDDRHL(1000)
EQUIVALENCE (DDRHL,NDDRHL,NCODHL),(NMDIHL,AMDHL)
C
C
C-----
C
      logical lvert
C-----
      real y(JPELEV)
      dimension ps(jplon,jplat)
      real x(KPELEV)
      real a(KPELEV+1),b(KPELEV+1)
      integer level(KPELEV)
      data ps_ref/100000./
C-----
C
1      format(' kfield=',i10)
      if (mlev.ne.KPELEV) stop 'stop in vert: mlev is wrong size'
C
C
      nlev=mlev
      c9o16 = 9./16.
      c1o16 = 1./16.
C
C Compute ps.
C
      kfield = 0
1000   kfield = kfield + 1
      if((kfield.eq.1).or.(kfield.eq.4)) go to 11
      if(kfield.eq.2) go to 12
      if(kfield.eq.3) go to 13
      if (kfield.gt.4) go to 1001
C
11     continue
      print 1, kfield
      do 5 i=1,jplon
      do 5 j=1,jplat
      ps(i,j) = bufin(i,j,2)
5      continue
      go to 14
C
12     continue
      print 1, kfield
      do 6 i=1,jplon
      do 6 j=1,jplat
      IF(I.GE.jplon-1) GO TO 101

```



```

        IF(I.LE.1      ) GO TO 101
        ps(i,j) = c9o16*( bufin(i ,j,2) + bufin(i+1,j,2) )
1         - c1o16*( bufin(i-1,j,2) + bufin(i+2,j,2) )
        GO TO 6
101      CONTINUE
        IF(I.EQ.jplon) GO TO 201
        ps(i,j) = 0.5*( bufin(i,j,2) + bufin(i+1,j,2) )
        GO TO 6
201      CONTINUE
        ps(i,j) = bufin(jplon,j,2)
6        continue
        go to 14
c
13       continue
c        Set jlatm1 = jplat - 1
c
        jlatm1 = jplat - 1
        print 1, kfield
        do 7 i=1,jplon
            do 7 j=1,jlatm1
                IF(J.GE.jplat) GO TO 102
                IF(J.LE.1      ) GO TO 102
                ps(i,j) = c9o16*( bufin(i,j ,2) + bufin(i,j+1,2) )
1                 - c1o16*( bufin(i,j-1,2) + bufin(i,j+2,2) )
                GO TO 7
102      CONTINUE
                IF(J.EQ.jplat) GO TO 202
                ps(i,j) = 0.5*( bufin(i,j,2) + bufin(i,j+1,2) )
                GO TO 7
202      CONTINUE
                ps(i,j) = bufin(i,jplat,2)
7        continue
14       continue
c
c
c        Loop over all points.
c
        do 130 ix=1,jplon
            do 130 iy=1,jplat
c
c        initialize coef.
c
c        Compute the old pressure and put it in array y.
c
        do 15 k=1,NLEVHL
            y(k)=alevhl(k,1)+alevhl(k,2)*ps(ix,iy)
15       continue
c
c        Compute the new pressure and put it in array x.
c        Also, initialize level(i).
c
        do 20 i = 1,nlev
            x(i)=(a(i)+a(i+1) + (b(i+1)+b(i))*ps(ix,iy) )/2.
            level(i) = 0
20       continue
c
c        Compute the coefficients for the Lagrange linear interpolation
c        and replace it in array x.
c
        do 25 i=1,nlev

```

```

do 30 k=1,NLEVHL-1
  if ( x(i).gt.y(k) .and. x(i).le.y(k+1) ) then
    x(i)=1-(x(i) y(k))/(y(k+1)-y(k))
    level(i)=k
  endif
30 continue

c
c   Do the following: If pnew(i) is
c   less than pold(1) then put level(i) = 1 and x(i) = 1.
c   That is, if the new level is above the old top level
c   put the field at the new level equal to the field at
c   the old top level.
c   if (level(i).eq.0 .and. x(i).le.y(1) ) then

    level(i)=1
    x(i)=1.
  else

c
c   Do the following: If pnew(i) is
c   greater than than pold(NLEVHL) then put level(i) = NLEVHL-1 and
c   x(i) = 0. That is, if the new level is below the old bottom level
c   put the field at the new level equal to the field at the old
c   bottom level.
c
    if (level(i).eq.0 .and. x(i).gt.y(NLEVHL) ) then
      level(i)=NLEVHL-1
      x(i)=0.
    endif
  endif
endif
if( (ix.eq.10) .and. (iy.eq.10) )
1   print*, level(i),x(i),y(level(i)),y(level(i)+1)
-----

25 continue

c
c   interpolate each field
c
k = kfield
do 110 j=1,mlev
  lev1=( NLPTHL(level(j))+1+NMPTHL(k) )
  lev2=( NLPTHL(level(j)+1)+1+NMPTHL(k) )

c
  lev3=j+2+(k-1)*mlev
  bufout(ix,iy,lev3) = bufin(ix,iy,lev1)*x(j)
1   + bufin(ix,iy,lev2)*(1-x(j))

c
  if( (ix.ne.10) .or. (iy.ne.10) ) go to 110
  print 234, k,j,lev1,lev2,lev3
234  format( ' k,j,lev1,lev2,lev3=',5i10)
  b3 = bufout(ix,iy,lev3)
  b1 = bufin (ix,iy,lev1)
  b2 = bufin (ix,iy,lev2)
  xj = x(j)
  print 456, b1,b2,b3,xj,ix,iy
456  format( ' b1,b2,b3,x=',4e13.3,' i,j=',2i4)
110  continue

130  continue
c

```

```

        go to 1000
1001  continue
c
c      modify the data description record, since the length
c      of each slab will be reduced from a factor of 16 to the
c      new factor of 7
c
      do 140 j=1,mlev
          NLPTHL(j)=(j-1)
          ALEVHL(j,2)=(b(j)+b(j+1))/2.
          ALEVHL(j,1)=(a(j)+a(j+1))/2.
140  continue
      do 145 j=1,NMLFHL
          NMPTHL(j)=(2+(j-1)*mlev)
145  continue
      NLEVHL=mlev
c
c      put the new single level in the buffer array again (newposition)
c
      do 160 j=1,NSLFHL
          l=NSPTHL(j) + 1
          if (j.gt.2) then
              NSPTHL(j)=1+mlev*NMLFHL+j-2
          else
              NSPTHL(j)=j-1
          endif
          k=NSPTHL(j)+1
          do 170 ix=1,jplon
              do 170 iy=1,jplat
                  bufout(ix,iy,k)=bufin(ix,iy,l)
170  continue
160  continue
      return
      end

```

APPENDIX B

MASTER/SLAVE INTERPOLATION PROGRAMS
INCLUDING BIXM.F77 AND BIXW.F77.

```

program bixM
-----
c          bixM.f77
c          P halton
c          11-June-1992
-----
c          This is a test program which will be incorporated
c          into the main HIRLAM program on the transputer to
c          enable the task CALPOR to perform its multi-level
c          calls to BIXINT in parallel, one call for each of
c          the vertical levels.
c
c          The application is configured by FCONFIG the
c          flood-fill configuration utility
-----
c
c          This test consists of two tasks:
c
c          (a) bixM.f77 -- this Master program will run on the
c                   the ROOT transputer.
c          (b) bixW.f77 -- the slave(worker) task which will run
c                   on all transputers including the root.
c
c          Task bixM consists of three THREADS:
c
c          (1) the MAIN thread
c          (2) the SEND_BIX thread
c          (3) the RECV_BIX thread
c
c          INPUT PARAMETERS:
c          -----
c
c          *KLON*      NUMBER OF GRIDPOINTS IN X-DIRECTION
c          *KLAT*      NUMBER OF GRIDPOINTS IN Y-DIRECTION
c          *KLEV*      NUMBER OF VERTICAL LEVELS
c          *KPBPTS*    NUMBER OF PASSIVE BOUNDARY LINES
c          *KSLPQI*    NUMBER OF ITERATIONS
c          *KSLINT*    TYPE OF INTERPOLATION FOR EACH ITERATION
c          *PARGA*     3 DIM. ARGUMENT FOR CALCULATION OF PALFA
c          *PARGB*     3 DIM. ARGUMENT FOR CALCULATION OF PBETA
c          *PARGG*     3 DIM. ARGUMENT FOR CALCULATION OF PGAMA
c          *L3DIM*     TRUE IF 3-DIMENSIONAL TRAJECTORIES
c          *LFGUES*    TRUE IF FIRST GUESS OF PALFA,PBETA AND PGAMA ARE
c                   TAKEN AS VALUES FROM THE PREVIOUS TIMESTEP
c
c          OUTPUT PARAMETERS:
c          -----
c
c          *PALFA*     DISPLACEMENT IN X-DIRECTION (NUMBER OF GRIDPOINTS)
c          *PBETA*     DISPLACEMENT IN Y-DIRECTION (NUMBER OF GRIDPOINTS)
c          *PGAMA*     DISPLACEMENT IN VERT. DIRECTION (NUMBER OF LEVELS)
c
c          EXTERNALS.
c          -----
c
c          *VCOPY*     COPY OF DATA ARRAY
c          *BIXINT*    INTERPOLATION BETWEEN GRIDPOINTS
c          -----

```

```

C*  DECLARATION OF GLOBAL PARAMETERS
C  -----
C
C*  PARAMETER STATEMENTS FOR DEFINITION OF INTEGRATION AREA
C
C    PARAMETER ( KLONG = 38, KLAT = 34, KLEV = 07 )
C
C    PARAMETER ( MSLAB = 3 , MBDPTS = 8 )
C
C*  PARAMETER STATEMENTS FOR ONE HORIZONTAL INDEX
C
C    PARAMETER ( MLNLT = KLONG*KLAT )
C
C-----
C
C    include 'paramk.inc'
C    include 'paramst.inc'
C
C    INTEGER KSLINT(KSLPQI),
C    *      KP(KLONG*KLAT), KQ(KLONG*KLAT), KR(KLONG*KLAT)
C    REAL  PARGA(KLONG*KLAT,KLEV), PARGB(KLONG*KLAT,KLEV),
C    *      PARGG(KLONG*KLAT,KLEV),
C    *      PALFA(KLONG*KLAT,KLEV), PBETA(KLONG*KLAT,KLEV),
C    *      PGAMA(KLONG*KLAT,KLEV),
C    *      PALFH(KLONG*KLAT), PBETH(KLONG*KLAT), PGAMH(KLONG*KLAT)
C    LOGICAL L3DIM , LFGUES, complete
C
C    include 'thread.inc'
C    include 'chan.inc'
C
C    external send_bix, recv_bix
C
C    Channel for communicating with SEND_BIX thread
C    integer ichan
C    common /internal_chan/ ichan
C
C    Common block for communication with RECV_BIX thread
C    integer level_done
C    common /level_control/ level_done
C
C    Set aside work space for SEND_BIX and RECV_BIX threads
C
C    integer send_ws( kbytes ), recv_ws( kbytes )
C    integer ichanaddr, level_tally, nargs
C
C    the following common area will be used by BIXM, SEND_BIX
C    and RECV_BIX threads
C
C    common /bix_data/
C    I  KP,   KQ,   KR,
C    R  PARGA, PARGB, PARGG,
C    R  PALFA, PBETA, PGAMA,
C    R  PALFH, PBETH, PGAMH
C
C-----
C
C    get sample data set which was trapped during a sequential run of
C    the HIRLAM model on the VAX-4200
C

```

```

jiter = 1
jk = 1
nargs = kparams
nsize = kbytes
LFGUES = .TRUE.

C   Ensure that no other threads are using the Run Time Library (RTL)
c
call f77_thread_use_rtl

CALL rdBIXINT (
I   KLON,KLAT,KLEV,JK,1,KSLINT(JITER),
I   IWEST,IEAST,ISOUTH,INORTH,
I   KP(1),KQ(1),KR(1),
R   PARGA(1,1),PALFA(1,JK),PALFH(1),PBETH(1),PGAMH(1),
R   PALFA(1,JK),PBETA(1,JK),PGAMA(1,JK),
L   L3DIM)

C   Finished with the Run Time Library (RTL), so free it...
call f77_thread_free_rtl

c
c   Init chan for communication with SEND_BIX
C
ichanaddr = f77_chan_address (ichan)
call f77_chan_init (ichanaddr)

c
c   Start the other threads...
c
call f77_thread_start( SEND_BIX, send_ws, nsize,
1   f77_thread_urgent,
2   nargs,
I   JK,1,KSLINT(1),
I   IWEST,IEAST,ISOUTH,INORTH,
L   L3DIM,LFGUES )

call f77_thread_start( RECV_BIX, recv_ws, nsize,
1   f77_thread_urgent,
2   nargs,
I   JK,1,KSLINT(1),
I   IWEST,IEAST,ISOUTH,INORTH,
L   L3DIM,LFGUES )

level_done = 0

c
c   Ask SEND_BIX thread to begin processing the data
c
call f77_chan_out_message(4, klev, ichanaddr)

level_tally = 0

c   This Master Thread just controls the other threads' work

do while( level_done .lt. klev )
c   -----
c   Force current MAIN thread to be momentarily unable to execute
c   in order to allow another thread at the same or lower priority
c   to resume execution in its place. Eventually, after say, one
c   timer tick the MAIN thread will resume.....
c   -----
do while (level_done .eq. level_tally)

```

```
        call f77_thread_deschedule
    end do
    level_tally = level_done  I updated by RECV_BIX
end do
```

```
c    return  I will be used when this routine is part of model
end
```

```
c-----
```

```
c
```



```

c-----
c          SEND_BIX.FOR
c
c          P Halton  IMS
c          14-June-1992
c-----
c
c          SUBROUTINE send_bix (
c          I  KPBPTS, KSLPQI, KSLINT,
c          I  IWEST, IEAST, ISOUTH, INORTH,
c          L  L3DIM, LFGUES )
c
c          include 'paramk.inc'
c
c          INTEGER KSLINT(KSLPQI),
c          *      KP(KLON*KLAT), KQ(KLON*KLAT), KR(KLON*KLAT)
c          REAL PARGA(KLON*KLAT,KLEV), PARGB(KLON*KLAT,KLEV),
c          *      PARGG(KLON*KLAT,KLEV),
c          *      PALFA(KLON*KLAT,KLEV), PBETA(KLON*KLAT,KLEV),
c          *      PGAMA(KLON*KLAT,KLEV),
c          *      PALFH(KLON*KLAT), PBETH(KLON*KLAT), PGAMH(KLON*KLAT)
c          LOGICAL L3DIM , LFGUES, complete
c
c          C
c          C*  DECLARATION OF GLOBAL PARAMETERS
c          C-----
c
c          the following common area will be used by BIXM, SEND_BIX
c          and RECV_BIX threads
c
c          common /bix_data/
c          I  KP,  KQ,  KR,
c          R  PARGA, PARGB, PARGG,
c          R  PALFA, PBETA, PGAMA,
c          R  PALFH, PBETH, PGAMH
c
c          include 'chan.inc'
c          include 'net.inc'
c
c          Channel for communicating with MAIN thread
c          integer ichan
c          common /internal_chan/ ichan
c
c          integer ichanaddr
c          ichanaddr = f77_chan_address( ichan)
c
c          call f77_thread_use_rtl
c          write(6,*)'send_bix: started'
c          call f77_thread_free_rtl
c
c

```

```

C
1044 continue
C
C      Wait for instructions from MAIN
C
C      call f77_chan_in_message( 4, levs_to_do, ichanaddr )
C
C      call f77_thread_use_rtl
C
C      write(6,*)' send_bix: levels to do = ',levs_to_do
C
C      Now prepare to send off the LEVELS to be processed by
C      BIXINT, the worker task. The value of KLEV should be
C      sufficient to determine the LEVEL being passed.....
C
C
C      ILNLT = KLON*KLAT
C      IWEST = KPBPTS + 2
C      ISOUTH = KPBPTS + 2
C      IEAST = KLON - 2 - KPBPTS
C      INORTH = KLAT - 2 - KPBPTS
C
C
C*     CHECK NUMBER OF ITERATIONS
C     -----
C
C     IF (KSLPQI.LT.1) THEN
C       WRITE(6,'(/,1X,'IN BIXM: KSLPQI MUST BE .GE. 1''')'
C       STOP
C     ENDIF
C     call f77_thread_free_rtl
C
C
C*     COMPUTE FIRST GUESS OF PALFA, PBETA AND PGAMA
C     -----
C
C     IF (LFGUES) THEN
C
C       DO 100 JK=1,levs_to_do    ! KLEV
C       DO 100 JL=1,ILNLT
C         PALFA(JL,JK) = 0.5*PALFA(JL,JK)
C         PBETA(JL,JK) = 0.5*PBETA(JL,JK)
C         PGAMA(JL,JK) = 0.5*PGAMA(JL,JK)
100    CONTINUE
C
C     ELSE
C
C       CALL VCOPY ( ILNLT*KLEV,PARGA(1,1),1,PALFA(1,1),1 )
C       CALL VCOPY ( ILNLT*KLEV,PARGB(1,1),1,PBETA(1,1),1 )
C       CALL VCOPY ( ILNLT*KLEV,PARGG(1,1),1,PGAMA(1,1),1 )
C
C     ENDIF
C

```

```

C
C* LOOP OVER VERTICAL LEVELS and LOOP OVER ALL ITERATIONS
C -----
C
DO 1000 JK=1,levs_to_do
DO 200 JITER=1,KSLPQI

E
C* INTERPOLATION IN *BIXINT*
C -----
C
CALL BIX_MAST (
I JK,1,KSLINT(JITER),
I IWEST,IEAST,ISOUTH,INORTH,
I KP(1),KQ(1),KR(1),
R PARGA(1,1),PALFA(1,JK),PALFH(1),PBETH(1),PGAMH(1),
R PALFA(1,JK),PBETA(1,JK),PGAMA(1,JK),
L L3DIM)

CALL BIX_MAST (
I JK,2,KSLINT(JITER),
I IWEST,IEAST,ISOUTH,INORTH,
I KP(1),KQ(1),KR(1),
R PARGB(1,1),PBETA(1,JK),PALFH(1),PBETH(1),PGAMH(1),
R PALFA(1,JK),PBETA(1,JK),PGAMA(1,JK),
L L3DIM)

IF (L3DIM)
* CALL BIX_MAST (
I JK,2,KSLINT(JITER),
I IWEST,IEAST,ISOUTH,INORTH,
I KP(1),KQ(1),KR(1),
R PARGG(1,1),PGAMA(1,JK),PALFH(1),PBETH(1),PGAMH(1),
R PALFA(1,JK),PBETA(1,JK),PGAMA(1,JK),
L L3DIM)
C
I = 966
ali =palfa(i,jk)
bei =pbeta(i,jk)
gai =pgama(i,jk)
print 123, ali,bei,gai,jk,jiter
123 format(' alfa,beta,gama=',3e13.6,' jk,jitr=',2i4)
200 CONTINUE
C
1000 CONTINUE
C
goto 1044
END
c

```

```

c-----
c          RECV_BIX.FOR
c          P halton
c          11-June-1992
c-----
c
c          SUBROUTINE RECV_BIX (
c          I  KPBPTS , KSLPQI , KSLINT,
c          I  IWEST,IEAST,ISOUTH,INORTH,
c          L  L3DIM, LFGUES )
c
c
c          Master Task bixM consists of three THREADS:
c
c          (1) the MAIN thread
c          (2) the SEND_BIX thread
c          (3) the RECV_BIX thread --- this routine
c
c-----
c*  DECLARATION OF GLOBAL PARAMETERS
c-----
c
c*  PARAMETER STATEMENTS FOR DEFINITION OF INTEGRATION AREA
c
c          PARAMETER ( MLON = 38, MLAT = 34, MLEV = 07 )
c
c          PARAMETER ( MSLAB = 3 , MBDPTS = 8 )
c
c*  PARAMETER STATEMENTS FOR ONE HORIZONTAL INDEX
c
c          PARAMETER ( MLNLT = MLON*MLAT )
c-----
c
c          include 'paramk.inc'
c          include 'parbix.inc'
c
c          INTEGER KSLINT(KSLPQI),
c          *      KP(KLON*KLAT), KQ(KLON*KLAT), KR(KLON*KLAT)
c          REAL PARGA(KLON*KLAT,KLEV), PARGB(KLON*KLAT,KLEV),
c          *      PARGG(KLON*KLAT,KLEV),
c          *      PALFA(KLON*KLAT,KLEV), PBETA(KLON*KLAT,KLEV),
c          *      PGAMA(KLON*KLAT,KLEV),
c          *      PALFH(KLON*KLAT), PBETH(KLON*KLAT), PGAMH(KLON*KLAT)
c          LOGICAL L3DIM , LFGUES, complete
c
c
c*  DECLARATION OF GLOBAL PARAMETERS
c-----
c
c          the following common area will be used by BIXM, SEND_BIX
c          and RECV_BIX threads
c
c          common /bix_data/
c          I  KP,  KQ,  KR,
c          R  PARGA, PARGB, PARGG,
c          R  PALFA, PBETA, PGAMA,
c          R  PALFH, PBETH, PGAMH

```

```

include 'net.inc'
include 'command.inc'

c
c   Common block for communication with RECV_BIX thread
integer level_done
common /level_control/ level_done

c
c
c   level_done = 0
call f77_thread_use_rtl
write(6,*)'recv_bix: started'
call f77_thread_free_rtl

C
C -----
100 continue

c
c   Wait here until a new packet arrives back from a WORKER
c
c   complete = .false.
kpac = 0
do while ( .not. complete )
    kpac = kpac + 1
    call f77_net_receive(len, kpacket(1,kpac), complete)
end do
call f77_thread_use_rtl
write(6,*)' RECV_BIX: received ',kpac,' packets '
-----

c
c   The result packets will contain all of the information
c   expected back by bixM master thread...
c
c   we now send the returned data to MASTER thread by means of
c   copying the contents of the packets to the shared common
c   area called BIX_DATA.
c
c -----

c
c   Copy all the parameters from the MASTER WORK common area...
c   into the BIX_DATA common area....
c
klevel = klevel9
kcall = kcall9
kint = kint9
klon1 = klon19
klon2 = klon29
klat1 = klat19
klat2 = klat29
l3dim = logic9

do k = 1, MLNLT
    kp(k) = kp9(k)
    kq(k) = kq9(k)
    kr(k) = kr9(k)

    palfa(k,klevel) = palfa9(k)
    pbeta(k,klevel) = pbeta9(k)
    pgama(k,klevel) = pgama9(k)

```

```

        palfh(k) = palfh9(k)
        pbeth(k) = pbeth9(k)
        pgamh(k) = pgamh9(k)
    end do

c      Check which result we have received back....
c      ie is it PARGA, PARGB or PARGG
c
    IF( .NOT. L3DIM) THEN
        IF( KCALL .EQ. 1) THEN
            do k = 1, MLNLT
                PALFA(k,klevel) = pres9(k)
            end do

            do k = 1, klev
                do j = 1, MLNLT
                    PARGA(j,k) = parg9(j,k)
                end do
            end do
        END IF
        IF( KCALL .EQ. 2) THEN
            do k = 1, MLNLT
                PBETA(k,klevel) = pres9(k)
            end do

            do k = 1, klev
                do j = 1, MLNLT
                    PARGB(j,k) = parg9(j,k)
                end do
            end do
        END IF
    END IF
    IF(L3DIM) THEN
        do k = 1, MLNLT
            PGAMA(k,klevel) = pres9(k)
        end do

        do k = 1, klev
            do j = 1, MLNLT
                PARGG(j,k) = parg9(j,k)
            end do
        end do
    END IF

c      Update level counter to indicate to the master task that
c      a new results packet has been received and identified.

        level_done = level_done + 1

        goto 100

c      return
c      end
c-----
c

```

```

-----
c
c          BIX_MAST.for
c
c      P Halton
c
c      14-June-1992
c
-----
c
c      SUBROUTINE BIX_MAST (
c      I   KLEVEL , KCALL , KINT
c      I   , KLONG1 , KLONG2 , KLONG1 , KLONG2
c      I   , KP   , KQ   , KR
c      R   , PARG , PRES , PALFH , PBETH , PGAMH
c      R   , PALFA , PBETA , PGAMA
c      L   , L3DIM )
c
c      *BIX_MAST* Interfaces between BIX_SEND and Worker Task BIX_SLAV
c
c      P Halton IMS
c
c      PURPOSE.
c      -----
c
c      To handle data packets being passed from original CALPRO
c      and SLDYNN routines to routine BIXINT.
c
c** INTERFACE.
c      -----
c
c      *BIX_MAST* IS CALLED FROM BIX_SEND
c
c      INPUT PARAMETERS:
c      -----
c
c      *KLEVEL*   CURRENT VERTICAL LEVEL
c      *KLONG1*   START INDEX IN X-DIRECTION
c      *KLONG2*   STOP INDEX IN X-DIRECTION
c      *KLONG1*   START INDEX IN Y-DIRECTION
c      *KLONG2*   STOP INDEX IN Y-DIRECTION
c      *KCALL*    = 1 : PREPARATION OF WEIGHTS IN ADDITION TO
c                THE INTERPOLATION PART
c                = 2 : INTERPOLATION PART ONLY
c      *KINT*     = 1 : BILINEAR INTERPOLATION
c                = 2 : BIQUADRATIC INTERPOLATION
c                = 3 : BICUBIC INTERPOLATION
c
c      *KP*      P
c      *KQ*      Q
c      *KR*      R
c
c      *PARG*    FIELD TO BE INTERPOLATED
c      *PALFH*   ALFA HAT
c      *PBETH*   BETA HAT
c      *PGAMH*   GAMA HAT
c      *PALFA*   ALFA
c      *PBETA*   BETA
c      *PGAMA*   GAMA
c      *L3DIM*   TRUE IF 3-DIMENSIONAL INTERPOLATION
c

```

```

C   OUTPUT PARAMETERS:
C   -----
C
C   none
C
C   EXTERNALS.
C   -----
C
C   *BIX_SLAV* - Interpolation Worker Task on Transputer Farm
C               which recieves packets from BIX_MAST and passes
C               the data to BIXINT for interpolation computation
C   -----
C*  DECLARATION OF GLOBAL PARAMETERS
C   -----
C       include 'paramk.inc'
C
C               for 38x34x7 we get....
C       include 'parbix.inc'
C
C   INTEGER KP(KLON*KLAT) ,   KQ(KLON*KLAT),   KR(KLON*KLAT)
C   REAL  PARG(KLON*KLAT,*), PRES(KLON*KLAT),
C   *     PALFH(KLON*KLAT) , PBETH(KLON*KLAT), PGAMH(KLON*KLAT),
C   *     PALFA(KLON*KLAT) , PBETA(KLON*KLAT), PGAMA(KLON*KLAT)
C   LOGICAL L3DIM, complete
C
C       include 'command.inc'
C       include 'thread.inc'
C       include 'chan.inc'
C
C
C   Copy all the parameters into the MASTER common area...
C
C   klon9  = klon
C   klat9  = klat
C   klev9  = klev
C   klevel9 = klevel
C   kcall9 = kcall
C   kint9  = kint
C   klon19 = klon1
C   klon29 = klon2
C   klat19 = klat1
C   klat29 = klat2
C   logic9 = l3dim
C
C   do k = 1, MLNLT
C       kp9(k) = kp(k)
C       kq9(k) = kq(k)
C       kr9(k) = kr(k)
C
C       palfa9(k) = palfa(k)
C       pbeta9(k) = pbeta(k)
C       pgama9(k) = pgama(k)
C
C       pres9(k) = pres(k)
C       palfh9(k) = palfh(k)
C       pbeth9(k) = pbeth(k)
C       pgamh9(k) = pgamh(k)
C   end do

```



```

do k = 1, klev
  do j = 1, MLNLT
    parg9(j,k) = parg(j,k)
  end do
end do

c
c We are now ready to send the newly formed COMMAND to the
c SLAVE tasks on the transputer farm.
c
c Send off the command packets ending with complete = .true.
c
c Send all but the last packet....
c
complete = .false.
intsiz = 250
isize = 1000      ! Bytes, Max = 1Kbyte, but allow 3 bytes for comm
iquot = kintsz / intsiz
iround = iquot * isize
irem = kintsz - iround

do kpac = 1, iround, intsiz
  call f77_net_send (isize, command(kpac),complete)
end do
kpac = iround + 1
complete = .true.
call f77_net_send (irem*4, command(kpac),complete)

return
end

```

SUBROUTINE VCOPY (KLEN, PA, KINCA, PB, KINCB)

```

c
c**** *VCOPY* - COPY OF ARRAY
c
c PURPOSE.
c -----
c
c COPY DATA VALUES FROM PA TO PB
c
c** INTERFACE.
c -----
c
c *VCOPY* IS AN AUXILIARY ROUTINE
c
c INPUT PARAMETERS:
c -----
c
c *KLEN*      NUMBER OF DATA VALUES
c *PA*        ARRAY WITH INPUT DATA
c *KINCA*     INCREMENT BETWEEN POINTS IN PA
c *KINCB*     INCREMENT BETWEEN POINTS IN PB
c
c OUTPUT PARAMETERS:
c -----
c
c *PB*        ARRAY WITH OUTPUT DATA
c
c EXTERNALS.
c -----
c

```

```

C   *SCOPY*   CRAY EQUIVALENT LIBRARY ROUTINE
C
C   -----
C
C*  DECLARATION OF GLOBAL PARAMETERS
C   -----
C
C     REAL PA(1), PB(1)
C
C   -----
C
C     IF (KLEN.LE.0) RETURN
C
C*  NON-LIBRARY CODE
C   -----
C
C     INCA = KINCA
C     INCB = KINCB
C
C     IA = 1 - INCA
C     IB = 1 - INCB
C     DO 100 JL=1,KLEN
C         IA = IA + INCA
C         IB = IB + INCB
C         PB(IB) = PA(IA)
100 CONTINUE
C
C     RETURN
C     END

```

```

-----
C
C
C      bixw.f77
C
C      P Halton    14-June-1992
C
-----
C      PURPOSE
C      -----
C      To receive data packets being passed from CALPRQ
C      and SLDYNM routines to routine BIXINT.
C
C**   INTERFACE.
C      -----
C
C      *BIXW* IS CALLED from f77_send_net interface routine
C
C      INPUT PARAMETERS: via the command package....
C      -----
C
C      *KLON*      NUMBER OF GRIDPOINTS IN THE X-DIRECTION
C      *KLAT*      NUMBER OF GRIDPOINTS IN THE Y-DIRECTION
C      *KLEV*      NUMBER OF VERTICAL LEVELS
C      *KLEVEL*    CURRENT VERTICAL LEVEL
C      *KLON1*     START INDEX IN X-DIRECTION
C      *KLON2*     STOP  INDEX IN X-DIRECTION
C      *KLAT1*     START INDEX IN Y-DIRECTION
C      *KLAT2*     STOP  INDEX IN Y-DIRECTION
C      *KCALL*     = 1 : PREPARATION OF WEIGHTS IN ADDITION TO
C                  THE INTERPOLATION PART
C                  = 2 : INTERPOLATION PART ONLY
C      *KINT*      = 1 : BILINEAR INTERPOLATION
C                  = 2 : BIQUADRATIC INTERPOLATION
C                  = 3 : BICUBIC INTERPOLATION
C      *KP*        P
C      *KQ*        Q
C      *KR*        R
C      *PARG*      FIELD TO BE INTERPOLATED
C      *PALFH*     ALFA HAT
C      *PBETH*     BETA HAT
C      *PGAMH*     GAMA HAT
C      *PALFA*     ALFA
C      *PBETA*     BETA
C      *PGAMA*     GAMA
C      *L3DIM*     TRUE IF 3-DIMENSIONAL INTERPOLATION
C
C      OUTPUT PARAMETERS:
C      -----
C
C      none
C
C      EXTERNALS.
C      -----
C
C      *BIXW* - Interpolation Worker Task on Transputer Farm
C              which recieves packets from BIX_MAST and passes
C              the data to BIXINT for interpolation computation
C
-----
C      program bixw

```

```

logical complete

include 'net.inc'
include 'paramk.inc'
include 'comslav.inc'  ! User written packet definitions
include 'parbix.inc'  ! Parameter declarations for BIXINT

100  continue          ! do forever

c      Wait here until a new packet arrives from BIX_MAST
c
      complete = .false.
      ipac      = 0
      do while( .not. complete .and. ipac .le. 200 )
          ipac = ipac+1
          call f77_net_receive( N,kpacket(1,ipac),complete)
      end do

c
c      We are ready to pass on the new COMMAND to the
c      original interpolation processes
c
      CALL BIXINT (
I      KLON9,KLAT9,KLEV9,KLEVEL9,KCALL9,KINT9,
I      KLON19,KLON29,KLAT19,KLAT29,
I      KP9(1),KQ9(1),KR9(1),
R      PARG9(1,1),PRES9(1),PALFH9(1),PBETH9(1),PGAMH9(1),
R      PALFA9(1),PBETA9(1),PGAMA9(1),
L      L30IM9)

      intsiz = 250
      isize  = 1000          ! Bytes, Max = 1Kbyte
      iquot  = kintsz / intsiz
      iround = iquot * isize
      irem   = kintsz - iround

c
c      Return the results packets back to Master Task bixM
c
      do kpac = 1, iround, intsiz
          call f77_net_send (isize, command(kpac),complete)
      end do
      kpac      = iround + 1
      complete = .true.
      call f77_net_send (irem*4, command(kpac),complete)

      goto 100

      end

```

```

SUBROUTINE BIXINT (
I  KLOM  , KLAT  , KLEV  , KLEVEL , KCALL  , KINT
I  , KLOM1 , KLOM2 , KLAT1 , KLAT2
I  , KP   , KQ    , KR
R  , PARG  , PRES  , PALFH  , PBETH  , PGAMH
R  , PALFA , PBETA , PGAMA
L  , L3DIM )
C
C**** *BIXINT* - INTERPOLATION FROM PARG TO PRES
C
C  PURPOSE.
C  -----
C
C  INTERPOLATION IN THE SEMI-LAGRANGIAN SCHEME
C
C** INTERFACE.
C  -----
C
C  *BIXINT* IS CALLED FROM *SLDYNM* AND *CALPOR*
C
C  INPUT PARAMETERS:
C  -----
C
C  *KLOM*      NUMBER OF GRIDPOINTS IN THE X-DIRECTION
C  *KLAT*      NUMBER OF GRIDPOINTS IN THE Y-DIRECTION
C  *KLEV*      NUMBER OF VERTICAL LEVELS
C  *KLEVEL*    CURRENT VERTICAL LEVEL
C  *KLOM1*     START INDEX IN X-DIRECTION
C  *KLOM2*     STOP  INDEX IN X-DIRECTION
C  *KLAT1*     START INDEX IN Y-DIRECTION
C  *KLAT2*     STOP  INDEX IN Y-DIRECTION
C  *KCALL*     = 1 : PREPARATION OF WEIGHTS IN ADDITION TO
C              THE INTERPOLATION PART
C              = 2 : INTERPOLATION PART ONLY
C  *KINT*      = 1 : BILINEAR INTERPOLATION
C              = 2 : BIQUADRATIC INTERPOLATION
C              = 3 : BICUBIC INTERPOLATION
C  *KP*        P
C  *KQ*        Q
C  *KR*        R
C  *PARG*      FIELD TO BE INTERPOLATED
C  *PALFH*     ALFA HAT
C  *PBETH*     BETA HAT
C  *PGAMH*     GAMA HAT
C  *PALFA*     ALFA
C  *PBETA*     BETA
C  *PGAMA*     GAMA
C  *L3DIM*     TRUE IF 3-DIMENSIONAL INTERPOLATION
C
C  OUTPUT PARAMETERS:
C  -----
C
C  *PRES*      INTERPOLATED FIELD
C
C  EXTERNALS.
C  -----
C
C  *ABGCRI*    CHECK LENGTH OF THE DISPLACEMENTS
C  *VERINT*    THREE-DIMENSIONAL INTERPOLATION BETWEEN GRIDPOINTS
C  *HORINT*    TWO -DIMENSIONAL INTERPOLATION BETWEEN GRIDPOINTS

```

```

C
C -----
C
C*  DECLARATION OF GLOBAL PARAMETERS
C -----
C
      INTEGER KP(KLON*KLAT) ,    KQ(KLON*KLAT),    KR(KLON*KLAT)
      REAL  PARG(KLON*KLAT,*),  PRES(KLON*KLAT),
*     PALFH(KLON*KLAT) ,  PBETH(KLON*KLAT),  PGAMH(KLON*KLAT),
*     PALFA(KLON*KLAT) ,  PBETA(KLON*KLAT),  PGAMA(KLON*KLAT)
      LOGICAL L3DIM

C
C -----
      INCLUDE 'paramm.com'

C
C*  DECLARATION OF LOCAL WORKSPACE IN COMMON *COMBIX*
C -----
C
      COMMON / COMBIX /
+     ZALFA(MLNLT,4), ZBETA(MLNLT,4), ZGAMA(MLNLT,4),
+     INDEXA(MLNLT) , INDEXB(MLNLT)

C
C -----
      DIMENSION ZFIX(3), ZLIM(3)
      DATA ZFIX /100.0,100.5,100.0/
      DATA ZLIM /1.000005,1.500005,2.000005/

C
C*  CHECK DIMENSION OF WORK SPACE ARRAYS
C -----
C
      IF (MLNLT.LT.KLON*KLAT) THEN
cph      WRITE(6,'(/,1X,'EXTEND WORK SPACE ARRAYS IN *BIXINT*')')
cph      WRITE(6,'(1X,'MLNLT,KLON*KLAT=',2I10)') MLNLT,KLON*KLAT
          STOP
      ENDIF

C
      ILOW = KLOW
      ILEV1 = KLEV-1
      ILEVEL = KLEVEL
      ISTART = (KLAT1 - 1)*KLOW + KLOW1
      ISTOP = (KLAT2 - 1)*KLOW + KLOW2

C
C*  PREPARATION OF WEIGHTS
C -----
C
      IF (KCALL.NE.1) GOTO 1000

C
C*  CHECK DISPLACEMENTS CLOSE TO BOUNDARIES IN *ABGCRI*
C -----
C
      ZGMIN = FLOAT(ILEVEL-KLEV) + 0.000005
      ZGMAX = FLOAT(ILEVEL-1 ) - 0.000005
      ZGDUM = 0.000005
      IF (ILEVEL.EQ.1 ) ZGDUM = -1.000005
      IF (ILEVEL.EQ.2 ) ZGDUM = -0.000005
      IF (ILEVEL.EQ.KLEV) ZGDUM = 1.000005

C
cph      CALL ABGCRI (KLOW,KLAT,ILEVEL,
cph      +           KLOW1,KLOW2,KLAT1,KLAT2,3,

```

```

cph + PALFA(1),PBETA(1),PGAMA(1),
cph + ZLIM(KINT),ZGMIN,ZGMAX,ZGDUM )
C
C* PREPARATION OF HORIZONTAL WEIGHTS
C -----
C
C Z100 = ZFIX(KINT)
C
C DO 220 JL=ISTART,ISTOP
C KP(JL) = IFIX( PALFA(JL) + Z100 ) - 100
C KQ(JL) = IFIX( PBETA(JL) + Z100 ) - 100
C INDEXA(JL) = JL - ILOM * KQ(JL) - KP(JL)
C PALFH(JL) = PALFA(JL) - FLOAT(KP(JL))
C PBETH(JL) = PBETA(JL) - FLOAT(KQ(JL))
220 CONTINUE
C
C IF (KINT.EQ.1) THEN
C
C* BILINEAR INTERPOLATION
C -----
C
C DO 290 JL=ISTART,ISTOP
C ZALFA(JL,1) = PALFH(JL)
C ZALFA(JL,2) = 1.0 - PALFH(JL)
C ZBETA(JL,1) = PBETH(JL)
C ZBETA(JL,2) = 1.0 - PBETH(JL)
290 CONTINUE
C
C ELSEIF (KINT.EQ.2) THEN
C
C* BIQUADRATIC INTERPOLATION
C -----
C
C ZR2 = 0.5
C
C DO 300 JL=ISTART,ISTOP
C Z1MA = 1.0 - PALFH(JL)
C Z1PA = 1.0 + PALFH(JL)
C Z1MB = 1.0 - PBETH(JL)
C Z1PB = 1.0 + PBETH(JL)
C
C ZALFA(JL,1) = ZR2 * PALFH(JL) * Z1PA
C ZALFA(JL,2) = Z1MA * Z1PA
C ZALFA(JL,3) = -ZR2 * PALFH(JL) * Z1MA
C ZBETA(JL,1) = ZR2 * PBETH(JL) * Z1PB
C ZBETA(JL,2) = Z1MB * Z1PB
C ZBETA(JL,3) = -ZR2 * PBETH(JL) * Z1MB
300 CONTINUE
C
C ELSEIF (KINT.EQ.3) THEN
C
C* BICUBIC INTERPOLATION
C -----
C
C ZR2 = 0.5
C ZR6 = 1.0/6.0
C
C DO 310 JL=ISTART,ISTOP
C Z1MA2 = 1.0 - PALFH(JL)*PALFH(JL)
C Z2MA = 2.0 - PALFH(JL)

```

```

Z1MB2 = 1.0 - PBETH(JL)*PBETH(JL)
Z2MB = 2.0 - PBETH(JL)
C
ZALFA(JL,1) = -ZR6 * PALFH(JL) * Z1MA2
ZALFA(JL,2) = ZR2 * PALFH(JL) * (1.0 + PALFH(JL)) * Z2MA
ZALFA(JL,3) = ZR2 * Z1MA2 * Z2MA
ZALFA(JL,4) = -ZR6 * PALFH(JL) * (1.0 - PALFH(JL)) * Z2MA
ZBETA(JL,1) = -ZR6 * PBETH(JL) * Z1MB2
ZBETA(JL,2) = ZR2 * PBETH(JL) * (1.0 + PBETH(JL)) * Z2MB
ZBETA(JL,3) = ZR2 * Z1MB2 * Z2MB
ZBETA(JL,4) = -ZR6 * PBETH(JL) * (1.0 - PBETH(JL)) * Z2MB
310 CONTINUE
C
ENDIF
C
C* PREPARATION OF VERTICAL WEIGHTS
C -----
C
IF (L3DIM) THEN
C
Z100 = ZFIX(KINT)
C
DO 230 JL=ISTART,ISTOP
KR(JL) = IFIX( PGAMA(JL) + Z100 ) - 100
INDEXB(JL) = ILEVEL - KR(JL)
PGAMH(JL) = PGAMA(JL) - FLOAT(KR(JL))
230 CONTINUE
C
IF (KINT.EQ.1) THEN
C
C* LINEAR INTERPOLATION
C -----
C
DO 295 JL=ISTART,ISTOP
ZGAMA(JL,1) = PGAMH(JL)
ZGAMA(JL,2) = 1.0 - PGAMH(JL)
295 CONTINUE
C
ELSEIF (KINT.EQ.2) THEN
C
C* QUADRATIC INTERPOLATION
C -----
C
ZR2 = 0.5
C
DO 305 JL=ISTART,ISTOP
Z1MG = 1.0 - PGAMH(JL)
Z1PG = 1.0 + PGAMH(JL)
C
ZGAMA(JL,1) = ZR2 * PGAMH(JL) * Z1PG
ZGAMA(JL,2) = Z1MG * Z1PG
ZGAMA(JL,3) = -ZR2 * PGAMH(JL) * Z1MG
305 CONTINUE
C
C* LINEAR INTERPOLATION IF QUADRATIC NOT POSSIBLE
C* -----
C
DO 306 JL=ISTART,ISTOP
IF (INDEXB(JL).LT.2) THEN
INDEXB(JL) = 2

```



```

ZGAMH      = PGAMA(JL) - FLOAT(IK)
ZGAMA(JL,1) = ZGAMH
ZGAMA(JL,2) = 1.0 - ZGAMH
ZGAMA(JL,3) = 0.0
ENDIF
IF (INDEXB(JL).GT.ILEV1) THEN
  INDEXB(JL) = ILEV1
  IK        = IFIX( PGAMA(JL) + ZFIX(1) ) - 100
  ZGAMH     = PGAMA(JL) - FLOAT(IK)
  ZGAMA(JL,1) = 0.0
  ZGAMA(JL,2) = ZGAMH
  ZGAMA(JL,3) = 1.0 - ZGAMH
ENDIF
306 CONTINUE
C
  ELSEIF (KINT.EQ.3) THEN
C
C* CUBIC INTERPOLATION
C -----
C
  ZR2 = 0.5
  ZR6 = 1.0/6.0
C
  DO 315 JL=ISTART,ISTOP
    Z1MG2 = 1.0 - PGAMH(JL)*PGAMH(JL)
    Z2MG  = 2.0 - PGAMH(JL)
C
    ZGAMA(JL,1) = -ZR6 * PGAMH(JL) * Z1MG2
    ZGAMA(JL,2) =  ZR2 * PGAMH(JL) * (1.0 + PGAMH(JL)) * Z2MG
    ZGAMA(JL,3) =  ZR2 * Z1MG2 * Z2MG
    ZGAMA(JL,4) = -ZR6 * PGAMH(JL) * (1.0 - PGAMH(JL)) * Z2MG
  315 CONTINUE
C
C* LINEAR INTERPOLATION IF CUBIC NOT POSSIBLE
C -----
C
  DO 316 JL=ISTART,ISTOP
    IF (INDEXB(JL).LT.3) THEN
      INDEXB(JL) = 3
      ZGAMA(JL,1) = PGAMH(JL)
      ZGAMA(JL,2) = 1.0 - PGAMH(JL)
      ZGAMA(JL,3) = 0.0
      ZGAMA(JL,4) = 0.0
    ENDIF
    IF (INDEXB(JL).GT.ILEV1) THEN
      INDEXB(JL) = ILEV1
      ZGAMA(JL,1) = 0.0
      ZGAMA(JL,2) = 0.0
      ZGAMA(JL,3) = PGAMH(JL)
      ZGAMA(JL,4) = 1.0 - PGAMH(JL)
    ENDIF
  316 CONTINUE
C
  ENDIF
C
  ENDIF
C
  1000 CONTINUE
C

```

```

C* INTERPOLATION IN *VERINT* AND *HORINT*
C -----
C
C IF (L3DIM) THEN
C
C* 3-DIMENSIONAL CASE ( HORIZONTAL AND VERTICAL )
C -----
C
C CALL VERINT (
I KLOW , KLAT , KINT ,
I KLOW1 , KLOW2 , KLAT1 , KLAT2 ,
I INDEXA(1), INDEXB(1),
R PARG(1,1), PRES (1),
R ZALFA(1,1), ZBETA(1,1), ZGAMA(1,1) )
C
C ELSE
C
C* 2-DIMENSIONAL CASE (HORIZONTAL)
C -----
C
C CALL HORINT (
I KLOW , KLAT , KINT ,
I KLOW1 , KLOW2 , KLAT1 , KLAT2 ,
I INDEXA(1),
R PARG(1,ILEVEL), PRES(1),
R ZALFA(1,1), ZBETA(1,1) )
C
C ENDIF
C
C RETURN
C END

```

```

SUBROUTINE HORINT (
I  KLON  , KLAT  , KINT
I  , KLON1 , KLON2 , KLAT1 , KLAT2
I  , KINDXA
R  , PARG  , PRES
R  , PALFA , PBETA )
C
C**** *HORINT* - HORIZONTAL INTERPOLATION
C
C  PURPOSE.
C  -----
C
C  TWO DIMENSIONAL (HORIZONTAL) INTERPOLATION
C
C**  INTERFACE.
C  -----
C
C  *HORINT* IS CALLED FROM *BIXINT*
C
C  INPUT PARAMETERS:
C  -----
C
C  *KLON*      NUMBER OF GRIDPOINTS IN X-DIRECTION
C  *KLAT*      NUMBER OF GRIDPOINTS IN Y-DIRECTION
C  *KINT*      INDEX FOR INTERPOLATION METHOD
C              KINT = 1 : BILINEAR
C              KINT = 2 : BIQUADRATIC
C              KINT = 3 : BICUBIC
C  *KLON1*     START INDEX IN X-DIRECTION
C  *KLON2*     STOP  INDEX IN X-DIRECTION
C  *KLAT1*     START INDEX IN Y-DIRECTION
C  *KLAT2*     STOP  INDEX IN Y-DIRECTION
C  *KINDXA*    ARRAY OF INDEXES FOR DISPLACEMENTS
C  *PARG*      ARRAY OF ARGUMENTS
C  *PALFA*     ARRAY OF WEIGHTS IN X-DIRECTION
C  *PBETA*     ARRAY OF WEIGHTS IN Y-DIRECTION
C
C  OUTPUT PARAMETERS:
C  -----
C
C  *PRES*      INTERPOLATED FIELD
C
C  EXTERNALS.
C  -----
C
C  NONE
C
C  -----
C*  DECLARATION OF GLOBAL PARAMETERS
C  -----
C
C  INTEGER KINDXA(KLON*KLAT)
C  REAL      PARG(KLON*KLAT) , PRES(KLON*KLAT) ,
C  *          PALFA(KLON*KLAT,*), PBETA(KLON*KLAT,*)
C
C-----
C
C  ILOM  = KLON
C  ISTART = (KLAT1 - 1)*KLON + KLON1

```

```

      ISTOP = (KLAT2 - 1)*KLN + KLN2
C
      IF (KINT.EQ.1) THEN
C
C*  BILINEAR INTERPOLATION
C  -----
C
      IM1M1 = - ILO - 1
      IMOM1 = - ILO
      IM1M0 = - 1
C
      DO 320 JL=ISTART,ISTOP
        IDIS = KINDXA(JL)
C
        PRES(JL) =
C
      +  PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM1M1)
      +                    + PALFA(JL,2) * PARG(IDIS+IMOM1) )
      + + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM1M0)
      +                    + PALFA(JL,2) * PARG(IDIS      ) )
320  CONTINUE
C
      ELSEIF (KINT.EQ.2) THEN
C
C*  BIQUADRATIC INTERPOLATION
C  -----
C
      IM1M1 = - ILO - 1
      IMOM1 = - ILO
      IP1M1 = - ILO + 1
      IM1M0 = - 1
      IP1M0 = 1
      IM1P1 = ILO - 1
      IMOP1 = ILO
      IP1P1 = ILO + 1
C
      DO 330 JL=ISTART,ISTOP
        IDIS = KINDXA(JL)
C
        PRES(JL) =
C
      +  PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM1M1)
      +                    + PALFA(JL,2) * PARG(IDIS+IMOM1)
      +                    + PALFA(JL,3) * PARG(IDIS+IP1M1) )
      + + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM1M0)
      +                    + PALFA(JL,2) * PARG(IDIS      )
      +                    + PALFA(JL,3) * PARG(IDIS+IP1M0) )
      + + PBETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM1P1)
      +                    + PALFA(JL,2) * PARG(IDIS+IMOP1)
      +                    + PALFA(JL,3) * PARG(IDIS+IP1P1) )
330  CONTINUE
C
      ELSEIF (KINT.EQ.3) THEN
C
C*  BICUBIC INTERPOLATION
C  -----
C
      IM2M2 = - ILO - ILO - 2
      IM1M2 = - ILO - ILO - 1
      IMOM2 = - ILO - ILO

```

```

IP1M2 = - ILOM - ILOM + 1
IM2M1 = - ILOM - 2
IM1M1 = - ILOM - 1
IMOM1 = - ILOM
IP1M1 = - ILOM + 1
IM2M0 = - 2
IM1M0 = - 1
IP1M0 = 1
IM2P1 = ILOM - 2
IM1P1 = ILOM - 1
IMOP1 = ILOM
IP1P1 = ILOM + 1

```

C

```

DO 340 JL=ISTART,ISTOP
  IDIS = KINDXA(JL)

```

C

```

  PRES(JL) =

```

C

```

+  PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2M2)
+                  + PALFA(JL,2) * PARG(IDIS+IM1M2)
+                  + PALFA(JL,3) * PARG(IDIS+IMOM2)
+                  + PALFA(JL,4) * PARG(IDIS+IP1M2) )
+ + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM2M1)
+                  + PALFA(JL,2) * PARG(IDIS+IM1M1)
+                  + PALFA(JL,3) * PARG(IDIS+IMOM1)
+                  + PALFA(JL,4) * PARG(IDIS+IP1M1) )

```

C

```

  PRES(JL) = PRES(JL) +

```

C

```

+  PBETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM2M0)
+                  + PALFA(JL,2) * PARG(IDIS+IM1M0)
+                  + PALFA(JL,3) * PARG(IDIS      )
+                  + PALFA(JL,4) * PARG(IDIS+IP1M0) )
+ + PBETA(JL,4) * ( PALFA(JL,1) * PARG(IDIS+IM2P1)
+                  + PALFA(JL,2) * PARG(IDIS+IM1P1)
+                  + PALFA(JL,3) * PARG(IDIS+IMOP1)
+                  + PALFA(JL,4) * PARG(IDIS+IP1P1) )

```

```

340 CONTINUE

```

C

```

  ENDIF

```

C

```

  RETURN
  END

```

```

SUBROUTINE VERINT (
I  KLOW  , KLAT  , KINT
I  , KLOW1 , KLOW2 , KLAT1 , KLAT2
I  , KINDXA , KINDXB
R  , PARG  , PRES
R  , PALFA , PBETA , PGAMA )
C
C**** *VERINT* - THREE DIMENSIONAL INTERPOLATION
C
C  PURPOSE.
C  -----
C
C  THREE DIMENSIONAL INTERPOLATION
C
C** INTERFACE.
C  -----
C
C  *VERINT* IS CALLED FROM *BIXINT*
C
C  INPUT PARAMETERS:
C  -----
C
C  *KLOW*      NUMBER OF GRIDPOINTS IN X-DIRECTION
C  *KLAT*      NUMBER OF GRIDPOINTS IN Y-DIRECTION
C  *KINT*      INDEX FOR INTERPOLATION METHOD
C
C              KINT = 1 : LINEAR
C              KINT = 2 : QUADRATIC
C              KINT = 3 : CUBIC
C
C  *KLOW1*     START INDEX IN X-DIRECTION
C  *KLOW2*     STOP  INDEX IN X-DIRECTION
C  *KLAT1*     START INDEX IN Y-DIRECTION
C  *KLAT2*     STOP  INDEX IN Y-DIRECTION
C  *KINDXA*    ARRAY OF INDEXES FOR HORIZONTAL DISPLACEMENTS
C  *KINDXB*    ARRAY OF INDEXES FOR VERTICAL  DISPLACEMENTS
C  *PARG*      ARRAY OF ARGUMENTS
C  *PALFA*     ARRAY OF WEIGHTS IN X-DIRECTION
C  *PBETA*     ARRAY OF WEIGHTS IN Y-DIRECTION
C  *PGAMA*     ARRAY OF WEIGHTS IN VERTICAL DIRECTION
C
C  OUTPUT PARAMETERS:
C  -----
C
C  *PRES*      INTERPOLATED FIELD
C
C  EXTERNALS.
C  -----
C
C  NONE
C
C  -----
C
C** DECLARATION OF GLOBAL PARAMETERS
C  -----
C
C  INTEGER KINDXA(KLOW*KLAT) , KINDXB(KLOW*KLAT)
C  REAL   PARG(KLOW*KLAT,*), PRES(KLOW*KLAT) ,
C  *      PALFA(KLOW*KLAT,*), PBETA(KLOW*KLAT,*),
C  *      PGAMA(KLOW*KLAT,*)
C
C  -----

```

```

LON = KLOM
ISTART = (KLAT1 - 1)*KLOM + KLOM2
ISTOP = (KLAT2 - 1)*KLOM + KLOM2
IF (KINT.EQ.1) THEN
C
C*
C-----
LINEAR INTERPOLATION
C
C
DO 320 JL=ISTART,ISTOP
IDIS = KINDXA(JL)
ILEV = KINDXB(JL)
ILM1 = ILEV - 1
ILM2 = ILEV + 1
IM1M1 = - ILOM + 1
IM1M2 = - ILOM
IP1M1 = - ILOM + 1
IP1M2 = 1
IM1M1 = - 1
IM1M2 = 1
IM1P1 = ILOM - 1
IM1P2 = ILOM
IP1P1 = ILOM + 1
DO 330 JL=ISTART,ISTOP
IDIS = KINDXA(JL)
ILEV = KINDXB(JL)
ILM1 = ILEV - 1
ILM2 = ILEV + 1
IM1M1 = - ILOM + 1
IM1M2 = - 1
IP1M1 = - ILOM + 1
IP1M2 = 1
IM1M1 = - 1
IM1M2 = 1
IM1P1 = ILOM - 1
IM1P2 = ILOM
IP1P1 = ILOM + 1
PRES(JL) = PGAMA(JL,1) * (
PALFA(JL,1) * PARG(IDIS+IM1M1,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILM1)
+ PALFA(JL,3) * PARG(IDIS+IP1M1,ILM1)
+ PALFA(JL,1) * PARG(IDIS+IM1M1,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILM1)
)
PRES(JL) = PGAMA(JL,1) * (
PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
)
IF (KINT.EQ.2) THEN
C
C*
C-----
QUADRATIC INTERPOLATION
C
C
DO 320 JL=ISTART,ISTOP
IDIS = KINDXA(JL)
ILEV = KINDXB(JL)
ILM1 = ILEV - 1
ILM2 = ILEV + 1
IM1M1 = - ILOM + 1
IM1M2 = - ILOM
IP1M1 = - ILOM + 1
IP1M2 = 1
IM1M1 = - 1
IM1M2 = 1
IM1P1 = ILOM - 1
IM1P2 = ILOM
IP1P1 = ILOM + 1
DO 330 JL=ISTART,ISTOP
IDIS = KINDXA(JL)
ILEV = KINDXB(JL)
ILM1 = ILEV - 1
ILM2 = ILEV + 1
IM1M1 = - ILOM + 1
IM1M2 = - 1
IP1M1 = - ILOM + 1
IP1M2 = 1
IM1M1 = - 1
IM1M2 = 1
IM1P1 = ILOM - 1
IM1P2 = ILOM
IP1P1 = ILOM + 1
PRES(JL) = PGAMA(JL,2) * (
PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
)
PRES(JL) = PGAMA(JL,2) * (
PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,1) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
)
END
```



```

C
+ PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2M2,ILM2)
+ PALFA(JL,2) * PARG(IDIS+IM1M2,ILM2)
+ PALFA(JL,3) * PARG(IDIS+IMOM2,ILM2)
+ PALFA(JL,4) * PARG(IDIS+IP1M2,ILM2) )
+ + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM2M1,ILM2)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILM2)
+ PALFA(JL,3) * PARG(IDIS+IMOM1,ILM2)
+ PALFA(JL,4) * PARG(IDIS+IP1M1,ILM2) ) )

```

```

C
PRES(JL) = PRES(JL) + PGAMA(JL,1) * (

```

```

C
+ PBETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM2M0,ILM2)
+ PALFA(JL,2) * PARG(IDIS+IM1M0,ILM2)
+ PALFA(JL,3) * PARG(IDIS      ,ILM2)
+ PALFA(JL,4) * PARG(IDIS+IP1M0,ILM2) )
+ + PBETA(JL,4) * ( PALFA(JL,1) * PARG(IDIS+IM2P1,ILM2)
+ PALFA(JL,2) * PARG(IDIS+IM1P1,ILM2)
+ PALFA(JL,3) * PARG(IDIS+IMOP1,ILM2)
+ PALFA(JL,4) * PARG(IDIS+IP1P1,ILM2) ) )

```

```

C
PRES(JL) = PRES(JL) + PGAMA(JL,2) * (

```

```

C
+ PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2M2,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M2,ILM1)
+ PALFA(JL,3) * PARG(IDIS+IMOM2,ILM1)
+ PALFA(JL,4) * PARG(IDIS+IP1M2,ILM1) )
+ + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM2M1,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILM1)
+ PALFA(JL,3) * PARG(IDIS+IMOM1,ILM1)
+ PALFA(JL,4) * PARG(IDIS+IP1M1,ILM1) ) )

```

```

C
PRES(JL) = PRES(JL) + PGAMA(JL,2) * (

```

```

C
+ PBETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM2M0,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1M0,ILM1)
+ PALFA(JL,3) * PARG(IDIS      ,ILM1)
+ PALFA(JL,4) * PARG(IDIS+IP1M0,ILM1) )
+ + PBETA(JL,4) * ( PALFA(JL,1) * PARG(IDIS+IM2P1,ILM1)
+ PALFA(JL,2) * PARG(IDIS+IM1P1,ILM1)
+ PALFA(JL,3) * PARG(IDIS+IMOP1,ILM1)
+ PALFA(JL,4) * PARG(IDIS+IP1P1,ILM1) ) )

```

```

C
PRES(JL) = PRES(JL) + PGAMA(JL,3) * (

```

```

C
+ PBETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2M2,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M2,ILEV)
+ PALFA(JL,3) * PARG(IDIS+IMOM2,ILEV)
+ PALFA(JL,4) * PARG(IDIS+IP1M2,ILEV) )
+ + PBETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM2M1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILEV)
+ PALFA(JL,3) * PARG(IDIS+IMOM1,ILEV)
+ PALFA(JL,4) * PARG(IDIS+IP1M1,ILEV) ) )

```

```

C
PRES(JL) = PRES(JL) + PGAMA(JL,3) * (

```

```

C
+ PBETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM2M0,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1M0,ILEV)
+ PALFA(JL,3) * PARG(IDIS      ,ILEV)
+ PALFA(JL,4) * PARG(IDIS+IP1M0,ILEV) )

```

```

+ PBFETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2P1,ILEV)
+ PALFA(JL,2) * PARG(IDIS+IM1P1,ILEV)
+ PALFA(JL,3) * PARG(IDIS+IMOP1,ILEV)
+ PALFA(JL,4) * PARG(IDIS+IP1P1,ILEV) ) )
PRES(JL) = PRES(JL) + PGAMA(JL,4) * (
PBFETA(JL,1) * ( PALFA(JL,1) * PARG(IDIS+IM2M2,ILP1)
+ PALFA(JL,2) * PARG(IDIS+IM1M2,ILP1)
+ PALFA(JL,3) * PARG(IDIS+IMOM2,ILP1)
+ PALFA(JL,4) * PARG(IDIS+IP1M2,ILP1) )
+ PBFETA(JL,2) * ( PALFA(JL,1) * PARG(IDIS+IM2M1,ILP1)
+ PALFA(JL,2) * PARG(IDIS+IM1M1,ILP1)
+ PALFA(JL,3) * PARG(IDIS+IMOM1,ILP1)
+ PALFA(JL,4) * PARG(IDIS+IP1M1,ILP1) ) )
PRES(JL) = PRES(JL) + PGAMA(JL,4) * (
PBFETA(JL,3) * ( PALFA(JL,1) * PARG(IDIS+IM2M0,ILP1)
+ PALFA(JL,2) * PARG(IDIS+IM1M0,ILP1)
+ PALFA(JL,3) * PARG(IDIS
,ILP1)
+ PALFA(JL,4) * PARG(IDIS+IP1M0,ILP1) )
+ PBFETA(JL,4) * ( PALFA(JL,1) * PARG(IDIS+IM2P1,ILP1)
+ PALFA(JL,2) * PARG(IDIS+IM1P1,ILP1)
+ PALFA(JL,3) * PARG(IDIS+IMOP1,ILP1)
+ PALFA(JL,4) * PARG(IDIS+IP1P1,ILP1) ) )
340 CONTINUE
C
C
ENDIF
C
RETURN
END

```

APPENDIX C

SAMPLE MASTER/SLAVE CONFIGURATION FILES

```

|      BIX.CFG
|
|      P. Halton    20-June-1992
|
|      Configuration file to place one copy of Master task, bixM, on
|      Root Transputer and one copy slave task, bixW, on P001 transputer
|-----
|
|      Declare all processors in the configuration
Processor Host
Processor Root
Processor P001

|      Declare the connections between processors
Wire ? Host[0] Root[0]
Wire ? Root[2] P001[1]

|      Declare the all Tasks and the number of ports for each
Task Afserver Ins=1 Outs=1
Task Filter   Ins=2 Outs=2 Data=10K
Task bixM     Ins=2 Outs=2 Data= 900K
Task bixW     Ins=1 Outs=1 Stack=1k Heap = 200k Opt = Stack Opt = Code

|      Declare connections between tasks
Connect ? Afserver[0] Filter[0]
Connect ? Filter[0]   Afserver[0]
Connect ? Filter[1]   bixM[1]
Connect ? bixM[1]     Filter[1]
Connect ? bixM[0]     bixW[0]
Connect ? bixW[0]     bixM[0]

|      Place the tasks on their respective processors.
Place Afserver Host
Place Filter   Root
Place bixM     Root
Place bixW     P001

```

```

|-----
|      FBIX.CFG
|
|      P HALTON    10-JUNE-1992
|
|      Specification of tasks and their size for the
|      Flood Fill configuration program, FCONFIG
|-----
Task Worker File=bixW Data=280K
Task Master File=bixM Data=1130K

```

```
REM    Bix.bat
REM
REM    P Halton, June 1992
REM
REM    Compile, link and configure the BIX application
t8f bixm/Zi/Zd
t8f bixw/Zi/Zd
t8f rdbixint/Zi/Zd
linkt/G/I @bixm.lnk, bixM.b4
linkt/G/I @bixw.lnk, bixW.b4
```

```
REM configure 'static' version of application
config bix.cfg bix.app
```

```
REM configure 'flood-filled' version of application
fconfig fbix.cfg fbix.app
```

```
| _____
|
|          BixM.lnk
bixm.bin
rdbixint.bin
\tf2v1\frtlt8.bin
\tf2v1\taskharn.t8
```

```
| _____
|
|          BixW.lnk
bixw.bin
\tf2v1\safrtlt8.bin
\tf2v1\taskharn.t8
```

APPENDIX D

SAMPLE MODEL OUTPUT LOG
FOR ONE TIME STEP

START FORECAST

Before call to timingzt1,zt2,ztotal =

0.000000E+00 -2.217773E-39 -2.893066E-39

After call to timingzt1,zt2,ztotal =

2.236160 -2.217773E-39 -2.893066E-39

0 2 1 86 8 25 0 8

USING CORIOLIS IMPLICIT SCHEME: T KITRMX= 2

NLSLAN=.T., LAMCHO IS SET EQUAL 2

NLSIMP = T

NLINMI = F

NLSLAN = T

NL2TIM = T

NLSL3D = T

NLPHYS = F

HYBRID-COORDINATES, HIRLAM LEVEL 2

MAIN DIMENSIONS:

NLOW = 38

NLAT = 34

NLEV = 7

NSLAB = 3

NBDPTS = 8

NPBPTS = 3

NMODES = 4

EXPERIMENT NAME : PAUL

GENERATED FROM START DATA FILE:

NCBASE = 860825

NTBASE = 0

NTDATA = 0

POLE = 30.0000000000

WEST = -34.5000000000

EAST = 21.0000000000

SOUTH = -24.0000000000

ANORTH = 25.5000000000

DLAMDA = 1.5000000000

DTHETA = 1.5000000000

FULL LEVELS:

FIRST PARAMETER =

0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000

0.0000000000 0.0000000000

SECOND PARAMETER =

0.0715989173 0.2148840130 0.3572940230 0.4997450113 0.6431530118

0.7860499620 0.9286330342

HALF LEVELS:

FIRST PARAMETER =

0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000

0.0000000000 0.0000000000 0.0000000000

SECOND PARAMETER =

0.0000000000 0.1431978345 0.2865701914 0.4280178547 0.5714721680

0.7148338556 0.8572660685 1.0000000000

```

-----
NSTEP      1 BEGINS HERE:
TDATA,NBDTIM,NBDDIF,TIMRAT= 10800.0    21600    21600    1.000
etadot = 0.000E+00 k= 1
etadot = -0.768E-06 k= 2
etadot = 0.575E-06 k= 3
etadot = 0.132E-05 k= 4
etadot = 0.230E-05 k= 5
etadot = 0.122E-05 k= 6
etadot = 0.437E-06 k= 7
time in dyn= 0.3361E+01
time in forward time step = 0.2101E+00
time in diffusion = 0.0000E+00
time in sldyn computing non linear terms= 0.1534E+01
time in intrp of sigmadot to full levels= 0.3720E+00
time interpolating hor. vel to mass points= 0.3071E+00
VERCFT, 1,MEAN=-2.74E-03,MIN=-7.03E-02( 29, 9),MAX= 4.86E-03( 33, 15)
VERCFT, 2,MEAN= 6.65E-04,MIN=-0.30 ( 12, 22),MAX= 7.76E-03( 5, 5)
VERCFT, 3,MEAN= 8.05E-03,MIN=-0.64 ( 12, 23),MAX= 3.62E-03( 5, 5)
VERCFT, 4,MEAN= 1.13E-02,MIN=-0.60 ( 12, 23),MAX= 1.75E-03( 5, 5)
VERCFT, 5,MEAN= 1.16E-02,MIN=-0.45 ( 12, 23),MAX= 4.53E-03( 5, 5)
VERCFT, 6,MEAN= 8.96E-03,MIN=-0.38 ( 19, 11),MAX= 7.19E-03( 5, 5)
VERCFT, 7,MEAN= 3.66E-03,MIN=-0.16 ( 18, 11),MAX= 4.38E-03( 5, 5)

time interpolating = 0.2006E+02
time in sldyn= 0.4814E+02
time spent in explicit adjustment = 0.7686E+00
corimp= 1 kitrmx= 2 epm= 0.000

EIGENVALUE for (2c/dt+M)= 0.1144E+06 L= 1
DIFF= 0.590E-03 MAX.DIFF= 0.753E-02 IMAX,LEVEL= 299 1
EIGENVALUE for (2c/dt+M)= 0.1144E+06 L= 1
DIFF= 0.568E-06 MAX.DIFF= 0.400E-05 IMAX,LEVEL= 204 1
EIGENVALUE for (2c/dt+M)= 0.2453E+05 L= 2
DIFF= 0.109E-03 MAX.DIFF= 0.131E-02 IMAX,LEVEL= 923 2
EIGENVALUE for (2c/dt+M)= 0.2453E+05 L= 2
DIFF= 0.320E-06 MAX.DIFF= 0.265E-05 IMAX,LEVEL= 241 2
EIGENVALUE for (2c/dt+M)= 0.4953E+04 L= 3
DIFF= 0.491E-04 MAX.DIFF= 0.792E-03 IMAX,LEVEL= 848 3
EIGENVALUE for (2c/dt+M)= 0.4953E+04 L= 3
DIFF= 0.376E-06 MAX.DIFF= 0.451E-05 IMAX,LEVEL= 848 3
EIGENVALUE for (2c/dt+M)= 0.1377E+04 L= 4
DIFF= 0.137E-04 MAX.DIFF= 0.270E-03 IMAX,LEVEL= 886 4
EIGENVALUE for (2c/dt+M)= 0.1377E+04 L= 4
DIFF= 0.228E-06 MAX.DIFF= 0.428E-05 IMAX,LEVEL=1085 4
EIGENVALUE for (2c/dt+M)= 0.4469E+03 L= 5
DIFF= 0.102E-04 MAX.DIFF= 0.160E-03 IMAX,LEVEL= 886 5
EIGENVALUE for (2c/dt+M)= 0.4469E+03 L= 5
DIFF= 0.310E-06 MAX.DIFF= 0.478E-05 IMAX,LEVEL= 886 5
EIGENVALUE for (2c/dt+M)= 0.1400E+03 L= 6
DIFF= 0.111E-05 MAX.DIFF= 0.173E-04 IMAX,LEVEL= 885 6
EIGENVALUE for (2c/dt+M)= 0.1400E+03 L= 6
DIFF= 0.502E-07 MAX.DIFF= 0.629E-06 IMAX,LEVEL= 885 6
EIGENVALUE for (2c/dt+M)= 0.2939E+02 L= 7
DIFF= 0.186E-06 MAX.DIFF= 0.348E-05 IMAX,LEVEL= 885 7
EIGENVALUE for (2c/dt+M)= 0.2939E+02 L= 7
DIFF= 0.114E-07 MAX.DIFF= 0.181E-06 IMAX,LEVEL= 885 7
K= 1 DAV= 1.6407E-06
K= 2 DAV= 6.6322E-06

```


K= 3 DAV= 1.1719E-05
K= 4 DAV= 8.8301E-06
K= 5 DAV= 7.9711E-06
K= 6 DAV= 7.8001E-06
K= 7 DAV= 8.3286E-06

time in helmholtz solver = 0.5088E+01
time spent in implicit adjustment = 0.1101E+01

ABS(DPS)/3H = 1.27 MB
MAXWIND = 42.59 M/S IN (X,Y,LEV)= 36 15 2
STPS = 99246.171875000008
STQ = 204.293426513672
STS = 0.000000000000
STPE = 25293983743.999999269647
STKE = 4627942.500000000514
STTE = 25298612224.000001091894