Meteorological modelling on the ICL Distributed Array Processor and other parallel computers

Glenn Derek Carver

Ph.D.
University of Edinburgh
1990

Declaration

I hereby declare that this thesis has been written and composed by myself and that the work herein is my own unless otherwise stated.

Abstract

Parallel computers are being increasingly used for meteorological modelling. It is therefore important to establish which types of parallel computer architectures are suitable for meteorological models and whether existing modelling techniques can be used efficiently.

A study of meteorological modelling techniques on the ICL Distributed Array Processor (DAP) is described. This computer has 1-bit processing elements, connected as a 64x64 array, that execute the same instructions at the same time. It is programmed in DAP FORTRAN, a version of FORTRAN that supports parallel data objects. The architecture of this computer requires a different approach to algorithm design from conventional computers.

Studies by other authors on applying gridpoint models to the DAP were reviewed. These studies concluded that global gridpoint models can make efficient use of the DAP, although the choice of finite difference grid is important because it affects the number of processors used. A regular latitude-longitude grid with Fourier filtering on polar latitudes appeared the best choice.

In a spectral meteorological model, the Legendre transforms account for most of the CPU time. Therefore, prior to the implementation of a barotropic spectral model on the DAP, efficient parallel Legendre transform algorithms were derived. The performance of these algorithms was dependent on the way the data were mapped onto the processor array. The best algorithms resulted from the data mappings that made the most efficient use of memory. The spectral model based on these algorithms was not as efficient as an equivalent gridpoint model, mainly because of the different storage requirements for the spectral, Fourier and gridpoint representations of the variables.

A two-dimensional finite element mesoscale model was implemented on the DAP to contrast the algorithms used for the gridpoint and spectral models. The finite element model was found to make efficient use of the processors and storage, if the equations were written to result in tridiagonal matrix equations. It was concluded that the finite element method is better suited to the DAP than the spectral method, although each is applied to different types of meteorological modelling problems.

The algorithms used for meteorological modelling on other parallel computers were reviewed. In particular, the sophisticated techniques used to multiprocess a spectral forecast model on the 4 processor CRAY X-MP were described.

In all the models studied, there was more parallelism available than could be exploited on the DAP. However, the lack of a fast data transfer facility, a limited processor memory and the array size constraints of DAP FORTRAN made the DAPs at Edinburgh University unsuitable for meteorological modelling problems in general. These objections do not apply to the latest generation DAP so this architecture was concluded to be suitable for gridpoint, spectral and finite element models.

CHAPTER 1

INTRODUCTION

With the development of the first electronic computer came the first successful numerical weather forecast at 500mb by Charney *et al.* (1950) using a barotropic model. Developments in numerical weather prediction led to the use of quasi-geostrophic models followed by primitive equation models in the late 1960s. Currently, models with a sophisticated representation of physical processes are used for daily medium range global forecasts, requiring typically a total of 10^{12} operations on 10^6 gridpoints for a 10 day forecast. By the 1990s, Bengtsson (1988) estimates that 10^{14} operations will be required by such models.

These advances in forecasting closely followed the rapid development in computer technology; roughly a ten-fold increase in computing speed every five years (Hockney and Jesshope, 1981). This increase is the result of improvements in hardware performance and the introduction of parallelism at all levels of computer architecture.

Two broad classes of parallel computer can be identified. The first is the pipelined computer of which the CYBER 205 and the CRAY 1 are examples. The second is the processor array architecture consisting of many interconnected processors. The ICL Distributed Array Processor (DAP) is an example of this class, having 1-bit processors, connected as a 64^2 array. Whilst the speed of each processor is slow, the performance of the array as a whole can be tens of millions of floating point operations per second (Mflops).

Up to now, meteorological models have been run on the pipelined class of computer. In the short term, this will continue because these machines offer the best performance and the code of these models has a long development time. However, in the long term, other architectural types may offer the best performance. It is therefore essential to gain experience and expertise in developing numerical weather prediction models on alternative parallel computers.

The motivation for any study of meteorological modelling on parallel computers is to establish whether existing modelling techniques can be used efficiently, to develop new algorithms for such techniques and to develop any

new techniques required for specific architectures. What is more, the facilities offered by parallel computer systems need to be reviewed: what programming languages are available, what programming environment and tools are provided, what peripheral devices are available and whether input/output facilities are sufficient. From such a study it should be possible to list the requirements of future computer systems for meteorological models, identify drawbacks of different architectures and discuss trade-offs from a meteorological viewpoint.

In this thesis, the application of numerical weather prediction to the ICL DAP is studied. To obtain the best performance, the processor array architecture of this machine requires a different approach to algorithm design. Furthermore, the programming language for the DAP is a parallel version of FORTRAN. Unlike the original DAPs, the current generation is built in VLSI and their physical size allows them to be used as add-on processors to workstations. Whilst these machines do not match the fastest pipelined computers available at present, arrays of 256^2 would give a performance of the order of several Gflops (10^9 floating point operations per second). A study of the suitability of this type of computer is therefore relevant to operational forecasting requirements. It could also be argued that research models that do not fully utilize the potential of a supercomputer would more suited to a local workstation with a DAP attached.

This thesis will demonstrate the feasibility of using processor arrays for meteorological modelling based on the implementation and performance study of two models, using the latest numerical methods, on the ICL DAP. Experience gained from this study should not be applicable solely to the DAP but also to other massively parallel computers. Comparison with the techniques used to implement models on other architectures might indicate the types of computer most suited to meteorological modelling. Guidelines could be put forward for the design of models on future parallel computers which have their origins in the computer architectures of the present.

The rest of this thesis is organized as follows. Chapter 2 reviews parallel computers, languages and algorithms. Chapter 3 reviews the various meteorological modelling techniques in use. Chapter 4 gives a general introduction to applying meteorological modelling to the DAP by reviewing past work. In chapter 5, new parallel algorithms for the Legendre transforms

are developed for the implementation of a barotropic spectral model on the DAP, described in chapter 6. Chapter 7 describes the implementation of a mesoscale finite element model on the DAP. Chapter 8 discusses the techniques used for meteorological modelling on other parallel architectures. Finally, chapter 9 presents conclusions and discusses the overall findings of this work.

CHAPTER 2

PARALLEL COMPUTERS, LANGUAGES AND ALGORITHMS

2.1. Introduction

In this chapter the ICL DAP is described, together with a brief description of some other parallel computers. It is important to review parallel computer architectures likely to be commercially available in the near future, to be able to comment on the architectures most suited to meteorological modelling and to anticipate future processing speeds. A discussion of programming languages for parallel computers is also presented. Finally, some of the considerations and problems involved in writing algorithms for parallel computers are described. Some well known algorithms for the ICL DAP, used in the meteorological models described in this thesis, are presented as examples.

2.2. Classification of parallel computers

The architecture of early computers is described as serial and referred to as the von Neumann organization. Flynn (1972) has classified this type of computer as a single instruction stream/single data stream (SISD) computer.

Parallelism can be introduced into computer architecture in several principal ways. From Hockney and Jesshope (1981), these are:

1. Pipelining - overlapping of separate operations in an assembly line fashion to improve the performance of an arithmetic or control unit.

2. Functional - providing several independent units for performing different functions, such as logic or addition, to operate simultaneously on different data.

3. Array - providing an array of identical processing elements operating simultaneously under the same instructions on different data.

4. Multiprocessing - the provision of several processors, each obeying its own instructions, with a facility for

interprocessor communication.

The idea of pipelining applies to basic operations, such as the steps in a floating point addition. If two or more pipes are present in a machine, different operations may be 'chained' to give an improved performance. For example, suppose a computer contained a multiplication pipe and an addition pipe. If a vector multiply instruction was followed by a vector add, chaining allows the addition pipe to take the results emerging from the multiplication pipe, without waiting for the first vector instruction to complete.

A pipelined vector computer is usually classified as a SISD computer (Hockney, 1977), although Flynn (1972) preferred to classify this type of architecture as single instruction stream/multiple data stream (SIMD). However, the term SIMD has become associated with arrays of processors working in lockstep (i.e. a common instruction stream) and is usually used to distinguish such array processors from pipelined computers. The classification of computers as multiple instruction stream/multiple data stream (MIMD) machines follows Flynn's (1972) definition and is used to refer to all forms of multiprocessors, where each processor obeys its own instructions.

This classification scheme, however, is very general. Shore (1973) and Hockey and Jesshope (1981) both describe more detailed classification schemes based on architectural configuration, rather than how the machine relates its instructions to the data being processed. However, these comprehensive schemes have not come into general use.

2.3. Description of some current and future parallel computers

There are now many parallel computers commercially available, with more used for research in universities. In this section, a brief overview is given of some parallel computers, most of which have been used for meteorological modelling.

2.3.1. Pipelined computers

2.3.1.1. Cray

The CRAY-1, first delivered in 1976, was the first commercially successful pipelined vector supercomputer. The CRAY X-MP is a multiprocessor upgrading of the CRAY-1 architecture, available with 1, 2 or 4 processors and central memory sizes of 2, 4, 8 or 16Mwords. A Solid-state Storage Device (SSD) is also available with memory sizes from 32 to 512Mwords. The SSD functions as a secondary memory and can be accessed using normal FORTRAN I/O statements. As the SSD is a nonrotating memory device, transfer rates are typically 2000Mbytes per second, 200 times faster than transfers from disk.

The clock period of the CRAY X-MP processor is 9.5nsecs. The peak speed of one processor is 210Mflops, so the peak performance of the 4 processor system is about 0.8Gflops.

The central memory is divided into 16 banks which are shared between the processors. As each processor has three computational ports, memory bank conflicts can arise. Larson (1988) describes some programming techniques to avoid memory contention.

The CRAY X-MP operating system allows multiple jobs to be executing simultaneously and programs to utilize several processors. Support for multiprocessing in programs is provided by the multitasking (or macrotasking) FORTRAN subroutine library and the microtasking FORTRAN compiler directives. As task creation overheads are relatively large, macrotasking is applied at the subroutine level, whereas the small overheads for microtasking mean it is applied at the loop level.

The CRAY Y-MP, due to be commercially available in late 1989, is an upgraded CRAY X-MP with 8 processors. Its peak performance is about 5Gflops.

The CRAY-2 is available with 2 or 4 processors. The combined peak performance of 4 processors is 1.8Gflops. The clock period is 4.1nsecs. Each processor is pipelined with separate functional units for integer, scalar and floating point arithmetic and eight 64 element vector registers.

The CRAY-2 has a large directly addressable shared memory of up to 256Mwords, arranged in 128 banks. In addition, each processor has a local memory of 16kwords used to temporarily hold scalar operands or vector segments during computation. The CRAY-2 offers the same parallel processing facilities as the CRAY X-MP.

The CRAY-3, due to be available in 1990, is an implementation of the CRAY-2 in gallium arsenide semiconductor technology. The machine is expected to have 8 or 16 processors and the performance of each is expected to be about 1Gflops.

2.3.1.2. ETA

The Engineering Technology Associates (ETA) company was founded in 1983, as an offshoot to the Control Data Corporation (CDC), with the aim of developing a 10Gflops supercomputer commonly known as the ETA-10. The ETA-10 can support up to 8 processors where each processor is based on the CDC CYBER 205 architecture. Each processor contains two vector pipelines but, unlike the CRAY machines, has no vector registers, fetching and saving vectors directly from memory instead. The two vector pipelines can operate concurrently with the scalar unit.

An unusual feature of the ETA-10 is that the processors are immersed in liquid nitrogen and cooled to a temperature of 77K. This enables the clock period to be reduced. The 8 processor ETA-10 is rated at 4.5Gflops for 64-bit and 9Gflops for 32-bit floating point arithmetic, with a 7nsecs clock period.

The ETA-10 is different from the CRAY computers as it has a hierarchical memory. Each processor is connected to its own 32Mbytes of local memory which in turn is connected to a shared memory of 256Mbytes to 2Gbytes in size. The shared memory is also connected to the I/O processors. The ETA-10 is a virtual memory machine, unlike the CRAY computers.

The parallel programming environment for the ETA-10 provides three types of approaches. The first is the use of compiler directives, the second is the provision of a multitasking library and the third, unique to ETA, is the provision of a FORTRAN-like language, used in a top-down design approach, which contains all the parallel constructs. This top layer then calls the FORTRAN subroutines to execute the computation. The memory hierarchy is reflected in

the complexity of the compiler directives and the multitasking library. Snelling (1988) reports that the ETA multitasking library has a total of 61 subroutines requiring a total of 187 parameters, compared to 16 subroutines and 22 parameters for the CRAY multitasking library.

2.3.1.3. IBM

The IBM corporation offer the IBM 3090/200 (2 processors) and the IBM 3090/400 (4 processors) with an optional vector facility for each processor. Each vector facility has two pipelined arithmetic units connected to 16 vector registers, each of which contains 128 32-bit elements. The peak performance of the 3090/400 is 430Mflops. The processors access a shared memory of 64Mbytes (model 200) or 128Mbytes (model 400), expandable to three times these sizes.

Macrotasking is available through a FORTRAN multitasking library. This is relatively simple, compared to that of CRAY, with 4 subroutines requiring a total of 5 parameters. No microtasking facilities are available.

2.3.1.4. Japanese supercomputers

The three largest Japanese computer manufacturers have been making pipelined supercomputers since the late 1970s. The current generation of supercomputers have only a single processor but have peak performances comparable to the CRAY and ETA machines.

Fujitsu have produced the 250Mflops VP-100, 500Mflops VP-200 and the 1Gflops VP-400. The machines have multiple vector units and a clock period of 7.5nsecs. They also contain a memory of 256Mbytes.

Hitachi have produced the 315Mflops S-810/10 and the 630Mflops S-810/20. This also has multiple vector units with a 14nsecs clock period. The S-810/20 may include 1Gbyte of solid state storage.

NEC make the 570Mflops SX-1 and the 1.3Gflops SX-2. Again, each machine has multiple vector units with a clock period of 6nsecs. The faster machine may include a 2Gbytes solid state storage device to go with 256Mbytes of main memory.

The next generation of Japanese supercomputers will be multiprocessors.

NEC recently announced the SX-3, due for release in the mid 1990s, which is a four processor machine capable of 22Gflops. Fujitsu are designing an 8 processor machine for the 1990s.

2.3.2. Array computers

Computers in this class include the SIMD type such as the DAP (described in detail in the next section) and the MIMD type such as the FPS T-series. The memory for these systems may be shared between processors through an interconnecting switching network or distributed local to each processor, with processors connected by a network or switch. Hockney (1985) discusses these architectural types in more detail.

2.3.2.1. Goodyear Aerospace Corporation

The Goodyear Aerospace Corporation delivered the SIMD Massively Parallel Processor (MPP) to the NASA Goddard Space Flight Centre in 1983. It consists of 1-bit processing elements (PEs) connected in a 128x128 mesh. Each PE has connections to its four nearest neighbours and has 1kbyte of memory. The machine was constructed in VLSI chips. The clock period is 100nsecs.

The MPP is designed principally for satellite picture processing but has also been used for fluid dynamics problems (Gallapoulos, 1984). The floating point performance of the machine is enhanced by the inclusion of a programmable shift register to give 200Mflops for 32-bit floating point multiplication.

Input and output on the MPP is accomplished by shifting data across the columns of the PE array, either from a separate host computer or an input interface. The MPP has only a global broadcast capability, unlike the DAP which has row and column broadcast capability. The MPP is programmed in a parallel version of Pascal.

2.3.2.2. Thinking Machines

The Connection Machine CM-1 built by Thinking Machines Incorporated has 65536 1-bit processors and is a SIMD machine. The recently introduced CM-2 adds 2048 Weitek floating point processors to give a peak performance of 32Gflops. Sixteen bit-serial processors are fabricated on a single chip and each processor has 8kbytes of local memory, so the machine contains

512Mbytes of memory as a whole. No code is stored in the local memory, the CM is used by a host computer to execute the parallel sections of a program; all serial code is executed on the host. Every two chips (32 1-bit processors) share a Weitek floating point unit.

Communication between processors on the same chip is achieved through a Benes network (Hockney and Jesshope, 1981) whilst the chips themselves are connected as a hypercube. In a hypercube, if all the processors are numbered 0 to n, processor i is connected to all processors j, such that the binary representations of i and j differ by 1 bit. See Hockney and Jesshope (1981) for more details. The hardware also supports communications on a two-dimensional grid, like the DAP, which is quicker than the general communication on the hypercube.

The Connection Machine can be programmed in parallel versions of FORTRAN, C and LISP. A useful feature of the CM is that the system supports virtual processors in that it can be programmed as if it has more processors than it actually has. Each processor simulates virtual processors by dividing its memory space into equal parts and sequentially serving the virtual processors. This feature is typically used to assign a virtual processor to each gridpoint in a numerical solution.

2.3.2.3. Floating Point Systems

The Floating Point Systems MIMD T-series is another distributed memory machine based on a hypercube network. Each node of the hypercube consists of an INMOS T400 transputer, 1Mbyte of memory and a Weitek floating point vector unit. The T400 transputer contains a 32-bit integer processor, 2Kbytes of RAM and four bi-directional serial communication ports, all on one chip in VLSI. The transputer acts as a controller to the vector unit and also performs integer arithmetic in parallel. The vector unit has pipelined functional units and vector registers. The peak performance of each node is 12Mflops. The largest possible T-series configuration would have 16384 nodes with a peak processing speed of 192Gflops. The computer can be programmed in OCCAM, FORTRAN or C.

2.3.2.4. Edinburgh Concurrent Supercomputer

When the ICL 2900 mainframe service at Edinburgh University was closed down, a replacement for the ICL DAPs was sought. This led to the Edinburgh Concurrent Supercomputer (ECS) project. The ECS is a MIMD machine consisting of T800 transputers as nodes on an electronically configurable network. The T800 transputer has a 32-bit floating point unit as well as an integer unit and 4Kbytes of memory instead of 2Kbytes on the T400. Each node on the ECS also has 4Mbytes of external memory. Each transputer has a peak performance of 0.8Mflops, although this can only be sustained if the data is stored in the internal memory.

The ECS is hosted by a microVAX with three 0.8Gbyte disks. It can be programmed in FORTRAN, C, OCCAM, or a mixture. At present, the largest single-user-resource offered on the ECS is 259 nodes, which would give a peak performance of approximately 200Mflops.

2.4. The ICL DAP

The design of the International Computers Limited (ICL) Distributed Array Processor (DAP) (Reddaway, 1973) was begun in 1972 and the first production machine was delivered in 1980, to Queen Mary College in London. The first DAP for Edinburgh University was delivered in 1982 and a second DAP delivered in 1983. The integration of the DAPs into the dual ICL 2976 mainframe service running the Edinburgh Multi-Access System (EMAS) operating system is described by Brown (1986) and Stephens and Yarwood (1986).

2.4.1. Architecture

The DAP is a SIMD computer comprising a 64x64 array of 1-bit processing elements. Each PE is connected to its four nearest neighbours and has its own 4Kbits of memory, expandable to 16Kbits (see Fig. 1). Each DAP at Edinburgh University had a memory size of 2Mbytes. The cycle time of the DAP is 200nsecs.

Unlike most supercomputers, the DAP was built from modest technology at low levels of integration. The simple nature of the bit processors meant the hardware was cheap and quick to develop and manufacture. The drawback

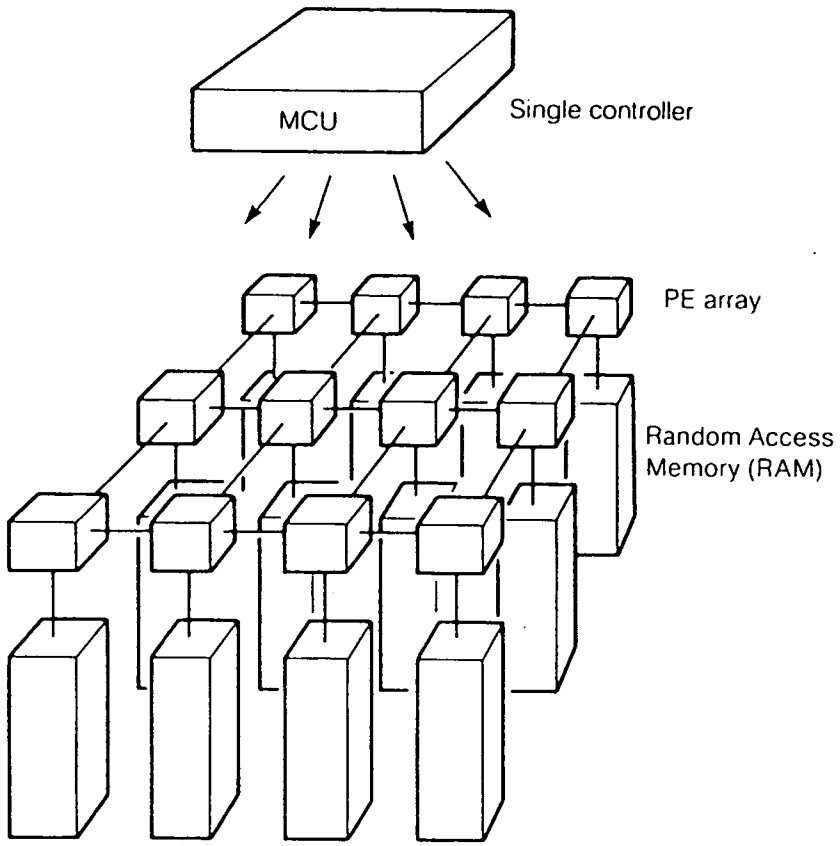


Figure 1. Schematic illustration of the DAP showing processor connections and processor memory. From Ettinger (1987).

was the physical size of the machine.

The decoding and issuing of instructions is performed by the Master Control Unit (MCU). The code for the DAP and the program's data are both stored in the processor memory. The MCU contains eight 64-bit registers which are connected to row and column highways leading to each processor. Thus register data can be broadcast to all the array, or to all rows or all columns.

The DAP forms an integral part of the host mainframe ICL 2900. It can provide memory in the conventional way, for example if the host system is being heavily used, or it can be instructed to execute autonomously. The DAP interface to the 2900 mainframe is provided by the DAP access controller and the column highway. Therefore, data transfer between the host and the DAP takes place 1 row of 64 bits at a time, requiring 1 cycle per access.

Each PE contains an activity register which is used to determine whether store operations are obeyed or not. Therefore the writing of data back to memory can be controlled for each individual PE, by the user, in the DAP program.

Flanders *et al.* (1977) and Hockney and Jesshope (1981) give more details on the architecture of the DAP.

2.4.2. Storage and processing modes

There are three formats in which data can be held in the DAP store. The first, known as FORTRAN storage mode, is the way the host stores data in the DAP memory (ICL, 1979). Two 32-bit words are held in a row of a DAP store plane. Before it can be used, the DAP program must reformat this data, into either the horizontal or vertical storage format.

The horizontal storage format is similar to FORTRAN storage mode except that each word is held one per row, right justified. This mode leads to vector processing where 64 numbers are stored on one plane and processed in parallel.

In the vertical storage format, words are held vertically in the store of each PE. This gives matrix processing in which 4096 numbers are processed in

parallel, but the arithmetic in this case is bit serial. Arithmetic operations take longer in vertical mode than in horizontal mode but the factor is less than 64 so that optimum processing is achieved in vertical mode.

2.4.3. Programming and performance

The DAP is programmed in DAP FORTRAN, which is based on FORTRAN77 but includes extensions to support vector and matrix processing. The language is described in a later section.

A program to use the DAP consists of a FORTRAN part that runs on the host and a DAP FORTRAN part that is called from the host part of the program. Brown (1986) describes the procedure for the preparation and running of DAP programs at Edinburgh University in more detail.

DAP FORTRAN does not support the FORTRAN READ and WRITE statements and all communication of data is made via COMMON blocks shared between the DAP and the host. The lack of DAP FORTRAN I/O statements means that debugging DAP FORTRAN programs becomes difficult.

As the DAP consists of 1-bit processing elements, basic arithmetic operations have to be built up in software. For the matrix processing mode, there is a strong dependency of performance on word length. DAP FORTRAN will allow floating point variables to be declared as 3 to 8 bytes in length, integers 1 to 8 bytes. For integer arithmetic, performance varies linearly with word length for addition and as the square of the word length for multiplication. The overheads of exponent and mantissa manipulation mean the same is not true for floating point addition and multiplication. However, for a typical mix of addition and multiplication, floating point operations show a linear trend with word length. There are obvious gains in performance to be made if block floating point arithmetic is used. In block floating point arithmetic, a single exponent is used for a data array. The largest value in the array determines the scale factor and exponent for the entire array. However, operations are still done in fixed point arithmetic as all values share the same exponent. If overflow occurs, the exponent increases and the entire array is shifted 1 bit.

For 32-bit floating point numbers in matrix mode, addition takes place at 25Mflops and multiplication at 14Mflops. Bit level algorithms can be used for

operations such as computing the sum, logarithm or square root of the 4096 values in the PE array. The relative performances of these operations to basic arithmetic operations is often very different to that found on conventional architectures. For example, the DAP FORTRAN version of the MAX function achieves a processing rate of 60Mflops whilst changing the sign of 4096 vertically stored numbers can be done at 10Gflops (Flanders *et al*, 1977).

2.4.4. Next generations

The second generation DAP, produced in 1986 and called the mil-DAP, was a 32x32 PE array in LSI. The clock cycle was 145nsecs and the machine had 2Mbytes of memory. It was hosted by an ICL Perq workstation and was mainly intended for signal processing.

The Active Memory Technology (AMT) company was set up in October 1986, as a spin off from ICL, to develop and market the DAP technology. They have produced the third generation DAP, the AMT DAP 500 and 600 series implemented in VLSI. The first machine, a DAP 510, was delivered in late 1987.

The third generation DAP has many improvements over the first generation, both in terms of hardware and software. One significant difference is that they are add-on processors to a SUN or VAX computer, rather than forming an integral part of a mainframe computer. They can either provide high performance for a workstation environment or act as a mainframe service.

The 500 series comprises a 32x32 array of PEs with either a 6MHz (167nsecs; model 506) or 10MHz (100nsecs; model 510) clock. The 600 series is a 64x64 array.

The PE array is controlled from the Master Control Unit as before, except that the DAP program code is now held in a separate code memory attached to the MCU. This has a minimum size of 0.5Mbytes and can be expanded to 2Mbytes. The minimum size for the PE memory is 32Kbits expandable up to 1Mbit, giving a maximum total memory of 128Mbytes for the 500 series or 512Mbytes for the 600 series.

A Host Connection Unit is responsible for all communication to the host. It contains a 32-bit microprocessor with various interfaces and connects to the MCU.

The PE in this generation is exactly the same as in the original DAP. However, an additional bit-plane exists between the processors and their memory to act as a fast data transfer channel between memory and peripheral devices, such as a high resolution colour display. Data can be transferred at a rate of 70Mbytes/sec utilizing just 6% of the processor cycles during the transfer. This facility, coupled with the DAP's ability to use short word lengths, makes it well suited to graphics and image processing applications.

The DAP 510 has a maximum floating point performance, on 32-bit words, of 60Mflops. This is 2.5 times faster than the 64x64 ICL DAP. The DAP 610 would therefore be rated at 240Mflops, nearly 10 times faster than the first generation DAP; the difference in clock period accounting for a factor of 2. A DAP 710 (2^7 or 128x128) would have a peak performance of about 1Gflops, 5 times faster than the MPP which also has a 128x128 array and a 100nsec clock.

The programming environment has also been improved. The AMT DAP can be programmed in FORTRAN-PLUS (DAP FORTRAN renamed) and the DAP assembly language APAL. DAP programs are developed on the host as before. Any run-time errors, such as divide by zero, are detected by the DAP which informs the user on the host and automatically enters an interactive debugging mode. The user can then request the value of variables by name. Alternatively, a dump can be made and saved for later analysis. When in this debugging mode, the user can resume or abandon the run at any time.

AMT recently announced (AMT, 1989) their intention to develop FORTRAN-PLUS (enhanced) and FORTRAN 8X compilers. The main new feature of FORTRAN-PLUS (enhanced) is the lack of size constraints for matrix and vector type variables. In FORTRAN-PLUS (or DAP FORTRAN) these types are constrained to the size of the PE array. The FORTRAN 8X language is attractive to AMT as it includes similar array processing statements to DAP FORTRAN. As it will also need to support all FORTRAN 77 instructions, this makes the DAP potentially available to more users.

In reaching conclusions about the suitability of the DAP to meteorological modelling, it is obviously important to take account of the facilities provided by this latest generation of the DAP.

2.5. Programming languages for parallel computers

2.5.1. Introduction

Every programming language is based on the type of computer system that the computer program is to run on. The sequential von Neumann architecture has therefore influenced the design of programs since the 1950s. With the development of parallel computers (SIMD and MIMD) the need arose for languages to reflect the new hardware, either by extracting parallelism from existing sequential programs (e.g. vectorization, parallel processing subroutine libraries, refined languages), by extending languages to include parallel constructs (e.g. DAP FORTRAN) or by designing new parallel languages (e.g. OCCAM). A great deal of work has been done in these areas and they are discussed in more detail in subsequent sections and by Jesshope (1987).

Hockney and Jesshope (1981) introduce the principle of the conservation of parallelism. This states that the degree of parallelism should not decrease from the algorithm development stage, through the programming stage to the code executing on the machine. This is desirable because it is much easier to translate from a parallel to a serial approach, rather than from a sequential to parallel approach which would require a detailed program analysis. Unfortunately, the compilers of pipelined vector computers violate this principle, in that intervention by the programmer is often required to achieve the desired performance. However, this method has been successful because of the long development time and the cost of large FORTRAN programs. It also allows programs to be ported between different computers, something that is generally not possible for the extended or new parallel languages without rewriting the code.

2.5.2. Data management

The extraction of parallelism from a program is the application of the Parallelization Principle (Klappholz *et al.*, 1987) by the compiler. Put simply, this is the analysis of the program's pattern of data access, as code sections can be performed in parallel if they do not write to the same area of memory. Thus the parallelization of code is intimately related to the access of data.

Data management is probably the most critical aspect of parallel processing. A multiprocessor system may support a shared memory directly

connected to the processors (e.g. CRAY X-MP), or connected via a switching network. The memory may be distributed, where each processor has direct connection to its own local memory (e.g. ICL DAP), or a memory hierarchy may exist. In the same way, a programmer may require that data is to be shared between all processors or is private to a processor. There may also be a need for semi-private data, which is to be shared between groups of processors. The availability of these data classes within a language will be strongly influenced by the intended architecture and have an effect on the design of the algorithms. Data security is also an issue, since unrepeatable results (known as races) can occur from improper use of shared data.

2.5.3. Models of parallelism

There are two ways in which parallelism can be exploited in applications. The first is 'data parallelism' where an operation or sequence of operations is applied simultaneously to a set of data values. This is SIMD mode. The second is known as 'process parallelism', where multiple operations, or multiple operation sequences, are applied simultaneously to the same or different data. Languages for SIMD computers need only support data parallelism, whereas both may be supported for MIMD computers. The requirements and features of languages that support these models of parallelism are described at length by Hockney and Jesshope (1981) and Jesshope (1987) and are summarized below.

2.5.3.1. Data parallelism

In a sequential language, operations on an array are applied on an array element basis within a loop. When these operations are independent of the data, parallel processing can be applied. Thus, it is natural to use the array as a basic parallel construct. The most general approach is where an array of any number of dimensions can be treated as a distinct object. Any reference to the array then refers to all the elements in the array. This approach has been adopted for the forthcoming FORTRAN 8X standard (Reid and Wilson, 1986).

Many languages, essentially those for processor arrays, have compromised this general approach and restricted the number of dimensions over which the array can be considered as a parallel object. Any further dimensions must be

indexed. DAP FORTRAN (ICL, 1979) uses this approach, where the first or first two dimensions only may be referenced in parallel and these dimensions are constrained to the size of the processor array. This has the disadvantage that the language becomes machine and array size dependent, making programs nonportable.

Indexing parallel data objects can be thought of as a rank reducing operation (Hockney and Jesshope, 1981). Thus, a two-dimensional matrix could be indexed as a vector or a scalar. More generally, it becomes necessary to specify regions of the array which are to be manipulated and updated. DAP FORTRAN provides good examples of these facilities and is discussed later.

2.5.3.2. Process parallelism

Process parallelism in programming languages has arisen from the need to exploit MIMD multiprocessor computers and concurrency in algorithms. Support for concurrent processes in a language can be provided by allowing process creation during execution or by assuming all processes to be active at the start of the program and remain so for its lifetime.

Process creation during program execution uses the fork/join approach. The CRAY multitasking library provides a `TSKSTART` routine to create a new process (or task) and a `TSKWAIT` routine which causes a process to wait until the specified process terminates. The PAR construct in OCCAM, placed around statements, uses the fork/join approach but can be applied to a piece of work of a shorter time duration than on the CRAY because of the smaller process creation overheads.

Process parallelism is an MIMD mode of operation in which the task of algorithm design is inherently more complex than for SIMD mode or the use of data parallelism. What is more, an MIMD algorithm will be harder to debug since new types of errors, not possible in SIMD (or SISD) programming, can occur. These are described in more detail in a later section. As discussed by Jesshope (1987), this additional complexity requires expressive programming languages with simple but powerful abstract constructs with which to exploit the concurrency in the algorithms.

2.5.4. Communication and synchronization

Communication and synchronization overheads are among the main reasons why the speed-up of a program on N processors is not N. SIMD machines are simpler than MIMD machines in this respect as no synchronization is required.

Communication between processes can be via an interconnecting network (e.g. a transputer based architecture), a communications buffer (e.g. the ETA-10) or by a shared variable in a global memory (e.g. CRAY X-MP). For a network system, messages or data can be passed directly between processes (direct-send systems) or left with a global entity for later collection by the relevant process (mailbox systems). The level at which communication is supported in a language varies. Since communication is generally an overhead (except in cases where the cost can be completely masked by concurrent processing), parallel MIMD algorithms should ideally contain a minimum of interprocessor communication coupled with a fast and efficient communication network.

Unless each process is independent, some means of synchronization will be required to ensure the correct behaviour of the program. There are two basic types of synchronization; data and control oriented.

Data oriented synchronization is used to synchronize the updating of variables which are shared between processes. Variables must have an associated status to indicate whether they are full or empty. The programming language described by Jordan (1987), 'The Force', supports this form of synchronization.

One control oriented synchronization concept is that of the barrier (Axelrod, 1986). A barrier defines a point in the control flow of an algorithm or program at which all processes must arrive before any are allowed to proceed further. A barrier is expensive in terms of communication since each process must communicate with every other process. Additionally, since all processes must wait at the barrier until the last arrives, the effects of fluctuations in process execution time or imperfect load balancing are maximized. However, one advantage of the barrier is that it can be implemented such that each process does not need to explicitly perform any communication (this can be hidden at a lower system level), so that programs

can be independent of the number of processes (Jordan, 1987). The fork/join approach to process control also provides a means of synchronization as mentioned previously.

Control oriented synchronization also uses the concept of a critical region. A critical region is a protected section of sequential code through which only one process is allowed to progress at any time. If any other processes reach this critical region whilst another process is executing it, they wait until that process has completed. This mutually exclusive condition is usually used for writing to shared memory locations. The LOCKs in the CRAY multitasking library (CRAY, 1986) are a facility for defining critical regions.

The concepts of communication and synchronization are unified in the paper by Hoare (1978), where simple input and output commands are introduced for communication between processes. If one process expects input from another, it will wait until the second process is ready to send. Likewise, a process will wait to send data until the receiving process is ready. Thus processes can be made to synchronize by communicating. This simple approach can be extended to the barrier concept. These ideas have since been incorporated into the OCCAM and ADA programming languages.

2.5.5. Parallelism from sequential languages

The pipelined vector class of computers use vectorizing compilers to generate parallel machine instructions from sequential code. This method is attractive because standard FORTRAN can generally be used, although for the best performance the programmer usually has to adopt a suitable programming style. The compiler will analyse DO loops, generally the innermost loop, substituting vector instructions. More intelligent compilers will analyse nested loops and can reorder them to remove a data dependency in an inner loop (Hockney and Jesshope, 1981).

In a loop, conditional statements and sequential data dependencies can inhibit vectorization. Conditional statements can often be vectorized by using a masking technique. Those loops with conditional statements that cannot be vectorized can usually be replaced with a vector function (e.g. the CRAY vector merge functions).

Facilities for FORTRAN programs to make use of multiple processors have

been provided in the form of compiler directives and subroutine libraries. Compiler directives take the form of a normal FORTRAN comment line with a special string of characters, including the directive, which the compiler recognizes. Compiler directives are advantageous since they do not require modifications to executable FORTRAN statements. Parallel processing subroutine libraries are provided by the major supercomputer manufacturers CRAY, ETA and IBM. These vary greatly in the facilities offered, as discussed in the survey by Snelling and Hoffmann (1988). All use the fork/join concept of process control discussed above. The libraries all reflect the underlying architecture and are therefore machine dependent.

The refined language methodology (Klappholz *et al*, 1987) offers an evolutionary approach to the problem of moving programs from SISD to MIMD machines. A range of tools is provided to support the programmer in developing parallel programs based on a refined sequential language (i.e. FORTRAN or C). Klappholz *et al.* (1987) describe the software tool Prefine, which analyses the pattern of data access within subroutines and between subroutines. The analysis of this tool can be combined with a sequential program in standard FORTRAN or C, which the compiler then exploits by using the parallelization principle to translate specifications of data access rights into parallel processes. The additional data access information enables a larger degree of parallelism to be detected. Furthermore, runtime error checking code can use the same information to prevent races. In essence, Prefine performs the translation from standard FORTRAN or C to the refined version of the language. The refined version is an extension of the original with added syntax for definition of data access rights. The refined languages are therefore still sequential languages, but provide an evolutionary step for moving large sequential programs to a safe parallel form, free from the errors associated with MIMD programs.

2.5.6. Extensions to existing languages

Language extensions for parallel processing use either data parallelism or process parallelism, occasionally both. Languages for SIMD computers use the data parallelism approach by incorporating parallel data objects into their syntax and providing facilities for manipulating these objects. Extensions for use with MIMD machines have generally supported process parallelism through the use of preprocessors or macros (Leasure, 1988) to hide machine

dependent instructions.

Extended languages to support data parallelism include; Actus II (a Pascal based language for the DAP, described by Perrott *et al*, 1987), DAP FORTRAN (described in more detail below) and FORTRAN 8X (Reid and Wilson, 1986). Both Actus II and FORTRAN 8X support the declaration of arrays as parallel objects with unconstrained rank (or number of parallel dimensions) and range (length of each dimension) unlike DAP FORTRAN which is more restrictive. All the languages have been extended so that functions and subroutines can be passed and can return parallel data objects. Actus II provides indexing facilities and operations for data alignment. FORTRAN 8X is expected to include powerful indexing facilities similar to DAP FORTRAN. For example, the statement, `WHERE(A.GT.0) B = B/A` performs the division on elements in `B` where the corresponding elements in `A` are nonzero (`A` and `B` are parallel data arrays). In DAP FORTRAN, this would be written as `B(A.GT.0) = B/A`. One important advantage with parallel data constructs is that the program becomes more concise and easier to read as, for example, loops over individual array elements are no longer needed.

The Force programming language (Jordan, 1987) is an example of a portable, parallel programming language based on FORTRAN, incorporating process control primitives which are translated to machine dependent statements using a macro preprocessor. The language is portable because the parallel instructions embody simple process parallelism constructs which are machine independent. Shared and private data are supported as well as data and control synchronization concepts. The Force has been implemented on a variety of computers (e.g. the CRAY-2), but can only be applied to shared memory multiprocessors.

2.5.6.1. DAP FORTRAN

As an example of an extended language supporting data parallelism and as the language is used for the work presented in this thesis, a brief review of DAP FORTRAN (ICL, 1979) is given here.

In DAP FORTRAN, arrays are declared in the usual way. For example,

```
DIMENSION  VECT(), MAT(,), MAT3(,,3)
```

defines VECT to be a vector of 64 elements, stored in horizontal format, with MAT a matrix of 64x64 elements, stored in the vertical format. Both are parallel data objects. Arrays or sets of vectors and matrices can be defined in the usual way; MAT3 is an array of 64x64x3 elements.

Operations on vectors and matrices take place on all elements simultaneously. For example, the FORTRAN code below,

```
      DO 1 J = 1, 64
      DO 1 I = 1, 64
    1 M1(I,J) = M1(I,J) + M2(I,J)
```

could be replaced in DAP FORTRAN, by,

```
      M1 = M1 + M2
```

Indexing the constrained (parallel) dimensions can be performed with integer, integer vector or logical indices, on the left and right hand side of assignment statements. For example, MAT(,3) would select a vector equal to the third column of the matrix. If IV() is an integer vector, whose elements all lie in the range 1 to 64, MAT(IV,) selects a vector containing MAT(IV(I),I) in element I. If LV() and LM(,) are a logical vector and logical matrix respectively, MAT(,LV) selects a column and MAT(LM) selects a scalar when used on the RHS of an assignment statement or in a procedure call. The logical vector and matrix must have only one element TRUE. When this indexing is used on the LHS of an assignment statement, this restriction does not apply. For example, MAT(LM) = ... would cause updating of the matrix elements only where the corresponding element of LM was TRUE. This powerful facility is known as masking. Any valid logical expression can be substituted in place of the logical matrix, as shown in the previous section. As logical data objects occupy one bit plane, logical operations are fast and have a negligible overhead in the indexing operations.

The movement of data between processors is done using functions. Planar and cyclic boundary conditions can be specified at the edges of the PE array. When planar conditions are specified, zeros are shifted in to replace boundary values. With cyclic conditions, data shifted off one edge of the array appear at

the opposite edge. For example, SHNP(MAT,3) would shift the matrix 3 PEs north (SHift-North-Planar) with planar boundary conditions. Data can also be shifted south, east and west across the array using similar functions. Functions to shift vectors also exist. The execution times for the shift functions are given in Table 1.

Other functions are provided to transform data in matrices. The TRAN function performs a matrix transpose, whilst REVC and REVR reverse the column and row ordering of the matrices. There are other DAP FORTRAN functions for logical operations, reduction operations and type conversion functions, the timings of some of which are presented in Table 2.

Most of the standard FORTRAN functions are included in DAP FORTRAN and extended to support data in vector and matrix mode. Timings of some of these functions together with the CPU times for basic arithmetic operations on the DAP are given in Table 3.

Matrices may also be treated as 'long vectors' in DAP FORTRAN, where the vector is formed by conceptually concatenating successive columns of the matrix. For certain applications, a long vector mapping of data onto the PE array is advantageous. A number of functions are available in DAP FORTRAN for long vector operations (e.g. shift functions).

2.5.7. Parallel languages

Several new languages have been designed which support process parallelism. Two examples are ADA and OCCAM, both described in Jesshope (1987).

ADA was designed in the mid-1970s for the U.S. Department of Defence. It supports process parallelism but not data parallelism. Processes can be created dynamically i.e. process control uses the fork/join concept. Processes communicate with each other by passing parameters. This also provides a synchronization mechanism in that if a process calls another it will wait until the called process is ready to receive the call.

OCCAM is a lower-level language in that it has a direct correspondence between hardware (the transputer) and code. Communication is via channels, with input and output statements representing primitive processes i.e. a single

Number of places (n)	2	4	8	16	32
Matrix shift	32	48	72	120	232
Vector shift	8	8	8	8	8

Table 1.

DAP routing operation times in μsecs for M = SHEC(M,n) and V = SHLC(V,n). From Fishbourne (1980).

Operation		Time (μsec)	
Z = M*S	Matrix x scalar.	120-296	(136-224)
Z = V*S	Vector x scalar.	64	(80)
Z = M*MATR(V)	Matrix x vector (expanded by rows).	304	(328)
S = SUM(M)	Summation of all elements of a matrix.	464	(488)
V = SUMC(M)	Summation along each row to give a column vector.	352	(368)
Z = MATR(V)	Expansion of a vector to a matrix by rows.	32	(32)
Z = REVC(M)	Reverse column ordering of matrix.	232	(232)
Z = TRAN(M)	Transpose matrix.	216	(216)
LS = ANY(LM)	Logical .OR. of all elements of a logical matrix.	2.25	(3.25)
LM = ALTC(S)	Generate logical matrix with alternate S columns .TRUE.	3.8	(4.1)
LM = LM1 .OR. LM2	Logical .OR. of two logical matrices.	2.1	(2.5)

Table 2.

DAP times for mixed mode operations and functions. All times are in μsecs for 32-bit floating point arithmetic and logical operations where appropriate. The times in brackets are with all the run-time checks performed. From Fishbourne (1980).

Operation	Matrix Mode	Vector Mode	Scalar Mode
Z = X + Y	152 (176)	64 (80)	24 (40)
Z = X * Y	272 (296)	64 (80)	40 (48)
Z = X / Y	376 (408)	136 (146)	80 (88)
Z = X ** 2	152 (160)	152 (160)	96 (104)
Z = Y	16 (16)	8 (8)	8 (8)
Z = LOG(X)	312 (328)	376 (384)	376 (384)
Z = SIN(X)	856 (872)	920 (928)	920 (928)
Z = COS(X)	848 (864)	920 (928)	912 (920)
Z = ABS(X)	24 (24)	16 (16)	8 (8)
Z = SQRT(X)	192 (208)	272 (280)	256 (264)

Table 3.

DAP times for basic arithmetic operations and standard functions. All times are in μsecs for 32-bit floating point numbers. The times in brackets are with all run-time checks carried out. Matrix mode processing uses 4096 numbers, vector mode 64 numbers and scalar mode 1 number. From Fishbourne (1980).

statement in OCCAM is a process. Synchronization is similar to ADA, as communication between two processes cannot proceed until they are both ready. Parallelism is introduced into OCCAM by the PAR statement, as described previously. Execution proceeds after the PAR statement only when every process within that PAR block has finished, thus providing another method of synchronization.

OCCAM can only be used for distributed memory systems since it does not provide for shared memory. ADA can be applied to both shared and distributed memory systems but is more suited to the former. In ADA, there is no language syntax available to specify which processes should be assigned to individual processors on a distributed system, unlike OCCAM.

2.6. Parallel programming and algorithms

2.6.1. Measuring computer performance

The most frequently used measure of a computer's performance is the peak floating point operation rate in millions of floating point operations per second (Mflops). Such figures are misleading on any computer but especially parallel computers because they do not take into account; finite vector length, memory latency, bank conflicts, multitasking overheads and so on. It is often necessary to qualify a Mflops rate by stating what floating point operation is being considered. For example, the rate at which floating point addition and multiplication are performed on the DAP differs by a factor of 1.7.

To obtain a measure of the likely performance of an application on a computer, benchmarks are often used. The kernel method of benchmarking involves extracting the core routines, responsible for a large proportion of the execution time, from a program and timing the execution of these routines. This method does not provide an accurate performance figure for the application as it is not based on a statistical collection of instructions. However, the kernel can be altered to suit the target architecture and provides a good indication of the likely performance.

There are several well known kernel benchmarks. The LINPACK benchmark (Dongarra, 1985) is based on the Basic Linear Algebra Subprograms (BLAS) and solves a dense system of linear equations. The benchmark is therefore

application specific. The more general Livermore Loops benchmark (Feo, 1988) represents the type of computational kernels typically found in large scale scientific computing. The kernels range from operations such as the calculation of an inner product and matrix multiplication, to searching and storing algorithms typical of Monte Carlo methods. The kernels are all extracts from production programs run at the Lawrence Livermore national laboratory and there are 24 loops in all. The loops contain code fragments that range from sequential to completely vectorizable and are also suited to parallel processing.

Hockney and Jesshope (1981) introduce two parameters, $n_{\frac{1}{2}}$ and r_∞, which are used to characterize the performance of a computer, serial or parallel. These parameters are related to the time taken for an arithmetic operation, t, on a vector of length n, by the equation,

$$t = (n + n_{\frac{1}{2}}) \; / \; r_\infty \qquad (2.6.1)$$

The r_∞ parameter is the maximum or asymptotic performance of the computer i.e. the maximum rate of computation in units of equivalent scalar operations performed per second, for a vector of infinite length. This gives an indication of the performance for long vectors and is a characteristic of the computer technology used. It is also a function of the type of operation under consideration. For pipelined computers it varies with dyadic, triadic, register-to-register and memory-to-memory operations. For a DAP it varies with addition or multiplication operations, floating point and integer arithmetic and the precision used.

The $n_{\frac{1}{2}}$ parameter is known as the half-performance length as it is the vector length, n, required to achieve half the maximum performance. This parameter is a measure of the amount of parallelism in the computer architecture. It varies from 0 for a serial computer, to infinity for an infinite array of processors. For a pipelined machine, this parameter is proportional to the vector start up time and is typically of the order of 100. For an array processor, such as the DAP, Hockney and Jesshope (1981) consider two cases. If the vector length is less than the number of processors, the array appears as an infinite array and the half-performance length is infinity. The time for the operation on the vector is just the time for one parallel operation. If n is more than the number of processors, N, the half-performance length is given

by $N/2$. Like the parameter r_∞, $n_{\frac{1}{2}}$ also depends on the precise operation being considered.

The half-performance length is used by Hockney and Jesshope (1981) as a measure of the relative performance of different algorithms on the same computer. The ratio $\nu = n_{\frac{1}{2}}/n$ represents how parallel the computer appears to the algorithm. For example, if ν is zero or small a sequential algorithm is appropriate, whereas if ν is large a highly parallel algorithm is appropriate.

For the DAP, the half-performance length serves to emphasize the highly parallel nature of the architecture. The use of $n_{\frac{1}{2}}$ and r_∞ therefore merely state the obvious, that execution time depends on the number of parallel operations. This method of algorithm performance analysis therefore has little use for processor array applications.

The parameter $s_{\frac{1}{2}}$ was introduced by Hockney and Snelling (1984) in order to assess the synchronization overhead in MIMD computers. If t is the time for a piece of work between synchronization points, the timing equation analogous to Eq.(2.6.1) is,

$$t = (s + s_{\frac{1}{2}}) \; / \; r_\infty \tag{2.6.2}$$

where s is the total number of floating point operations during this piece of work. The parameter $s_{\frac{1}{2}}$, expresses the overhead in terms of how many floating point operations could have been performed in the time taken for synchronization. When s is equal to $s_{\frac{1}{2}}$, half the time is spent on useful arithmetic, the other on synchronization. This parameter therefore indicates whether the work under consideration is of sufficient duration for efficient multiprocessing. Hockney (1988) measured this parameter for a CRAY X-MP/2 (2 processors) and obtained a value in the range 2000-6000flop , depending on the synchronization method used. For the DAP, $s_{\frac{1}{2}}$ is zero.

Benchmarking can omit some important issues, such as the I/O requirements of the program or whether bottlenecks occur with the full program. The full program may not fit entirely in memory so the effect of disks and so on is unaccounted for. However, benchmarks are useful because of their simplicity. They are economical to test on several architectures and serve to indicate which computers should be investigated further in a procurement process. Meteorological benchmarks are discussed in chapter 8.

2.6.2. SIMD programming

In this and the following section, some of the issues in programming parallel computers are discussed. This section is concerned with SIMD computers, more specifically processor arrays, which present different issues to MIMD computers.

2.6.2.1. Data mapping

The SIMD processor array computer is often considered a specialized machine whereas the pipelined MIMD computer is thought of as a general purpose computer (Suarez, 1988). The reason for this is that the application must contain arrays with at least as many elements as the number of processors. However, this level of parallelism is generally easy to achieve with atmospheric models and MIMD programming is more complex, as will be seen in the next section.

The most important issue in programming SIMD processor arrays is the mapping of data i.e. the placement of array elements to processors. This is because it not only affects the efficiency of the program, since as many processors as possible should be kept doing useful work, but also because it affects the parallel algorithms used. On a processor array, there is a much stronger relationship between the storage mapping and the algorithm than in pipelined or sequential machines.

The need to exploit all the processors constrains the problem size, for efficiency reasons. For example, grids in meteorological models should ideally be multiples of the array size. Also, meteorologists find it useful to vary the resolution of models in order to test their behaviour. This would require good support from the software environment on the computer.

Mapping data arrays larger than the DAP can be done in two ways. The first is to divide the domain into sections or 'sheets', where each sheet is the same size as the DAP array as shown in Fig. 2. The sheets are then stored successively in the PE memory. This arrangement has a disadvantage in that the boundary processors in each sheet have to be treated separately if they require access to the data in neighbouring gridpoints in another sheet. The other method is to assign neighbouring gridpoints to each processor. This is known as a 'crinkled' mapping and illustrated in Fig. 3. In this mapping, each

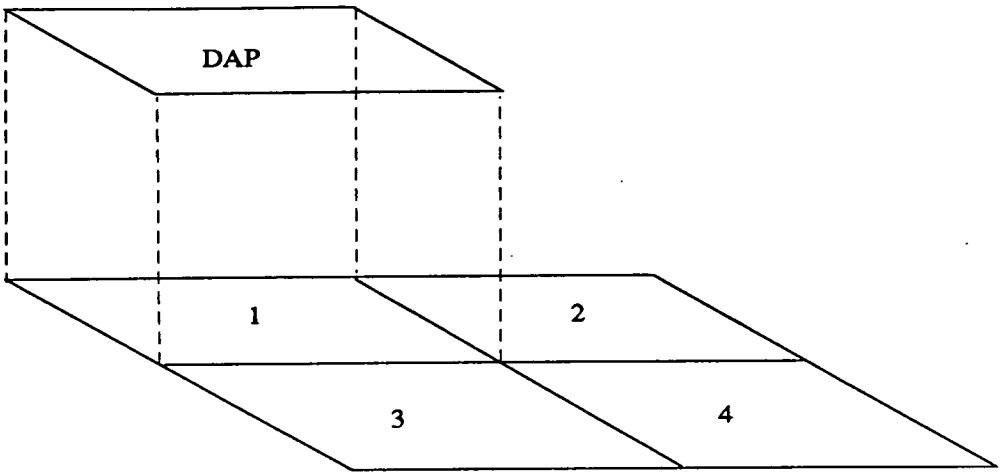


Figure 2. Sheet mapping of arrays onto the DAP.

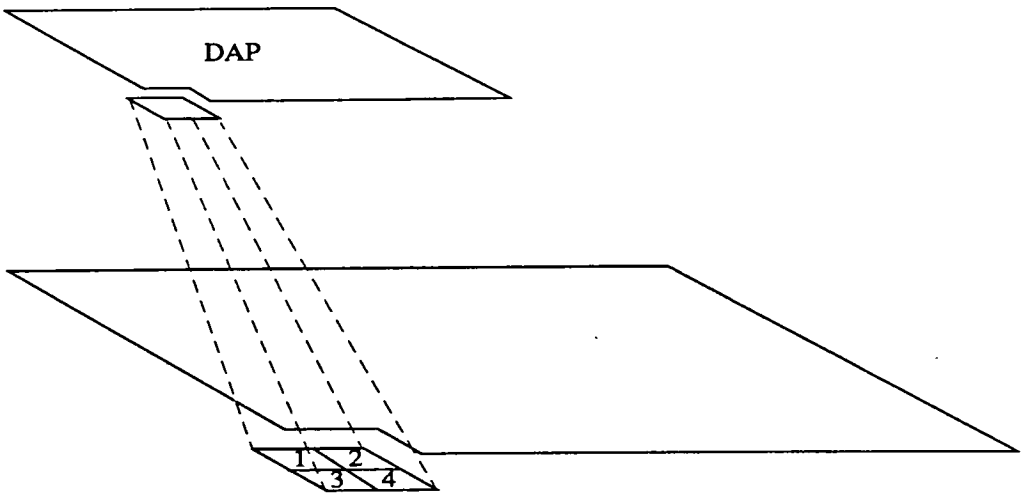


Figure 3. Crinkled mapping of arrays onto the DAP.

PE can access data from neighbouring points either in its own memory or that of a neighbour from processors nearby. The routing required for this approach is less than for sheet mapping and additional boundary calculations do not arise. The mapping of irregularly shaped grids to the DAP depends on the precise problem and cannot be discussed generally. Some examples of mappings of nonrectangular arrays in finite difference and spectral models are given in chapters 4 and 5 respectively.

2.6.2.2. Data routing

The routing or movement of data between PEs during a program can be regarded as an overhead to the parallel approach. The size of this overhead depends on the amount of routing required and the ratio of the cost of routing to that of the arithmetic. On the DAP, routing a matrix by one PE costs about a tenth of a 32-bit floating point operation. Other transformations, such as a matrix transpose, are more expensive. The routing overhead therefore also depends on the type of routing. The relatively low cost of routing on the DAP generally leads to acceptable routing overheads. However, for block point or integer arithmetic, the relative cost of routing increases and may become significant (Hockney and Jesshope, 1981).

2.6.2.3. Performance

To indicate the performance of parallel computers, a speedup ratio is often used. This is the ratio of the time on a single processor of a computer to that on all the processors. For processor arrays, speedup ratios are somewhat meaningless, since they cannot be sensibly used in SISD mode, unlike MIMD machines. A more meaningful performance measure is the average number of processors kept busy during the program. To define this efficiency, E, the total time of a program, T, is divided into time slices, t_p, where the number of processors doing useful work in each time slice stays constant at m_p. The efficiency can then be defined as,

$$E = \sum_{p=1}^{P} (t_p/T)(m_p/n_p) \qquad (2.6.3)$$

where n_p is the number of processors actually used at the p^{th} stage and P is the total number of time slices.

Consider a simple example where each gridpoint takes a time t_i to compute where,

$$t_i = t_{MAX} \quad \text{if} \quad i \leq rN \tag{2.6.4}$$

$$t_i = ft_{MAX} \quad \text{if} \quad i > rN$$

where N is the number of gridpoints so that a fraction, r, of the gridpoints take the maximum time, t_{MAX}, to be computed. The time, T_1, for a serial version is then the sum of the times for the individual points,

$$T_1 = t_{MAX}N[\ 1 + r(1 - f)\ ] \tag{2.6.5}$$

The time, T_S, for the SIMD implementation is t_{MAX} if the number of processors is also N. The efficiency for the SIMD case applied to this example, using Eq.(2.6.3), is calculated with P = 2, $n_p = N$ for p = 1 and 2, as,

$$E = f + (1\text{-}f)r \tag{2.6.6}$$

The performance of an application on a SIMD computer is degraded by operations which do not involve all the available processors, as expressed by Eq.(2.6.6). The suitability of an application to a processor array architecture therefore depends on the fraction of the total work that uses all the available processors. One example is the conditional operations found in convective parametrizations; processors only do useful work if the air at that gridpoint is saturated.

In Eq.(2.6.6), f represents the fraction of the total time spent executing fully parallel operations and r represents the fraction of the array actually doing useful work during the remaining time. By plotting curves for constant values of E against f and r, as shown in Fig. 4, it is possible to gauge the effect of operations with idle processors.

Fig. 4 shows that as f tends to unity, when most of the time is spent using all processors, there is a negligible effect of r on the overall efficiency of the program. Even with half the time spent using half the number of processors, the overall efficiency, E, is still 75%.

The degradation in SIMD efficiency arising from the convective

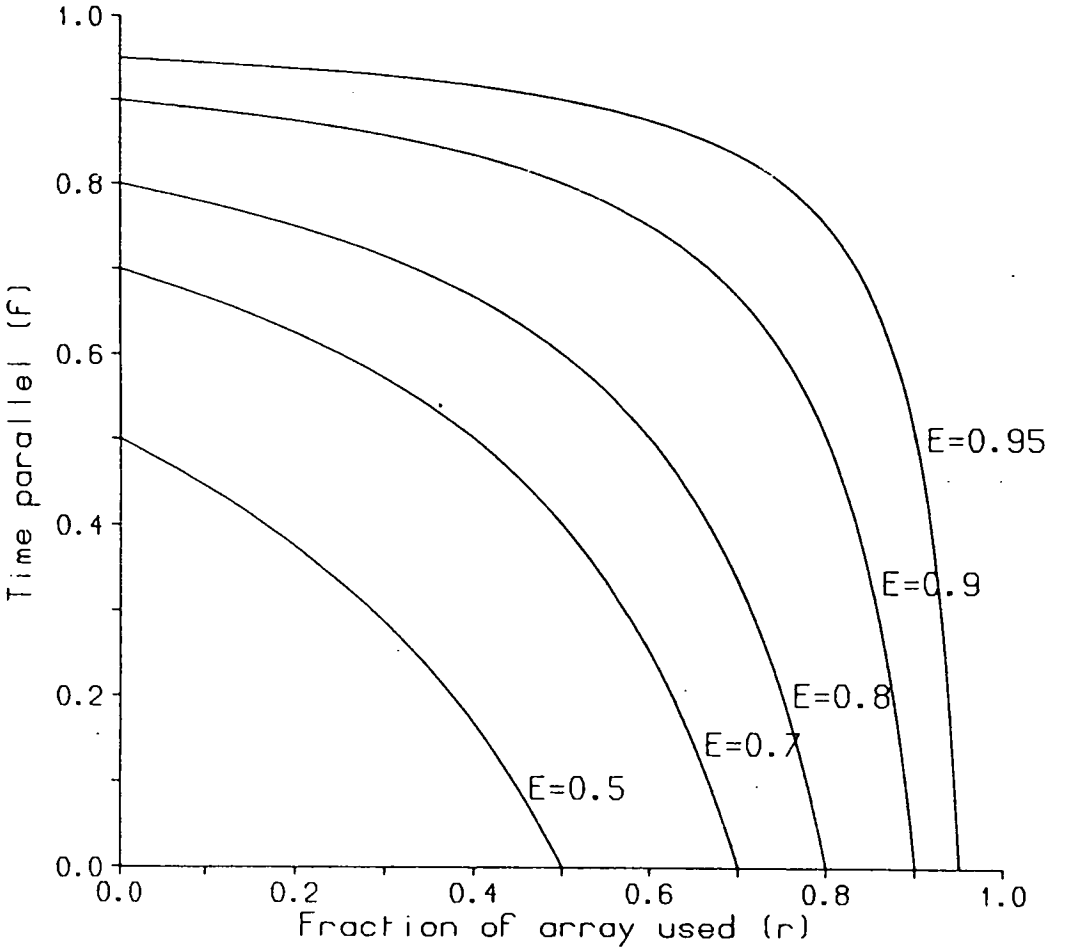


Figure 4. Variation of overall efficiency, E, of a program with a fraction, f, of the work fully parallel and a fraction, r, of the processors doing useful work for the remaining time of the program.

parametrizations can be estimated by suitable choices for f and r. Reddaway *et al.* (1976) used the values f = 0.75 and r = 0.15 for their study of the estimated DAP performance for the Meteorological Office's forecast model. The conditional operations are very inefficient but because they only make up a small part of the total number of operations, the overall efficiency, from Eq.(2.6.6), is still acceptable at 79%. The degradation of performance for models containing conditional operations, involving only a part of the PE array, does not appear to make them unsuitable for implementation on SIMD computers, as might at first be anticipated.

2.6.3. MIMD programming

2.6.3.1. Amdahl's law

Now suppose the above example is implemented on a MIMD computer with p processors. Assuming $p < N$ each processor is assigned several gridpoints. The ideal case is when each processor takes a time T_1/p. However, since portions of work are only available as combinations of the individual components ft_{MAX} and t_{MAX}, in general it will not be possible to find a combination that gives each processor work of a duration T_1/p. Instead the work is shared between processors such that each one computes for a time duration of $(T_1-\alpha)/p$, where α is chosen so that this work is a combination of the basic portions of work available and is constant for each processor. However, one processor must also compute the remaining gridpoints, the time for which is given by α. The time for the MIMD implementation, T_M, the maximum processing time of any processor, is therefore given by,

$$T_M = [(T_1 - \alpha) / p] + \alpha \qquad (2.6.7)$$

If the speedup ratio for the MIMD implementation, S_M, is defined as the ratio of the time for the uniprocessor, serial version to that of the MIMD version, using Eq.(2.6.5) gives,

$$S_M = p / (1 - \beta + \beta p) \qquad (2.6.8)$$

where the expression $\alpha = \beta T_1$ has been substituted and the parameter β represents the fraction of the serial time that is computed sequentially in the MIMD implementation. Eq.(2.6.8) is known as Amdahl's law (Amdahl, 1967) and is represented in Fig. 5 for several values of p. The curves show that

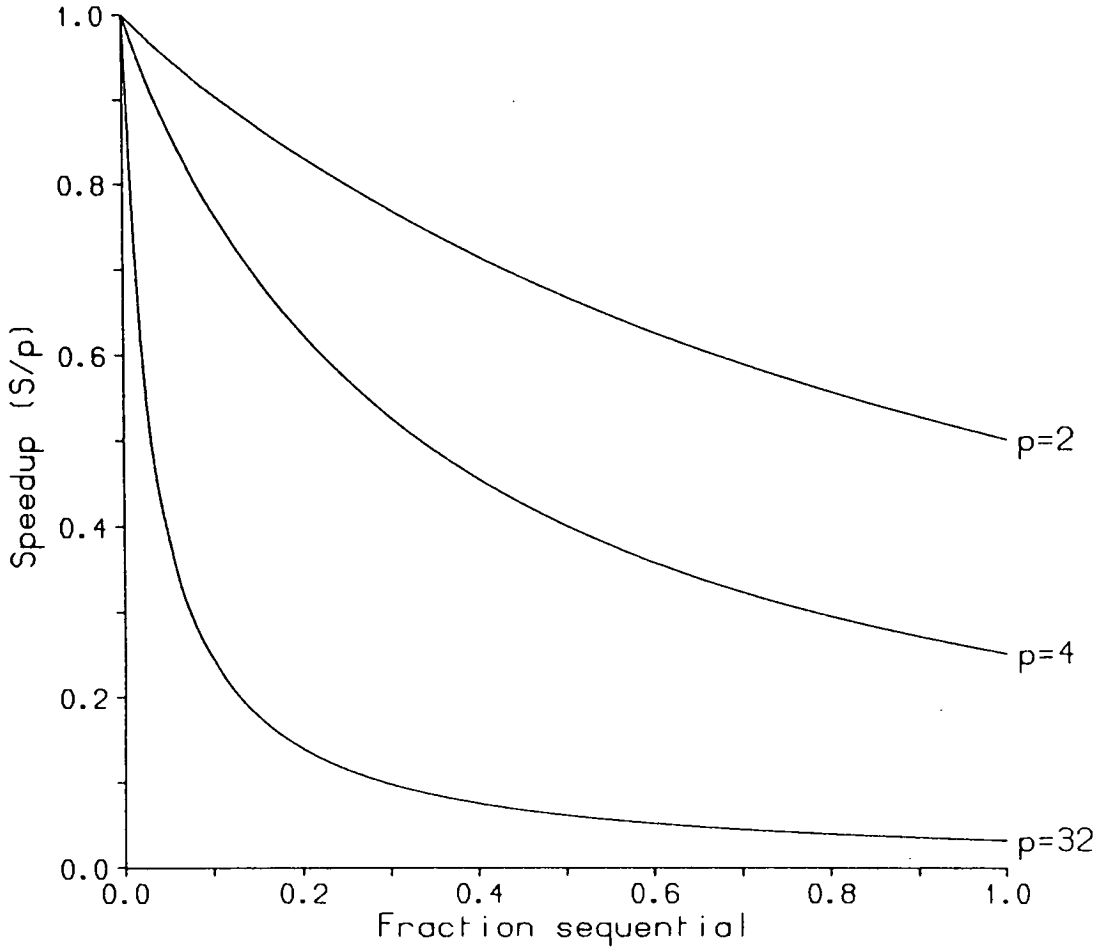


Figure 5. Amdahl's law for several values of p, the number of processors. Speedup is shown as the fraction of the number of processors.

significant speedups are not possible unless significant portions of the program can be multiprocessed. This effect becomes more noticeable with an increasing number of processors. Another feature is that, for a fixed percentage of sequential time, the speedup does not increase as fast as the number of processors. As p approaches infinity, so the speedup S_M in Eq.(2.6.8) converges to β^{-1}. For a large number of processors, the execution (wall-clock) time becomes dominated by the sequential part of the program.

However, considering the example of the previous section, when $p = N$, processing can proceed in the SIMD mode where the speedup behaves as in Fig. 4 and the effect of processing that takes place on just some of the processors, is less marked than for the MIMD case. This would suggest massive parallelism is the best approach for these kind of problems, rather than a computer with a few processors.

2.6.3.2. Granularity and overheads

On a MIMD computer, the grain of an algorithm is the time taken to execute a segment of work between two synchronization points. Granularity is sometimes also measured in terms of floating point operations. The granularity of an algorithm can determine the best architecture or the best multiprocessing method to use (e.g. CRAY macrotasking or microtasking), as the cost of overheads will limit the smallest grain size that can be profitably exploited. The $s_{\frac{1}{2}}$ parameter, described previously, limits the grain size in terms of the overhead from synchronization.

Overheads also arise from the finite time for communication between processors. This is often the main overhead for distributed memory, multiprocessor systems especially when communication cannot be overlapped with computation.

Memory bandwidth and memory latency also contribute to overheads, not just in multiprocessing. The memory bandwidth of a machine can lead to bank conflicts if the processors require more memory references than the memory banks can service. Memory latency is the time taken for memory access. As computer systems are built with larger memories, the latency can become significant. This is especially true of systems where processors are connected to memory via a network of some kind. On vector machines, latency is not

important so long as it is small compared to the length of the vector registers (Hsiung, 1988).

The effects of overheads on the speedup of an application on a parallel computer can be modelled in a similar way to Amdahl's law and a similar curve results. For a given overhead, it is possible to determine the potential speedup for a particular grain size (Larson, 1988).

On a SIMD machine, synchronization and process control overheads do not occur. Furthermore, on the DAP, bank conflicts cannot occur and memory latency is negligible.

2.6.3.3. Scheduling

Two commonly used techniques for scheduling work on MIMD computers are static and dynamic scheduling. Perhaps the most commonly used approach is static scheduling, where work is assigned to a processor before runtime. This can be used when the work in a loop is to be multiprocessed and the CPU time is approximately the same for each loop iteration. A subset of the iterations is assigned explicitly to each process. This simple method has the advantage that the work schedule is under the control of the programmer.

However, if the time duration of each iteration of the loop varies, it is possible for the processor workload to be uneven leading to a loss of performance through Amdahl's law. In these circumstances a dynamic or self scheduling approach is best. This technique maintains a list or a counter to indicate the loop iterations to be done. Each process accesses and updates the list to get its next piece of work. Those processes that have iterations of a short time duration access the list more often to get more work, so that some processes may process more iterations than others. Whilst this method is more flexible and achieves a better overall performance with iterations assigned to processors at runtime, it has several disadvantages. If the time duration of an iteration is data dependent, the work schedule for processes is indeterminate and may result in indeterminate or irreproducible results. Also the work schedule is no longer controlled by the programmer and more complex control logic is required. An example of dynamic scheduling applied to a spectral meteorological model is presented in chapter 8.

The dynamic scheduling technique also incurs an overhead from having to protect the list or counter whilst being updated. This is a critical region as only one process must be allowed access to the list at any one time. If the average granularity of the iterations is large compared to this overhead, the iterations may be allocated one at a time. If the average is small compared to the time for the overhead, the iterations should be allocated in blocks.

2.6.3.4. MIMD bugs

Programming MIMD computers is undoubtedly more complex and requires more development time than for SIMD computers. Furthermore, entirely new types of bugs can arise which do not occur in programming SIMD or SISD computers. Snelling (1988c) and Snelling and Hoffmann (1988) have divided these new problems into five classes. These are the stampede effect, bystander effect, deadlock effect, irreproducibility effect and the Heisenberg effect.

The stampede effect occurs when a process, for whatever reason, encounters an error. The other processes continue to execute for a time and 'stampede' over the evidence, leaving little or no information for debugging. This effect cannot occur on distributed memory computers.

The bystander effect (bystanders are assumed innocent) occurs when one process corrupts another's data. Although the failed process is the obvious place to start debugging, the problem is actually with another process. This type of error can occur in a distributed memory system although it is less likely than in a shared memory system.

Deadlock is perhaps the simplest problem in multiprocessing. It occurs when all processes are awaiting input from another process. Therefore, all the processes have stopped at or near the code that caused the problem, making the problem easy to identify.

The irreproducibility effect is a result of the nondeterministic nature of parallel processing. For example, a program may begin to fail because of a change in the timing of processes relative to one other, perhaps because a critical region of code has not been identified. The problem may be especially hard to track down if the program does not always fail. Another example is when a global quantity is accumulated from contributions from processes.

The value of this quantity can be different on different runs if the processes add their contributions in a different order, because of the finite precision of numbers on a computer. Whilst these differences are small, typically the same magnitude as those arising from using a different compiler, they make the testing of new code virtually impossible.

Lastly, the Heisenberg effect occurs when debugging is in progress. Additional diagnostic code is often used to help in finding bugs. However, it can happen that when this extra code is used it perturbs the execution environment (i.e. alters the timing of processes relative to each other) such that the problem never occurs or manifests itself in a completely different form.

The occurrence of these effects or the degree to which they occur depends on the architecture of the computer and the multiprocessing environment. This is discussed further in Snelling and Hoffmann (1988).

2.6.4. Some algorithms for the DAP

In this section, some parallel algorithms for the DAP are presented. These algorithms are used in the following chapters. Hockney and Jesshope (1981) describe some additional parallel algorithms for the DAP.

2.6.4.1. Cascade-sum

An important example of the tree-height reduction technique is the cascade-sum algorithm for the evaluation of,

$$S = \sum_{i=1}^{N} x_i \qquad (2.6.9)$$

In a serial approach this would require N-1 additions. However, by evaluating some of these in parallel, the sum can be performed in $\log_2 N$ stages, as illustrated in Fig. 6. On the DAP, this would be done by first assigning all N elements to processors. The first stage would involve shifting the data by one PE and adding it to unshifted data to form the sum of adjacent pairs. The next stage uses a shift of two PEs. In general, at the l^{th} stage a shift of 2^l PEs is required (Hockney and Jesshope, 1981).

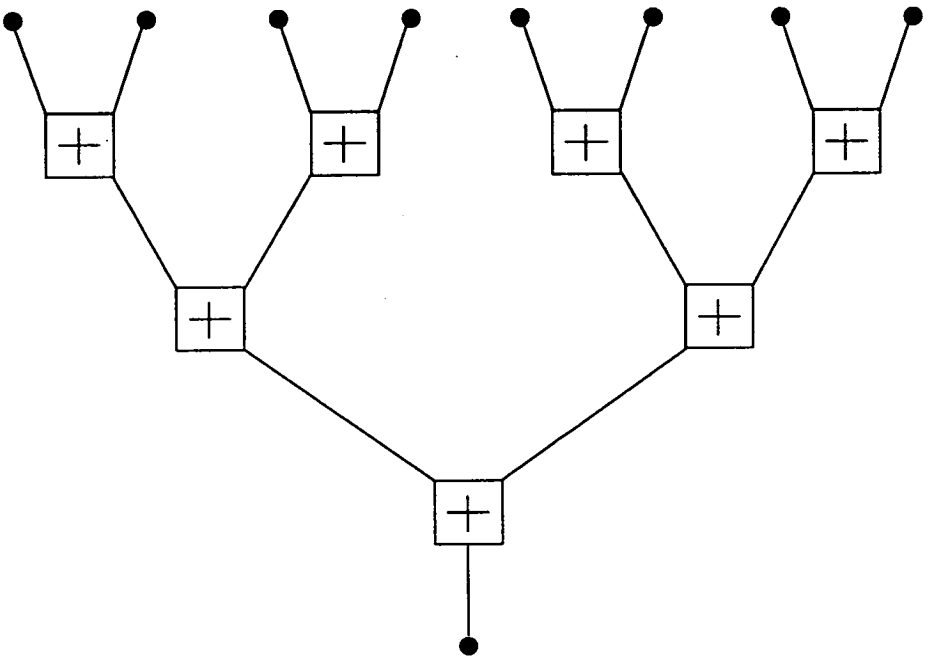


Figure 6. Parallel evaluation of the sum of 8 numbers.

However, it is possible to exploit additional parallelism at the bit level on computers with 1-bit processors such as the DAP. For example, if N processors are available, only half are actually used at the first stage. It is therefore possible to share the addition operations between pairs of processors; one calculating the least significant part, the other the most significant part. The addition would then take half the time. Similarly, at the next stage, the additions can be shared between 4 processors and so on. This technique, combined with block point arithmetic, is used in the DAP FORTRAN intrinsic function SUM which returns the scalar sum of a matrix or vector (Flanders *et al*, 1977). To evaluate the sum of a matrix requires the equivalent of three matrix additions, illustrating the flexibility of the bit serial processors.

2.6.4.2. Fast Fourier Transform

The Fast Fourier Transform (FFT) is a well known algorithm and its implementation on the DAP has already been described by Jesshope (1980) and Hockney and Jesshope (1981) among others. A brief description of the DAP algorithm is given here as FFTs are used in global gridpoint models (for filtering) and spectral models.

A complex Fourier transform on N points is written as,

$$z(k) = \sum_{j=0}^{N-1} x(j) w_N^{jk} \qquad (2.6.10)$$

where k = 0,1....N-1 and

$$w_N = e^{2\pi i/N}$$

It is assumed that N=2^M i.e. a radix-2 transform. The FFT algorithm is derived by writing $x(j)$ as two sequences,

$$x_0(j) = x(2j), \qquad j=0,...(N/2)-1$$

$$x_1(j) = x(2j+1), \qquad j=0,...(N/2)-1$$

which, when substituted into Eq.(2.6.10), give the recurrence relations,

$$z(k) = z_0(k) + w_N^k z_1(k) \qquad (2.6.11)$$

$$z(\tfrac{1}{2}N+k) = z_0(k) - w_N^k z_1(k)$$

where $k = 0,\ldots\tfrac{1}{2}N-1$. Thus, the original transform has been reduced to the calculation of two transforms over half the original length. This process is repeated M times until N 1-point transforms have to be calculated and the transform of a point is the point itself. The FFT algorithm consists therefore of $\log_2 N$ recurrence relations of the form Eq.(2.6.11). The flowchart in Fig. 7 illustrates the algorithm by showing the movement of data and the multipliers, w^k, that are applied at each stage.

If this procedure is applied to the DAP, Fig. 7 also illustrates the routing requirements of the algorithm. In long vector mode, the DAP could be used to calculate a 4096 point transform. In matrix mode, 64 FFTs of 64 points each could be calculated. Although there are a significant number of routing operations required by the algorithm, because arithmetic is relatively slow, the routing only accounts for 10-20% of the overall cost (Flanders *et al*, 1977). Hockney and Jesshope (1981) describe the DAP algorithm in more detail. Essentially, at each stage a complex multiplication and addition are required which have a parallelism of N, so that the available parallelism is constant throughout the procedure. The number of operations required is $O(N\log_2 N)$.

The data are transformed into bit reversed order i.e. the 4th point (0100) is transformed to the 2nd point (0010), as shown in Fig. 7. Depending upon the application, this may need to be ordered correctly by further routing (Flanders, 1982), or additional temporary storage and routing during the transform can be used to leave the data in normal order.

2.6.4.3. Tridiagonal systems

Tridiagonal systems are an important class of linear algebraic equations and occur frequently in meteorological modelling. However, solution techniques based on Gaussian elimination are unsuitable for a processor array as they are sequential algorithms. Instead parallel versions of the cyclic reduction and Jacobi algorithms are preferred.

The implementation of the parallel cyclic reduction algorithm on the DAP is described by Whiteway (1979). Consider 3 consecutive equations of a

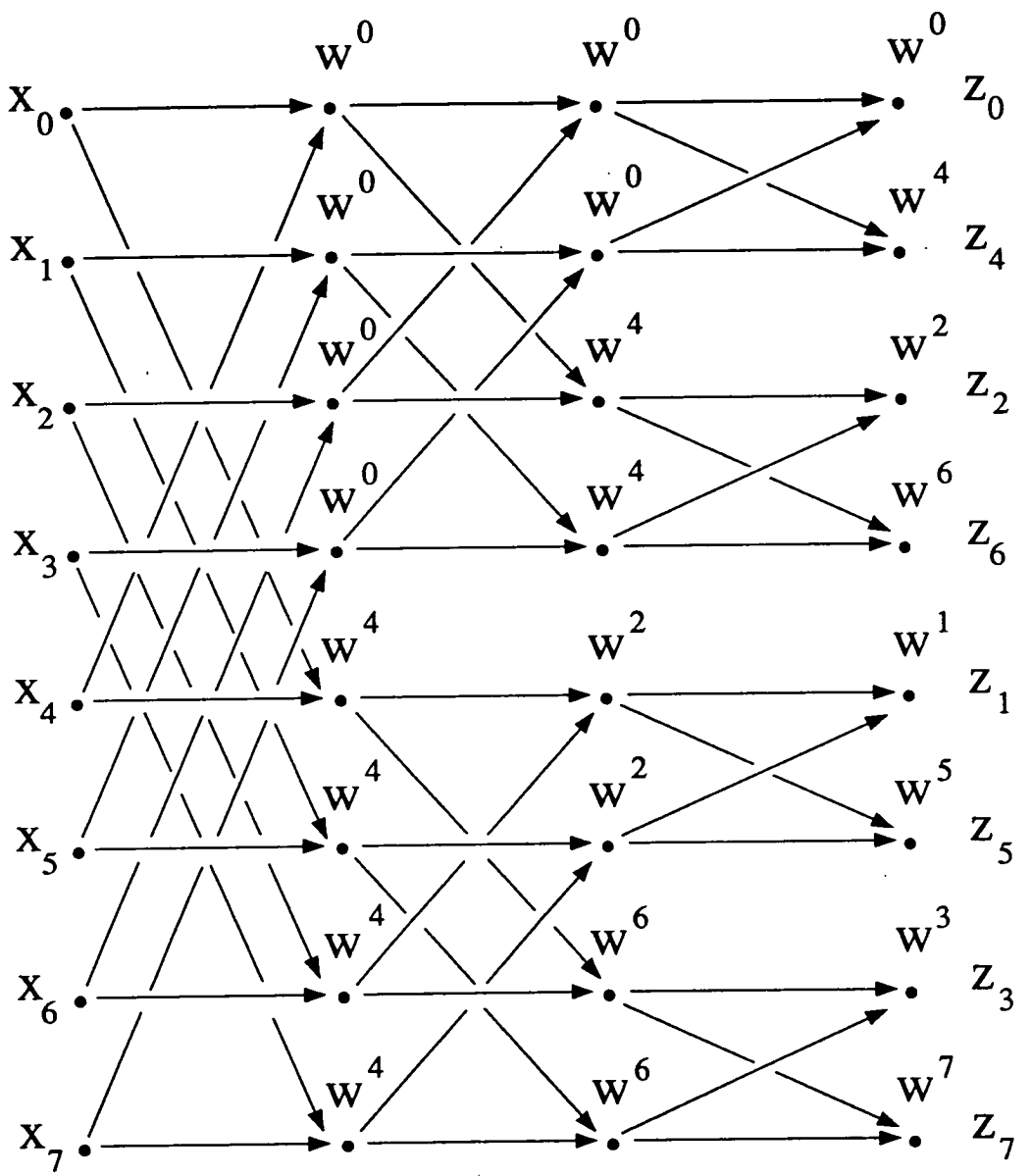


Figure 7. Flowchart illustrating the 4 stages for a FFT algorithm on 8 points. The multipliers, W^k, applied at each stage are also shown.

tridiagonal matrix equation of length N, $\underline{\underline{A}}\underline{x} = \underline{y}$,

$$\text{(a)} \quad a_{i-1}x_{i-2} + x_{i-1} + c_{i-1}x_i = y_{i-1}$$

$$\text{(b)} \quad a_i x_{i-1} + x_i + c_i x_{i+1} = y_i$$

$$\text{(c)} \quad a_{i+1}x_i + x_{i+1} + c_{i+1}x_{i+2} = y_{i+1}$$

The term $a_i x_{i-1}$ can be eliminated from (b) by subtracting a_i times (a). Similarly, $c_i x_{i+1}$ can be eliminated from (b) by subtracting c_i times (c). Doing this and dividing by the main diagonal gives,

$$a_i^{(1)}x_{i-2} + x_i + c_i^{(1)}x_{i+2} = y_i^{(1)}$$

so that the off-diagonals are now twice the distance from the main diagonal. Repeating the process gives the general equation at step j as,

$$a_i^{(j)}x_{i-k} + x_i + c_i^{(j)}x_{i+k} = y_i^{(j)} \tag{2.6.12}$$

where $k = 2^j$. As soon as $i-k$ or $i+k$ are outside the range 1 to N, the term can be considered zero. After $\log_2 N$ steps, all such terms are eliminated and only the main diagonal remains. On the DAP, the algorithm can be applied to all x_i in parallel. Planar boundary conditions are used so that terms shifted in outside the range 1 to N are zero. As for the FFT, the DAP may be used to solve a system with $N = 4096$ or 64 systems each of length 64. Whilst this direct method always completes in $\log_2 N$ steps, it is sometimes possible to halt the procedure without loss of accuracy if the matrix is sufficiently diagonally dominant (Hockney and Jesshope, 1981). This is because the diagonal dominance increases at each successive stage. If at any stage in the reduction, the inverse of the diagonal dominance exceeds the arithmetic precision, the procedure can be stopped.

However, this method is not fully efficient on the DAP, because as the algorithm proceeds fewer PEs are performing useful work. At the last stage the off-diagonals are only half their original length. An alternative solution procedure is the iterative Jacobi method (Golub and Van Loan, 1983). For this method, the tridiagonal matrix, $\underline{\underline{A}}$, is split into strictly lower and upper triangular matrices, $\underline{\underline{L}}$ and $\underline{\underline{U}}$ respectively. The Jacobi iteration step is then,

$$\underline{x}^{(k+1)} = \underline{y} - (\underline{\underline{L}} + \underline{\underline{U}})\underline{x}^{(k)} \qquad (2.6.13)$$

It is possible to show that this iteration converges only if the tridiagonal matrix, $\underline{\underline{A}}$, is diagonally dominant (Bowgen, 1981a). The application of Eq.(2.6.13) to the DAP is straightforward and can be done efficiently.

The convergence rate of the Jacobi method depends on the diagonal dominance of the system. Since the cyclic reduction algorithm increases the diagonal dominance, Bowgen (1981a) developed a hybrid algorithm which uses one pass of this method to improve the convergence rate of the Jacobi scheme. This adds an overhead due to the more expensive step of the cyclic reduction algorithm and because routing in the Jacobi iteration has to increase as the diagonals are further apart. The optimum number of passes of the cyclic reduction algorithm depends on the initial diagonal dominance of the equations.

The cyclic reduction algorithm and the hybrid Jacobi solver (with one pass of the cyclic reduction algorithm) are used in subroutines in the DAP FORTRAN subroutine library (supplied originally by Queen Mary College and now by AMT) for the solution of tridiagonal equations. Several versions are available, including the solution of 64 systems of length 64 each.

2.6.5. Notation

With data arrays mapped into parallel data objects in DAP FORTRAN, it is useful to have a notation to express the way in which the data are mapped. A simple notation used by the author is of the form,

$$(i, j, k) \rightarrow \{ l, m, n \} \qquad (2.6.14)$$

where l, m and n can all be functions of i, j and k. This mapping expression is interpreted to mean the value at the coordinate (i,j,k) is stored at the PE in the l^{th} row, m^{th} column and the n^{th} matrix, assuming that the data are stored as a matrix array.

DAP FORTRAN routing functions can therefore be presented in terms of their translation on the mapping expression Eq.(2.6.14). For example, the TRAN, REVC and REVR functions perform the following translations,

$$\text{TRAN}\{ i, j \} = \{ j, i \}$$

$$\text{REVC}\{ i, j \} = \{ i, 65\text{-}j \} \qquad (2.6.15)$$

$$\text{REVR}\{ i, j \} = \{ 65\text{-}i, j \}$$

The shift functions have analogous expressions.

This notation was found to be useful, not only in representing the way data was stored on the DAP, but also in determining the routing between two mappings. It is similar to the low-level, bit oriented notation developed by Flanders (1982) for expressing data mappings and determining routing operations in the DAP assembly language, APAL.

3.1. Introduction

Before discussing the implementation of meteorological models on parallel computers, particularly the ICL DAP, it is necessary to have an understanding of the techniques that are used to solve the governing equations. In this chapter, the methods used to approximate temporal and spatial terms are reviewed.

Time differencing methods are reviewed first. The finite difference method is then reviewed to illustrate the potential areas of difficulty in implementing a model on a parallel computer and also to enable a complete understanding of the contents of chapter 4. Then, the spectral and finite element methods of the Galerkin approach are reviewed, as they are used in the models developed and described in this thesis.

3.2. Time differencing

Many methods of integrating the time dependent equations that occur in meteorology have been proposed. These methods have been reviewed by Haltiner and Williams (1980) and Mesinger and Arakawa (1976) and are summarized below. The choice of time scheme is not influenced by the computer architecture in use, since the parallelism is associated with the spatial dimensions. However, storage requirements may be a consideration.

3.2.1. Explicit schemes

The most commonly used explicit scheme is the leapfrog scheme defined by,

$$dF/dt \simeq (F(t+\Delta t) - F(t-\Delta t)) / 2\Delta t \qquad (3.2.1)$$

It is a neutral scheme and simple to use. However, it has several disadvantages, in common with all other three time level schemes. First, a computational mode is present in the solution, which tends to amplify for nonlinear problems resulting in a separation of the solution on odd and even time levels. A time filter or the occasional use of a scheme with good

damping properties, such as the Euler-backward or Matsuno scheme (Matsuno, 1966), can be used to control this mode. Second, two initial fields are required to start the integration. Last, it is generally true that models using three time levels require more memory or I/O than those requiring two. Also, the leapfrog scheme is unstable when applied to friction terms. Another method, usually the forward scheme in gridpoint models or an implicit method in spectral models, has to be used.

Another explicit scheme that has received much attention is the Lax-Wendroff scheme (Haltiner and Williams, 1980). This is a two-step scheme that has good selective damping properties. It uses two time levels, avoiding the disadvantages of the leapfrog scheme. Gadd (1978a, 1980) added a simple modification to significantly reduce the phase speed errors. The resulting scheme has fourth order accurate phase speeds but a second order accurate truncation error. Carpenter (1981) and Collins (1983) developed variants of the scheme with third and fourth order accurate truncation error, respectively.

It is well known that the maximum allowable timestep for an explicit time scheme is restricted by the Courant-Friedrichs-Lewy (CFL) condition. In primitive equation models, the timestep is restricted by the speed of the gravity wave modes. Several techniques have been used to overcome this restriction so that the timestep is limited only by the more significant but slower moving meteorological waves. One method involves treating those terms responsible for the generation of gravity waves implicitly, the so called semi-implicit method discussed in the next section. Another efficient method is known as the splitting technique, pioneered by Marchuk (1974). In this approach the horizontal advection terms are integrated separately from the gravity wave terms, using a timestep limited only by the windspeed. The gravity wave terms are integrated using a smaller timestep. Different time integration schemes can be used for these advective and adjustment stages.

This split explicit scheme is used operationally by the U.K. Meteorological Office (Gadd, 1978b, 1980). A forward-backward scheme (Mesinger and Arakawa, 1976) is used for the adjustment stage whilst the modified Lax-Wendroff scheme of Gadd (1978a) is used for the advective terms. One advantage of the scheme is that the memory requirements of the model are reduced, as only one time level of data needs to be stored (Cullen, 1983).

3.2.2. Implicit schemes

Implicit time integration methods, such as the trapezoidal (or Crank-Nicolson) scheme, are usually unconditionally stable and the timestep is chosen on the basis of accuracy requirements. The principal difficulty is that the solution of a large system of simultaneous equations is required at each step. In general, nonlinear equations lead to complicated systems that can only be solved by iterative methods. Efficient techniques have been developed to solve these systems. The splitting method of Marchuk (1974) or the alternating-direction-implicit method (ADI) (Gustafsson, 1971) have been successfully used by reducing the dimensionality of the problem and solving sets of equations in only one spatial dimension.

3.2.2.1. Semi-implicit scheme

Perhaps the most commonly used method for extending the CFL limit is the semi-implicit scheme, first used by Robert (1969) in a spectral model. In this method, only linear terms generating gravity waves are treated implicitly. A Helmholtz or Poisson equation has to be solved each timestep, but this added computational cost is offset by a timestep up to six times greater than that for a fully explicit scheme.

In spectral models on a sphere, the semi-implicit scheme is particularly efficient since, as shown later, the basis functions are eigenfunctions of the Poisson operator. Daley (1980) estimated the additional overhead for a spectral model using a semi-implicit scheme to be 3%. For a gridpoint model this figure might be as large as 50% (Robert *et al*, 1972). The semi-implicit scheme has also been successfully implemented in a finite element model by Staniforth and Mitchell (1977), although a Helmholtz equation still has to be solved.

Despite its obvious advantages, the semi-implicit method has several drawbacks. First, the formulation of the scheme in baroclinic models is not entirely straightforward. Second, Simmons *et al.* (1978) have shown that under certain conditions, the scheme is unstable whatever the choice of timestep. Last, the scheme achieves its stability by slowing down the fastest gravity wave modes, resulting in a poorer representation of the geostrophic adjustment process. This also allows the possibility of interactions with the Rossby waves although there is no documented evidence of this so far.

3.2.3. Semi-Lagrangian scheme

A method that has recently been the subject of much research is the semi-Lagrangian scheme. It was first used by Sawyer (1963) based on a trajectory method proposed by Wiin-Nielson (1959). The scheme achieves long timesteps by treating the advective terms in a Lagrangian sense. That is, at each time level, the properties of each gridpoint are determined by calculating a back trajectory of the fluid parcel at that point over the last time interval.

Robert (1981, 1982) combined a semi-Lagrangian treatment of advection with the semi-implicit scheme in a finite difference shallow-water model. Robert *et al.* (1985) extended the approach to a multi-level model. They found that the timestep could be further increased by a factor of six over that for a semi-implicit scheme alone. Bates and McDonald (1982) and Bates (1984) combined the scheme with the split explicit and ADI integration techniques. Staniforth and Temperton (1986) applied the semi-Lagrangian semi-implicit approach to the variable resolution finite element model of Staniforth and Mitchell (1978). They found that the combined scheme could again be used successfully with timesteps greater by a factor of six.

Although the semi-Lagrangian semi-implicit scheme has been implemented successfully in baroclinic finite difference and barotropic finite element models, only recently has the method been implemented in a spectral model. Ritchie (1987) made a preliminary study of advection on a Gaussian grid using a semi-Lagrangian scheme before applying the method to a spectral shallow-water model on the sphere (Ritchie, 1988). It was found necessary to reformulate the spectral model to use wind components as the prognostic variables, rather than the more usual vorticity and divergence formulation, to be consistent with the semi-Lagrangian approach. Ritchie (1988) found that, like finite difference and finite element implementations, the semi-Lagrangian semi-implicit scheme gave acceptable results using a timestep greater by a factor of six than that for an Eulerian semi-implicit version.

Another advantage of the semi-Lagrangian approach is that the scale dependent phase speed errors are less than in an Eulerian approach. This led Ritchie (1985) to use the method just for the moisture equation in the baroclinic model of Staniforth and Daley (1979). One technical difficulty with

the scheme however, is that field values on neighbouring latitudes must be available. Spectral models are usually designed to process one latitude at a time to reduce core storage (Baede *et al*, 1979).

3.2.4. Time filtering

The computational mode associated with three level time integration schemes is undesirable. Although frequent use of another scheme with selective damping properties is one way of reducing the effect of this mode, the preferred and more flexible approach is to use a time filter. The Asselin (1972) filter, originally developed by Robert (1966), is generally used and takes the form,

$$\bar{F}(t) = F(t) + \nu[\ \overline{F(t-\Delta t)} - 2F(t) + F(t+\Delta t)\ ] \qquad (3.2.2)$$

The filter parameter, ν, must be less than 0.5 and an overbar represents a filtered value. Asselin (1972) found that the filter is a strong damper of the computational mode and of two grid length noise in explicit, semi-implicit and implicit time schemes. Since the filter is linear, it can be applied in spectral, Fourier and gridpoint space.

Schlesinger *et al.* (1983), Sakellarides (1984) and Deque and Cariolle (1986) have made more detailed studies of the effects of the Asselin filter. In particular the last two references show the filter can destabilize a numerical solution when damping terms are present in the equations and a timestep close to the CFL limit is selected.

3.3. Finite difference methods

3.3.1. Choice of grid for modelling on the sphere

A major aspect in the design of a model to forecast for all or part of the globe is the choice of grid. There has been much literature in the past on this subject and the main approaches are reviewed here. A good review is given by Williamson (1979).

3.3.1.1. Map projections

Map projections represent the surface of the sphere on a plane in some way. However, such projections cannot retain all the properties of the sphere.

A conformal projection preserves the angle between intersecting lines of latitude and longitude. Commonly used conformal mappings are the polar stereographic and Mercator projections. It is not possible to define a conformal mapping that represents the entire globe on a finite plane and so conformal mappings are often used for limited area modelling (e.g. Staniforth and Mitchell, 1977). However, by overlapping projections it is possible to cover the entire globe (e.g. Phillips, 1959; Stoker and Isaacson, 1975).

One type of nonconformal projection is the equal area grid, where the area in each grid square is approximately constant. Kurihara (1965) developed an equal area grid in which the number of gridpoints decreased as the poles were approached and the latitudes were at equally spaced intervals. However, the gridpoints no longer lined up in the north-south direction and the finite difference schemes were derived by considering fluxes across grid boxes (Bryan, 1966). Holloway and Manabe (1971), Dey (1969), Sankar-Rao and Umscheid (1969) and Grimmer and Shaw (1967) have all tested the Kurihara grid and found there is insufficient resolution in polar regions, resulting in serious errors.

Sadourney (1975a) used a cylindrical equal area grid where every latitude has the same number of points. The latitudinal spacing was varied to achieve the equal area. He tested the grid using Rossby-Haurwitz waves as used by Phillips (1959) and achieved reasonably accurate results. However, the poleward increase in spacing between latitudes allows the possibility of the refraction of poleward propagating waves.

3.3.1.2. Spherical geodesic grids

The spherical geodesic grid was developed (Williamson, 1968; Sadourney *et al.*, 1968) to provide a homogeneous spherical grid, where not only are the grid areas equal but they are also of the same shape. The finest resolution grid with these properties is an icosahedron constructed within a sphere, consisting of 20 equilateral triangular faces with 12 vertices connected by great circles. Each vertex therefore has 5 nearest neighbours. This grid,

however, does not provide sufficient resolution and so the two conditions above must be relaxed slightly. Each of the major triangles is subdivided into smaller triangles. Each vertex of these smaller triangles has 6 nearest neighbours.

Williamson (1970) found that the slight irregularity of the grid gave only first order accuracy. A $2\frac{1}{2}°$ degree resolution grid was required so that truncation errors did not dominate in divergence calculations. Williamson (1971) developed nonconserving second order schemes to overcome this problem. Although this type of grid has not been used much by meteorologists, it appears to be a natural choice for a finite element model based on triangular elements (Cullen, 1974b).

The spherical geodesic grid has several computational disadvantages. The indexing of the gridpoints is complex and more storage space is required than for an equivalent latitude–longitude grid. The complex indexing together with the number of nearest neighbours being either 5 or 6 would present difficulties in implementing this type of grid on a parallel computer based on an array of processors connected in a regular mesh.

3.3.1.3. Latitude–longitude grids

The easiest grid to use for equations written in polar spherical coordinates is one in which the gridpoints are at regularly spaced intervals of latitude and longitude. A difficulty, however, is that the wind components are undefined at the poles. Modifications to the differencing scheme are therefore necessary at or near the poles. From a programming point of view, it is easiest not to use pole gridpoints but to place the nearest latitude half the latitudinal spacing from the poles. No special section of code is then needed and finite differences can be applied by using the gridpoints opposite, across the pole.

3.3.1.4. The pole problem

If the spacing between gridpoints decreases as the poles are approached, the CFL limit requires a small and impractical timestep for stability. This is a problem for the latitude–longitude grid and it has been overcome in several ways.

Grimmer and Shaw (1967) and Corby *et al.* (1972) allowed the timestep to

decrease as the poles were approached in the integration. Although the results were satisfactory and the scheme stable, Grimmer and Shaw (1967) pointed out that over half of the computing time was spent on integrating the northernmost rows, about 2% of the hemisphere.

Another approach is to use a coarser grid near the pole. The Kurihara grid is one example. Washington and Kasahara (1970) used a similar grid but with more points around the poles than the Kurihara grid, to get better resolution in the polar regions. A slightly different approach was adopted by Gates and Riegel (1962) who integrated at each point on a regular latitude-longitude grid, but as the poles were approached the longitudinal differences were taken over larger intervals which were multiples of the basic longitudinal spacing. Their results were not as accurate as those obtained on a Kurihara type grid.

One method of ensuring stability on a latitude-longitude grid is to apply some form of filtering in the polar regions to remove the short wavelengths. Umscheid and Sankar-Rao (1971) tested the smoothing algorithm used in the Mintz-Arakawa general circulation model (Gates *et al*, 1971). They found that although it was a strong damper of short waves it also damped the longer waves. They also tested a longitudinal Fourier filter, where the amplitudes of the waves above a certain cutoff frequency were set to zero. This frequency was a function of latitude and the filter was only applied to polar latitudes. The results showed this technique could be used successfully, without loss of accuracy in the large scale flow. Further studies by Holloway *et al.* (1973), Williamson and Browning (1973) and Umscheid and Bannon (1977) confirmed this. In the ECMWF gridpoint forecast model, Burridge and Haseler (1977) reduced the amplitude of the short waves rather than eliminating them entirely. Temperton (1977) showed that this reduced the effect on the unfiltered model modes.

3.3.2. Finite difference approximations of spatial derivatives

Centred differencing is commonly used in finite difference models to approximate spatial derivatives. For example, the approximation,

$$\frac{\partial u}{\partial x} \simeq (u_{i+1} - u_{i-1}) / 2\Delta x \qquad (3.3.1)$$

is straightforward and second order accurate. One sided differences are

sometimes used and are known as upstream or downstream differences depending on the direction of the advecting flow field (Mesinger and Arakawa, 1976). Unlike the centred scheme, they are only first order accurate and have a wavelength dependent damping. They have occasionally been used in meteorology since they use less gridpoints and therefore have a smaller domain of influence (e.g. Miller and Thorpe, 1981).

3.3.2.1. Higher order schemes

Although computationally efficient to use, centred differences have poor phase properties (Mesinger and Arakawa, 1976). Both the accuracy and the phase behaviour can be improved by using more points. An explicit fourth order difference approximation to the first derivative would be,

$$\frac{\partial u}{\partial x} \simeq \frac{4(u_{i+1} - u_{i-1})}{6\Delta x} - \frac{(u_{i+2} - u_{i-2})}{12\Delta x} \tag{3.3.2}$$

This scheme requires more computation and cannot be applied at the boundaries of the solution domain. Another problem arises when Eq.(3.3.2) is used with an implicit time differencing scheme, such that a Helmholtz type equation has to be solved. The coefficient matrix that has to be inverted is then pentadiagonal and therefore computationally more expensive to solve.

It is possible to derive fourth order accurate schemes without using more points than for the second order scheme Eq.(3.3.1). The price is to make these schemes, known as compact differencing schemes, implicit (Navon and Riphagen, 1979; Haltiner and Williams, 1980; Chang and Shirer, 1985). The method would approximate the first derivative by,

$$aw_{i-1} + bw_i + cw_{i+1} = du_{i-1} + eu_i + fu_{i+1} \tag{3.3.3}$$

where,

$$(\frac{\partial u}{\partial x})_i = w_i \tag{3.3.4}$$

The values of the coefficients in Eq.(3.3.3) are determined by requiring the scheme to be accurate up to and including fourth order expressions for u. The resulting scheme for this example is the same as the finite element approximation with regular spacing. However, the approximations obtained from compact differencing and finite elements are usually totally different.

Chang and Shirer (1985) compared finite element and compact difference schemes for geostrophic adjustment and vorticity advection problems. They found the finite element method to be superior for the adjustment problem. Although compact differences gave the best results in the advection problem they noted that the finite element results could have been improved.

3.3.3. Staggered grids

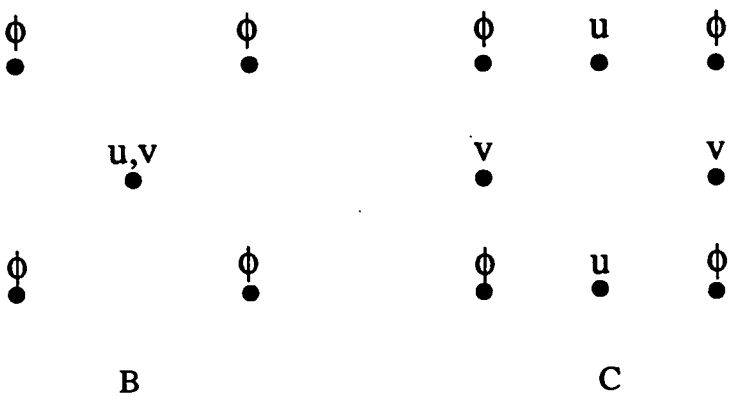
When solving the equations used for meteorological modelling, it is uneconomical and unnecessary to store all the variables at every gridpoint. Savings in storage space can be achieved if a staggered grid is used, in which the variables are held at alternate points. Staggered schemes can also reduce the problem of solution separation that occurs with leapfrog time differencing on an unstaggered grid. Also, noise can be generated in the pressure field if an unstaggered grid is used.

Arakawa and Lamb (1977) discussed five different arrangements of the dependent variables using linearized shallow-water equations. They found that the simulation of the geostrophic adjustment process is highly dependent on the form of the staggering used. The best scheme has since become known as the Arakawa 'C' grid, although the 'B' grid analysed by Arakawa and Lamb (1977) can also exhibit similar properties. Both grids are shown in Fig. 8. Haltiner and Williams (1980) and Mesinger and Arakawa (1976) both review the Eliassen grid (also shown in Fig. 8) which was proposed for a baroclinic primitive equation model. This scheme is staggered in both space and time and has excellent geostrophic adjustment properties as the arrangement of variables at each timestep is optimal for the principal terms.

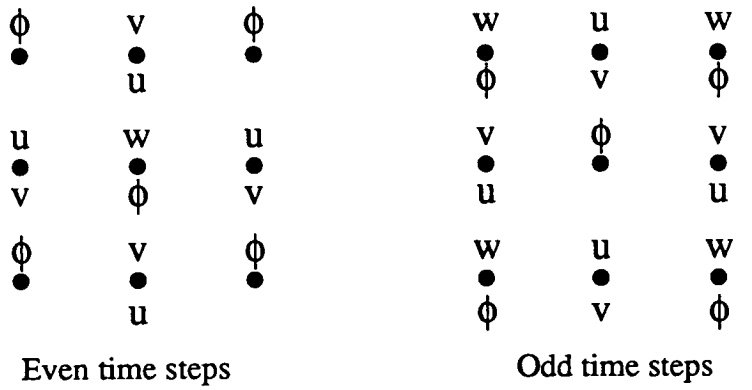
3.4. Galerkin techniques: Spectral method

3.4.1. Introduction

The use of spectral methods in atmospheric models was first proposed by Blinova (1942) in the USSR, who advocated using a linearized model based on spherical harmonics for long-range forecasting. In 1954, Silberman's use of the method was the first of several theoretical studies. The method was found to have useful properties but was costly in storage and computation, as the interaction coefficient method was used to compute the nonlinear terms. It



Arakawa grids



Eliassen grids

Figure 8.

was concluded that only low-resolution spectral models were feasible.

Orszag (1970) and Eliasen *et al.* (1970) independently developed the transform method, for integrating the nonlinear terms, removing the storage problem and substantially reducing the computation required. Higher resolution models became feasible and subsequent research has been devoted to developing multi-level spectral models, particularly for forecasting. At current resolutions, spectral models appear to be superior to gridpoint models for the same computing resources (Jarraud and Girard, 1983).

This section gives a general overview of the spectral method. More detailed descriptions are given by Bourke *et al.* (1977), Machenhauer (1979) and Jarraud and Simmons (1983).

3.4.2. General method

The basis of the spectral method is to expand the variables in the equations as a series. To do this, it is convenient to regard the variables as functions in a complex, infinitely-dimensional function space or 'Hilbert space', H (Courant and Hilbert, 1953). This space can be defined by a complete set of linearly-independent basis functions defined on a fundamental domain, so that any function in H may be expanded as a linear combination of these basis functions (Mandl, 1957). The set of basis functions can be regarded as forming a coordinate system spanning the Hilbert space H.

The equations used in numerical weather prediction can be written generally as,

$$\frac{\partial F}{\partial t} = A(F) \tag{3.4.1}$$

where F is a function of space and time and represents any prognostic variable and A represents the corresponding spatial operator, generally a combination of linear and nonlinear terms.

In expanding F, it is assumed that a suitable set of orthonormal basis functions, ψ_n, are known. For example, on a sphere, spherical harmonics form such a set, whereas on a plane, double Fourier series would be appropriate. It will also be assumed that the boundary conditions of the domain can be satisfied by a proper choice of any finite dimensional set of basis functions. F

may therefore be written as,

$$F = \sum_{n=1}^{\infty} c_n \psi_n \tag{3.4.2}$$

where the c_n, for $n = 1$ to ∞, are functions of time only and the ψ_n are functions of space only. As the basis functions are orthonormal, the coefficients can be determined by,

$$c_n = (F,\psi_n) = \int_\tau F \psi_n^* \, d\tau \qquad \text{for} \quad n = 1 \text{ to } \infty \tag{3.4.3}$$

By taking the scalar product of both sides of Eq.(3.4.1) with the basis functions, the evolution equations for the time dependent coefficients can be obtained by,

$$dc_n/dt = (A(F),\psi_n) = \int_\tau A(F)\psi_n^* \, d\tau \qquad \text{for} \quad n=1 \text{ to } \infty \tag{3.4.4}$$

It is computationally impossible to deal with an infinite number of basis functions. Instead F must be approximated by a linear combination of basis functions that span a finite dimensional subspace of H, denoted by H'. So, F is approximated by,

$$F' = \sum_{n=1}^{N} d_n \psi_n, \qquad F' \simeq F \tag{3.4.5}$$

The best approximation to F in a least squares sense, is the projection of F onto the finite dimensional space H' (Machenhauer, 1979). Hence d_n is set equal to c_n for n=1 to N.

Eq.(3.4.1) must now be solved using F' rather than F. However, since a truncated representation of F is used, F' will not in general satisfy Eq.(3.4.1) e.g. if nonlinear terms are present in the operator A. A residual R, which is a function of F', can be defined by,

$$R(F') = \frac{\partial F'}{\partial t} - A(F') \tag{3.4.6}$$

To ensure that F' satisfies the equation Eq.(3.4.1) as accurately as possible, it is desirable to minimize the residual $R(F')$ averaged over the fundamental domain, by some criteria. The norm $N(R)$ of the residual,

$$N(R) = (R(F'),R(F')) = (\partial F'/\partial t - A(F'),\partial F'/\partial t - A(F')) \tag{3.4.7}$$

can be minimized. The function obtained by A acting on F', $A(F')$, may be expanded as a linear combination of the basis functions,

$$A(F') = \sum_{n=1}^{\infty} \alpha_n(c_1,c_2,\cdots c_N)\psi_n \tag{3.4.8}$$

where the coefficients α_n are functions of c_i for $i = 1$ to N. This resulting function can be separated into two parts,

$$A(F') = A_1(F') + A_2(F') \tag{3.4.9}$$

$$A_1 = \sum_{n=N+1}^{\infty} \alpha_n\psi_n, \qquad A_2 = \sum_{n=1}^{N} \alpha_n\psi_n$$

Substituting this into Eq.(3.4.7), expanding the product and using Eq.(3.4.5) and Eq.(3.4.9) gives,

$$(R,R) = \sum_{n=1}^{N} |\frac{dc_n}{dt} - \alpha_n(c_1,c_2,\cdots c_N)|^2 + \sum_{n=N+1}^{\infty} |\alpha_n|^2 \tag{3.4.10}$$

The first term of the above equation involves coefficients of the projection of $A(F')$ onto H', whilst the second, which may arise from nonlinear interactions, involves the projection of $A(F')$ onto the complement of H'. If the second term is ignored (assuming it is small), as it would otherwise result in a complicated system of equations, Eq.(3.4.10) is minimized if,

$$\frac{dc_n}{dt} = \alpha_n(c_1, c_2, \ldots c_N) \qquad \text{for } n=1 \text{ to } N \tag{3.4.11}$$

That is, Eq.(3.4.11) gives N first order ordinary differential equations which can be solved for $c_n(t)$, for n = 1 to N, subject to certain boundary conditions. If the set of coefficients $\{c_n\}$ are known at a particular time, standard differencing techniques can be used for the time derivative in Eq.(3.4.11) to calculate the $\{c_n\}$ at a later time.

The coefficients α_n are given by,

$$\alpha_n = (A(F'), \psi_n) \qquad \text{for } n=1 \text{ to } N \tag{3.4.12}$$

So, Eq.(3.4.11) becomes,

$$dc_n/dt = \int_\tau A(F')\psi_n^* \, d\tau \qquad \text{for } n=1 \text{ to } N \tag{3.4.13}$$

Comparison with Eq.(3.4.4) shows that Eq.(3.4.13) may be obtained by taking the scalar product of the truncated equation with the basis functions ψ_n, for n = 1 to N. If the residual is expanded as a linear combination of basis functions, from Eq.(3.4.11) it is easy to see that the projection of $R(F')$ onto H' vanishes and $R(F')$ only exists in the complement of H'.

The initial conditions, F_0, will not, in general, be represented exactly using the truncated set of basis functions and so a best fit using a minimization of the norm $N(F_0' - F_0)$ is usually made i.e.

$$F_0' = \sum_{n=1}^{N} c_n^0 \psi_n, \qquad F_0' \simeq F_0 \tag{3.4.14}$$

where,

$$c_n^0 = (F_0, \psi_n) \qquad \text{for } n = 1 \text{ to } N \tag{3.4.15}$$

3.4.3. Some properties of the spectral method

Since interactions between the components spanning H' and components spanning the complement of H' are excluded, in theory the approximate solution F' is calculated with no aliasing errors. In practice however, in meteorological models, the nonlinear terms in A of Eq.(3.4.1) are calculated by transform methods using a grid with sufficient resolution to ensure alias free calculation of the quadratic terms only (Hoskins and Simmons, 1975). However, the aliasing caused by triple product terms is generally negligible, although some effects can sometimes be observed (Jarraud and Simmons, 1983). The lack of aliasing errors, means that nonlinear instability, as described by Phillips (1959), cannot occur. In addition, no phase errors will be introduced by the linear terms of A since these are represented exactly. A cause of computational dispersion is thereby eliminated (Machenhauer, 1979).

The neglect of interactions between members of H' and members of the complement of H' is an important source of error in spectral models. Although the projection of the residual $R(F')$ on H' vanishes (i.e. the error in satisfying the equation is orthogonal to the basis functions that span H'), neglect of interactions involving components outside H' causes errors in the components of the basis functions spanning H' (Haltiner and Williams, 1980).

Since $R(F')$ is orthogonal to the basis functions that span H', it will also be orthogonal to any function in H'. In particular,

$$(R,F') = \int_{\tau} R(F')F'^{*}\, d\tau = 0 \qquad \text{for} \quad n=1 \text{ to } N \tag{3.4.16}$$

which is important as it expresses the conservation properties of the technique. It can be shown (Machenhauer, 1979) that the spectral method conserves the first and second moments of F', subject to time differencing errors. For example, in a model with vorticity as a prognostic variable, vorticity and squared vorticity would both be conserved. For shallow-water and primitive equation models however, energy is not exactly conserved as it is a triple product. However, in practice, as the nonlinear terms are calculated accurately with the spectral method, energy is nearly conserved (Bourke, 1972).

3.4.4. Basis functions

The choice of basis functions depends on the geometry of the problem to be solved. For global or hemispheric problems, spherical harmonics are the natural choice. An important property of spherical harmonics is that they are eigenfunctions of some of the operators that typically occur in the operator A of Eq.(3.4.1), simplifying the equation. No pole problem occurs as the harmonics satisfy the boundary conditions; if continuous variables are used no discontinuities at the poles are encountered. Hence the use of vorticity and divergence as prognostic variables. Spherical harmonics are also useful as they converge rapidly for sufficiently smooth functions. Therefore spectral models need fewer degrees of freedom compared to equivalent grid point models, to achieve comparable results.

Spherical harmonics are defined by,

$$Y_{m,n}(\lambda,\mu) = P_{m,n}(\mu)e^{im\lambda} \tag{3.4.17}$$

where λ is the longitude, μ is $\sin\theta$ for θ as latitude and the $P_{m,n}$ are the associated Legendre functions of the first kind of order m and degree n. Here, m is the zonal wavenumber and n is known as the total wavenumber or the two dimensional index (Jarraud and Simmons, 1983). Also, $n-|m|$ is the number of zeros between the poles and $n-|m|+1$ can be considered as a pseudo-latitudinal wavenumber (Doron *et al*, 1974).

There are many definitions of the $P_{m,n}$ because of the freedom of choice of the normalization constant. Following Jarraud and Simmons (1983), a suitable definition is,

$$P_{m,n}(\mu) = \left[\frac{(2n+1)(n-|m|)!}{(n+|m|)!} \right]^{1/2} \frac{(1-\mu^2)^{|m|/2}}{2^n n!} \frac{d^{n+|m|}}{d\mu^{n+|m|}}(\mu^2 - 1)^n \tag{3.4.18}$$

which gives,

$$P_{-m,n} = P_{m,n} \tag{3.4.19}$$

The spherical harmonics are eigenfunctions of the two-dimensional Laplacian on a sphere,

$$\nabla^2 Y_{m,n} = \frac{-n(n+1)}{a^2} Y_{m,n} \tag{3.4.20}$$

where a is the radius of the sphere. This holds only when a is a constant, which is not generally true for meteorological models. For example, in a sigma coordinate model a will not be constant along a coordinate surface. However, this approximation is a good one (except possibly in mountainous regions) and consistent with others used in the derivation of the primitive equations. Spherical harmonics are also eigenfunctions of the zonal derivative,

$$\frac{\partial Y_{m,n}}{\partial \lambda} = im Y_{m,n} \tag{3.4.21}$$

Following the general method outlined in the previous section, the scalar product on the sphere is defined by,

$$(f,g) = \frac{1}{4\pi} \int_{-1}^{1} \int_{0}^{2\pi} f g^* \, d\lambda \, d\mu \tag{3.4.22}$$

The constant outside the integral may be chosen freely and only changes the length or norm of the function. Eq.(3.4.22) implies that,

$$(Y_{m,n}, Y_{m',n'}) = \frac{1}{4\pi} \int_{-1}^{1} \int_{0}^{2\pi} Y_{m,n} Y^*_{m',n'} \, d\lambda \, d\mu = \delta_{mm'} \delta_{nn'} \tag{3.4.23}$$

Thus the infinite set $\{Y_{m,n};\ m=-\infty,\infty,\ n=-\infty,\infty\}$ form an orthonormal set of complex valued basis functions defined on the sphere. Any function, F, may therefore be expanded as,

$$F(\lambda,\mu,t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} F_{m,n}(t) Y_{m,n}(\lambda,\mu) \tag{3.4.24}$$

with the coefficients $F_{m,n}$ given by,

$$F_{m,n}(t) = (F, Y_{m,n}) = \frac{1}{4\pi} \int_{-1}^{1} \int_{0}^{2\pi} F(\lambda,\mu,t) Y^*_{m,n} \, d\lambda \, d\mu \qquad \text{for all } m,n \tag{3.4.25}$$

The property Eq.(3.4.19) implies that,

$$Y_{-m,n} = Y^*_{m,n} \tag{3.4.26}$$

and,

$$F_{-m,n}(t) = F^*_{m,n}(t) \tag{3.4.27}$$

This means that the model only needs to deal explicitly with the $F_{m,n}$ for $m \geq 0$. The values of $F_{m,n}$ for $m<0$ can be obtained by taking the complex conjugate of the values of $F_{m,n}$ for positive m. This halves the storage requirements of the model variables.

Another property that arises from the definition Eq.(3.4.18) is that,

$$P_{m,n}(\mu) = 0 \qquad \text{if } |m| > n \tag{3.4.28}$$

This can be used to write Eq.(3.4.24) as,

$$F(\lambda,\mu,t) = \sum_{m=-\infty}^{\infty} \sum_{n=|m|}^{\infty} F_{m,n}(t)\, Y_{m,n}(\lambda,\mu) \tag{3.4.29}$$

From Eq.(3.4.18) it can also be shown that,

$$P_{m,n}(-\mu) = (-1)^{n+|m|} P_{m,n}(\mu) \tag{3.4.30}$$

That is, the Legendre functions are antisymmetric about the equator when $n+|m|$ is odd and symmetric when $n+|m|$ is even. This property can be used to reduce the computations necessary for the Legendre transforms as described later.

Recurrence relations are used to compute the values of the $P_{m,n}$ and also $dP_{m,n}/d\mu$ as the $Y_{m,n}$ are not eigenfunctions of the meridional derivative. They are (Hobson, 1931),

$$\varepsilon_{m,n+1} P_{m,n+1}(\mu) = \mu P_{m,n} - \varepsilon_{m,n} P_{m,n-1} \tag{3.4.31}$$

$$P_{m+1,m+1} = [\, ((2m+3)/(2m+2))(1-\mu^2)\,]^{1/2} P_{m,m} \tag{3.4.32}$$

$$(\mu^2 - 1)\frac{dP_{m,n}}{d\mu} = n\varepsilon_{m,n+1}P_{m,n+1} - (n+1)\varepsilon_{m,n}P_{m,n-1} \tag{3.4.33}$$

$$\varepsilon_{m,n} = \left(\frac{n^2 - m^2}{4n^2 - 1}\right)^{1/2} \tag{3.4.34}$$

A more stable formula for high resolution is used at ECMWF (Jarraud and Simmons, 1983).

3.4.5. Truncation

A truncated set of basis functions must be used to solve the equations. As the variables are real-valued functions, from Eq.(3.4.27), m must satisfy $-M \leq m \leq M$. However, there is no restriction on the limit of the index n. So, the truncated set gives the approximation F' to F as,

$$F' = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} F_{m,n} Y_{m,n} \tag{3.4.35}$$

As m and n increase they correspond to features with decreasing horizontal scales. By careful choice of the limits M and $N(m)$ it is therefore possible to select the desired scales and features to be represented. This is somewhat analogous to choosing the size of the grid spacings in finite difference models.

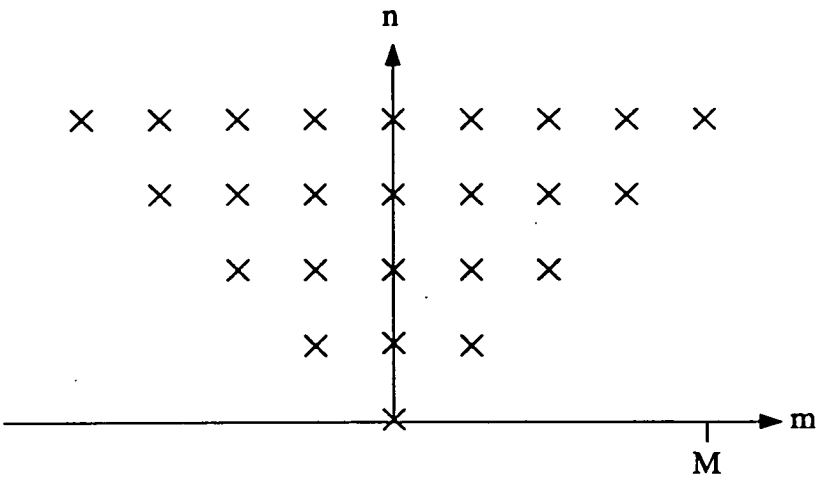
The two most common truncations used are known as triangular and rhomboidal truncation. These are best illustrated by their form in the complex, spectral plane and shown in Fig. 9. For triangular truncation,

$$N = M \tag{3.4.36}$$

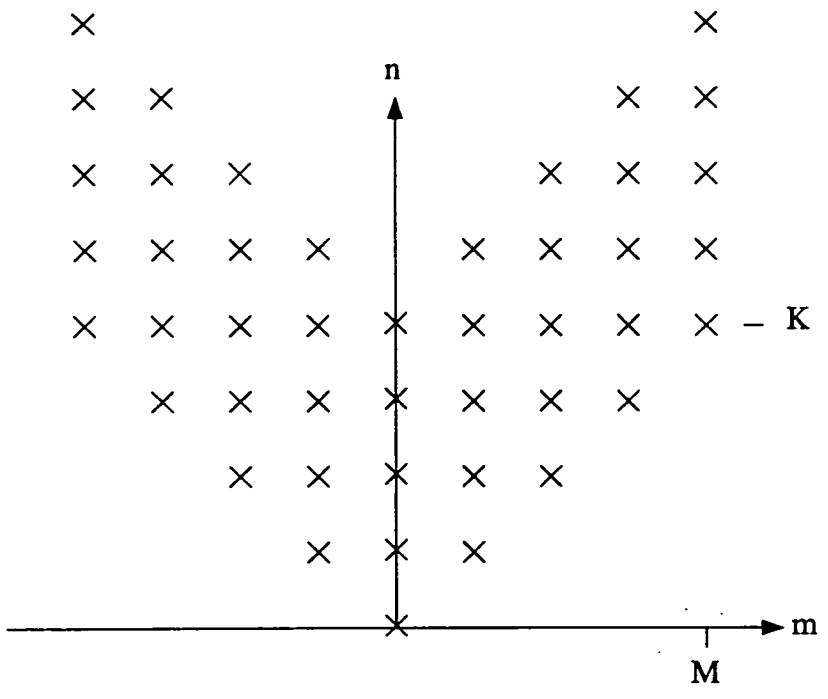
whilst for rhomboidal truncation,

$$N = K + M, \quad n = K + |m| \tag{3.4.37}$$

A property of the triangular truncation is that it is isotropic (Jarraud and Simmons, 1983). That is, it is invariant under rotation about an arbitrary axis through the centre of the sphere, so that the resolution is uniform on the sphere and the truncated field is the same whatever the position of the pole.



Triangular truncation



Rhomboidal truncation

Figure 9. Illustration of triangular and rhomboidal spectral truncation.

Therefore, for triangular truncation, all components with a horizontal scale smaller than that represented are consistently neglected.

On the other hand, rhomboidal truncation is only invariant under rotation about the Earth's axis, so that there is uniform resolution in the east-west direction but varying resolution in the north-south direction. With the same number of degrees of freedom as triangular truncation, the rhomboidal truncation gives increased resolution at high latitudes to the medium part of the spectrum (not just in the meridional direction) at the expense of lower longitudinal resolution at low latitudes. Rhomboidal truncation can be thought of as being analogous to a regular latitude-longitude grid whereas triangular truncation resembles a latitude-longitude grid with a decreasing number of points on a latitude circle approaching the pole.

Early spectral models employed the rhomboidal truncation, as Ellsaesser (1966) showed that for coarse resolutions this truncation could maximize the variance retained for the kinetic energy at 500mb. Later, however, Baer (1972) made a more detailed analysis of the kinetic energy from two winter months at several levels and his results favoured the triangular truncation.

Comparisons of the results from models using rhomboidal and triangular truncation have been made (e.g. Simmons and Hoskins, 1975; Daley and Bourassa, 1978; Jarraud and Simmons, 1983). The results were generally similar, with triangular truncation better at high levels.

From a computational point of view, triangular truncation offers several advantages over rhomboidal truncation. Since shorter waves are represented with rhomboidal truncation, the timestep must be smaller. Also, the Gaussian grid on which the nonlinear terms are calculated is larger by about 20%. These two considerations lead to an increase in computer time of about 25% for the rhomboidal truncation.

3.4.6. The transform method

Whilst linear terms can be computed in spectral space, the approach for nonlinear terms is not so straightforward. Early spectral models used the method of Silberman (1954), known as the interaction coefficient method. This involves substituting the spectral expansions directly into the equations and evaluating the extensive summations. Although selection rules (Baer and

Platzmann, 1961) can be used to reduce the number of computations, this is still of the order of N^5 where N is the truncation wavenumber (Orszag, 1970). Computer storage requirements are also large and become prohibitive as resolution increases.

A further problem was that the effects of physical parametrizations could not be included. As a consequence, only low resolution spectral models were possible until the transform method was developed independently by Orszag (1970) and Eliassen *et al.* (1970). In this approach, the nonlinear products are formed by transforming the required variables to gridpoint space, computing the products locally at each point and then transforming back to spectral space. The number of operations using this technique is proportional to N^3 (Orszag, 1970), a substantial reduction compared to the interaction coefficient method. Additionally, a substantial saving in storage space is achieved.

The transform method can be demonstrated by rewriting Eq.(3.4.25) as,

$$F_{m,n} = \frac{1}{2}\int_{-1}^{1}\left\{\frac{1}{2\pi}\int_{0}^{2\pi} F\, e^{-im\lambda}\, d\lambda\right\} P_{m,n}\, d\mu \tag{3.4.38}$$

That is, a Fourier transform,

$$F_m(\mu,t) = \frac{1}{2\pi}\int_{0}^{2\pi} F(\lambda,\mu,t)\, e^{-im\lambda}\, d\lambda \tag{3.4.39}$$

to give the Fourier coefficients at each latitude, followed by a direct Legendre transform,

$$F_{m,n}(t) = \frac{1}{2}\int_{-1}^{1} F_m(\mu,t) P_{m,n}(\mu)\, d\mu \tag{3.4.40}$$

to give the spectral coefficients. The inverse spectral transform Eq.(3.4.24) can be separated in the same way into inverse Legendre and Fourier transforms.

Considering the longitudinal integral first (the Fourier transform), this can be computed exactly by the trapezoidal quadrature formula if a sufficient number of points are chosen (Machenhauer and Rasmussen, 1972). The

integral becomes,

$$F_m(\mu_j,t) = \frac{1}{I}\sum_{i=1}^{I} F_{i,j}(t)\exp\{-im\lambda_i\} \tag{3.4.41}$$

using a regular set of points, λ_i, in the east-west direction. These are given by,

$$\lambda_i = 2\pi i / I \qquad \text{for } i = 0 \text{ to } I\text{-}1 \tag{3.4.42}$$

which is exact for $I\text{-}1 \geq N$, where N is the maximum wavenumber of the series to be summed (Machenhauer, 1979). The integrand of Eq.(3.4.41) is the product of three trigonometric functions, as $F_{i,j}$ represents the result of calculating the nonlinear terms (the product of two functions) each with a maximum zonal wavenumber M. The result is therefore exact if I satisfies,

$$I \geq 3M + 1 \tag{3.4.43}$$

(Orszag, 1971). In practice, the number of points, I, is chosen to allow the use of the Fast Fourier Transform (FFT) algorithm (Cooley and Tukey, 1965).

The latitudinal integral Eq.(3.4.40) is computed using the Gaussian quadrature formula (Eliasen *et al*, 1970),

$$F_{m,n} = \frac{1}{2}\sum_{j=1}^{J} g_j F_m(\mu_j,t) P_{m,n}(\mu_j) \tag{3.4.44}$$

which is exact for an integrand that is a polynomial in μ of degree $\leq 2J\text{-}1$. The μ_j are the zeros of $P_{0,J}$, that is,

$$P_{0,J}(\mu_j) = 0 \qquad \text{for } j = 1 \text{ to } J \tag{3.4.45}$$

which is the condition used to determine the Gaussian latitudes, μ_j, of the grid. They can be approximated by the zeros of Bessel functions (Abramowitz and Stegun, 1965) and then iterated using a Newton-Raphson procedure until the desired accuracy is achieved. The Gaussian coefficients or weights, g_j, are determined by,

$$g_j = \frac{2(1 - \mu_j^2)(2J - 1)}{[JP_{0,J-1}(\mu_j)]^2} \tag{3.4.46}$$

using the normalized values of the $P_{m,n}$ given in Eq.(3.4.18)

Substituting the spectral expansions for the variables into the product of two functions (the calculation at each gridpoint), shows that the integrand of Eq.(3.4.40) is a polynomial in μ of degree $3M$ for triangular truncation and $2M+3K$ for rhomboidal truncation. Therefore the number of latitudes, J, must satisfy,

$$J \geq (3M + 1) / 2 \tag{3.4.47}$$

and,

$$J \geq (2M + 3K + 1) / 2 \tag{3.4.48}$$

for triangular and rhomboidal truncation respectively.

Unfortunately, there is no fast algorithm analogous to the FFT for the Legendre transform, although optimization is possible as described below. It is not surprising therefore that the Legendre transforms account for a significant fraction of the total computational cost of any spectral model.

The grid on which the nonlinear products are calculated is regular in the east-west direction but irregular in the north-south direction. However, the irregularity decreases with increasing resolution. A major advantage of the transform method over the interaction coefficient method is that the use of a grid in physical space allows the effects of the physical processes to be included.

3.4.6.1. Optimization of the transforms

The Legendre transforms may be optimized by using the property Eq.(3.4.30), that the Legendre functions are either symmetric or antisymmetric about the equator, to halve the number of necessary computations (Machenhauer, 1979; Jarraud and Simmons, 1983).

Optimization of the FFT is also possible. Since the spectral coefficients are conjugate-symmetric from Eq.(3.4.27), the transform of real data of length N can be done using a complex transform of length $N/2$ by treating the even

values as the real part and the odd values as the imaginary part (Hockney and Jesshope, 1981; Orszag, 1971). Also, it is possible to improve the FFT algorithm if I can be factorized into mutually prime numbers (Temperton, 1983).

3.5. Galerkin techniques: Finite element method

3.5.1. Introduction

Although the use of the finite element method in engineering had become standard by the early 1970s (e.g. Zienkiewicz, 1971), the first meteorological applications of the method were only just beginning to appear in the literature (e.g. Wang *et al.*, 1972; Cullen, 1973). Finite element barotropic models were developed by Cullen (1974b) and Staniforth and Mitchell (1977). The global model of Cullen (1974b) used triangular elements mapped onto a icosahedron. A baroclinic model was developed by Staniforth and Daley (1979). Boundary layer finite element models have also been developed independently by Mailhot and Benoit (1982) and Chang *et al.* (1982). The finite element technique has also been successfully used for the vertical discretization in sigma coordinate primitive equation models, particularly those incorporating a spectral representation in the horizontal. Work on the Canadian spectral and finite element models using finite elements in the vertical has been performed by Staniforth and Daley (1977), Cote *et al.* (1983) and Beland and Beaudoin (1985). More recent work on the inclusion of finite elements in the vertical for the ECMWF spectral forecast model has been done by Burridge *et al.* (1986) and Steppler (1986). Whilst at current model resolutions the finite element method is not superior to the spectral technique for global models (Cullen, 1974b), it is being increasingly used by research workers developing mesoscale and limited area models, as demonstrated by the list of active mesoscale research groups in Pielke (1984).

In this section, the general approach of the method is described along with a discussion of its properties and the treatment of boundary conditions. Most of this review is a summary of the papers by Cullen, Staniforth and of selected text books.

3.5.2. General method

The finite element method is usually introduced using variational principles (e.g. Strang and Fix, 1973). However, the Galerkin method is more suited to time dependent problems and does not rely on finding a variational form of the problem.

Like the spectral method, the finite element method is best described in terms of Hilbert spaces and their norms. The admissible functions to a Hilbert space must have finite energy,

$$N(f) = \int_\tau f^2 \, d\tau \; < \infty \tag{3.5.1}$$

where $N(f)$ is the norm. The space of functions satisfying Eq.(3.5.1) is denoted by H^0. The superscript denotes how many derivatives of f are required to have finite energy.

The Galerkin finite element method proceeds as for the spectral method. The general form of meteorological equations, Eq.(3.4.1), is written as,

$$w = A(F) \tag{3.5.2}$$

where $w = \partial F / \partial t$. This is multiplied by some test function, v, in some test space, V, and integrated over the domain to give,

$$(w,v) = (A,v) \tag{3.5.3}$$

This must hold for each function v in V.

If V is H^0, the equation is said to hold in a strong sense. The solution must therefore lie in the space H_B^{2m} where $2m$ is the order of the operator A and B denotes that the full boundary conditions must be satisfied. However, by choosing a test space $V = H^s$, this permits (by integration by parts) s of the derivatives in A to be shifted from F to v. The solution therefore need only be sought in H^{2m-s}.

As s increases and only $2m-s$ derivatives of F need have finite energy (by shifting s of them to v) to qualify for the solution space, the only boundary conditions which can be applied directly are those of order less than $2m-s$. This is an important point in the finite element method. Those boundary

conditions of order less than $2m-s$ are known as the essential conditions in that the test functions v must satisfy them. Those conditions of order greater than or equal to $2m-s$ are known as natural boundary conditions as the Galerkin problem can be formulated so that the test functions v automatically satisfy them.

When $m=s$, the test space is the same as the solution space (the Ritz case), as is usual for meteorological applications using the finite element method. The solution space is then denoted by H_E^m where the E denotes that only the essential boundary conditions need to be satisfied. The test functions, v, must lie in the solution space H_E^m.

The Galerkin method is a discretization of the continuous equation Eq.(3.5.3). The Ritz case is used, where the functions, ϕ_n, n=1 to N, form a set for the space H_E^m. Any function in H_E^m can be expanded as,

$$f = \sum_{n=1}^{N} q_n \phi_n \tag{3.5.4}$$

where the q_n are the expansion coefficients.

Since the test functions v in Eq.(3.5.3) can also be written as Eq.(3.5.4) it is sufficient to use ϕ_n in Eq.(3.5.3),

$$(\sum_{n=1}^{N} w_n \phi_n, \phi_m) = (A(\sum_{n=1}^{N} F_n \phi_n), \phi_m) \qquad \text{for } m=1 \text{ to } N \tag{3.5.5}$$

Eq.(3.5.5) can then be solved by a suitable choice of basis functions. Unlike the spectral method, the basis functions usually employed in the finite element method are not orthogonal, although they are nearly so.

3.5.3. Some properties of the finite element method

Papers by Cullen (1973, 1974a, 1974b) illustrate that finite element models can achieve better results than finite difference models of second order that use four times the number of gridpoints. Aliasing is eliminated in finite element models, although not in the sense of the spectral method. Instead short wave interactions are heavily damped (Cullen, 1973). Also, phase propagation is accurate and nonlinearities are handled well. The finite element

technique, as a Galerkin method, has good conservation properties, depending on the problem.

One advantage over spectral methods is that the finite element method can be applied to irregular domains and have irregular spacing with little additional effort. Grids that vary with time are also possible. An advantage over finite difference techniques is that boundary conditions can be incorporated in a more mathematically consistent way. Also, the finite element method implies the form of the field between nodes from the basis functions used. In the finite difference method nothing is known about the shape of the field between gridpoints, which therefore leads to aliasing.

The disadvantages of the finite element method are that first the resulting scheme is more difficult to program; large systems of linear equations have to be solved. Noise can also be a problem with finite element models (Cullen, 1976) and one known cause is when all the variables are carried at each nodal point (Haltiner and Williams, 1980). However, in the barotropic model of Staniforth and Mitchell (1977), the vorticity and divergence equations were used as in spectral models. This was found to reduce the noise problem considerably. Williams and Zienkiewicz (1981) have shown that using different basis functions for the height and velocity variables can also give good results.

3.5.4. Basis functions

The key difference between the spectral and finite element methods is that the basis functions used in the former are defined over the whole domain whereas for the latter they are nonzero only over a small, local area. It is this local nature of the basis functions that gives the finite element method its flexibility in solving problems with irregular geometry.

The domain is first divided into a mesh of nodal points. The placement of nodal points depends on the choice of basis functions; triangular elements would require the region to be divided into triangular regions. For general problems, this will require some approximation of the boundary (Strang and Fix, 1973). However, at any point in the region the mesh can be refined to give a higher resolution (Staniforth and Mitchell, 1978). The choice of the type of basis function depends on the desired accuracy of the solution and on the

geometry of the problem itself.

The simplest basis function is the one-dimensional Chapeau function. It is defined by,

$$\begin{aligned} \phi_n &= 0 && \text{for } x < x_{n-1} \text{ and } x > x_{n+1} \\ \phi_n &= (x-x_{n-1})/(x_n-x_{n-1}) && \text{for } x_{n-1} \leq x \leq x_n \\ \phi_n &= (x_{n+1}-x)/(x_{n+1}-x_n) && \text{for } x_n \leq x \leq x_{n+1} \end{aligned} \tag{3.5.6}$$

and is 1 only at x_n and zero elsewhere. This is an interpolatory basis function since for an expansion,

$$u(x) = \sum_{n=0}^{N} u_n \phi_n(x) \tag{3.5.7}$$

then,

$$u(x_n) = u_n \tag{3.5.8}$$

Thus, unlike the spectral method, the expansion coefficients u_n are the field values at the nodal points x_n. This is not true for higher order elements.

The simplest basis function in two dimensions is the rectangular bilinear piecewise basis function. It is the product of two one-dimensional Chapeau functions i.e.

$$e_p(x,y) = \phi_i(x)\phi_j(y) \tag{3.5.9}$$

where $p = p(i,j)$. This is also an interpolatory basis function.

Any function $u(x,y)$ can be expanded as,

$$u(x,y) = \sum_{p=0}^{M} u_p e_p(x,y) \tag{3.5.10}$$

but from Eq.(3.5.9), this can be written as,

$$u(x,y) = \sum_{i=0}^{N} \sum_{j=0}^{N} u_{i,j} \phi_i(x)\phi_j(y) \tag{3.5.11}$$

As the solution space is a product of one-dimensional spaces, the finite

element matrices resulting from the approximation of the linear operators in Eq.(3.5.2) may also be decomposed into the product of one-dimensional matrices. This results in reduced computation. This property of the bilinear elements has been used by Staniforth and co-workers in developing their barotropic and baroclinic models (Staniforth, 1987).

3.5.5. Approximation of some simple terms

In this section, the finite element method is illustrated for some simple terms which form the components for solving meteorological problems and will be used for the model developed in chapter 7. In particular, the treatment of the nonlinear advective terms is described, where the precise approach is important. The one-dimensional examples discussed below are a summary of the review by Cullen (1979).

3.5.5.1. First derivative

Consider the equation,

$$w = \partial u/\partial x \tag{3.5.12}$$

over the interval x=0 to x where $x=\Sigma_{i=0}^{N-1}\Delta x_i$. Following the Galerkin procedure, the variables w and u are expanded as,

$$w = \sum_{i=0}^{N} w_i\phi_i(x), \qquad u = \sum_{i=0}^{N} u_i\phi_i(x) \tag{3.5.13}$$

Substituting into Eq.(3.5.12) and taking the scalar product gives,

$$\sum_{i=0}^{N} w_i \int_0^X \phi_j\phi_i \, dx = \sum_{i=0}^{N} u_i \int_0^X \phi_j d\phi_i/dx \, dx \qquad \text{for all } j \tag{3.5.14}$$

If the integrals are written as matrix elements,

$$M_{ji} = \int \phi_j\phi_i \, dx, \qquad P_{xji} = \int \phi_j \, d\phi_i/dx \, dx \tag{3.5.15}$$

then Eq.(3.5.14) can be written as,

$$\underline{\underline{M}}\,\underline{w} = \underline{\underline{P}}_x\,\underline{u} \tag{3.5.16}$$

where $\underline{u}$ and $\underline{w}$ are vectors containing the nodal values. $\underline{\underline{M}}$ is the so-called mass matrix (Strang and Fix, 1973) and both $\underline{\underline{M}}$ and $\underline{\underline{P}}_x$ are often referred to in the literature as Galerkin or projection operators (Burridge *et al*, 1986).

The integrals in Eq.(3.5.15) are evaluated by substituting for ϕ_n using Eq.(3.5.6). Since the ϕ_n are locally defined, only neighbouring elements interact. Thus, although the basis functions are not orthogonal they are nearly so and the evaluation of the integrals is straightforward.

To evaluate $\underline{\underline{P}}_x$, the derivative of the basis functions is required, which is,

$$d\phi_i/dx = \begin{cases} 1/\Delta x_{i-1} & \text{for } x_{i-1} < x < x_i \\ -1/\Delta x_i & \text{for } x_i < x < x_{i+1} \\ 0 & \text{elsewhere} \end{cases} \tag{3.5.17}$$

This leads to,

$$P_{x_{i,i-1}} = -1/2, \qquad P_{x_{i,i+1}} = 1/2, \qquad P_{x_{i,i}} = 0 \tag{3.5.18}$$

together with,

$$M_{i,i-1} = \Delta x_{i-1} / 6, \qquad M_{i,i+1} = \Delta x_{i+1} / 6 \tag{3.5.19}$$
$$M_{i,i} = (\Delta x_{i-1} + \Delta x_i) / 3$$

so the resulting scheme for a node j is,

$$\frac{w_{j-1}}{6}\Delta x_{j-1} + \frac{w_j}{3}(\Delta x_{j-1} + \Delta x_j) + \frac{w_{j+1}}{6}\Delta x_j = \frac{1}{2}(u_{j+1} - u_{j-1}) \tag{3.5.20}$$

The solution of Eq.(3.5.16) is given by,

$$\underline{w} = \underline{\underline{M}}^{-1}\underline{\underline{P}}_x\,\underline{u} \tag{3.5.21}$$

Since the mass matrix is positive definite it is invertible and Eq.(3.5.21) can be solved. Both matrices (as with all matrices resulting from the use of one-dimensional linear basis functions) are tridiagonal. Higher order basis functions and problems in more than one dimension lead to higher bandwidth matrices.

Eq.(3.5.21) is implicit and therefore the solution at each node must be

obtained simultaneously. The additional cost to solving Eq.(3.5.21), when compared with an explicit finite difference scheme, is offset by its superior accuracy. With an irregular spacing, as above, the solution is first order accurate. For regular spacing the usual truncation error analysis shows the method to be fourth order accurate (Cullen, 1976). This high accuracy is a case of superconvergence, where the accuracy of the solution at the nodes is of a higher accuracy than at non-nodal points (Strang and Fix, 1973; de Boor, 1974). For free boundary nodes (free meaning no boundary conditions are imposed) the accuracy is only first order. The boundary nodes are therefore likely to be a source of error. This can be improved by including more nodes in the equation for the solution of the boundary node as in the procedure given in the appendix to Beland and Beaudoin (1985). However, the modified mass matrix is then no longer tridiagonal. The approach of Beland and Beaudoin (1985), similar to the approach used in compact differencing, can also be used to obtain a scheme for the first derivative that is fourth order accurate with irregular spacing and the resulting matrices are tridiagonal.

For the two dimensional problem, Eq.(3.5.12) is solved over a domain x=0 to X, z=0 to Z. The variables are expanded as in Eq.(3.5.13) but using the bilinear basis functions defined in Eq.(3.5.9). Applying the Galerkin procedure gives,

$$\sum_{p=0}^{P} w_p \int_D e_q e_p \, dx dz = \sum_{p=0}^{P} u_p \int_D e_q \partial e_p / \partial x \, dx dz \quad \text{for all } q \tag{3.5.22}$$

By introducing the matrices,

$$M_{qp} = \int_D e_q e_p \, dx dz \tag{3.5.23}$$

$$P_{xqp} = \int_D e_q \partial e_p / \partial x \, dx dz$$

Eq.(3.5.22) can be written in matrix form as before,

$$\underline{\underline{M}}\,\underline{w} = \underline{\underline{P}}_x \underline{u} \tag{3.5.24}$$

except $\underline{w}$ and $\underline{u}$ are vectors holding all the nodes in the two-dimensional domain in some particular nodal ordering (e.g. along successive horizontal

levels). The mass matrix in Eq.(3.5.23) is block tridiagonal where each block is itself tridiagonal.

Staniforth and Mitchell (1977) describe how the use of the bilinear basis function enables Eq.(3.5.22) to also be solved as a set of one-dimensional problems. To see this, for the LHS of Eq.(3.5.22) substitute for the bilinear basis using Eq.(3.5.9) and separate the integrals to give,

$$\sum_{p=0}^{P} w_p \int_D e_q e_p \, dx dz = \sum_{i=0}^{I} \sum_{k=0}^{K} w_{ik} \int_0^X \phi_j \phi_i \, dx \int_0^Z \phi_l \phi_k \, dz \tag{3.5.25}$$

Following the same procedure for the RHS and using matrix notation, Eq.(3.5.22) becomes,

$$\underline{\underline{N}}(\underline{\underline{M}}\,\underline{w})^T = \underline{\underline{N}}(\underline{\underline{P}}_x\,\underline{u})^T \tag{3.5.26}$$

where, $\underline{\underline{N}}$ is the mass matrix for the vertical only, $\underline{\underline{M}}$ is the horizontal mass matrix and $\underline{\underline{P}}_x$ is the projection matrix defined in Eq.(3.5.15). The variables w and u are now represented by matrices rather than vectors. The approach of Staniforth and Mitchell (1977) is to first solve $\underline{\underline{M}}\,\underline{w}$ and then $\underline{u}$. However, as pointed out by Staniforth (1987), multiplying Eq.(3.5.26) by $\underline{\underline{N}}^{-1}$ reduces it to,

$$\underline{\underline{M}}\,\underline{w} = \underline{\underline{P}}_x\,\underline{u} \tag{3.5.27}$$

Thus the computation of the derivative becomes less expensive as not only is the RHS simpler but only tridiagonal systems need to be solved. This simplification could not be done if the e_p were not the product of one-dimensional basis functions or if the domain was not rectangular.

3.5.5.2. Approximation of products

Now consider the simplest nonlinear equation in one dimension,

$$w = uv \tag{3.5.28}$$

Following the Galerkin procedure, the resulting finite element equation is,

$$\sum_{i=0}^{N} w_i \int_0^X \phi_k \phi_i \, dx = \sum_{i=0}^{N} \sum_{j=0}^{N} u_i v_j \int_0^X \phi_k \phi_i \phi_j \, dx \qquad \text{for all } k \tag{3.5.29}$$

Evaluation of these integrals gives,

$$\int \phi_k^3 \, dx = (\Delta x_{k-1} + \Delta x_k) / 4$$

$$\int \phi_{k-1}\phi_k^2 \, dx = \int \phi_{k-1}^2\phi_k \, dx = \Delta x_{k-1} / 12 \qquad (3.5.30)$$

$$\int \phi_{k+1}\phi_k^2 \, dx = \int \phi_{k+1}^2\phi_k \, dx = \Delta x_k / 12$$

which, substituting into Eq.(3.5.29) gives the following equation for a node k,

$$\frac{w_{k-1}}{6}\Delta x_{k-1}+\frac{w_k}{3}(\Delta x_{k-1}+\Delta x_k)+\frac{w_{k+1}}{6}\Delta x_k = \frac{1}{12}[\Delta x_k(u_k+u_{k+1})(v_k+v_{k+1})+ \qquad (3.5.31)$$

$$2u_kv_k(\Delta x_{k-1}+\Delta x_k)+\Delta x_{k-1}(u_k+u_{k-1})(v_k+v_{k-1})]$$

It is possible to write this in matrix form by introducing the projection matrix $\underline{\underline{P}}_u$ (Cote *et al.*, 1983) where,

$$P_{ukj} = \sum_{i=0}^{N} u_i \int_0^X \phi_k\phi_j\phi_i \, dx \qquad (3.5.32)$$

so that Eq.(3.5.29) can be written as,

$$\underline{\underline{M}}\,\underline{w} = \underline{\underline{P}}_u\,\underline{v} \qquad (3.5.33)$$

where $\underline{\underline{M}}$ is the mass matrix defined in Eq.(3.5.15). The solution to Eq.(3.5.33) again requires the solution to a set of simultaneous equations.

Cullen (1976) has analysed the error associated with this scheme and shown it to be fourth order accurate. Using the basic operations of differentiation and multiplication it is possible to have a fourth order accurate finite element model. An example of this is the barotropic model of Staniforth and Mitchell (1977).

Also important is the scheme's treatment of short wave interactions. This can be examined by substituting Fourier modes of the form,

$$u = \exp\{ikx_n\}, \qquad v = \exp\{ilx_n\}, \qquad w = K(k,l)\exp\{i(k+l)x_n\} \tag{3.5.34}$$

into Eq.(3.5.31). Solving for $K(k,l)$ gives,

$$K(k,l) = (3 + \cos k + \cos l + \cos(k+l)) \,/\, (4 + 2\cos(k+l)) \tag{3.5.35}$$

This function is plotted in Fig. 10 for $k=l$ and $k=2\pi/\lambda$ where λ is the wavelength, assuming a constant spacing. From this figure it can be seen that those short wave interactions that would result in aliasing are heavily damped (although not shown, this is also true when $k \neq l$). With the possibility of aliasing greatly reduced, nonlinear instability as described by Phillips (1959) is less likely to occur. Cullen (1973) found this to be the case in practice.

Although the Galerkin formulation reduces aliasing, it has been found that Eq.(3.5.28) can be computed by,

$$w_n = u_n v_n \tag{3.5.36}$$

when either u or v is a smoothly varying function (Staniforth and Mitchell, 1977). This point collocation approach is still fourth order accurate but is much cheaper to compute.

For the product of two functions in two dimensions the procedure is identical to that used for the first derivative. Again, the equation may be solved using one dimensional matrices. The finite element approximation becomes,

$$\underline{\underline{M}}(\underline{\underline{N}}\,\underline{\underline{w}}^{\mathsf{T}})^{\mathsf{T}} = \underline{\underline{F}} \tag{3.5.37}$$

where $\underline{\underline{F}}$ represents the matrix resulting from the computation of the RHS to the Galerkin equation of Eq.(3.5.28) in two dimensions. The elements of $\underline{\underline{F}}$ are given by,

$$F_{i,j} = \frac{\Delta x \Delta z}{9}\Big[\, P_{i-\frac{1}{2},j-\frac{1}{2}} + P_{i,j-\frac{1}{2}} + P_{i+\frac{1}{2},j-\frac{1}{2}} + P_{i-\frac{1}{2},j} + \tag{3.5.38}$$

$$P_{i,j} + P_{i+\frac{1}{2},j} + P_{i-\frac{1}{2},j+\frac{1}{2}} + P_{i,j+\frac{1}{2}} + P_{i+\frac{1}{2},j+\frac{1}{2}} \,\Big]$$

where,

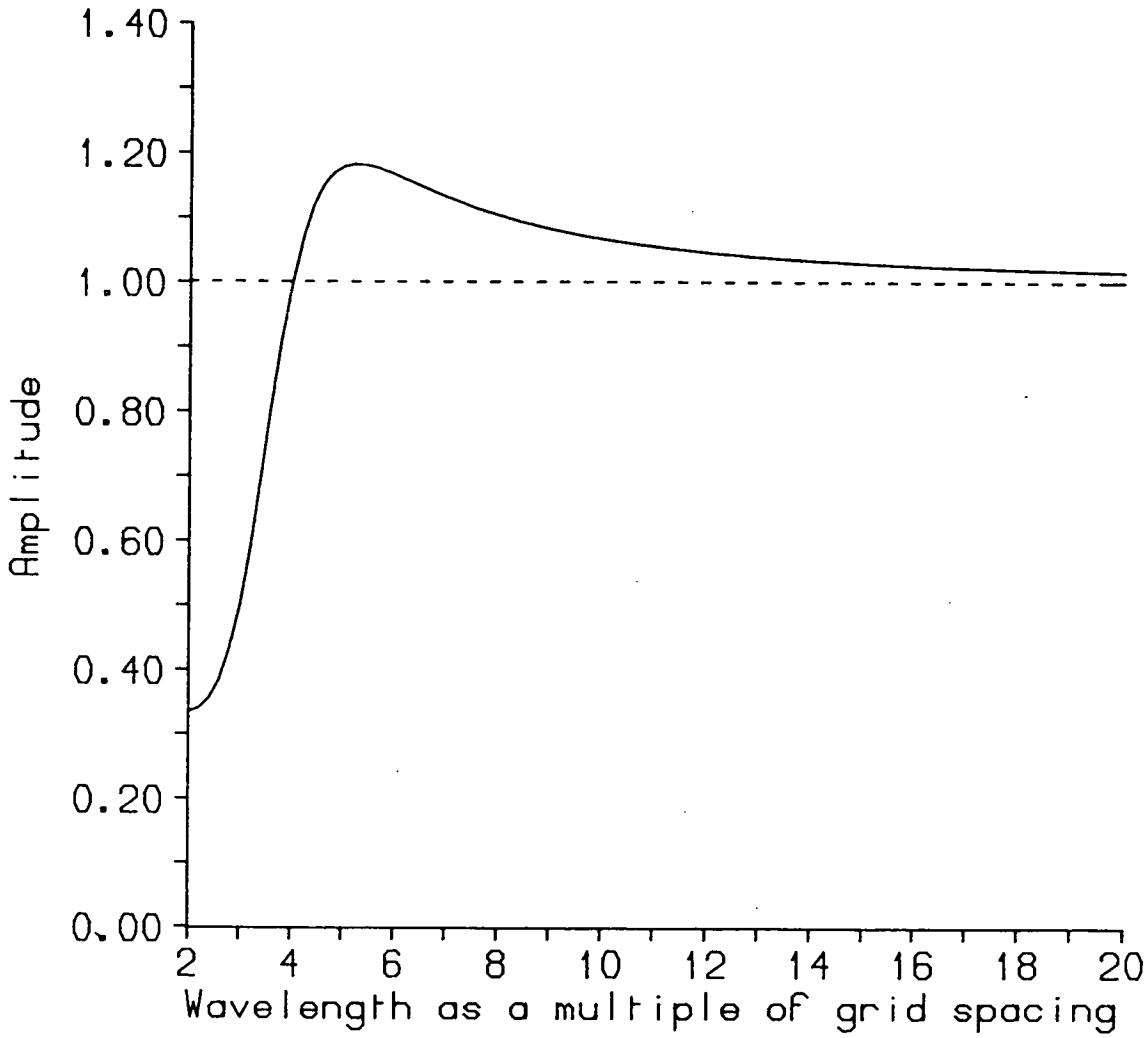


Figure 10. Galerkin scheme for $w = uv$ (for waves of equal wavelength.)

$$P_{i,j+\frac{1}{2}} = u_{i,j+\frac{1}{2}}\, v_{i,j+\frac{1}{2}} \tag{3.5.39}$$

$$u_{i,j+\frac{1}{2}} = (u_{i,j} + u_{i,j+1}) \,/\, 2$$

and equal spacing in both directions is assumed for clarity. The procedure to solve Eq.(3.5.37) is therefore to first solve,

$$\underline{\underline{A}}^{T} = \underline{\underline{M}}^{-1}\, \underline{\underline{F}} \tag{3.5.40}$$

and then solve,

$$\underline{\underline{w}}^{T} = \underline{\underline{N}}^{-1}\, \underline{\underline{A}} \tag{3.5.41}$$

3.5.5.3. Approximation of advective terms

The finite element approximation of the one-dimensional advective term,

$$w = u\partial v/\partial x \tag{3.5.42}$$

is now considered. In finite difference approximations, care must be taken to ensure that the scheme does not lead to nonlinear instability.

The procedure is as before, the variables are approximated and the scalar product with the basis functions is taken over the domain. The resulting scheme for a node i is (Cullen, 1979),

$$\frac{1}{6}[w_{i-1} + 4w_i + w_{i+1}] = \frac{1}{2\Delta x}[(\frac{1}{3}u_{i-1} + \frac{2}{3}u_i)(v_i - v_{i-1}) + \tag{3.5.43}$$

$$(\frac{1}{3}u_{i+1} + \frac{2}{3}u_i)(v_{i+1} - v_i)]$$

where regular spacing is assumed.

The analysis of Cullen (1979) shows that this scheme has a leading truncation error term of $\Delta x^4/40$. However, using the basic operations of differentiation and multiplication it is possible to compute Eq.(3.5.42) using an alternate approach, as described in Cullen (1974a). First compute $s=\partial v/\partial x$ so that s is fitted to a piecewise linear function. Then compute the product $w=us$ using the finite element scheme for products. Since these individual operations are fourth order accurate, the combined scheme gives the solution at the nodes to fourth order accuracy. This split or two stage scheme has

been shown by Cullen (1974a) to have a leading error term of $\Delta x^4/240$ so that it is more accurate than the single stage scheme above. However, it is no longer a conservative scheme. Even so, when the two stage scheme was used in the model of Staniforth and Mitchell (1977) the total energy and mass of the barotropic model were conserved to a few percent over a run of 50 days.

The accuracy of both schemes can be illustrated by substituting Fourier modes of the form given by Eq.(3.5.34), assuming regular unit spacing. For Eq.(3.5.43),

$$K(k,l) = \frac{\mathrm{i}(\sin(k+l) + 2\sin l - \sin k)}{2 + \cos(k+l)} \tag{3.5.44}$$

whilst the response of the two stage scheme is,

$$K(k,l) = \frac{3\sin k}{2+\cos k} \; \frac{3 + \cos k + \cos l + \cos(k+l)}{4 + 2\cos(k+l)} \tag{3.5.45}$$

By comparison, the response of the second order finite difference approximation,

$$w_i = \frac{1}{4\Delta x} [(u_i + u_{i-1})(v_i - v_{i-1}) + (u_i + u_{i+1})(v_{i+1} - v_i)] \tag{3.5.46}$$

is,

$$K(k,l) = \mathrm{i}(\sin l - \sin k + \sin(k+l)) \; / \; 2 \tag{3.5.47}$$

Fig. 11 shows a comparison of Eq.(3.5.44), Eq.(3.5.45) and Eq.(3.5.47) plotted for $k=l$ Both the finite element schemes are superior to the finite difference scheme. The two stage scheme is the most accurate at a wavelength of $4\Delta x$ but appears to be slightly less accurate for wavelengths between $5\Delta x$ and $8\Delta x$

The extension to two dimensions for the single step scheme is straightforward and follows the procedure for the approximation of a product in two dimensions. For the two stage scheme, the approach is a combination of the two dimensional schemes used for the first derivative and product.

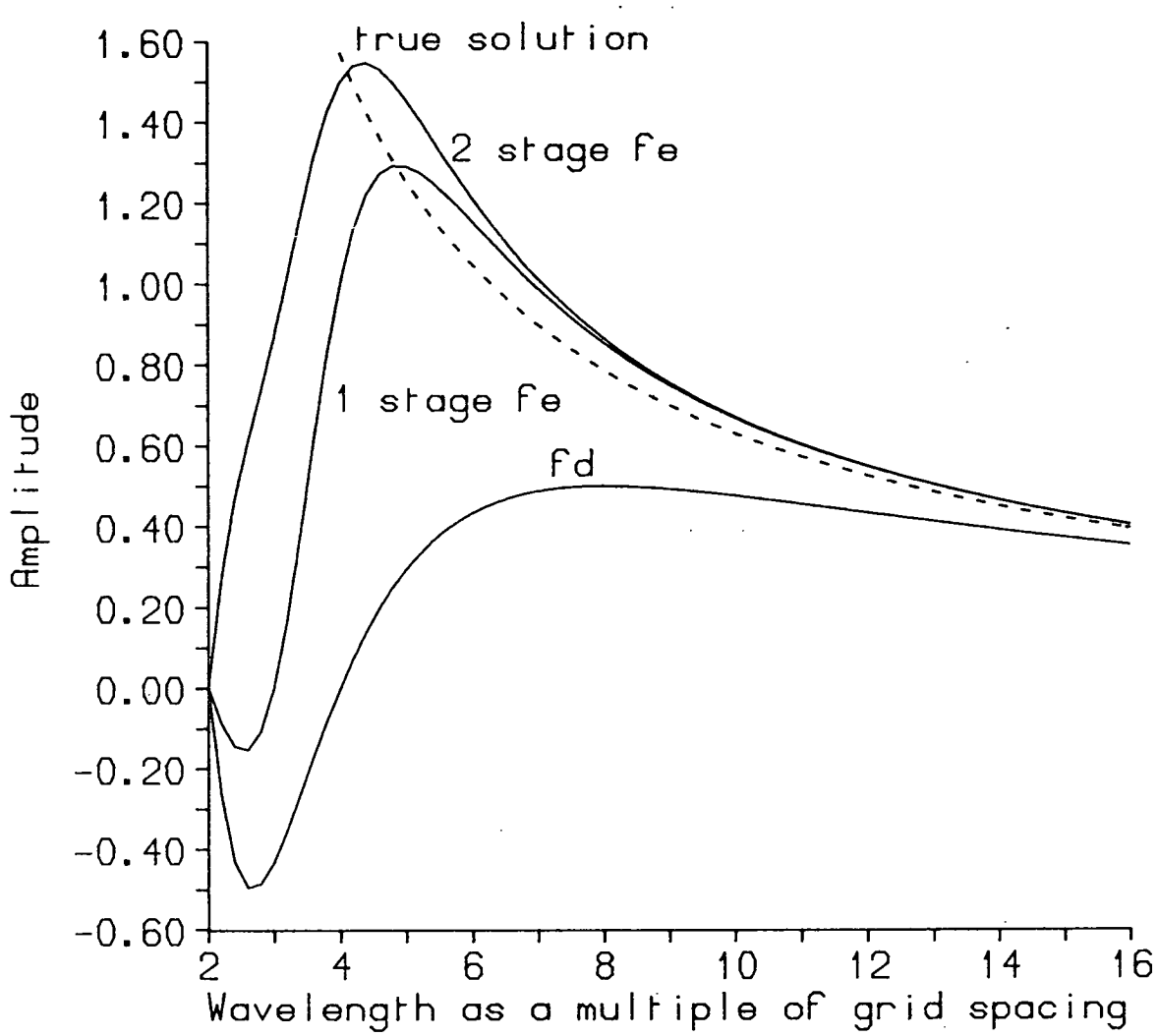


Figure 11. Comparison of Galerkin schemes and a finite difference scheme for a non-linear advection term.

3.5.6. Stability and phase properties

To examine the stability criterion of the finite element method consider the one dimensional advection equation,

$$\partial u/\partial t = c\partial u/\partial x \tag{3.5.48}$$

where c is a constant. Following the procedure in Haltiner and Williams (1980), assume a solution of the form,

$$u = \lambda^n e^{ikx} \tag{3.5.49}$$

where λ^n represents an amplification factor at time level n and $\lambda=\lambda^{n+1}/\lambda^n$. The Galerkin scheme for Eq.(3.5.48) is given by Eq.(3.5.20) so,

$$\frac{1}{6}[\dot{u}_{i-1} + 4\dot{u}_i + \dot{u}_{i+1}] = \frac{c}{2\Delta x}[u^n_{i+1} - u^n_{i-1}] \tag{3.5.50}$$

The time derivative is approximated using the leapfrog time scheme.

Substituting Eq.(3.5.49) into Eq.(3.5.50) and requiring that λ be bounded for all n gives,

$$\Delta t \leq \frac{\Delta x}{c\sqrt{3}} = \frac{0.57\Delta x}{c} \tag{3.5.51}$$

The fourth order accurate finite difference scheme,

$$\partial u/\partial x = [\ 8(u_{i+1} - u_{i-1}) - (u_{i+2} - u_{i-2})\ ] \ /\ 12\Delta x \tag{3.5.52}$$

by comparison requires a timestep of,

$$\Delta t \leq \frac{3\Delta x}{4c} = \frac{0.75\Delta x}{c} \tag{3.5.53}$$

for stability. The finite element approximation is more restrictive. This has led to the use of implicit, semi-implicit and semi-Lagrangian time integration schemes for finite element models.

To study the phase properties of the scheme, a solution of the form,

$$u = \exp\{i(kx + \sigma t)\} \tag{3.5.54}$$

(Cullen, 1973), is substituted into Eq.(3.5.50) and Eq.(3.5.52) and σ solved for. Fig. 12 shows the behaviour of the phase velocity, where the ratio of σ to the

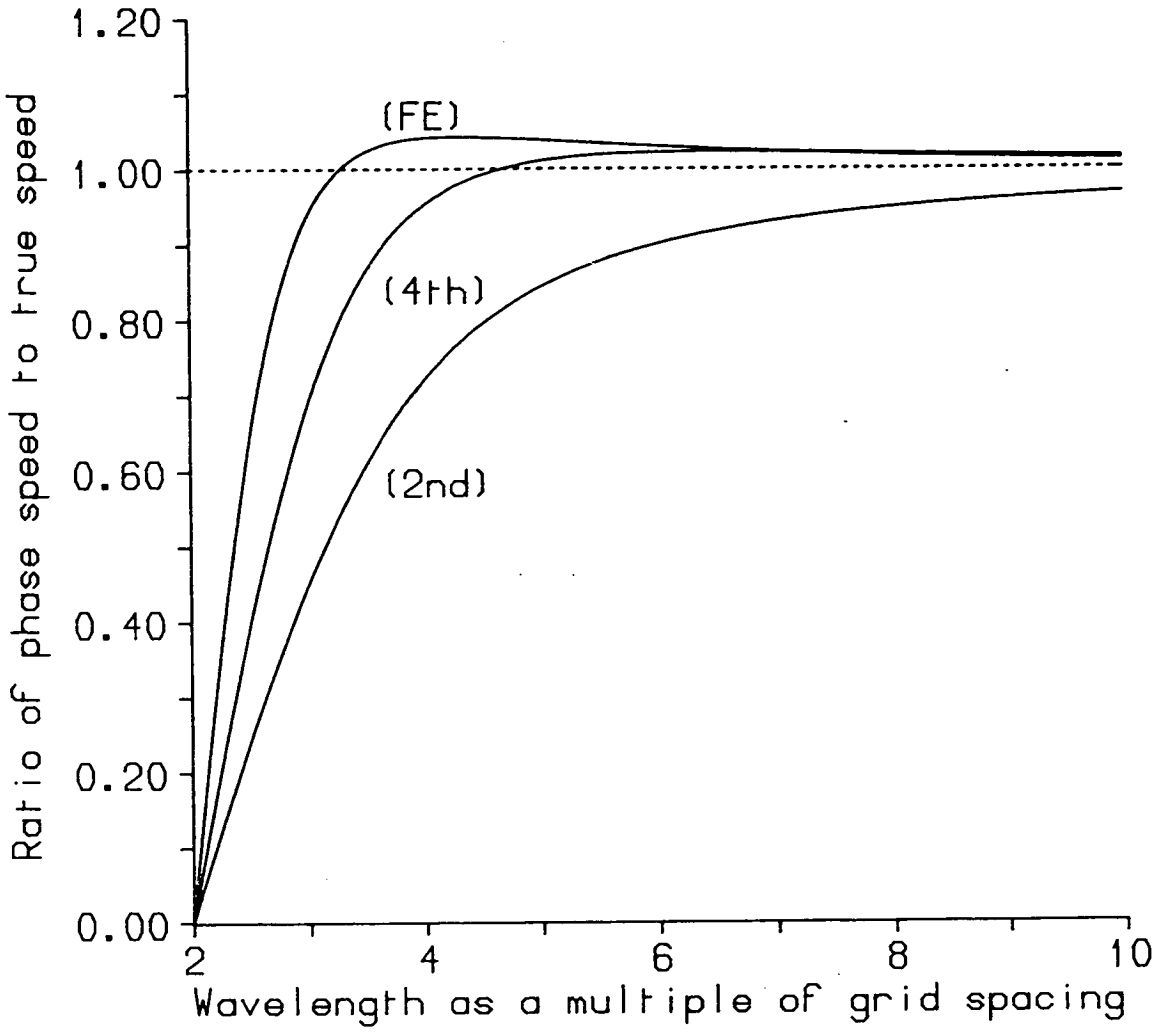


Figure 12. Comparison of the phase properties of finite element and finite difference schemes.

true phase velocity $2\pi c/\lambda$ is plotted against wavelength. Also shown is the phase behaviour of a second order finite difference approximation to Eq.(3.5.48). A timestep that is 75% of the maximum allowable for each scheme is assumed in each case to illustrate the behaviour for a timestep close to that used in practice.

Fig. 12 shows that the finite element scheme is more accurate; even $3\Delta x$ waves are propagated at almost the correct phase speed. However, it can also be seen that, like the fourth order finite difference scheme, the short waves have a leading phase error. Nevertheless, the phase speeds of the waves in the finite element method are very accurate, a property noted by early literature on the use of the technique for meteorology (Cullen, 1973, 1974a; Staniforth and Mitchell, 1977). The penalty in the timestep is therefore due to the superior treatment of the shortest waves.

3.5.7. Boundary conditions

The incorporation of boundary conditions into a finite element problem is perhaps more mathematically consistent than in finite difference methods. The subject is not discussed to any length in meteorological literature on finite element applications. Therefore this section discusses some of the theory and finer points to the inclusion of boundary conditions.

3.5.7.1. Homogeneous conditions

To illustrate the application of boundary conditions the equation,

$$-\frac{d^2u}{dx^2} = f \tag{3.5.55}$$

is to be solved over a region x=0 to X. Since this is a second order equation, the second derivative of the solution, u, must have finite energy. That is, u must reside in the H^2 space.

Following Strang and Fix (1973), Eq.(3.5.55) is solved subject to the boundary conditions,

$$\begin{aligned} u(0) &= 0 \\ du(X)/dx &= 0 \end{aligned} \tag{3.5.56}$$

The solution must now reside in the space H^2_B, where the subscript denotes that Eq.(3.5.56) is satisfied.

Applying the Galerkin method to Eq.(3.5.55) and integrating the LHS by parts gives,

$$\int_0^X \frac{du}{dx}\frac{d\phi_i}{dx}dx + \left[\frac{du}{dx}\phi_i\right]_0^X = \int_0^X f\phi_i\, dx \qquad \text{for } i=1 \text{ to } N \tag{3.5.57}$$

where the basis functions, ϕ_i, can reside in H^1, as the order of the equation has been reduced. This implies that ϕ_i may be piecewise linear functions.

The integrated term at $x=X$ vanishes since $du(X)/dx=0$ from the boundary condition Eq.(3.5.56). Since the ϕ_i belong to the homogeneous space H^1, $\phi_i(0)=0$ for all i and the integrated term at $x=0$ also vanishes. The resulting equation is then solved in the usual way by approximating u and f, where the basis ϕ_i are defined on N nodes for $x=i\Delta x$ and $i=1$ to N. There are thus N unknowns for the $N+1$ nodes as the solution at $x=0$ is known.

The essential condition is satisfied by the finite element expansion for u. This is not true of the natural condition but the Galerkin equation to be solved ensures that u satisfies it. This is the classic finite element problem as discussed by Strang and Fix (1973) and Mitchell and Wait (1977).

3.5.7.2. Inhomogeneous conditions

Consider the solution to Eq.(3.5.55) but with the condition,

$$u(0) = g \tag{3.5.58}$$

This is an essential inhomogeneous condition and must therefore be satisfied by the solution in the space H^1_E.

The functions in H^1_E are not linearly independent as it is possible to write,

$$(u_1 + u_2)|_{x=0} = 2g \tag{3.5.59}$$

That is, the sum of two such functions does not satisfy Eq.(3.5.58). However, the function obtained as the difference of two such functions,

$$v = u_1 - u_2 \tag{3.5.60}$$

must belong to the homogeneous space H_0^1, such that $v(0)=0$. This homogeneous space was the solution space in the previous section.

If the same procedure as in the homogeneous case is followed, Eq.(3.5.55) is multiplied by a basis ϕ_i and integrated over the domain. The integrated term at $x=X$ vanishes as before. However, for the term at $x=0$, since nothing is known of the derivative at that point and the basis ϕ_i can be freely chosen, it is necessary to insist that $\phi_i(0)=0$ for all i as in the homogeneous case. This is an important point to understand in the finite element method. That is, whilst the solution must reside in the H_E^1 space, the test and basis functions must reside in the homogeneous space H_0^1. This is true for homogeneous and inhomogeneous boundary conditions (Strang and Fix, 1973; Conner and Brebbia, 1976). The inhomogeneous essential boundary condition does not alter the Galerkin equation for Eq.(3.5.55) in any way.

Now consider the approximation of u. As before, u is expanded in terms of the basis ϕ_i that span the homogeneous space H_0^1. To satisfy Eq.(3.5.58), the function ϕ_0 is introduced. So,

$$u \simeq g\phi_0 + \sum_{i=1}^{N} u_i\phi_i \tag{3.5.61}$$

where $\phi_0(0)=1$ at $x=0$. The function ϕ_0 need not be of the same form as the ϕ_i but is usually taken to be so. Burridge *et al.* (1986) give examples of different boundary elements for the vertical discretization of primitive equation models. However, although of the same form, this function is treated differently from all other ϕ_i. The Galerkin equation must not be minimized with respect to ϕ_0. In other words, there are $N+1$ nodes, the value at one is fixed and so N equations for the remaining nodes must be obtained.

Consider the inhomogeneous Cauchy condition,

$$\frac{du(X)}{dx} + \alpha u(X) = b \tag{3.5.62}$$

This condition need not be satisfied by the approximation to u but it does require a modification to the Galerkin equation so that it is a natural boundary condition for the problem. The procedure is exactly as above, resulting in the

Galerkin equation Eq.(3.5.57). As before, the integrated term disappears at x=0 but at x=X Eq.(3.5.62) is used so that the equation becomes,

$$\int_0^X \frac{du}{dx}\frac{d\phi_i}{dx}\,dx - \delta_{iN}(b - \alpha u_N) = \int_0^X f\phi_i\,dx \qquad \text{for } i=1 \text{ to } N \tag{3.5.63}$$

where δ_{iN} is used since $\phi_i(X)$=1 only when i=N and $u(x)$=u_N, as the piecewise linear basis is used. The condition Eq.(3.5.62) modifies the LHS and the RHS of the resulting matrix equation whereas the essential inhomogeneous condition above, Eq.(3.5.58), will only modify the RHS. However, they are only local changes, involving modifications to the equations of the boundary nodes. Nevertheless, since the equations are solved simultaneously the boundary value can affect any interior node on the first timestep. This is in contrast to the explicit finite difference method, where the effect of the boundaries propagates into the interior of the domain as the integration in time progresses. This therefore suggests that it is more important in finite element models to have accurate boundary conditions.

In practice however, it is possible to ignore the boundary conditions until after the projection matrices have been assembled. This is because of the local nature of the changes that need to be made. Thus, u would be written as,

$$u \simeq \sum_{i=0}^{N} u_i\phi_i \tag{3.5.64}$$

The Galerkin equation is then solved for i=0 to N. The essential and natural conditions can then be introduced in the same way, by modifying the appropriate equation. For example, the equation for the node i=0 without any boundary condition is,

$$(u_0 - u_1) \,/\, \Delta x = f_0 \tag{3.5.65}$$

which is replaced by,

$$u_0 = g \tag{3.5.66}$$

in the matrix equation. The natural condition is imposed in the same way. This is slightly inefficient for the solution of the system, as the known values

could be moved to the RHS. However, as pointed out by Conner and Brebbia (1976), this often presents technical difficulties as the program has to renumber rows and columns in the matrices.

3.5.8. Initial conditions

In providing the initial conditions for a finite element model given a continuous function u, the problem is to determine the best way to approximate the function and obtain the finite element expansion coefficients. For linear elements, since the basis function is interpolatory (from Eq.(3.5.8)), one way of determining the expansion coefficients is simply to take the value of u at each of the nodes. The approximation error can be shown to be $O(\Delta x^2)$ (Cullen, 1979).

A better approach is to use the Galerkin method. If u is approximated by,

$$u \simeq u' = \sum_{i=1}^{N} u_i \phi_i \tag{3.5.67}$$

then multiplying the residual $u-u'$ by the basis function and integrating over the domain gives,

$$\sum_{i=1}^{N} u_i \int_{\tau} \phi_j \phi_i \, dx = \int_{\tau} u \phi_j \, dx \qquad \text{for } j=1 \text{ to } N \tag{3.5.68}$$

The LHS leads to the mass matrix as before whilst the RHS integral has to be numerically evaluated using some form of quadrature, the most common and efficient of which is Gaussian quadrature (Krylov, 1962). Duller and Paddon (1984) give a parallel implementation of this for a finite element program on the ICL DAP.

Following Cullen (1976), the effect of this procedure can be seen by assuming that,

$$u = \mathcal{R}e\{e^{ikx}\} \tag{3.5.69}$$

and that the expansion coefficients can be written as,

$$u_i = \alpha(k)\exp\{ikx_i\} \tag{3.5.70}$$

Substituting into Eq.(3.5.68) and solving for α gives,

$$\alpha(k) = \frac{12(\ 1\ -\ \cos(k\Delta x)\ )}{k^2\Delta x^2(4\ +\ 2\cos(k\Delta x))} \tag{3.5.71}$$

This is plotted in Fig. 13 against wavelength expressed as a multiple of the grid spacing. It can be seen that projecting the data onto the nodes using a Galerkin procedure virtually eliminates all waves of length less than $2\Delta x$ i.e. those that would be misrepresented on the grid.

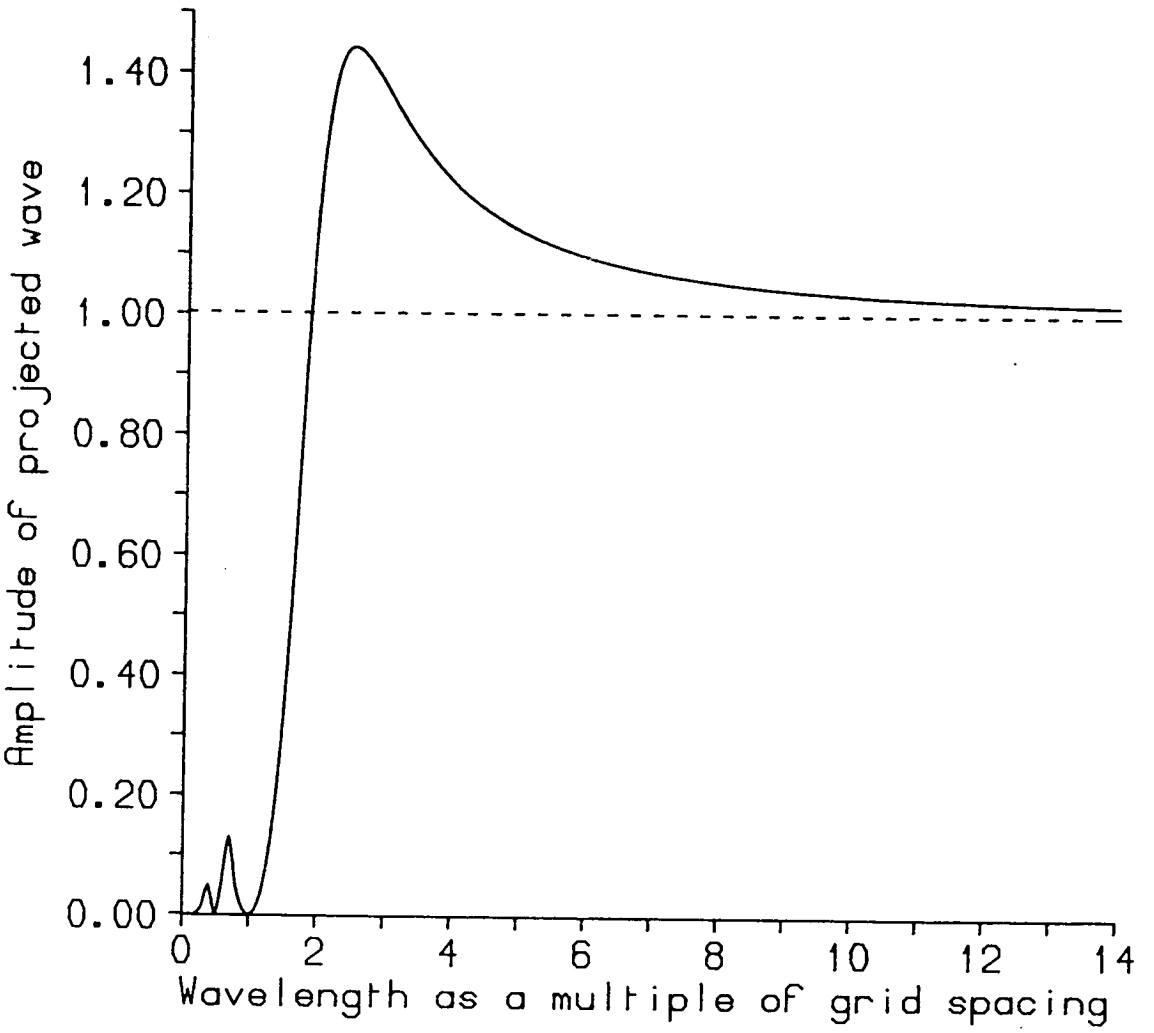


Figure 13. Response of the finite element projection operator.

CHAPTER 4

METEOROLOGICAL MODELLING ON THE ICL DAP

4.1. Introduction

This chapter reviews the previous meteorological modelling studies on the ICL DAP. The processor array has a natural analogue to the finite difference method and this formed the basis for the initial studies on using the DAP for meteorological modelling, reviewed in the next two sections. As the spectral method is preferred for global modelling, it is important to try to apply it to the DAP. The first study of this method on the DAP is described in section 3. The finite element method has already been shown by other authors (e.g. Lai and Liddell, 1987a) to be suited to the DAP for engineering problems. No meteorological studies using this technique have been conducted on the DAP to the author's knowledge.

4.2. Studies using the Meteorological Office operational suite

Hunt (1974a,b) and Reddaway (1976) studied the application of the DAP to the operational suite in use by the U.K. Meteorological Office at that time. As the machine was not yet built, these studies were estimates of the DAP execution time based on operation counts from the code listings.

Hunt (1974a) considered the implementation of the initialization program that prepared a balanced initial state for the general circulation forecast model. The input to the program was height and humidity, mapped to a rectangular three dimensional grid. The wind field, height and humidity (adjusted to be convectively stable) were output from the program. The grid had 66x50 points with 10 vertical levels. The study assumed a 64x64 DAP so that each PE processed one column.

Hunt (1974a) found that much of the original FORTRAN code could be directly expressed in parallel form, the remainder consisting of conditional operations and the calculation of boundary values. The conditional operations occurred in the adjustment for convective instability and the height field was adjusted so that the balance equation for the streamfunction was elliptic. Adjustments were made with logical masks to ensure the correct PEs were updated. Although inefficient, these calculations formed only a small part of the total number of operations.

About 65% of the arithmetic operations were spent solving Poisson equations. A parallel version of the serial solution procedure was shown to be inefficient and Hunt (1974a) discussed other methods but made no estimate of the likely improvements in performance.

The overall efficiency of the DAP program expressed as the percentage of PEs doing useful work was estimated as 50%. For 32-bit floating point precision, the estimated performance was about 6 times that of the program running on the IBM 360/195 used by the Meteorological Office at that time. Hunt (1974a) noted that some parts of the program appeared suitable for calculation using block point arithmetic. Assuming the entire program was coded using block point arithmetic, the estimated performance on the DAP would be 18 times that on the IBM. The initialization program fitted entirely into the 2Mbyte DAP store.

If a larger grid was used, it would have been processed in sections on the DAP. On the other hand, if the array size was a multiple of the grid size (e.g. a 128x128 DAP), Hunt (1974a) described how several levels could be processed in parallel for some but not all parts of the program, depending on the relationship between levels. First, in some calculations, each level was processed independently. Second, some calculations used values at adjacent levels or, last, a progressive calculation was made in which the results at one level influenced values at successive levels. The second case could be implemented in parallel if the data was mapped to the PE array to allow the communication required. The latter case could not be implemented in parallel and alternative algorithms would be needed.

Hunt (1974b) studied the implementation of the Meteorological Office 11 level general circulation model on the DAP. The grid of this model had 90 latitudes with a varying number of gridpoints around each latitude. In the rows nearest the equator there were 180 points. In other rows, the number of points was chosen to make the spacing in terms of distance approximately equal to that at the equator. However, close to the poles, the spacing was reduced to give a minimum of 16 points per row. This grid arrangement meant that each point had two neighbours in the same row, one or two in the adjacent row nearer the equator and one, two or occasionally three in the row nearer the pole.

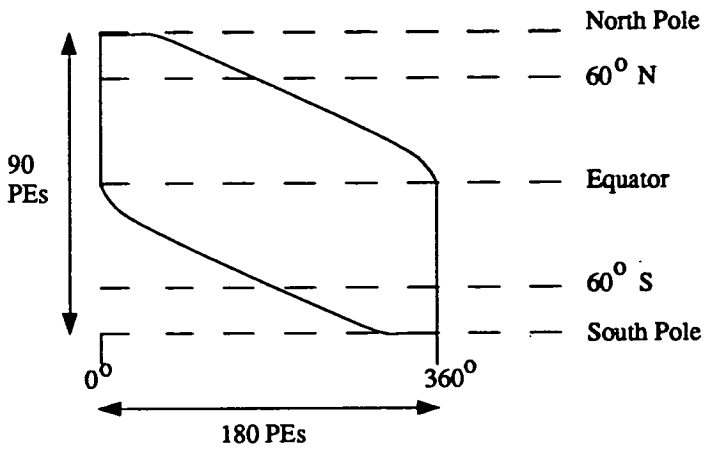
Hunt (1974b) discussed various methods of mapping this grid onto the PE array. Each PE was assigned one vertical column and each latitude row corresponded to part of a row of PEs. In the northern hemisphere, consecutive PEs were assigned starting from the west edge of the PE array. In the southern hemisphere they were assigned from the east edge, as depicted in Fig. 14a. At 60°N and 60°S, the poleward latitudes were then wrapped around onto the grid of 180x60 PEs as shown in Fig. 14b. Within this rectangle 96% of the PEs were being utilized. Assuming a 128x128 DAP, the grid could be stored by using two rows of PEs for each row of the 180x60 array. Another method considered was, starting from Fig. 14a, the grid was displaced at the equator and wrapped around in the east-west direction interchanging hemispheres, as shown in Fig. 14c. Utilization of PEs was 90% in this case.

For a number of horizontal gridpoints greater than the number of PEs, Hunt (1974b) showed it was possible for each PE to store two columns of gridpoints. One method was for each PE to store two adjacent columns from neighbouring rows, so that on successive processing steps odd and even rows were advanced alternately. Another method split the rows into two sections corresponding to east and west hemispheres. If the PE array size was a multiple of the grid size, as for the initialization program, Hunt (1974b) found that parts of the physics section of the model prevented several levels from being processed simultaneously.

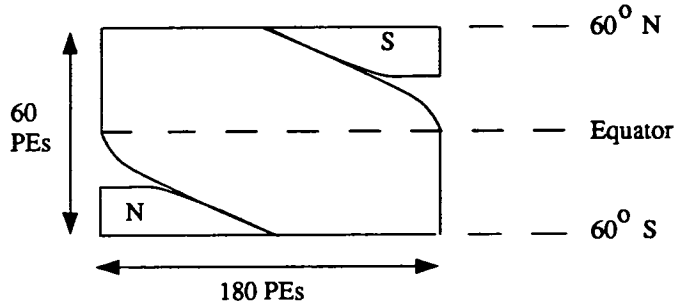
As the grid was irregular, the finite difference equations were formulated in terms of fluxes (Corby *et al*, 1977). The serial method for solving the finite difference equations was found by Hunt (1974b) to be readily expressed in parallel form. Some logical masks were used during the routing operations to ensure correct positioning of fluxes, especially across the Greenwich meridian.

At the two rows nearest the poles, the timestep was halved to ensure stability and the calculations were made twice. Hunt (1974b) discussed some methods to reduce the inevitable loss of efficiency in the use of the PE array that this introduced. A different treatment of polar latitudes, such as Fourier filtering, might be more appropriate for the DAP.

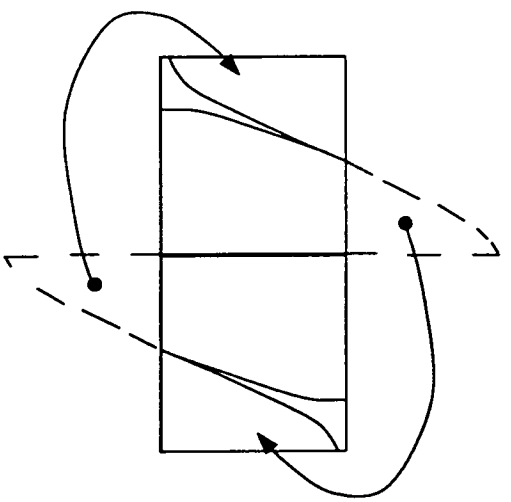
As for the initialization program, the convection subroutines introduced an inefficiency because some PEs, particularly around the polar regions, were idle



(a) Preliminary step in mapping the grid of the Meteorological Office general circulation model to the DAP.



(b) Gridpoints north and south of 60°N and S are wrapped around onto a 180x60 array of PEs.



(c) Alternatively, displace PEs in southern hemisphere in (a) and wrap around in east-west direction.

Redrawn from Hunt (1974b).

Figure 14.

during these calculations. The analysis of the code by Hunt (1974b), showed that about 16% of the total operations were conditional. Assuming an arbitrarily small number of PEs did useful work during the conditional operations, the minimum percentage of PE utilization for the model was given as 74%. An upper limit of 89% was obtained by Hunt (1974b) by assuming all conditional operations fully utilized the PE array. Both these figures assumed the same number of gridpoints as processors and therefore need to be weighted by the ratio of the number of gridpoints to processors given above. A realistic estimate of the overall efficiency might be 75%, higher if the grid was expanded to use all the PEs. Hunt (1974b) also estimated the amount of routing required during each timestep. This routing occurred solely in the dynamics calculations and represented about 2.5% of the total estimated CPU time per model step.

The possibility of using block point arithmetic or a reduced precision representation of numbers was suggested by Hunt (1974b). A method was suggested where variables at one time-level are held in 32-bit precision, whereas values at the previous step are held as differences at reduced precision. Although this reduces storage requirements, extra operations would be required for this approach so it is not obvious that the CPU time would be reduced. No other details were given of where reduced precision or block point representation could be used.

Hunt (1974b) concluded that the general circulation model was suited to the DAP, especially when the grid was chosen to match the number of processors. A 128x128 DAP was estimated to give a factor of 10 speedup over the performance of programs on the IBM 360/195.

Reddaway *et al.* (1976) conducted a study of the application of the Meteorological Office's operational suite of programs on the ICL DAP, comprised of programs for; data extraction from a database, data analysis, initialization, forecast and output. The model considered was the hemispheric octagon model on a polar stereographic projection. This model mapped onto the PE array in a straightforward manner. Reddaway *et al.* (1976) used the estimates of Hunt (1974a,b) for the initialization and forecast model to obtain an estimate of the performance for the complete suite of programs. Overall a factor of 13 over the throughput of the IBM 360/195 was estimated for a 64x64 DAP. However, this relied on the assumption that block point

representation could be used for the initialization and forecast model programs. If 32-bit floating point values were used the factor would be about 5.

4.3. A study using finite difference and spectral models

The unfinished Ph.D. thesis of Fishbourne (1980) contains a description of the application of finite difference and spectral meteorological models to the ICL DAP. As far as the author is aware, this is the only other study of meteorological modelling on the DAP. An incomplete copy of Fishbourne's thesis was obtained during this study at Edinburgh University and the main findings of his work are presented here.

4.3.1. Mapping grids to the DAP

Fishbourne (1980) studied the problem of mapping global finite difference grids onto the processor array. He considered the equal-area type of grid, as used by the Meteorological Office and described in the study of Hunt (1974b), unsuitable for a DAP implementation. This is because of the unused PEs and the nonuniform nature of the finite difference schemes, resulting in a loss of efficiency. Fishbourne (1980) favoured the latitude-longitude grid as it maps directly to the DAP and the finite difference equations take the same form throughout the region, making them well suited to parallel processing. The only drawback is the smoothing or filtering required near the poles for stability, which must not present a large overhead for efficiency reasons.

4.3.2. Finite difference models

Fishbourne (1980) developed several finite difference global models for the 64x64 DAP at Queen Mary College in London, two shallow-water equation models and a dry multi-level primitive equation model. The first shallow-water model was based on the description of the general circulation model of Arakawa and Lamb (1977), which used the equations in flux form. The second shallow-water model was based on that of Sadourney (1975a) written in spherical coordinates. This was similar to the first model but with the equations in advective form and with a different treatment of the equations at the poles.

The models used a regular latitude-longitude grid with 64 latitudes and 64

points around each latitude. The variables were held on an Arakawa 'C' grid. Points around a latitude were mapped along rows of the PE array and the meridians along columns. Height was defined at the poles but not velocity. The distribution of variables on the grid and the PE array is illustrated in Fig. 15. Since the wind components were not defined at the poles there were some unused elements in these matrices. There was also a redundancy in the use of processors to store the value of the height at the poles. However, these slight inefficiencies in the use of the PE array allowed the full parallelism of the finite difference calculations to be exploited.

To facilitate the difference calculations in the north-south direction near the poles, planar conditions for the respective edges of the PE array were used. This removed the need for any masks and allowed the polar operations to be done in parallel with those on the rest of the grid. Cyclic boundary conditions were used in the east-west direction to remove the need for any special code to compute the values at these edges.

The use of a regular latitude-longitude grid in both models ensured that the form of the equations was the same at each gridpoint, except at the polar gridpoints. Fishbourne (1980) noted there were two types of calculations for these. For the first type, the equations formed a subset of those required for the rest of the grid. Logical masks were used to prevent writing of the unrequired results for the polar points, adding a negligible overhead, so that all points were computed simultaneously. In the second type, the equations at the poles and those for the rest of the grid did not share any common operations e.g. the continuity equation takes a completely different form at the poles. As latitudes were stored on rows of the PE array, this type were computed in vector mode, which was more efficient than masked matrix operations.

Fishbourne (1980) applied a Fourier filter to latitudes greater than 45°N and 45°S. This meant filtering was applied to three variables with 32 rows each. Fishbourne (1980) chose to implement this filtering using a complex FFT on each row of the PE array. Two of the variables were used as real and imaginary coefficients for the FFT for the 16 northern and 16 southern rows, whilst the third variable was routed from these rows to the 32 unused rows about the equator for the FFT and then routed back afterwards. The CPU time for this routing was small compared to the time for the transform. The

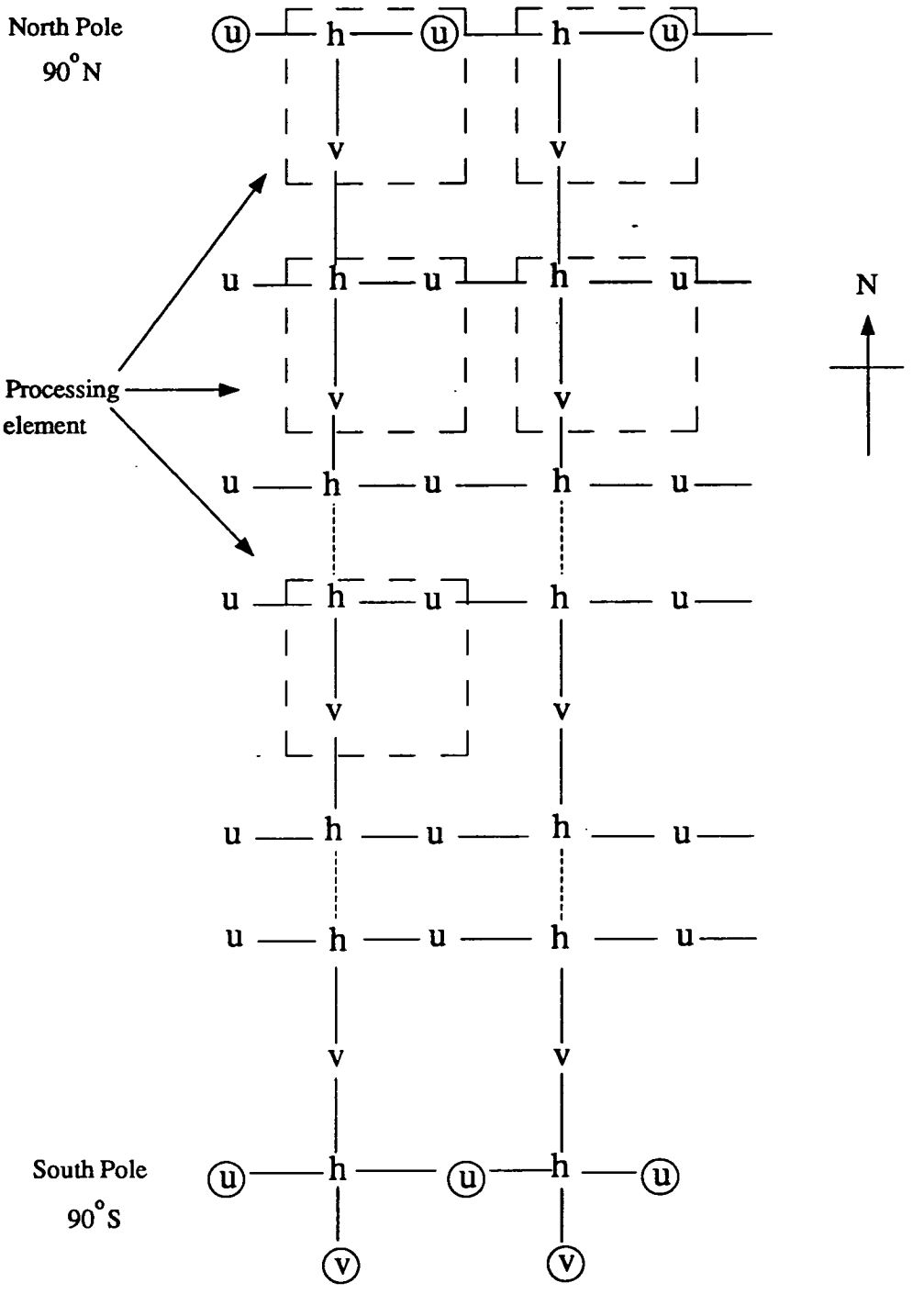


Figure 15. Arrangement of variables for the global shallow-water model of Fishbourne (1980). Values contained within a processing element are shown for some processors. The circled values of the velocities at the poles are not used and are set to zero.

filtering therefore made full use of the PE array, except that the imaginary coefficients for the 32 rows about the equator was zero as there were only 3 variables.

For a grid larger than the DAP array, Fishbourne (1980) favoured the crinkled mapping over the sheet mapping for several reasons. It allows the cyclic boundary conditions in the east-west direction to be used. As some points are held in the same PE, the routing for difference or averaging calculations is reduced. However, the amount of code has to be increased as, for the 128x64 grid considered by Fishbourne (1980), the relative position of the east-west neighbours in the PE memories is different between alternate gridpoints.

Fishbourne (1980) measured the DAP CPU time per timestep of the flux model to be 62msecs and the advective model to be 48msecs. The latter was faster because of the simpler finite difference calculations. Fishbourne (1980) does not give the model's performance rates but from the operation counts presented in his thesis, these can be calculated as 11.5Mflops and 10.2Mflops for the flux and advective models respectively. These are overestimates as Fishbourne (1980) does not give the percentage of PEs performing useful work during the calculations.

The Fourier filtering accounted for about 46% and 58% of the CPU time for the flux and advective models respectively. This was not a reflection of the inefficiency of the FFT algorithm, but resulted from the ratio of the number of arithmetic operations in the FFT to that in the rest of the model. This large overhead led Fishbourne (1980) to speculate that a nonuniform grid (such as that used in Hunt, 1974b) might give a better performance, although the inevitable mapping problems and extra routing required for the finite differences might offset any advantage. Fishbourne (1980) noted that the time for the FFT could be decreased by about 10% by leaving the transformed data in bit reversed order, since this data is then transformed back to gridpoint space directly after the amplitudes are modified. Fishbourne (1980) also pointed out that for a multi-level model, the relative cost of the filtering would be much less.

About 10% of the total CPU time per step for both models was spent computing the operations specific to the poles in vector mode. Fishbourne

(1980) regarded this as a small overhead. A timestep of 5 minutes was used for both models.

The shallow-water model in flux form was extended by Fishbourne (1980) to a dry, multi-level sigma coordinate model. The same 64x64 latitude-longitude Arakawa 'C' grid was used. This increased the number of potential processes in the spatial dimensions to a maximum of NLEVx4096, where NLEV is the number of levels. This number of processes was not possible throughout the whole procedure, however, as the calculations at the top and bottom levels differed from those at other levels. However they generally formed a subset and masked operations allowed all the levels to be processed simultaneously. In addition to the spatial parallelism, the temperature and humidity (not used by Fishbourne) calculations are largely SIMD, offering a factor of two in the number of available processes. The parallelism available in a SIMD sense therefore varied throughout the processing steps but was greater than could be exploited by the DAP.

The model was implemented on the DAP by processing the layers sequentially; the most natural way to deal with more potential processes than available processors. The number of vertical levels was varied from 3 to 8 to compare the performance of the model. The Fourier filtering was applied to latitudes greater than 60°N and 60°S, ten rows in each hemisphere, so that a complex transform was used to filter 6 variables. The full PE array was used for filtering by routing data from the polar latitudes in place of the unfiltered rows, as for the shallow-water models. The model required that 3NLEV+1 sets of derivatives were also filtered. The time for the Fourier transforms was reduced by removing the routing required to convert the data from bit reversed to normal ordering. Fishbourne (1980) obtained a 20% decrease in execution time for the FFTs when this was done.

In examining the performance of the model, Fishbourne (1980) showed that the execution time of those routines that depended on NLEV varied almost linearly. The proportion of the model CPU time spent executing in matrix mode was 83%, the proportion in vector mode was 7.5% and scalar mode operations accounted for 2%. The remaining proportion of the time, 7.5%, was spent routing and broadcasting data, most of this was in the FFTs. The computations in the polar regions accounted for nearly all of the time spent in vector mode. The FFTs in this model accounted for some 25% of the

execution time. This is a smaller fraction than before because of the increased finite difference calculations, an optimized FFT routine and the application of filtering to fewer latitudes.

Fishbourne (1980) found that with 3 levels one timestep required 176msecs, whilst with 8 levels the time was 449msecs. These times are 2.8 and 7.2 times the CPU time for the shallow-water model. The size of the DAP memory prevented the model from being run with greater than 8 levels. If humidity had been included, only a 6 level model would have been possible.

Fishbourne (1980) gives a performance rate of 18Mflops for the model, with 14Mflops for the finite difference calculations only, excluding Fourier filtering, vector and scalar mode operations. These rates are greater than those for the shallow-water models and the overall rate is 67% of the DAP peak performance for 32-bit floating point addition or 93% of the performance attained with an equal number of matrix additions and multiplications (19.3Mflops). These show that the model is well suited to the DAP.

This model's storage requirements and performance could have been improved. For example, Fishbourne (1980) noted that some repeated calculations could be avoided by using temporary storage. He did not discuss the use of lower precision or block point arithmetic.

4.3.3. Spectral model

To compare with the finite difference models, Fishbourne (1980) implemented the global spectral shallow-water model described by Hoskins (1973) on the DAP. Only a brief review of the main points of this work is given here as more detailed comments are made in the next chapter. The DAP FORTRAN code for Fishbourne's spectral model was made available to the author at Edinburgh University but did not compile and was not run on the Edinburgh University DAPs.

Fishbourne (1980) chose a triangular truncation at wavenumber 42 for the model. In analysing the potential parallelism available, he commented that this varied significantly during each timestep, more so than for the gridpoint model. The available parallelism of each variable varied between gridpoint, Fourier and spectral space. Additional parallelism existed from the same operations on different variables. However, at some stages, the total number

of available processes was less than the available processors. For example, in calculating the spectral coefficients of the wind components, the procedure was parallel in all the real and imaginary components of the two fields, giving a total of $2(M+1)(M+2)$ processes where M is the truncation wavenumber. With M=42, this is less than 4096, the number of available processors. However, at other points, such as the spectral transforms, there were more potential processes than processors. Careful consideration of the data layout on the PE array was thus required for an efficient implementation.

The storage requirements of the Legendre polynomials and their derivatives was recognized as considerable by Fishbourne (1980), who chose to pack the polynomials and compute the derivatives every timestep. This resulted in a significant overhead to the Legendre transforms. More details are given in the next chapter.

The model was found to take 1.38msecs per timestep. When compared with the gridpoint shallow-water models on an equivalent grid, this is about a factor of 10 greater. This is also 3 times the cost of the 8 layer primitive equation model on the 64x64 grid. This implementation by Fishbourne (1980) is clearly not competitive.

The spectral transforms accounted for 99.2% of the CPU time with the Legendre transforms alone accounting for 82.7%. The routing was found to account for 11.6% of the CPU time per timestep. Fishbourne (1980) does not give a performance figure in Mflops for this model.

The storage requirements of the model were such that Fishbourne (1980) calculated that a maximum of 3 layers would be possible for a 3 dimensional primitive equation model, without the need for external storage. Of the memory used by the program, 35% was for the storage of the Legendre polynomial values.

4.4. Arithmetic precision and block point arithmetic

Although the execution time of programs on the DAP can be decreased by using lower precision and block point arithmetic, none of the above references studied the effect of these changes on the CPU time and on the model results. It must be demonstrated that lower precision and block point arithmetic do not adversely affect the meteorological results for any model. This would

require a detailed study and would probably be model dependent. Hence, such a study was not done for the models described in the following chapters. However, this section reviews some previous work done on the effect of precision on model results.

4.4.1. Precision requirements for meteorological modelling

The experiments of Williamson and Washington (1974) used the NCAR global circulation model (Kasahara and Washington, 1971) on a CDC 6600 computer. They examined the importance of precision in short term forecasts and climate simulations using 48, 24 and 21-bit mantissa arithmetic.

They used a 48-bit forecast to derive the global root-mean-square (rms) errors in wind and temperature for the 24 and 21-bit forecasts and found that there was a rapid error growth in the first 12 hours of approximately one order of magnitude every hour. After a day, the growth rate decreased and became comparable to the growth rate from observational errors. They concluded that since the rapid error growth dominated the accumulation of round-off error and typical observational errors are greater than round-off errors, lower precision arithmetic did not significantly affect the result for short range forecasts. However, they noted that the rapid error growth was caused by the latent heating term as, when this was set to zero, the rapid growth did not occur.

For long-term forecasts over 80 days they compared the results of integrations using 48 and 24-bit mantissa arithmetic. Again, although they found minor differences, they concluded that 24-bit mantissa arithmetic does not significantly change the results as there seemed to be no tendency for round-off error to dominate.

The reply of Kurihara and Tuleya (1974) to the work of Williamson and Washington (1974) described differences in the results of a hurricane simulation model when run with 27-bit mantissa arithmetic on a UNIVAC 1108 computer and 24-bit mantissa arithmetic on an IBM 360 machine. With the higher precision the conservation of mass was perfect, however, with the lower precision a small but systematic decrease was noted. They also found that the heat budget of the model became inconsistent as more energy was lost through round-off error than added to the system.

They made several recommendations to correct the problems associated with low precision. In particular, the use of double precision with a 'rounding up or down' formula when a small term, comparable to or slightly larger than the round-off error, is added to a large term. Also the moist processes should be calculated in double precision.

Searle and Davies (1975) ran a four level model on an Atlas computer (40-bit mantissa) and on an IBM 360, in single (24-bit mantissa) and double precision (56-bit mantissa), to compare the effect of mantissa length. Using a timestep of 30 minutes they found that after 30 days, significant differences occurred in the eddy kinetic energy between the two runs on the IBM machine. The run on the Atlas machine showed noticeable differences to both runs on the IBM machine after 15 days. They suggested that tiny differences in the initial eddies became significant when the model eddies were amplified to near peak values.

The work of Baede *et al.* (1976) examined the effect of precision in more detail. They described two experiments; one using the adiabatic baroclinic spectral model of Hoskins and Simmons (1975), the other using the GFDL general circulation gridpoint model described by Miyakoda (1973). Both models were run on an IBM 360, in single and double precision and a 48-bit mantissa CDC 6600.

The spectral model was run for 8 days with a triangular truncation at wave number 21, for timesteps of 30 and 90 minutes, using a semi-implicit time integration scheme. Comparing the amplitudes and phases of the dependent variables in all runs they found no differences when the values were printed to 4 significant figures. Mass was formally conserved in the model and when the change in the total mass was studied, it was almost identical between runs. This agreement showed that time truncation errors rather than round-off errors dominated the nonconservation of mass, although for Kurihara and Tuleya (1974) round-off error seemed to be the cause of the problem.

In studying the energy conservation they found no difference between the CDC run and the double precision IBM run. However, the single precision run on the IBM showed a large deviation of several orders of magnitude from the previously calculated values. On closer examination they found that this large change could be attributed to a change in a single bit. Baede *et al.* (1976)

concluded that 24 bit mantissa arithmetic was sufficient for dynamics purposes as at least four figure accuracy could be obtained during an 8 day integration.

For their second experiment, the GFDL model was integrated over 10 days. Most of the dynamics and physics, apart from the moist processes, was carried out in single precision, although at many points in the GFDL code double precision arithmetic was used following the recommendations of Kurihara and Tuleya (1974). In running the model on the IBM computer, the original model was used with the moist processes in double precision, but on the CDC 6600 these processes were calculated using single precision (48-bit mantissa).

The results of Baede *et al.* (1976) paralleled those of Williamson and Washington (1974), as the global vertically integrated rms errors of wind and temperature showed the same initial rapid error growth. However, they also found that the errors were much greater over the tropics and equatorial regions. They concluded that the wind and temperature discrepancies originated in the equatorial regions and propagated into the mid-latitudes. The errors also showed a pronounced vertical structure.

The initial growth of discrepancies was attributed to the moist convective adjustment (MCA) scheme as, after only a few hours, local discrepancies of several degrees were observed in the tropics, at first near the ground but later in the mid-troposphere. This agreed with the conclusion of Williamson and Washington (1974). On further investigation they found that it was not the calculation of the MCA that was strongly computer dependent but instead the process that determined whether MCA occurred. However, as the decision making was performed in single precision on the CDC and double precision on the IBM computer, the length of the mantissas were comparable in both runs. Therefore Baede *et al.* (1976) suggested that compiler differences played a role. They concluded by stating that the precise nature of the problem was not clear but the MCA part of the code required double precision arithmetic and very careful coding.

From this review, it can be concluded that the dynamics calculations for the models presented in the following chapters can be coded using 32-bit precision. It is not clear, however, if the word length can be reduced still

further. The limit is the numerical inaccuracy introduced by the spatial and temporal truncation errors, as there should be enough precision to avoid round-off error (and its growth) exceeding the truncation error. The error in observations could also be used as a limit. As model resolutions increase and timesteps decrease correspondingly (ignoring advances in time integration techniques), the differences or increments in the model equations for each timestep become smaller. Therefore, the future trend might be for an increase in the arithmetic precision used for meteorological models.

For spectral models, the effect of the arithmetic precision on the Legendre transforms would be a concern. At high wavenumbers and close to the poles, the Legendre polynomials tend to zero. Therefore, high precision might be needed to allow for these small contributions. Alternatively, an algorithm that starts the summation from the high wavenumbers to accumulate these small contributions first might be suitable for use with a lower precision.

For the physics part of the models, it could be argued on the one hand that the required precision should be based on the accuracy of the approximations made in each parametrization scheme. On the other hand, the conditional part of the convective parametrization would seem to require as large a word length as possible. However, the reviewed studies may not be relevant to current convective parametrization schemes.

It would seem, therefore, that there is scope for employing reduced precision in meteorological models, although not for all parts of the program. As mentioned in chapter 2, the CPU time for calculations depends linearly on the precision.

4.4.2. Block point arithmetic

The use of block point arithmetic instead of floating point arithmetic for meteorological applications was suggested by Hunt (1974a,b). In block point arithmetic, a common exponent is held for each value on the PE array; hence it is best suited to fields without a large variation in their maximum and minimum values (e.g. water vapour).

Block point arithmetic is not available in DAP FORTRAN. However, it can be simulated using integer arithmetic. Tests using a one-dimensional linear advection equation model showed that integer arithmetic required

approximately a third less CPU time than the equivalent floating point precision.

4.5. Discussion

The previous studies of finite difference models reviewed in this chapter showed them to be suited to the DAP as they are efficient with low overheads (routing, nonmatrix mode operations). The appropriate choice of grid for global models on the DAP is not entirely clear. For a regular latitude-longitude grid, the overhead caused by the need for FFTs may offset the added complexity of finite difference operations on an equal-area type grid and the inefficiency caused by mapping this grid to the DAP. For example, Hunt (1974b) estimated the routing overhead to be 2.5%, whereas Fishbourne (1980) measured it as 7.5%.

In comparison, the spectral model of Fishbourne (1980) was expensive and had greater storage requirements. Any improvement to the spectral model would need to be made to the Legendre transforms since these accounted for 83% of the processing time. The following chapter describes improved parallel algorithms for these transforms.

CHAPTER 5

PARALLEL LEGENDRE TRANSFORM ALGORITHMS

5.1. Introduction

In this chapter, parallel algorithms for the inverse and direct Legendre transforms are developed prior to the implementation of a spectral shallow-water model on the DAP described in the next chapter. These transforms accounted for 83% of the CPU time of the spectral model of Fishbourne (1980). Efficient algorithms are therefore essential for an efficient implementation of the model. Furthermore, the Legendre transforms have to use the nonrectangular data structure of the spectral coefficients which is likely to inhibit their efficiency. To simplify the derivation of the algorithms, transforms that do not use the symmetry property Eq.(3.4.30) are first derived.

5.2. Data mapping

Before the algorithms are developed, methods for mapping the spectral data on the DAP must be considered, as the algorithm is dependent on the data storage format. The optimum algorithm will be the one which takes the least time to execute. It need not have the least storage requirements although this is also an important issue. Methods for mapping the two most commonly used truncations, triangular and rhomboidal, onto the DAP array are described.

5.2.1. Real spectral coefficients

The spectral coefficients require the least amount of space of the three representations of a variable (gridpoint, Fourier and spectral). For a single level model, at resolutions practical for the DAP, the space used by the spectral coefficients is not critical. However, it should be noted that in spectral models, the spectral coefficients must be held complete in store. For high resolution, multi-level spectral models (e.g. Baede *et al*, 1979) where latitude rows can only be processed and held in memory one at a time, the space used by the spectral coefficients is therefore important.

The earliest spectral models used a rhomboidal truncation and the spectral coefficients could be conveniently stored in two dimensional arrays since for each m, the number of coefficients over all n is constant. When triangular

truncation became more popular, one dimensional arrays or vectors were used to hold the spectral coefficients.

The spectral coefficients were traditionally stored column-wise in these one dimensional arrays but Baede *et al.* (1979) found this arrangement inhibited vectorization of the Legendre transforms. This problem was overcome by storing the coefficients in a diagonal-wise manner (Fig. 16). A similar approach could be used for storage of spectral coefficients on the DAP. Considering only the real parts such a mapping would be,

$$(m, n) \rightarrow \{ m + 1, n - m + 1 \} \tag{5.2.1}$$

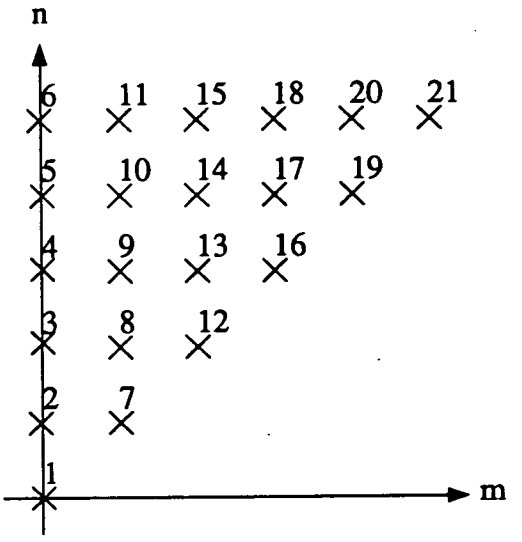
using the notation introduced in chapter 2. Diagonals are stored down columns of the DAP matrix.

If the DAP processor array is used in 'long-vector' format, where columns of the array are assumed to be concatenated (ICL, 1979), the spectral coefficients can be stored in the DAP analogous to the use of one dimensional arrays on serial or vector machines using either column-wise or diagonal-wise arrangements. This makes efficient use of storage. For a triangular truncation M, the number of real and imaginary spectral coefficients to be stored for each variable is given by,

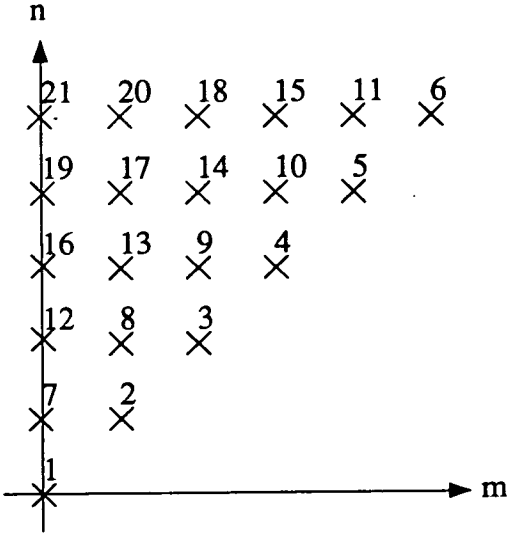
$$N_T = (M+1)(M+2) \tag{5.2.2}$$

For example, for M=42, two variables could be held completely in one DAP matrix, each occupying 46%. An equivalent rhomboidal truncation would have the same number of degrees of freedom and therefore occupy the same amount of space.

This mapping generally results in a column of the processor array containing coefficients for several values of m (column-wise storage) or diagonals (diagonal-wise storage) for both rhomboidal and triangular truncations. This means that the $(m,n)^{th}$ coefficient will be stored at different locations on the PE array for different resolutions, implying that the transform algorithms will apply only to a particular resolution. They will also be inefficient as they involve summations over μ and n and the required coefficients may exist on different rows or columns of the processor array. Despite the storage efficiency of this mapping, it would seem unsuitable for



(a) Column-wise storage for triangular truncation.



(b) Diagonal-wise storage for triangular truncation.

Figure 16. The number of each spectral coefficient indicates its storage position in a one-dimensional array.

use with the transforms.

Perhaps the most obvious mapping is to store the spectral coefficients as they appear in spectral space. Spectral coefficients triangularly truncated could fit into the upper or lower triangular part of the PE array (assuming for now a sufficiently low resolution). This mapping could be written as,

$$(m, n) \rightarrow \{ m + 1, n + 1 \} \tag{5.2.3}$$

Unlike the long-vector format, wavenumber m increases solely with the row index and n increases with the column index. This has the advantage that indexing the array is straightforward. Increasing the resolution involves simply adding another column without disturbing the existing spectral coefficients.

For rhomboidal truncation, the coefficients would be more efficiently stored if transformed slightly from their representation in spectral space. This mapping is identical to the one that might be used for serial or vector machines and is given by Eq.(5.2.1). Although indexing is slightly more complex than for triangular truncation, increases in resolution would be similarly straightforward.

Two methods exist for storing arrays larger than the DAP processor array. These are the sheet and crinkle methods described in chapter 2. Either can be used in mapping the spectral coefficients for a high resolution.

For the crinkled mapping, storing the coefficients of the summation index (total wavenumber or latitude) in a single processor or among neighbouring processors would give efficient transforms. For a sheet mapping, the spectral coefficients could be divided to make full use of the available space, for example by dividing the coefficients in half about a specified zonal wavenumber.

5.2.2. Imaginary spectral coefficients

So far only the storage arrangement of the real part has been considered. Perhaps the simplest approach for storing the imaginary part is to use the same mapping as the real part, requiring the use of a separate matrix. However, this is inefficient; half or less of the matrix would hold data. The advantage would be that selection of coefficients for the Legendre transforms

is straightforward as the mapping of the real and imaginary coefficients is the same. The spectral model of Fishbourne (1980) used this approach.

By storing the imaginary coefficients in the same matrix as the real coefficients not only are the storage requirements for the spectral coefficients halved, but so is the time taken for any computation in spectral space. While this is expected to be a small percentage of the total CPU time per timestep, it would be more beneficial with the use of a semi-implicit time scheme.

The disadvantage of storing the imaginary coefficients in the same matrix is that routing operations will be needed to reformat them for the Legendre transforms. To see how costly this would be, assume that the mapping used for the imaginary part is a transpose operation of the real part. Using the time of 216μsecs given in chapter 2 for the TRAN function, the overhead per timestep per variable will be 0.432msecs, assuming TRAN is called once for each of the inverse and direct Legendre transforms. By comparison, the gain from the reduction of operations in spectral space (diffusion, time differencing and filtering) will be approximately 1.85msecs. It is therefore advantageous in CPU time and storage space to store real and imaginary coefficients in the same matrix, for resolutions at which this is possible.

To minimize the routing to reformat the imaginary coefficients, some of the DAP FORTRAN intrinsic functions can be used. One possible mapping of the imaginary parts uses the REVC and REVR functions, which reverse the columns and rows of a matrix respectively (Eq.(2.6.15)). Applying these operations to the mapping for the real coefficients of,

$$\text{Real: } (m, n) \rightarrow \{ m + 1, n + 2 \} \tag{5.2.4}$$

which is obtained from Eq.(5.2.3) with the leading diagonal left free, gives a mapping for the imaginary coefficients of,

$$\text{Imag: } (m, n) \rightarrow \{ 64 - m, 63 - n \} \tag{5.2.5}$$

Another possibility would be to transpose the real coefficients using the TRAN function (Eq.(2.6.15)), to give an imaginary coefficients' mapping of,

Imag: (m, n) → { $n + 2$, $m + 1$ } (5.2.6)

The shift functions together with REVC and REVR can be used to obtain further possibilities. Suppose each column of the real coefficients is shifted n+1 places north cyclically and the rows reversed. The resulting mapping would be,

Imag: (m, n) → { $64 + m - n$, $63 - n$ } (5.2.7)

It occupies the same array space as Eq.(5.2.5) but the m and n axes are orientated differently. Alternatively, the rows of the real coefficients could be shifted and then reversed. This cannot be applied to the rhomboidal truncation. The mappings, Eq.(5.2.5), Eq.(5.2.6) and Eq.(5.2.7) are shown schematically in Fig. 17.

The orientation of the axes of the imaginary coefficients is different in each mapping and to the real coefficients. This may be influential on the efficiency of the Legendre transforms in the way the coefficients are selected from the matrix. Although the functions used above have similar execution times and therefore the differences in CPU time of various algorithms may not vary much, it is not clear if this is the case for the complete model. Therefore, several versions of the Legendre transform algorithms are developed in this chapter, corresponding to each of the mappings discussed above.

5.2.3. Legendre polynomials

It is a computational advantage to store the Legendre polynomials and their derivatives. In this section, the mapping of these values within the DAP store is considered.

The polynomial values require a three dimensional array unlike the coefficients for the Fourier transform and the space required to store them is considerable. One method of storing the polynomials would be that used for the spectral coefficients, where the data for each latitude would be held in a separate matrix. The mapping expression would be,

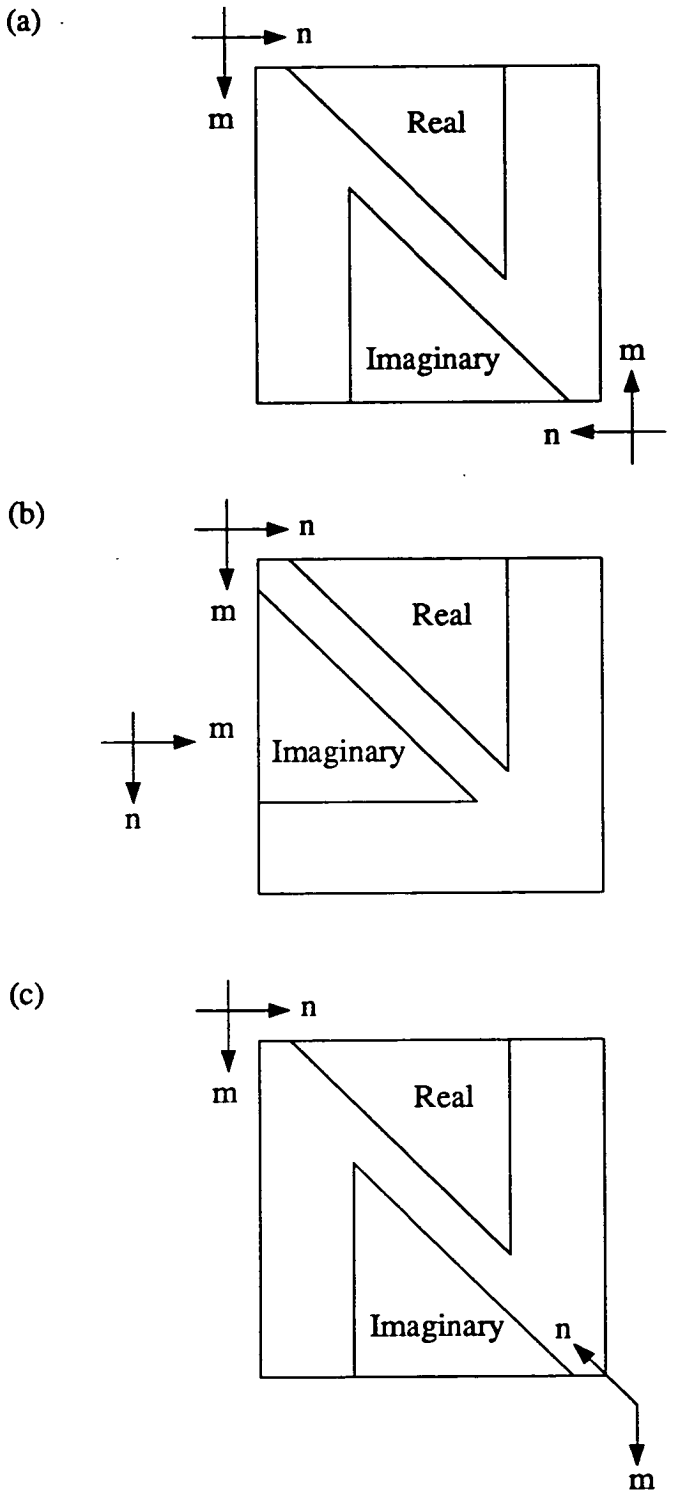


Figure 17. Illustration of possible mappings of imaginary spectral coefficients. (a) REVC and REVR, (b) TRAN, (c) SHIFT and REVC.

$$P_{m,n}: \quad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, m+1,\, n+2,\, j \,\} \tag{5.2.8}$$

where j=1 to J and J is the number of latitudes. The third index, j, on the RHS refers to the matrix number with positive increasing down in the DAP store.

This arrangement only utilizes 23% of the total space occupied by the polynomials. By also storing the derivatives in the array, 46% of the total area could be used. Whilst this is not efficient use of store, a more serious problem is that as many matrices as latitudes are required. For example, at T42, half the available DAP memory (2048 planes) would be used, assuming 32-bit precision and the symmetry property was not used.

Fishbourne (1980) overcame this problem by packing the polynomials so that each matrix held all the values for three latitudes. He obtained the derivatives by the recursion formula Eq.(3.4.33) at each timestep. This method introduces an overhead in routing the polynomials to the correct format and in calculating the derivatives. It has the advantage, however, that the values can be unpacked into a different format for the inverse and direct Legendre transforms. The number of planes required at T42 with packing is 704, with 69% of the array in use. The unpacking overheads are undesirable since the performance of the model will depend strongly on the performance of the Legendre transforms. An alternative storage arrangement is therefore required that does not introduce any overheads and improves on the storage efficiency and reduces the number of planes required.

Since the Legendre polynomials occupy a three dimensional area of storage, it is possible to store them such that either the n or m axis lies vertically in the store. In either case, the number of matrices required is M+1, 1376 planes at T42, rather than J. Although twice as much store as Fishbourne's packing approach, it is a third less storage than when latitude is mapped down the store, with 34% of the space used to hold the polynomials. No unpacking is required and the derivatives do not have to be calculated each step, since they may also be stored in the same area increasing the use of space to 68%. If the derivatives were also stored in Fishbourne's model, the total number of planes needed would become 1408.

If m is mapped down the store, the Legendre values can be stored as,

$$P_{m,n}: \quad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, j,\, n+2,\, m+1 \,\} \tag{5.2.9}$$

The derivatives may be stored as,

$$dQ_{m,n}/d\mu: \quad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, j,\, M-n+1,\, M-m+1 \,\} \tag{5.2.10}$$

This arrangement is illustrated schematically in Fig. 18.

If n is mapped vertically down the store, one possible mapping could be,

$$P_{m,n}: \qquad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, m+1,\, j,\, n+1 \,\} \tag{5.2.11}$$

$$dQ_{m,n}/d\mu: \qquad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, 64-m,\, j,\, M+1-n \,\}$$

This is shown in Fig. 19.

Within these different approaches there are several ways in which to map the Legendre polynomials. The options are: (i) whether to map m or n up or down the DAP store, (ii) how to orientate the other axes, (iii) where to position the origin and (iv) how to map the derivatives. Much of the discussion in mapping the real and imaginary parts of the spectral coefficients is relevant to mapping the Legendre values. Mappings need to be selected in such a way as to reduce any routing in selection of rows or columns of values during the Legendre transforms. If m or n is mapped such that it increases vertically down the store a simple mapping expression results (Eq.(5.2.9) or Eq.(5.2.11)). That is, indexing matrices of polynomials involves one integer scalar addition. For the derivatives, the mapping expression is more expensive. Selection of a matrix now involves an addition and a subtraction. Thus the amount of computation involved in selecting components of the array is directly related to the complexity of the mapping expression. Therefore, by comparing mapping expressions between spectral coefficients and Legendre values it is possible to determine the amount of routing required.

The mapping expression of the derivatives could be simplified by storing them with their m axis increasing down the store whilst the n axis of the polynomials increases down the store,

$$P_{m,n}: \qquad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, m+1,\, j,\, n+1 \,\} \tag{5.2.12}$$

$$dQ_{m,n}/d\mu: \qquad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, n+2,\, j,\, m+1 \,\}$$

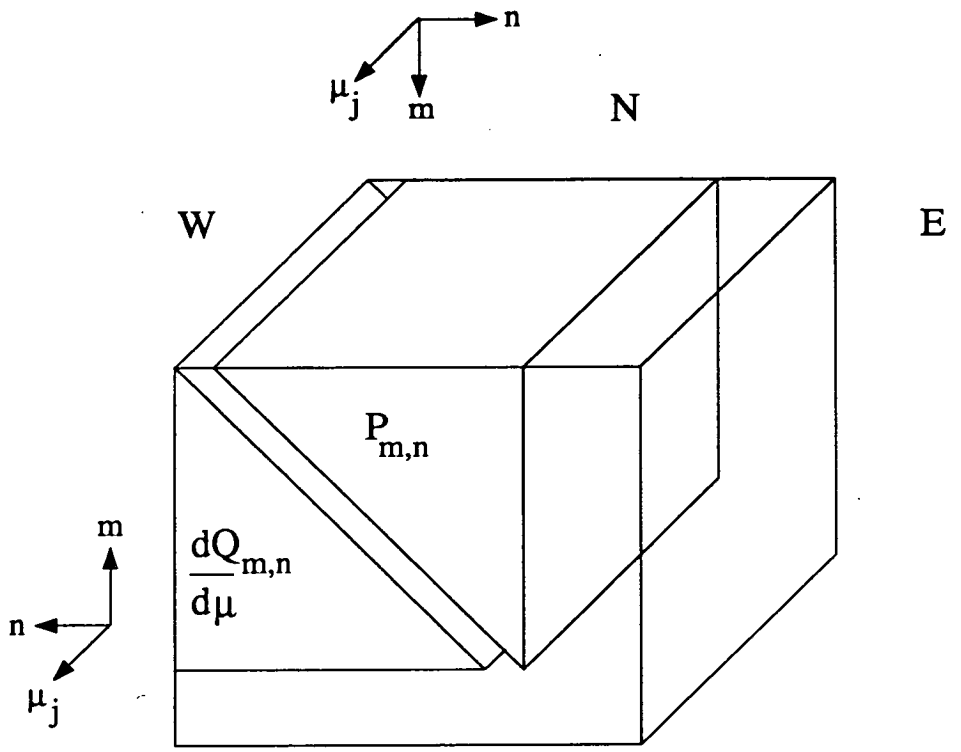


Figure 18. Schematic illustration of one possible storage format for the Legendre polynomials and their modified derivatives when the m axis of the polynomials is mapped vertically.

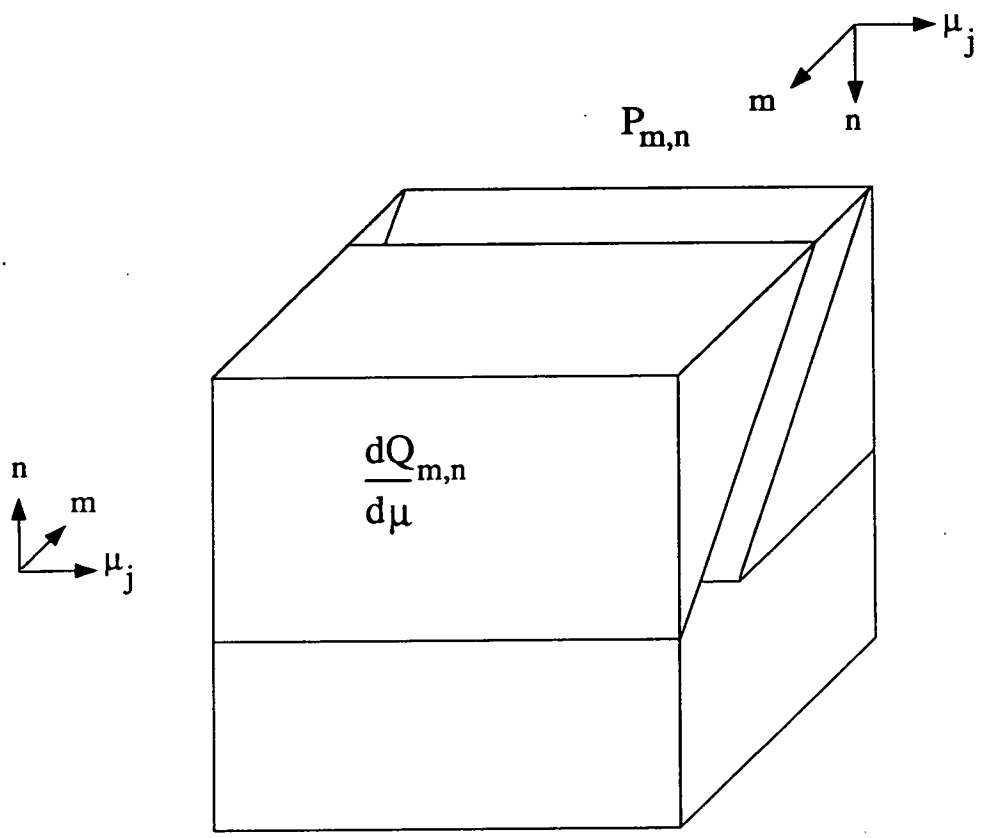


Figure 19. Schematic illustration of one possible storage format for the Legendre polynomials and their modified derivatives when the n axis of the polynomials is mapped vertically.

analogous to the transpose mapping of the imaginary spectral coefficients. It is possible to devise another mapping analogous to Eq.(5.2.7).

It is not possible to completely determine the optimum data mapping for the Legendre values until the transform algorithms are considered. However, from the above discussion the algorithms must use a mapping of the Legendre values such that wavenumbers n or m are vertical in the DAP store. It will also be beneficial to store the derivatives in the same array space as the polynomials. Another aspect is the data mapping expected by the FFTs. Whilst these are performed independently of latitude, the FFT algorithm will expect m to be mapped regularly on either rows or columns.

5.3. Inverse Legendre transform

In this section, the parallel algorithms for the inverse Legendre transform are developed. The symmetry property of the Legendre polynomials is not used. The inverse transform to be required in the spectral model of chapter 6 is given by,

$$F_m(\mu_j,t) = \sum_{n=|m|}^{M} F_{m,n}(t)P_{m,n}(\mu_j) \quad \text{for} \quad m=0 \text{ to } M \tag{5.3.1}$$

and is the simpler of the two Legendre transforms. It consists of two steps, multiplying the spectral coefficients by the Legendre polynomials and then for each m and μ summing over n.

The algorithms for the three storage arrangements of the Legendre polynomials are introduced first. Then the most efficient is examined in more detail, to be able to decide on the best precise mappings for the real, imaginary and Legendre coefficients.

5.3.1. Algorithms

In this section, only a general mapping of the spectral coefficients or Legendre polynomials is assumed. Any extra routing needed when the precise mappings are subsequently decided, is assumed to have a negligible effect on the execution time of the algorithms presented here. This is reasonable as arithmetic operations are more costly than routing operations.

5.3.1.1. Latitude vertical

Although the type of mapping exampled by Eq.(5.2.8) where μ is vertical in the DAP store was deemed to be unsatisfactory, the algorithm arising from this arrangement is described to enable comparison with later algorithms since it is essentially the method used by Fishbourne (1980). For either the real or imaginary coefficients, the algorithm is, for each latitude,

1. Multiply the spectral coefficients by the Legendre polynomials. This is done in matrix mode and is parallel in m and n.
2. Sum along rows (or columns, depending on the data mapping) of processors to give a vector result.

The real and imaginary coefficients are transformed sequentially. Some routing is required before the loop to format the imaginary coefficients correctly. Step 2 implies the m and n axes must be parallel to rows or columns and long-vector storage cannot be used. The estimated time for each pass is therefore,

$$T = 544 + 704 = 1248 \text{ µsec.} \tag{5.3.2}$$

As real and imaginary coefficients are stored in the same matrix, the opportunity to multiply the spectral coefficients by the polynomials in parallel exists, removing a multiplication on each pass, as noted by Fishbourne (1980). However, the summation along n must be done separately for the real and imaginary parts, whatever the mapping. As the polynomials and their derivatives must be stored in the same array, some routing and an addition become necessary to create the working polynomial arrays at each pass. Using the timings given in chapter 2, the total time (ignoring assignments and masking) per pass for this method is,

$$\begin{aligned} T &= 152 + 272 + 704 + r \\ &= 1128 + r \text{ µsecs} \end{aligned} \tag{5.3.3}$$

where r represents the time taken for the routing operation. For this method to be faster, $r < 120$µsecs. None of the imaginary part mappings discussed

earlier used routing operations which took this time or less to execute (see function timings given in chapter 2). Transforming the real and imaginary parts sequentially is therefore better.

An estimated execution time, based on the DAP FORTRAN code in Appendix A, is given in Table 4. Fishbourne (1980) gives an estimated CPU time of 89.28msecs and a measured CPU time of 96.63msecs for his inverse Legendre transform routine. The overhead incurred by unpacking the polynomials can be calculated to be between 8% and 17% by comparing Fishbourne's timings to that in Table 4. The higher figure is derived from the measured CPU time, the lower from the estimate given by Fishbourne. This overhead therefore makes a significant contribution to the execution time of the algorithm. However, if the available memory was such that the Legendre polynomials had to be packed, this overhead would be unavoidable.

The efficiency of an algorithm can be defined using Eq.(2.6.3). Only the operations in the loop over latitude are considered. The first step, the multiplication by the Legendre polynomials, has $m_1=\frac{1}{2}(M+1)(M+2)$ and t_1=272μsecs. For the second step, the summation, only $M+1$ rows will be in use, so $m_2=64(M+1)$. Substituting into Eq.(2.6.3) for T42 gives an efficiency of E=0.48 or 48%. It is assumed that the DAP FORTRAN summation function uses 100% of the processors. This is not strictly true as the columns contain different amounts of spectral coefficients to be summed. However, the algorithm used by this function (see chapter 2) spreads the work across more processors than those that contain data initially, so this assumption should not significantly affect the calculated efficiency. The inverse Legendre transform has $2\text{NLAT}(M+1)^2$ floating point operations, where NLAT is the number of latitudes. Using the estimated time from Table 4, an estimated performance rate, at T42, of 2.9Mflops is obtained.

5.3.1.2. *m* vertical

Suppose the Legendre polynomial mapping, Eq.(5.2.9), is used, in which the m axis lies vertically in the store. The algorithm becomes, for each zonal wavenumber,

1. Select a vector of spectral coefficients for all n. Broadcast and multiply to the Legendre polynomials.

$P_{m,n}$ mapping	Loop length	Estimated time (msecs)
Latitude vertical	64	82.45
m vertical	43	58.23
n vertical	43	41.71

Table 4.

Estimated execution times for three inverse Legendre transform algorithms derived from different mappings of the Legendre polynomials.

2. Sum along processors (rows or columns) to give a vector result over latitudes.

The algorithm is now sequential in m and parallel in n and latitude. The real and imaginary parts are treated sequentially.

Although each pass through the loop includes a broadcast of data, compared to Fishbourne's (1980) algorithm (latitude vertical) the loop length is now only 43 instead of 64. As the broadcast function is inexpensive, this method should be faster than the previous one. The estimated time in Table 4 shows the algorithm takes 29% less time than Fishbourne's. This gives an estimated performance rate of 4.1Mflops.

The algorithm still suffers from inefficient use of the array. At best, when m=0, 67% of the array is in use. At worst, when m=M, only 1.5% or 1 row or column is doing useful work. Whilst each row or column represents a latitude, the number of PEs holding useful data along the n axis varies between 1 and 43, so that, as for the latitude vertical algorithm the summation involves more processors than necessary. To compute the efficiency, the broadcast, multiplication and summation operations in each pass are considered. As the number of processors performing useful work changes with m, there are 3x43 timeslices for Eq.(2.6.3), giving an overall efficiency of 0.70. The decrease in execution time and improvement in the performance rate is consistent with an increase in the efficiency.

5.3.1.3. n vertical

The final mapping discussed for the Legendre polynomials was when the n axis was mapped vertically. The algorithm becomes, for each wavenumber n,

1. Select a vector of spectral coefficients for all m.
2. Broadcast to all latitudes and multiply by the Legendre polynomials.
3. Add product to the partial sum of products computed on previous pass.

This is parallel in latitude and m but sequential in n. The summation function

SUMC or SUMR is now replaced by a computationally cheaper matrix addition. As before, real and imaginary parts are transformed sequentially. The estimated timing in Table 4 confirms the superiority of this algorithm. The estimated performance rate is 5.7Mflops for this algorithm, almost twice the performance rate of the latitude vertical algorithm.

To compute the efficiency, the broadcast, multiplication and addition operations are considered for each wavenumber n. Using Eq.(2.6.3) gives E=0.34. This is less than the efficiencies for the m vertical and latitude vertical algorithms because the DAP FORTRAN function to sum across rows or columns, which is efficient in its use of processors, is no longer used. If all the processors were doing useful work during the loop, the performance rate would be 17Mflops. Thus 34% of the potential performance is achieved, consistent with the calculated efficiency.

The overall efficiency is only a third of the available processors. This is mainly due to the decreasing length of the selected vector as n decreases, an inefficiency common to all of the algorithms discussed. To improve this and keep a constant vector length it is necessary to pack the spectral coefficients and the Legendre polynomials. A simple method is where the spectral data are separated into two halves about $n=n'$ to give a rectangle. The loop length is halved as the coefficients for two values of n are stored in the same column. However, it is possible to show that no overall decrease in execution time occurs because of the additional work required in the loop to unpack the coefficients. A rhomboidal truncation, mapped as Eq.(5.2.1) would give a rectangular data structure without the need for packing. Minor optimizations of the code are possible e.g. some gain may be made by making use of vector operations rather than matrix ones when n is 0 or 1, since only one or two rows respectively are in use. A much greater improvement would result if this algorithm was coded in assembler, to use the spare processors during the arithmetic operations. For example, at n=31, half the PE array is unused so each processor could process 16 bits of each word, halving the computation time at this stage. At n=15, each processor would be assigned 8 bits of each word and so on.

5.3.2. Choice of mappings

Having determined the most efficient algorithm, the next step is to select the precise mappings for the real and imaginary spectral coefficients and the Legendre polynomials. The key point to the algorithm as far as the real coefficients are concerned is that a vector of coefficients over all m for each n is selected. All these coefficients must lie completely in a row or column to ensure efficient selection. Thus, a mapping using long-vector format or packing would be unsuitable. Furthermore, although not as important, to simplify the selection of a vector during the loop and avoid any unnecessary integer scalar arithmetic or initial routing, the n axis should increase with either the row or column index. For the m axis, its mapping should be such as to avoid any routing before the inverse FFT. The algorithm used for the FFT (see chapters 2 and 6) expects the m axis to increase with the row or column index and that m=0 maps to the first row. The mapping that satisfies these criteria is given by Eq.(5.2.4).

Some of the above points also apply to the Legendre polynomials. The m axis must be mapped across rows or columns as for the real coefficients. To avoid any integer scalar arithmetic during array indexing, the n axis should increase down the store with the values for n=0 as the first matrix. Although calculations are independent of latitude, it seems preferable to map latitude regularly across the columns of the DAP array. A suitable mapping of the Legendre polynomials is therefore given by Eq.(5.2.11).

The imaginary coefficients must be assigned to a work matrix before the loop, such that selection of a vector for each n gives a mapping for the m axis the same as for the polynomials. So, when each vector is selected, the m axis should be mapped as,

$$(m) \rightarrow \{ m + 1 \} \qquad (5.3.4)$$

Indexing in the loop is simpler if the n axis increases with rows or columns. Only Eq.(5.2.6), the imaginary coefficients' mapping given by use of the TRAN function achieves this. Transforming the imaginary parts would therefore involve no routing at all before the loop, merely masked assignment to a work array. Inside the loop a row vector is selected and broadcast rather than a column vector as for the real coefficients. Unfortunately, this method was not considered until after access to the DAPs at Edinburgh University was no

longer possible. The original approach routed the imaginary coefficients to the same mapping as the real coefficients.

Two other possible mappings of the imaginary coefficients were considered. The first used the DAP functions that reversed rows and columns, Eq.(5.2.5) i.e. the m and n axes are both reversed. Using the estimate of Table 4 and replacing the routing operation time by the time for the combined REVC and REVR operation gives an estimate of 41.98msecs. This code was timed to the nearest second over 10000 repetitions and executed in 52.5msecs. Thus the possible error is 0.1msecs. The overhead from high level language manipulations is therefore 25% and consistent with the 20% overhead found by Hockney and Jesshope (1981) for DO loops performing matrix operations. The time for a transform using the imaginary coefficients' mapping Eq.(5.2.6) would be 52.0msecs, by subtracting the cost of the REVC and REVR functions. The penalty for using Eq.(5.2.5) instead of Eq.(5.2.6) is therefore negligible at 1% of the CPU time.

The other mapping used the matrix shift functions instead of the reversal functions to give Eq.(5.2.7). Since the m axis increases with the row index only shifts are required to align the m=0 coefficients with the first row. Due to the uncertainty in the time for the shift operation, an estimate of the routine is not given. The measured time was 51.9msecs, again with a possible error of 0.1msec.

To conclude, it has been shown that by mapping the imaginary parts by a TRAN function no initial routing is needed. However, other possible mappings in which routing is necessary cause a negligible increase in the execution time of the transform. It seems therefore that the imaginary parts mapping can be freely chosen. However, the effect of the mapping options on the direct Legendre transform should be considered before deciding on the precise mapping. Fishbourne (1980) gives a CPU time of 96.63msecs for his inverse Legendre transform. The n vertical algorithm is therefore faster by a factor of 1.9.

5.4. Direct Legendre transform

The algorithms for the direct Legendre transform are now derived. The same approach as for the inverse transform is used. The algorithms arising from the three Legendre polynomial mappings are discussed generally and the most efficient is then examined in more detail. Any subsequently necessary routing is assumed to be negligible compared to the difference in execution times of the algorithms. The mapping that gave the most efficient algorithm for the inverse transform may not give the most efficient direct transform.

The direct Legendre transform, to be required by the model is of the form,

$$F_{m,n} = \sum_{j=1}^{J} h_j(\mu_j)\{B_m dQ_{m,n}/d\mu - iA_m P_{m,n}\} \tag{5.4.1}$$

where,

$$h_j = g_j / [2(1 - \mu_j^2)] \tag{5.4.2}$$

and,

$$\frac{dQ_{m,n}(\mu_j)}{d\mu} = (1 - \mu_j^2)\frac{dP_{m,n}(\mu_j)}{d\mu} \tag{5.4.3}$$

Since the derivatives are stored in the same array as the polynomials, the possibility exists of computing the two products and their summations in parallel. That is, writing Eq.(5.4.1) as,

$$F_{m,n} = \sum_{j=1}^{J} h_j B_m dQ_{m,n}/d\mu - \sum_{j=1}^{J} ih_j A_m P_{m,n} \tag{5.4.4}$$

algorithms should be developed that compute these two terms in parallel and then as the final step evaluate the difference to form the spectral coefficients.

5.4.1. Algorithms

5.4.1.1. Latitude vertical

First, consider the case where the polynomials are mapped with latitude increasing down the store. Since the final result is spectral coefficients, it will be beneficial to map the polynomials for each latitude exactly as the real coefficients. Likewise, the derivatives values are best mapped as the imaginary coefficients.

Using this storage arrangement the algorithm is,

1. Multiply the modified Gaussian weights to the Fourier coefficients.

2. For each latitude: Select a vector of wavenumbers to be multiplied by the polynomials and broadcast to a work matrix.

3. Select a vector of wavenumbers to be multiplied by the derivatives and broadcast (using masking) to the same work matrix. Routing will be required to position the Fourier coefficients.

4. Multiply the work matrix by the Legendre data. This is parallel in n, m and the two products. End of latitude loop.

5. Compute difference, using routing, to give spectral coefficients.

The transform of the imaginary parts is done after the real parts and is as above, except step (5) is a summation.

By treating the products in parallel, one addition and multiplication in matrix mode are avoided. The overhead in forming the Fourier matrix is slight as it involves matrix assignments.

An estimate of the time for the algorithm is given in Table 5, based on the code given in Appendix A. The number of floating point operations required for this transform is given by 4NLAT(M+1)(M+3)-(M+1)(M+2) which, for M=42, gives

$P_{m,n}$ mapping	Loop length	Estimated time (msecs)
Latitude vertical	64	97.54
m vertical	43	83.58
n vertical	43	57.78

Table 5.

Estimated execution times for three direct Legendre transform algorithms derived from different mappings of the Legendre polynomials.

an estimated performance rate of 5Mflops. The execution time is only 18% more than that of the inverse transform routine with the same Legendre data mapping. This is because the two products involving the polynomials and their derivatives are computed in parallel. If the products were computed sequentially, the extra operations would give an estimate of 153.86msecs, an increase of 58% on the parallel version.

To calculate the efficiency using Eq.(2.6.3), only the operations in the loop are considered. This gives E=0.37. This efficiency is greater than for the inverse Legendre transform, because the derivatives are involved.

5.4.1.2. m vertical

Consider the algorithm when the Legendre polynomials are mapped with m down the store. The procedure becomes,

1. Multiply the modified Gaussian weights to all the Fourier coefficients.
2. For each m: Select a vector for m over all latitudes from A_m.
3. Broadcast only to the processing elements in the work matrix that are multiplied by the polynomials.
4. Select vector for m from B_m.
5. Broadcast to the processing elements in the work matrix that are multiplied by the Legendre polynomial derivatives.
6. Multiply the work matrix by the array containing the polynomials and derivatives. This is parallel in n, μ and the products involving the polynomials and derivatives.
7. Sum along each latitude to give a vector result of the two required products. End of loop.
8. Route the products and compute difference to give the final spectral coefficients.

Like the previous algorithm, the imaginary coefficients are transformed sequentially. The amount of work within each pass of the loop has increased but the loop length has decreased. The timing estimate in Table 5 shows a decrease in time by 14% from the latitude vertical algorithm. This is almost half the decrease achieved in the inverse transform between these mappings. The estimated performance rate of this algorithm is 6Mflops.

To compute the efficiency of this algorithm, the routing, broadcast, multiplication and summation operations are taken into account. During the multiplication and addition stages, 68.8% of the processors are performing useful work. Unlike the inverse transform case, this is constant throughout the loop. The only operation in which the number of usefully active processors varies with the wavenumber m is the routing for the Fourier coefficients. Using Eq.(2.6.3) gives E=0.59. This is an increase on the last algorithm and consistent with the improved performance.

5.4.1.3. n vertical

The time for the m vertical algorithm could be improved if the broadcasting and routing during each pass could be reduced. This can be achieved if the n axis of the Legendre data is mapped vertically in the store.

Since the loop variable is n, the Fourier coefficients are effectively constants during the loop and masked assignment to a work matrix can be done outside the loop. Care must be taken, however, to ensure that the Fourier coefficients are placed correctly. At the start of the loop over n, either the polynomials or the derivatives will be mapped such that n=42 for one and n=0 for the other. As the loop index increases, the number of occupied rows decreases for one and increases for the other. Thus by assigning all the Fourier coefficients to be multiplied by the Legendre data for which n=42 at the start of the loop to the Fourier work matrix, only those Fourier coefficients multiplied by the Legendre data for which n increases down the store, need to be updated. The other set of Fourier coefficients can be overwritten.

Since a total of 44 rows or columns will be in use, the remaining 20 rows can be used to store the Fourier coefficients multiplied by the Legendre values for which n increases down the store. Thus, only after each 21 passes of the loop does the Fourier work matrix need to be updated. One masked matrix

assignment is faster than 21 vector assignments. A higher resolution would result in less unused rows or columns and therefore the Fourier work matrix would need to be updated more often. The steps of the algorithm are,

1. Multiply the modified Gaussian weights to all the Fourier coefficients.

2. Mask assign the correct Fourier coefficients valid up to n=20 to the Fourier work matrix.

3. For each n: Multiply the Fourier work matrix by the Legendre data array. This is parallel in m, μ and the two products.

4. Sum along the lines of latitude to give a vector containing the two products.

5. If n=20 or n=41, update the Fourier work matrix. End of loop.

6. Route the two products and compute the sum and difference to give the real and imaginary spectral coefficients.

As before, the real and imaginary parts are transformed sequentially. The DAP FORTRAN code is given in Appendix A.

The estimated time for this algorithm is given in Table 5. A reduction in time of 31% on the previous algorithm has been achieved, resulting from less work in the loop although the loop length is the same. The algorithm is also 40% faster than the latitude vertical algorithm. The inverse n vertical transform algorithm was 50% faster than the inverse latitude vertical algorithm. The estimated performance rate for this algorithm is 8.5Mflops.

Calculating the efficiency of each pass through the loop is straightforward. The Fourier work matrix updates at n=20 and 41 are ignored as they contribute little to the overall time. Like the previous algorithm, during the multiplication 68.8% of the array is doing useful work. The summation along latitudes takes place on the same percentage of rows or columns so the efficiency using

Eq.(2.6.3) is given by E=0.69. Therefore, nearly 70% of the processor array is doing useful work on average. It is this higher efficiency that enables the direct transform algorithm to execute in a time close to that of the inverse transform. In serial terms, the amount of computation for the direct transform is over twice that for the inverse transform. As the parallel version takes only 1.4 times the estimated time of the inverse transform, this shows that extra parallelism is available in the direct transform and it has been exploited successfully.

The direct transform algorithm of Fishbourne (1980) also uses the n sequential approach. However, he incurs an additional cost from unpacking the polynomials and computing the derivatives at each timestep. The two products are computed sequentially but the summation over latitude is done concurrently for both products. Comparing the time given by Fishbourne (1980) for his direct transform to that of the n vertical algorithm above shows the latter method to be about 3 times faster than Fishbourne's. However, the storage required by his algorithm is half that of this one. When symmetry is included in the Legendre transform algorithms, the storage requirement will be halved and an additional gain in execution time will result.

5.4.2. Choice of mappings

In this section, given the most efficient algorithm, the precise data mappings and movements are studied in detail. The relationships between the data mappings are first examined.

The best algorithm is achieved when the n axes of the Legendre polynomials and their derivatives are mapped down the DAP store. The first stage of this algorithm is the multiplication of the Fourier coefficients by the polynomial values. The Fourier coefficients are mapped as,

$$(m, \mu_j) \rightarrow \{ m + 1, j \} \qquad (5.4.5)$$

with real and imaginary values in separate matrices. The routing required to position the Fourier coefficients to be multiplied by the derivatives can be minimized if the m axis of the derivatives also increases with rows, as only shifts are required. These could be kept to a minimum if there were no free rows between the polynomials and the derivatives. However, it was shown in the last section how the 20 free rows can be used to advantage in the

algorithm. The arguments for selecting Eq.(5.2.11) as the polynomial mapping for the inverse Legendre transform also apply for the direct transform. A mapping satisfying the above criteria for the derivatives is,

$$dQ_{m,n}/d\mu: \quad (m, n, \mu_j) \rightarrow \{ 64 + m - n, j, 43 - n \} \tag{5.4.6}$$

This uses the shift functions as discussed in the section on mapping the Legendre data. Another option would be to use the mapping of Eq.(5.2.11) for the derivatives. A reversal of rows (REVR), rather than shifts, would be necessary in order to map the B_m coefficients correctly. The effect on the algorithm is discussed below.

The next stage in the algorithm is the summation. This produces a vector containing the two products. For any n, the first n+1 elements of this vector hold the polynomial product values, the last 43−n elements hold the derivative product values. Within this vector, the products will be mapped the same as the row mapping of the polynomials and their derivatives. Furthermore, if this vector is assigned to the n+1th column of a work matrix, the column mapping of the work matrix will be the vertical (matrix) mapping of the polynomials. Using the mapping of Eq.(5.4.6) as an example this could be written as,

$$P_{m,n} \text{ product:} \quad \text{SUMC}\{ m + 1, j, n + 1 \} = \{ m + 1, n + 1 \}$$

$$dQ_{m,n}/d\mu \text{ product:} \quad \text{SUMC}\{ 64+m - n, j, 43 - n \} = \{ 64+m - n, 43 - n \} \tag{5.4.7}$$

as the SUMC function sums along j. Therefore, Eq.(5.4.7) expresses the change of mapping that takes place during this stage of the algorithm.

Following the calculation of the products, they are combined to form the spectral coefficients. This involves another change of mapping. The closer the mapping Eq.(5.4.7) is to the spectral coefficients' mapping, the less routing will be necessary. For example, in Eq.(5.4.7), the Legendre polynomial product mapping needs only one shift east to be the same as the real part mapping. Likewise, the derivative product mapping is similar to the imaginary coefficients' mapping of Eq.(5.2.7), requiring an eastward shift of 20 places.

The key point here is that the mapping of the polynomials should be closely related to the mapping of the real coefficients for efficiency. Similarly, the mapping of the derivatives should be closely related to the mapping of the

imaginary parts. Routing is needed to map the derivative product as the real coefficients, and the polynomials as the imaginary coefficients, to form the real and imaginary spectral coefficients.

Having seen the relationships between the data mappings, the algorithms resulting from the different imaginary part mappings are now examined. Although varying the way they were mapped had a negligible impact on the execution time of the inverse Legendre transform, this should be verified for the direct transform. Assume that the same type of mapping is used for the imaginary coefficients and the derivatives i.e. a mapping using the shift functions, reversal of rows and columns or a matrix transpose.

With the mapping of Eq.(5.4.6), assignment to the Fourier work matrix for the B_m coefficients only requires shifts and is therefore economical. However, as n increases the row in which the m=0 coefficients reside changes. Therefore, on each pass of the loop over n, the B_m coefficients only must be shifted one place south. Masked assignment is necessary so as not to disturb the A_m coefficients. The algorithm can still make use of the 20 free rows as described in the previous section.

The routing required when combining the products can be determined by using the mapping expressions. Suppose that the real coefficients are to be formed and mapped as $\{m+1,n+2\}$. The derivative product is mapped as $\{64+m-n,43-n\}$ from Eq.(5.4.7). By knowing the change of mapping effected by the DAP matrix functions (Eq.(2.6.15)) the necessary routing operations can be determined. These are a cyclic shift of each row, the number of shifts depending on n, followed by REVC and a shift west of 20 processors.

Consider the algorithm when Eq.(5.2.11) is used as the mapping of the derivatives. Forming the Fourier work matrix becomes marginally more expensive using REVR rather than a shift of 20 places. During the loop the products are calculated in reverse order; the polynomial product starting from n=0 and increasing, the derivative product from n=42 and decreasing. From the mapping expression in Eq.(5.2.11), it can be seen that the mapping of m does not alter with n and no shift is required during the transform loop.

For this mapping, different routing operations will be required to form the spectral coefficients. Applying the summation function SUMC to the mapping of

Eq.(5.2.11) gives,

$$dQ_{m,n}/d\mu \text{ product:} \quad (\ m,\ n\ ) \rightarrow \{\ 64 - m,\ 43 - n\ \} \tag{5.4.8}$$

This has to be routed to the real coefficients' mapping as before. First, as the directions of the m and n axes in Eq.(5.4.8) are reversed, the functions REVC and REVR are applied, followed by a shift west of 20 PEs. The routing for the polynomial product to the imaginary coefficients' mapping is obtained in a similar way. Comparison of these routing operations to those described above using Eq.(5.4.6) show that these are slightly computationally cheaper.

Both this method and the previous one were coded and timed. The results are given by the first two entries in Table 6. The mapping of Eq.(5.4.6) for the derivatives is denoted by the word SHIFT because of the use of the shift functions to map the data correctly. Likewise for the imaginary spectral coefficients mapping of Eq.(5.2.7). For the mapping Eq.(5.2.11) of the derivatives and associated imaginary parts mapping of Eq.(5.2.5) the word REV is used because of the use of the REVR and REVC functions. To time the subroutines, each was called 10000 times and the CPU time obtained to the nearest second. All times therefore have a possible error of 0.1msecs.

The times in Table 6 clearly show the overhead in shifting the Fourier matrix at every loop pass. The measured times can be compared with the estimates of Table 5. The difference in CPU times resulting from changes in the routing are much smaller than the differences due to changes in the mapping of the polynomials (latitude, m or n vertical), validating the assumption made at the start of the previous section. The SHIFT algorithm is 3.5% more expensive than the REV algorithm.

Use of the TRAN function to map the imaginary coefficients gave the best inverse transform algorithm, removing all routing. As before, the derivatives should be mapped in such a way to give a product mapping that is as close as possible to the imaginary part mapping. The derivatives are separated by the 20 free rows to keep the algorithm efficient.

A derivative mapping of,

Imaginary coefficients mapping	Derivatives mapping	Time (msecs)
SHIFT	SHIFT	61.4
REV	REV	59.3
SHIFT	REV	59.7
TRANS	REV	59.7

Table 6.

The CPU times for the direct Legendre transform for different data mappings of the imaginary spectral coefficients and Legendre polynomial derivatives. Refer to text for explanation of mappings.

$$dQ_{m,n}/d\mu: \quad (m, n, \mu_j) \rightarrow \{ n + 22, j, m + 1 \} \tag{5.4.9}$$

will give a product mapping of,

$$dQ_{m,n}/d\mu \text{ product}: \quad \{ n + 22, m + 1 \} \tag{5.4.10}$$

The routing needed to transform Eq.(5.4.10) to the real part mapping is a shift north of 20 places followed by a transpose. This is the cheapest so far.

However, from Eq.(5.4.9) the derivatives have the m axis mapped down the store so that on each pass of the loop over n for the polynomials, a vector is selected from the B_m Fourier coefficients and broadcast to the area of the Fourier work matrix to be multiplied by the derivatives. This will make the algorithm more costly. It is essentially a combination of the n vertical and m vertical algorithms discussed in the previous section.

The most efficient algorithm therefore results when the mapping Eq.(5.2.11) is selected for the derivatives; other mappings introduce additional operations into the loop. In the above discussion, the routing in the final stage of the algorithm has been minimized by relating the mapping of the derivative product (and hence that of the derivatives) to that of the imaginary spectral coefficients.

Suppose that Eq.(5.2.11) is used for the derivatives but an imaginary coefficients' mapping that is not based on the reversal of rows and columns is used. Although the time for the direct transform will increase because of the extra routing, the combined times of the inverse and direct transform may decrease, because of a decrease in the inverse transform time. Using the mapping expressions as above, it is straightforward to determine the different routing operations necessary. Only the routing used in forming the real and imaginary coefficients at the end of the direct transform algorithm is affected.

Table 6 shows the actual timings of two of these combined mapping algorithms for the direct transform. The transpose mapping Eq.(5.2.6) is used for the imaginary part in one, the shifted m axis mapping Eq.(5.2.7) the other. The increase in CPU time of these methods over the case where both imaginary part and derivatives are mapped using reversal of m and n axes is seen to be small since the modifications occur outside the main loop.

The sum of the inverse and direct transform times gives 111.8msecs for

the REV mapping of the imaginary coefficients, 111.6msecs for the SHIFT mapping and 111.7msecs for the TRAN mapping based on a matrix transpose operation. For the TRAN mapping, an estimate is used, obtained by subtracting the time for the REVC and REVR functions from the measured time of the REV algorithm. These figures show that overall the fastest method for the direct transform does not give the best time, the shifted m axis mapping of the imaginary coefficients is marginally superior, a difference of 0.2% to the execution time of the REV mapping. A REV mapping of the derivatives is assumed in all cases.

To summarize, the precise mapping of the imaginary coefficients is not important to the efficiency of the transforms although the mapping of the Legendre polynomials and the derivatives is. Other aspects, such as conceptual simplicity and conciseness of code, may affect the choice of mapping.

5.5. Inclusion of symmetry

In this section, the modifications necessary to the data mappings and the parallel Legendre transform algorithms to make use of the symmetry property Eq.(3.4.30) are described. The n vertical algorithms derived in the preceding section are used, as this mapping of the Legendre data gave the best algorithms. The choice of mapping for the imaginary coefficients is arbitrary and so the REV mapping is used to be consistent with the derivatives' mapping. The amount of computation required to calculate the inverse and direct Legendre transforms can be reduced by using the property Eq.(3.4.30). The storage requirements for the Legendre polynomials and their derivatives are halved.

Assume that the latitudes are numbered from the North Pole to the equator i.e. j=1 to $J/2$, and that $-\mu_j$ represents a latitude in the southern hemisphere. Writing Eq.(5.3.1) as,

$$F_m(\mu_j) + F_m(-\mu_j) = \sum_{n=|m|}^{M} F_{m,n}(P_{m,n}(\mu_j) + P_{m,n}(-\mu_j)) \qquad (5.5.1)$$

$$F_m(\mu_j) - F_m(-\mu_j) = \sum_{n=|m|}^{M} F_{m,n}(P_{m,n}(\mu_j) - P_{m,n}(-\mu_j))$$

and using the symmetry relation Eq.(3.4.30) it can be shown that the symmetric (a_m) and antisymmetric (b_m) Fourier coefficients are computed by,

$$a_m = \sum_{n=|m|}^{M} F_{m,n} P_{m,n} \qquad \text{for } |m|+n \text{ even} \tag{5.5.2}$$

$$b_m = \sum_{n=|m|}^{M} F_{m,n} P_{m,n} \qquad \text{for } |m|+n \text{ odd}$$

for j=1 to $J/2$ only. The Fourier coefficients are then given by,

$$F_m(\mu_j) = a_m(\mu_j) + b_m(\mu_j) \tag{5.5.3}$$

$$F_m(-\mu_j) = a_m(\mu_j) - b_m(\mu_j)$$

The direct Legendre transform required by the spectral model in chapter 6 is given by Eq.(5.4.1). Using Eq.(3.4.33) and Eq.(3.4.46), it can be shown that,

$$g(\mu_j) = g(-\mu_j), \qquad h(\mu_j) = h(-\mu_j) \tag{5.5.4}$$

and,

$$\frac{dQ_{m,n}(-\mu_j)}{d\mu} = (-1)^{|m|+n+1} \frac{dQ_{m,n}(\mu_j)}{d\mu} \tag{5.5.5}$$

Rewriting Eq.(5.4.1) as,

$$F_{m,n} = \sum_{j=1}^{J/2} h_j(\mu_j)[B_m(\mu_j) \frac{dQ_{m,n}(\mu_j)}{d\mu} - iA_m(\mu_j)P_{m,n}(\mu_j)] \tag{5.5.6}$$

$$+ h_j(-\mu_j)[B_m(-\mu_j) \frac{dQ_{m,n}(-\mu_j)}{d\mu} - iA_m(-\mu_j)P_{m,n}(-\mu_j)]$$

and using Eq.(3.4.30), Eq.(5.5.4) and Eq.(5.5.5), it can be shown that the direct transform can be computed by,

$|m|+n$ even

$$b_m = B_m(\mu_j) - B_m(-\mu_j) \tag{5.5.7}$$

$$d_m = A_m(\mu_j) + A_m(-\mu_j)$$

$$F_{m,n} = \sum_{j=1}^{J/2} h_j[b_m dQ_{m,n}/d\mu - i d_m P_{m,n}] \qquad (5.5.8)$$

and,

$|m|+n$ odd

$$a_m = B_m(\mu_j) + B_m(-\mu_j) \qquad (5.5.9)$$

$$c_m = A_m(\mu_j) - A_m(-\mu_j)$$

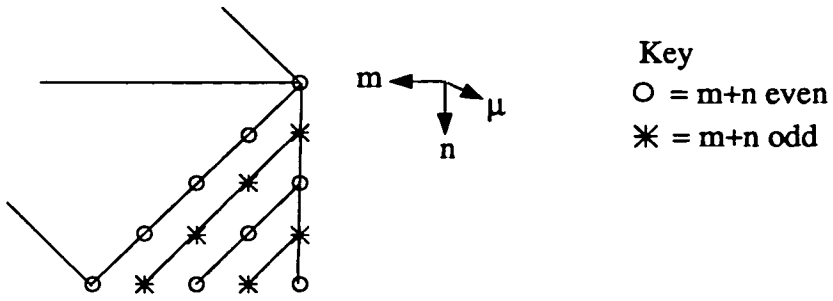
$$F_{m,n} = \sum_{j=1}^{J/2} h_j[a_m dQ_{m,n}/d\mu - i c_m P_{m,n}] \qquad (5.5.10)$$

5.5.1. Storage of spectral data

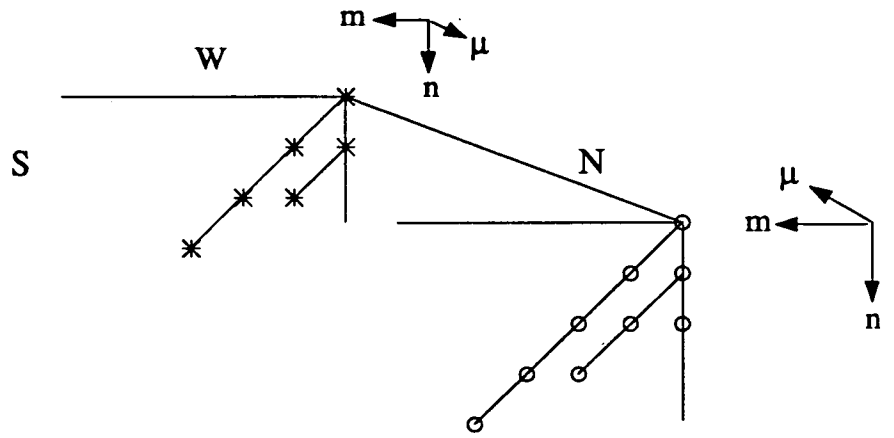
Using the symmetry property Eq.(3.4.30), the number of latitudes for which values of Legendre polynomials are required is halved. For T42 this means that half of the DAP array is then free, because latitude is mapped across columns. One possibility would be to copy the polynomials in each half and transform two variables simultaneously, but since there are an odd number of prognostic variables and this does not reduce the storage required, better use can be made of the available processors.

There are several mappings that can be formulated to make use of the available space. The first possibility would be to separate coefficients with odd and even $m+n$, or symmetric and antisymmetric values. The symmetric coefficients could be stored in one half of the DAP array, the antisymmetric coefficients in the other half.

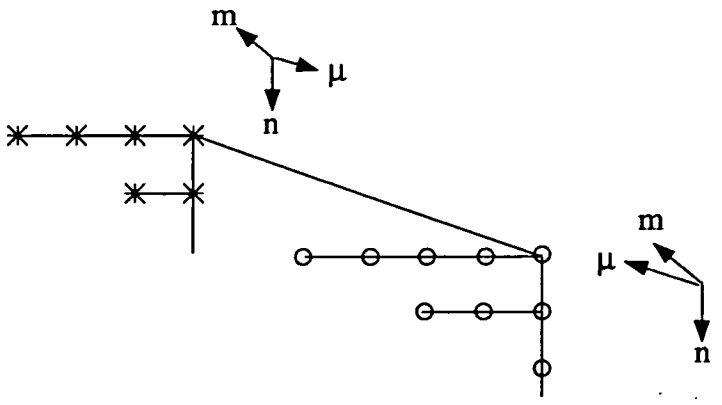
Since symmetric or antisymmetric coefficients are represented by diagonals in the (m,n) plane, the amount of storage required will not be reduced, as the depth of store depends on the length of the diagonals. However, a reduction in storage can be obtained if each vertical column is moved up by m matrices. This means that diagonals of $m+n$ odd and even are now stored on a DAP plane. Fig. 20 illustrates the steps in remapping the data



(a) The original Legendre polynomial mapping viewed from the side of the DAP store.



(b) The storage arrangement after separation of symmetric and antisymmetric values to separate halves of the DAP array.



(c) The separated coefficients are shifted vertically up the DAP store to achieve a reduction in storage. Diagonals now lie horizontally on DAP planes.

Figure 20. Illustration of the remapping of the Legendre polynomials for T4 resolution in order to use the symmetry property.

from the original nonsymmetric mapping. The mapping expression for this storage arrangement is,

$$P_{m,n}: \quad (\, m,\, n,\, \mu_j \,) \rightarrow \{\, m+1,\, j+32,\, 1+(n-m)/2 \,\} \qquad m+n \text{ even}$$

$$(\, m,\, n,\, \mu_j \,) \rightarrow \{\, m+1,\, j,\, (n-m+1)/2 \,\} \qquad m+n \text{ odd} \qquad (5.5.11)$$

Only $(M+2)/2$ or 22 matrices (704 planes) at T42 are now required to store the coefficients. For odd resolutions, the storage requirement is $(M+1)/2$ matrices.

As for the nonsymmetric storage mappings, the derivatives can be stored in the same array. From Eq.(5.5.5), the symmetric and antisymmetric coefficients of the derivatives can also be separated. A mapping expression similar to Eq.(5.5.11) results except that the n axis increases up through the DAP store.

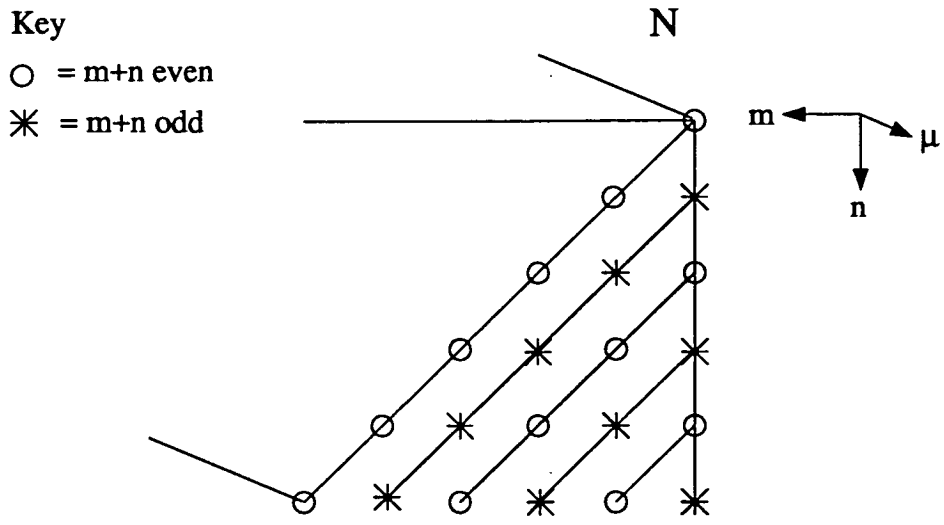
It was shown in the previous section that for efficiency, the mappings of the spectral coefficients should be related to the mappings of the Legendre polynomials and their derivatives. The inverse and direct Legendre transforms use and return a vector of spectral coefficients. In both cases, the mapping of data in this vector should match the mapping of Legendre values. This implies that diagonals should be stored in columns i.e.

$$F_{m,n} \text{ real}: \quad (\, m,\, n \,) \rightarrow \{\, m+1,\, n-m+1 \,\} \qquad (5.5.12)$$

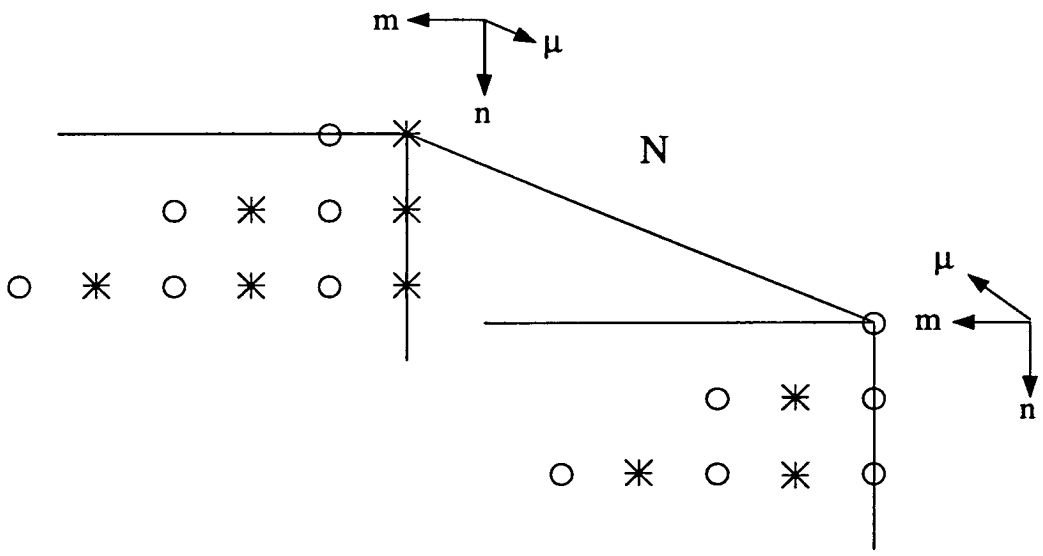
The imaginary coefficients would be mapped by applying the REVC and REVR functions to Eq.(5.5.12). This mapping is identical to the mapping Eq.(5.2.1) discussed for the rhomboidal truncation.

It is also possible to use the mapping Eq.(5.2.4) with the Legendre transforms that use the symmetry property, whilst keeping an efficient storage method for the Legendre polynomials. The n axis of the Legendre values must be kept vertical in the DAP store.

Each column of the original spectral coefficients' mapping Eq.(5.2.4) contains coefficients that alternate $m+n$ odd and even, so the Legendre values cannot be separated into $m+n$ odd and even as before. However, if the Legendre polynomials are separated so that values with odd and even n are stored in each half of the DAP array, calculations for one n odd and even could proceed in parallel. Fig. 21 illustrates how this mapping is obtained from



(a) Shows the original Legendre polynomial mapping before symmetry.



(b) Shows storage format after coefficients with odd and even n are separated into each half of the DAP array.

Figure 21. Illustration of the remapping of the Legendre polynomials, for T5 resolution, in order to use the symmetry property. Coefficients of odd and even n are separated.

the data mapping used for the nonsymmetric transforms. The mapping expression is,

$$P_{m,n}: \quad (m, n, \mu_j) \rightarrow \{ m+1, j+32, 1+n/2 \} \qquad n \text{ even}$$

$$(m, n, \mu_j) \rightarrow \{ m+1, j, 1+n/2 \} \qquad n \text{ odd} \qquad (5.5.13)$$

Each half contains a mix of symmetric and antisymmetric coefficients unlike Eq.(5.5.11). It will be seen however, that because the spectral coefficients' mapping matches that of the Legendre data in both cases, the transform algorithms are almost the same.

To store the polynomial derivatives in the same array, the above procedure is followed. The details of the mappings will be left until the transforms are considered. The used fraction of the array holding the Legendre values does not alter from that given for the nonsymmetric case, although less total storage is required.

5.5.2. Inverse Legendre transform

In this section the modifications necessary to the nonsymmetric inverse Legendre transform are described. The original spectral coefficients' mapping Eq.(5.2.4) and associated symmetric mapping of the Legendre polynomials Eq.(5.5.13) are used, although the changes necessary to use the remapped spectral coefficients Eq.(5.5.12) together with Eq.(5.5.11) are also described.

The algorithm commences as before, the real and imaginary coefficients are separated into work matrices. This code would be the same regardless of the spectral coefficients' mapping. The main difference is in the loop, where the symmetric and antisymmetric products are computed in parallel. On each pass of the loop, two vectors are selected from the spectral coefficients, one for n odd and one for n even. These are multiplied by their respective Legendre polynomials in each half of the DAP array. This step involves more work than the previous algorithm as each vector must first be broadcast and mask-assigned to the appropriate half of a work matrix.

Although the amount of data routing has increased in the loop, the loop length is now only 22 since two values of n are computed in parallel and therefore half the number of arithmetic operations in matrix mode are

required. However, after the loop, an additional step to form the Fourier coefficients from their symmetric and antisymmetric parts is necessary (Eq.(5.5.3)). Before describing this step, it is necessary to determine exactly how latitude is mapped for the Fourier data in each half of the DAP array. The example of Eq.(5.5.13) assumed j to increase with the column index for both odd and even n, in which case, a shift of 32 processors would be required to map one half onto the other. Alternatively, j could increase from columns 1-32 and decrease from 33-64, requiring a reversal of columns. From the timings of DAP functions given in chapter 2, a shift of 32 processors takes the same time as the REVC function. The symmetric mapping using REVC is chosen for which the mapping expression is,

$$\begin{aligned} P_{m,n}: \quad & (m, n, \mu_j) \rightarrow \{ m+1, j, 1+n/2 \} && n \text{ even} \\ & (m, n, \mu_j) \rightarrow \{ m+1, 65-j, 1+n/2 \} && n \text{ odd} \end{aligned} \qquad (5.5.14)$$

After the loop, the symmetric (a_m) and antisymmetric (b_m) Fourier coefficients are mapped as,

$$\begin{aligned} a_m: \quad & (m, \mu_j) \rightarrow \{ m+1, j \} && \text{for } m \text{ even} \\ & (m, \mu_j) \rightarrow \{ m+1, 65-j \} && \text{for } m \text{ odd} \\ b_m: \quad & (m, \mu_j) \rightarrow \{ m+1, 65-j \} && \text{for } m \text{ even} \\ & (m, \mu_j) \rightarrow \{ m+1, j \} && \text{for } m \text{ odd} \end{aligned} \qquad (5.5.15)$$

where j=1 to 32 and a_m and b_m are those in Eq.(5.5.3). This storage format is illustrated in Fig. 22. From Eq.(5.5.3), the b_m coefficients must be negated for the southern hemisphere before the a_m and b_m coefficients are added together, in parallel for both hemispheres.

The DAP FORTRAN code for this algorithm is given in Appendix A. Using this code, an estimate of the execution time of the routine can be made. This estimate and the measured CPU time of the routine are given in Table 7. As before, the measured time is subject to a possible error of 0.1msecs. An overhead of 21% applies to the estimated time, consistent with overheads found previously. The number of floating point operations required for this

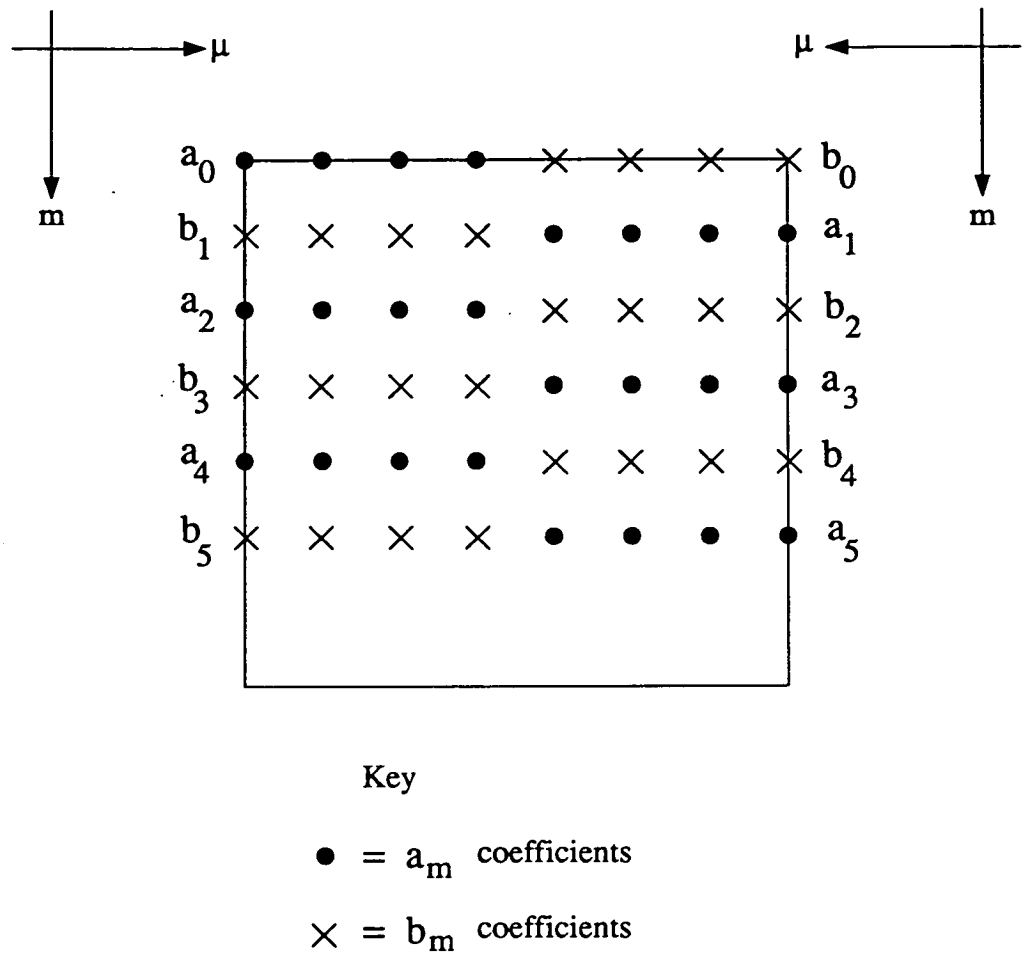


Figure 22. Illustration of the mapping of the symmetric (a_m) and antisymmetric (b_m) components of the Fourier coefficients before recombination in the inverse Legendre transform.

Transform	Estimated time (msecs)	Measured time (msecs)	Overhead
Inverse	27.13	32.8 ± 0.1	21%
Direct	49.2 [1]	58.0 ± 0.1 [1]	20%
	34.6 [2]	41.5 [2]	

Table 7.

Estimated and measured times for the inverse and direct Legendre transforms that use the symmetry property of the Legendre polynomials.

[1] This estimate uses the the measured time of the SUM2C function.

[2] This assumes the SUM2C function takes the same time as the SUMC function. The measured time in this case is the estimated time plus 20% overhead.

transform is $\frac{1}{2}$NLAT[$(M+1)(M+6)+M(M-1)$] which gives a performance rate of 3.7Mflops.

Comparing the measured time with the 52.5msecs for the nonsymmetric algorithm gives a speedup ratio of 1.6 or a decrease of 37.5%. Although the loop length has been halved, the additional cost of forming the work matrix inside the loop and combining the symmetric and antisymmetric parts after the loop reduce the speedup and the performance rate.

The efficiency of this algorithm can be calculated using Eq.(2.6.3) to be 0.31. Only the operations inside the loop are considered since the initial and final operations only account for 4% of the execution time. This efficiency is less than the nonsymmetric transform because two broadcasts are now used, where each writes to half the PEs written to in the nonsymmetric version. This efficiency could be increased and the performance improved by coding the routine in the DAP assembly language to take advantage of the spare processors as described for the nonsymmetric version.

Suppose the mappings of Eq.(5.5.12) and Eq.(5.5.11) were used. No changes to the initial operations in the algorithm are required. Furthermore, no changes to the code within the loop are necessary since the data are properly positioned relative to each other. The only change necessary is to the formation of the full Fourier coefficients from the symmetric and antisymmetric parts after the loop, as each half of the DAP array contains coefficients for $m+n$ odd or even instead of n odd or even. This simplifies the operations necessary to form the full Fourier coefficients. The difference this makes to the algorithm timing however is negligible. A more important point is that the algorithm can use different spectral data mappings, as long as the spectral coefficients are mapped correctly, relative to the Legendre polynomials. Only the final routing operations will need to be changed.

5.5.3. Direct Legendre transform

Following on from the inverse transform, the modifications to the direct transform to make use of the symmetry property are described. As before, the original spectral coefficient mapping is retained and the necessary changes to the algorithm to use the alternative mapping are described later. The mapping of the derivatives will also need to be altered.

As for the inverse transform, the direct transform algorithm will be similar to the nonsymmetric version. As the derivatives remain stored in the same array as the Legendre polynomials, their products can be computed in parallel. In addition, for each product, a vector result for two values of n will be obtained. The loop length will again be $M/2$ rather than M.

After the multiplication by the Gaussian weights, the second step, new to the algorithm, is to compute the symmetric and antisymmetric parts of the Fourier coefficients using Eq.(5.5.7) and Eq.(5.5.9). Since this additional work is outside the loop over n it represents a small increase in the total work of the algorithm. Only the code in this step will change if the alternative data mapping is used, since the mapping of the symmetric and antisymmetric parts depends on the mapping of the Legendre values.

The formation of the symmetric and antisymmetric Fourier coefficients is complicated because on each row $m+n$ is alternately odd and even. Thus d_m and c_m from Eq.(5.5.7) and Eq.(5.5.9) are stored alternately on each row. Fig. 23 shows how the coefficients must be created so they are multiplied correctly to the Legendre polynomials. The a_m and b_m coefficients to be multiplied to the derivatives have to be similarly created and stored. Since the mapping of the derivatives has not been finalized yet, this will be discussed later.

The next step is to form the Fourier work matrix, as before, to be multiplied by the Legendre data array. If the problem of avoiding an overlap of coefficients is ignored for the moment, the following stage is to sum along each half row to give a vector result of each product for one odd and one even n. These two vectors would then be stored in a product matrix as before. Two issues arise. First, a new function is needed to sum each half of a row separately. Unfortunately, as the Edinburgh University DAPs were to be taken out of service soon after this symmetry work was started, an optimized function written in the DAP assembly language (APAL) that summed each half was not developed. Instead the function was written using two calls of SUMC with appropriate masking. The measured time of the new algorithm suffered as a consequence. However, since an optimized function would take no more time than the SUMC function, the measured time could be made more realistic by assuming the same CPU time for the new function (SUM2C) as SUMC.

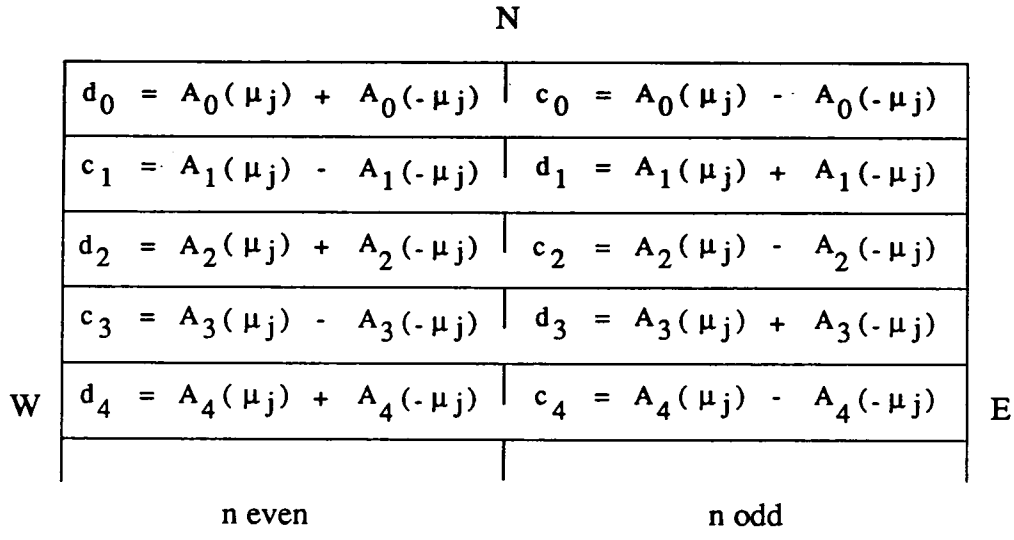


Figure 23. Illustration of the first 5 rows of the DAP matrix showing how the symmetric and antisymmetric Fourier coefficients are mapped for the Legendre polynomials in the direct Legendre transform.

The second issue is that the mapping of the product vectors has to be determined. The polynomial product vectors are assigned to columns of the product matrix with n increasing with the column index. In other words, multiply the mapping of n in Eq.(5.5.14) by 2 to give,

$$P_{m,n} \text{ product:} \quad (m, n) \rightarrow \{ m + 1, n + 2 \} \tag{5.5.16}$$

The derivatives, separated into n odd and even, are stored with n decreasing down the store. When assigning the derivative product vectors to the matrix, n must decrease with the column index of the matrix. This means that the parity of the truncation wavenumber M determines the half of the array that n odd and n even for the derivatives map to. That is, if M is odd, derivatives with n odd map to the west half of the array, as the first vector from this half contains the values for n=0 for the polynomials and n=M for the derivatives. Mapping derivatives with n even to the west would have given the coefficients for n=M-1 from the first vector, M from the second. This would be in the wrong order when assigned to the product matrix. The opposite is true for even M, which is the case for the model described in the next chapter. This means the mapping of the polynomial derivatives using the symmetry property is,

$$\begin{aligned} dQ_{m,n}/d\mu: \quad (m, n, \mu_j) &\rightarrow \{ 64 - m, j, \tfrac{1}{2}(42-n) + 1 \} \qquad n \text{ even} \\ (m, n, \mu_j) &\rightarrow \{ 64 - m, 65 - j, \tfrac{1}{2}(42-n) + 1 \} \quad n \text{ odd} \end{aligned} \tag{5.5.17}$$

and the derivative product mapping is,

$$dQ_{m,n}/d\mu \text{ product:} \quad (m, n) \rightarrow \{ 64 - m, 44 - n \} \tag{5.5.18}$$

Whether M is odd or even, within a column on a DAP plane there must always be a total of M+2 Legendre values.

Eq.(5.5.17) shows how the symmetric (b_m) and antisymmetric (a_m) Fourier coefficients that are multiplied to the derivatives have to be stored (Fig. 24). They are calculated efficiently using routing and masking operations in a way similar to the c_m and d_m coefficients. The DAP FORTRAN code to compute the transform is given in Appendix A. As explained above, the code is dependent on the parity of the maximum wavenumber. This dependency will also apply to the alternative mapping Eq.(5.5.12) discussed earlier.

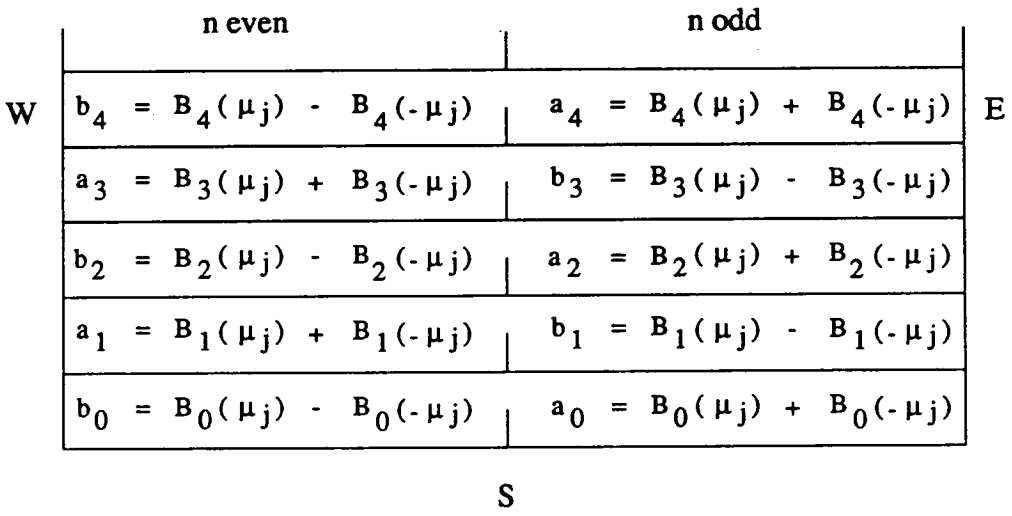


Figure 24. Illustration of the last 5 rows of the DAP matrix showing how the symmetric and antisymmetric Fourier coefficients are mapped for the Legendre derivatives in the direct Legendre transform.

The Legendre polynomials have to be stored to a resolution of M=43 to allow for the inverse transform of the velocities. This means 22 matrices are required for storage, half that when the symmetry property was not used; 17% of the available DAP store. Fig. 25 illustrates how the Legendre values appear in the DAP store.

The reassignment of the Fourier work matrix, constructed from the symmetric and antisymmetric coefficients, during the loop is now considered. The loop length becomes 22. For T42, on the west half of the PE array, the polynomials use the first row only of the first matrix whilst the derivatives use the last 43 rows. On the east half however, the polynomials use the first two rows, the derivatives the last 42 rows. Therefore, although 20 rows are free in each half, those in the eastern half are displaced one row south relative to those in the western half. The first loop over index n must therefore go only as far as n=19, or the 10th matrix from Eq.(5.5.14). This is one less than in the nonsymmetric case. Similarly, the second loop can only go as far as n=39, before the Fourier work matrix has to be altered, again one less than before.

The DAP FORTRAN for this transform is given in Appendix A. An estimate of the execution time for this code is given in Table 7. Two values are given. The first assumes that the summation function SUM2C takes the same time as SUMC. The second uses the measured time of the SUM2C function of 684msecs. The measured time of the algorithm, therefore, should only be compared to the second estimate. The overhead is found to be 20%. This algorithm requires $2NLAT(M+1)(M+6)-(M+1)(M+2)$ floating point operations, so the measured performance is 4.5Mflops but the potential performance would be 6.3Mflops. Like the inverse transform, a decrease over the nonsymmetric direct transform has resulted.

Comparison with Table 6 shows that the inefficient summation function means the symmetric transform routine gives no improvement over the nonsymmetric version. However, if the summation function took the same time as the DAP FORTRAN function SUMC, an overhead of 20% would imply a measured time of 41.5msecs. This is 30% less than the time for the nonsymmetric algorithm. The speedup ratio would therefore be 1.43, less than that for the inverse transform. Unlike the inverse transform, the work in the loop over n has only slightly increased (if an efficient SUM2C function is assumed). So the poorer speedup ratio is a result of the increased work

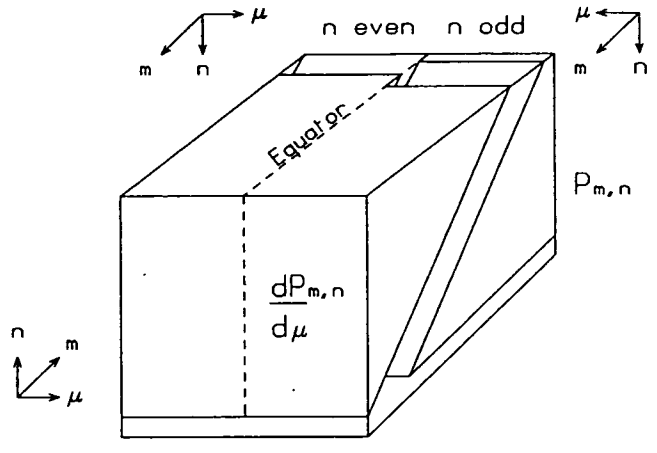


Figure 25. Schematic illustration of the storage format for the Legendre polynomials and their derivatives. Reproduced from Carver (1988).

outside the loop contributing more to the overall time, since the time spent in the loop has now decreased. Operations outside the loop account for about 12% of the CPU time. The efficiency of the code inside the loop is the same as the nonsymmetric case at 0.69.

For the alternative mapping for which diagonals are stored down columns, only the computation of the symmetric and antisymmetric coefficients has to be changed, which becomes simpler. For example, the corresponding storage to Fig. 23 would be d_m entirely in one half and c_m entirely in the other. Similarly for a_m and b_m. The storage of the derivatives is still subject to the restriction that the total number of Legendre values in each column must be M+2. The product mapping follows from the Legendre values as before.

5.6. Summary

By considering different storage arrangements of the Legendre polynomials and their derivatives, the most efficient algorithms and mappings were devised. In both cases, the best algorithms arose from the most efficient mappings. Inefficiencies from the triangular nature of the data could not be overcome by simple packing strategies, although this property allowed derivatives and polynomials to be stored in the same array. This reduced storage requirements and allowed the summations required for the direct transform to be computed in parallel. For the data arrangement chosen, the ratio of the time of the direct transform to that of the inverse transform is 1.13 for these parallel algorithms. For serial algorithms this ratio would be greater than 2, which illustrates the success of the exploitation of the extra parallelism available in the direct transform.

The Legendre transform algorithms developed in this chapter achieved better performances than those of Fishbourne (1980). Although with the nonsymmetric algorithms, the storage requirements of the Legendre data were more than in Fishbourne's algorithms, when the symmetry property was used, the storage requirements became the same (704 planes) and a further improvement in performance over Fishbourne's algorithms resulted.

A detailed account was given of the effect of different mappings for the data, within the context of the most efficient algorithm. The relationships between the mappings of the data involved in the transform, particularly for

the direct case, were established. These were then used to determine combinations of mappings of Legendre values and spectral coefficients that would be expected to give the best performance. Therefore, the storage of data on the DAP plays an important role, more so than on a serial machine.

CHAPTER 6

A BAROTROPIC SPECTRAL MODEL ON THE ICL DAP

6.1. Introduction

In this chapter the implementation of a spectral model on the ICL DAP is discussed in detail. Since the spectral method is preferred to the finite difference method for global models, it is important to study the implementation of this type of model on a processor array computer. The algorithms used to solve the spectral equations have a large degree of inherent parallelism but are different from those used for finite difference models. Therefore new parallel algorithms will need to be developed. Also, as the number of degrees of freedom of a variable is different in spectral, Fourier and gridpoint representations, particular attention will need to be paid to the choice of data mapping. Since the spectral method is usually only applied in the horizontal, it is sufficient to implement a single level model on the DAP. The shallow-water equations on a sphere are therefore used for the model.

In the next section, the model equations are formulated and the computational procedure described. The third section presents a preliminary discussion on what are likely to be the key issues in developing an efficient model for the DAP. The following two sections deal with the actual implementation of the model on the computer, followed by sections that analyse the performance of the model and present results to show the model's veracity. Finally, conclusions on the suitability of the DAP to spectral models are presented.

6.2. Description of the model

6.2.1. Spectral equations

The nondimensional shallow-water equations used for the spectral model are those of Hoskins (1973). They are,

$$\frac{\partial \xi}{\partial t} = \frac{-1}{1-\mu^2}\frac{\partial}{\partial \lambda}[(\xi+f)U] - \frac{\partial}{\partial \mu}[(\xi+f)V]$$

$$\frac{\partial D}{\partial t} = \frac{1}{1-\mu^2}\frac{\partial}{\partial \lambda}[(\xi+f)V] - \frac{\partial}{\partial \mu}[(\xi+f)U] - \nabla^2[\phi' + \frac{U^2+V^2}{2(1-\mu^2)}] \qquad (6.2.1)$$

$$\frac{\partial \phi'}{\partial t} = \frac{-1}{1-\mu^2}\frac{\partial}{\partial \lambda}[\phi' U] - \frac{\partial}{\partial \mu}[\phi' V] - \bar{\phi} D$$

To dimensionalize these equations, the radius of the Earth, a, is used as the length scale and the reciprocal of the angular velocity, Ω^{-1}, is used as the time scale. Also, ϕ' is the departure of the geopotential height of the fluid from some mean value given by,

$$\bar{\phi} = gH/(a^2\Omega^2) \tag{6.2.2}$$

where H is the mean height of the fluid. The velocity components U and V are given by $u\cos\theta$ and $v\cos\theta$ respectively, where θ is the latitude, $\mu = \sin\theta$ and λ is the longitude. The divergence is represented by D and the relative vorticity by ξ.

Following the procedure described in chapter 3, the prognostic variables are approximated (assuming a triangular truncation) by,

$$\begin{aligned}
\xi \simeq \hat{\xi}(\lambda,\mu,t) &= \sum_{m=-M}^{M} \sum_{n=|m|}^{M} \xi_{m,n}(t)\, Y_{m,n}(\lambda,\mu) \\
D \simeq \hat{D}(\lambda,\mu,t) &= \sum_{m=-M}^{M} \sum_{n=|m|}^{M} D_{m,n}(t)\, Y_{m,n}(\lambda,\mu) \\
\phi' \simeq \hat{\phi}'(\lambda,\mu,t) &= \sum_{m=-M}^{M} \sum_{n=|m|}^{M} \phi'_{m,n}(t)\, Y_{m,n}(\lambda,\mu)
\end{aligned} \tag{6.2.3}$$

The expansion coefficients are determined by Eq.(3.4.25). The ^ denoting an approximation will now be dropped on the understanding that the truncated variables are being used unless otherwise stated.

Introducing the streamfunction ψ and velocity potential χ, the relative vorticity and divergence can be expressed as,

$$\xi = \nabla^2\psi, \qquad D = \nabla^2\chi \tag{6.2.4}$$

Substituting expansions for the variables into the above equations and using Eq.(3.4.20) in nondimensional form gives the relations,

$$\psi_{m,n} = -\xi_{m,n} / [n(n+1)] \qquad (6.2.5)$$

$$\chi_{m,n} = -D_{m,n} / [n(n+1)]$$

The velocities, U and V, can be written as,

$$U = \frac{\partial \chi}{\partial \lambda} - (1-\mu^2)\frac{\partial \psi}{\partial \mu} \qquad (6.2.6)$$

$$V = \frac{\partial \psi}{\partial \lambda} + (1-\mu^2)\frac{\partial \chi}{\partial \mu}$$

Taking the scalar product of both of these equations, substituting expansions for ψ and χ and using the relations Eq.(3.4.23), Eq.(3.4.33) and Eq.(6.2.5), gives the expansion coefficients of U and V as,

$$U_{m,n} = -i\alpha_{m,n}D_{m,n} + \beta_{m,n+1}\xi_{m,n+1} - \beta_{m,n}\xi_{m,n-1} \qquad (6.2.7)$$

$$V_{m,n} = -i\alpha_{m,n}\xi_{m,n} - \beta_{m,n+1}D_{m,n+1} + \beta_{m,n}D_{m,n-1}$$

where,

$$\alpha_{m,n} = m / [n(n+1)] \qquad (6.2.8)$$

$$\beta_{m,n} = \varepsilon_{m,n} / n$$

and $\varepsilon_{m,n}$ is given by Eq.(3.4.34).

A problem with calculating the spectral coefficients $U_{m,n}$ and $V_{m,n}$ is that the series expansions must extend one degree above that of the other variables to be consistent with their truncation, as discussed by Machenhauer (1979) and easily seen from Eq.(6.2.7). The disadvantages of calculating the coefficients of U and V from the coefficients of vorticity and divergence are that more storage space is required and the truncation is not the same throughout the model. By substituting expansions for ψ and χ into Eq.(6.2.6) and using Eq.(6.2.5), it is possible to obtain the gridpoint values of U and V (which is the only representation of the wind used in the model) from the spectral coefficients of ξ and D. This approach, although more expensive computationally, has been used in the ECMWF spectral model (Baede *et al.*

1979) to avoid the disadvantages described above.

Following the procedure described in chapter 3, the truncated spectral equations are obtained by substituting the expansions for the variables into the governing equations (except for the nonlinear terms) and taking the scalar product with the basis $Y_{m,n}$. After integrating by parts and using the property that $P_{m,n}(\pm 1) = 0$, the equation for the relative vorticity becomes,

$$\frac{d\xi_{m,n}}{dt} = \frac{1}{4\pi}\int_{-1}^{1}\int_{0}^{2\pi} \{[(\xi+f)V]\frac{dP}{d\mu}_{m,n} - \frac{im}{1-\mu^2}[(\xi+f)U]P_{m,n}\} e^{-im\lambda} \, d\lambda \, d\mu \quad (6.2.9)$$

The divergence equation becomes,

$$\frac{dD_{m,n}}{dt} = \frac{1}{4\pi}\int_{-1}^{1}\int_{0}^{2\pi} \{\frac{im}{1-\mu^2}[(\xi+f)V]P_{m,n} + [(\xi+f)U]\frac{dP}{d\mu}_{m,n}\} e^{-im\lambda} \, d\lambda \, d\mu \quad (6.2.10)$$

$$+ \frac{n(n+1)}{4\pi}\int_{-1}^{1}\int_{0}^{2\pi} [\frac{U^2+V^2}{2(1-\mu^2)}]P_{m,n}e^{-im\lambda} \, d\lambda \, d\mu + n(n+1)\phi'_{m,n}$$

after using Eq.(3.4.20) and the continuity equation becomes,

$$\frac{d\phi'_{m,n}}{dt} = \frac{1}{4\pi}\int_{-1}^{1}\int_{0}^{2\pi} \{[\phi' V]\frac{dP}{d\mu}_{m,n} - \frac{im}{1-\mu^2}[\phi' U]P_{m,n}\} e^{-im\lambda} \, d\lambda \, d\mu - \bar{\phi} D_{m,n} \quad (6.2.11)$$

6.2.2. Calculation of nonlinear terms

To compute the nonlinear terms in the prognostic spectral equations the transform method is used. The method is now illustrated for the vorticity equation.

The first stage involves the computation of the gridpoint values of vorticity and the wind components from their spectral coefficients. From Eq.(3.4.35), for any variable F, these are computed by the inverse Legendre transform,

$$F_m(\mu_j,t) = \sum_{n=|m|}^{M} F_{m,n}(t)P_{m,n}(\mu_j) \quad \text{for } m=0 \text{ to } M \tag{6.2.12}$$

followed by an inverse Fourier transform,

$$F(\lambda_i,\mu_j,t) = \sum_{m=-M}^{M} F_m(\mu_j,t)\exp(im\lambda_i) \quad \text{for all } \mu_j \tag{6.2.13}$$

For the velocities U and V, the summation in the inverse Legendre transform must extend to M+1.

The products $[(\xi+f)U]_{i,j}$ and $[(\xi+f)V]_{i,j}$ are computed once the required gridpoint fields have been obtained. This is a local computation as the calculation of the product at any point involves only values at that point. Any linear operations can be applied in gridpoint space.

To compute the vorticity tendency, a Fourier transform of the products is performed,

$$A_m(\mu_j) = \frac{1}{2\pi}\int_0^{2\pi} [(\xi + f)U]_{i,j}\, e^{-im\lambda}\, d\lambda \tag{6.2.14}$$

and,

$$B_m(\mu_j) = \frac{1}{2\pi}\int_0^{2\pi} [(\xi + f)V]_{i,j}\, e^{-im\lambda}\, d\lambda \tag{6.2.15}$$

followed by a direct Legendre transform,

$$\dot{\xi}_{m,n} = \frac{1}{2}\int_{-1}^{1} \{B_m \frac{dP_{m,n}}{d\mu} - \frac{im}{1-\mu^2}A_m P_{m,n}\}\, d\mu \tag{6.2.16}$$

The quadrature formulae Eq.(3.4.41) and Eq.(3.4.44) are used to compute the integrals Eq.(6.2.14), Eq.(6.2.15) and Eq.(6.2.16). The integrals in Eq.(6.2.10) and Eq.(6.2.11) are computed in a similar way to give the tendencies of the divergence and the geopotential.

6.2.3. Inclusion of diffusion

To prevent spectral blocking (Gordon and Stern, 1974; Machenhauer, 1979), a linear diffusion term is included to remove energy from the smallest scales. For a variable f, which may be ξ or D only, the diffusion term is applied as,

$$\dot{f} = \dot{f}^* - (-1)^p K \nabla^{2p} f \tag{6.2.17}$$

where K is a nondimensional diffusion coefficient. The term f^* represents the tendency of f computed without any damping. Thus, the diffusion can be applied as a correction to the adiabatic tendencies.

Transforming to spectral space gives,

$$\dot{f}_{m,n} = \dot{f}^*_{m,n} - [n(n+1)]^p K f_{m,n} \tag{6.2.18}$$

where the property Eq.(3.4.20) has been used. Since a Laplacian does not have to be solved, an implicit time scheme can be used with little added computational cost i.e.

$$f_{m,n}(t+\Delta t) = f^*_{m,n}(t+\Delta t) \; / \; \{1 + 2\Delta t \; (n(n+1))^p K\} \tag{6.2.19}$$

where $f^*_{m,n}$ represents the values of f computed from the adiabatic tendency.

The model uses a sixth order diffusion term (p=3) to affect the long wavelengths as little as possible. The value of the diffusion coefficient used was 1.3×10^{26} $m^6 s^{-1}$. Table 8 shows the e-folding times for several wavenumbers using this value of K.

6.2.4. Time differencing

Once the tendencies of the prognostic variables have been computed, they are integrated forward in time using the leapfrog scheme Eq.(3.2.1). The Asselin time filter Eq.(3.2.2) is applied to each of the prognostic variables at each timestep to control the time-level splitting associated with the leapfrog scheme and also to provide additional smoothing of the short wavelengths. A weak filter parameter of ν=0.005 is used.

The linear stability condition is,

Wavenumber, n	e-folding time (days)
1	7.4×10^{8}
5	2.2×10^{5}
10	4.5×10^{3}
15	431
20	80
25	22
30	7.4
35	3.0
40	1.4
42	1.0

Table 8.

The e-folding times for selected wavenumbers using a sixth order diffusion term and a diffusion coefficient of 1.3×10^{26} $m^6 s^{-1}$ as used in the spectral model.

$$\Delta t \leq 1 / |\sigma_{max}| \qquad (6.2.20)$$

where σ_{max} is the frequency of the fastest oscillating mode. This can be determined by computing the eigenfrequencies of the linearized spectral equations. As shown by Machenhauer (1979), for spectral models that are not too severely truncated, the timestep limit is approximately given by,

$$\Delta t \leq 340 / \sqrt{[M(M+1)]} \quad \text{minutes} \qquad (6.2.21)$$

where M is the truncation wavenumber. For M = 42, this implies that,

$$\Delta t \leq 8 \text{ minutes.} \qquad (6.2.22)$$

Tests confirmed this limit and a timestep of 5 minutes was used for the model.

6.2.5. Computational procedure

Assuming the values of each variable, ξ, D and ϕ' are available at two time levels t and $t-\Delta t$, where the $t-\Delta t$ values are filtered, the computational procedure for each variable (in turn) is given below.

1. The gridpoint values of U and V are computed from their spectral values by an inverse Legendre transform followed by an inverse Fourier transform. These only need be computed once each step.

2. Gridpoint calculations are performed to evaluate the nonlinear terms.

3. Fourier transforms of the nonlinear products are used to obtain their Fourier coefficients.

4. Legendre transforms are performed to evaluate the spectral coefficients of the nonlinear terms.

5. If necessary, spectral calculations for the variable are performed to compute the spectral tendency.

6. The variable is integrated forward in time.

7. Time filtering and diffusion are applied.

6.3. Preliminary discussion on implementing the model on the DAP

6.3.1. Introduction

In this section some important issues in the implementation of the model on the ICL DAP will be discussed; namely the mapping of the data on the PE array, the choice of resolution and truncation.

Each model variable has three representations, spectral, Fourier and gridpoint and the mapping of data to the DAP will be different in each case. The number of degrees of freedom of each representation is also different. This means it will not be possible to achieve maximum use of the processors for all representations, in contrast to a gridpoint model.

This difficulty poses the problem of how to make best use of the array. A reasonable aim might be to attain maximum efficiency, in storage and processing, in the representation used most often. However, for the spectral shallow-water model of Fishbourne (1980), the transform stages, during which the representation of a variable changes, accounted for 99% of the processing time. Therefore it would seem that the correct approach should be to optimize the transform stages. Efficient parallel Legendre transform algorithms were developed in the previous chapter. The model is based on the nonsymmetric Legendre transform algorithms, although the effect of the symmetric algorithms on the model's performance is considered.

6.3.2. Data mapping

6.3.2.1. Spectral coefficients

The conclusions from the previous chapter show that the real spectral coefficients can be mapped as Eq.(5.2.4) with the imaginary coefficients mapped according to Eq.(5.2.5). The Legendre polynomials and their modified derivatives are mapped according to Eq.(5.2.11).

6.3.2.2. Fourier and gridpoint data mappings

The Fourier coefficients exist as an intermediate stage between spectral and gridpoint space. No calculations other than the transforms are done in Fourier space. For a T42 resolution, the number of Fourier coefficients is

greater than the space available in one DAP matrix and two matrices are required, the real coefficients are stored in one matrix and the imaginary coefficients in another. The precise mapping depends not only on the Legendre transforms but also on the FFTs. The FFT algorithm described in chapter 2 is used, in which the coefficients are required with wavenumber m increasing with the row index. The FFTs are performed on each latitude and are independent of each other. Therefore the latitude mapping will be completely determined by the Legendre transforms. The FFT will determine the longitude mapping of gridpoint data on the PEs. The amount of space required for the gridpoint representation of a variable is the greatest of all three representations. At T42, two matrices are needed.

6.3.3. Constraints on model formulation

6.3.3.1. Architectural constraints

Given that the dimensions of each representation of the data are different, it will generally be impossible to efficiently utilize the DAP array in all three representations. This is an important difference from finite difference models for which high efficiency is possible if all processors hold gridpoints. This is an aspect of spectral models that would seem to make them unsuitable for a SIMD processor array architecture.

The dimensions of the PE array impose constraints on the formulation of the model if the data are to utilize the array as much as possible. This applies equally to all classes of model; finite difference, spectral and finite element. These constraints will restrict the allowable resolutions of such models. This is discussed for the spectral model in the next section.

If the resolution is reduced, the execution time of a spectral model on the DAP may not decrease as much as the same model on a vector machine (multiprocessor or otherwise). This is because certain calculations would still require the same number of operations although less of the array would be used. It ought to be possible to use any spare processor time in a MIMD processor array for other tasks or jobs. Thus, of the processor array architectures, the SIMD type will impose perhaps the severest constraints on the model formulation, as the penalty for not making optimum use of the processors is likely to be the highest. This underlines the need for efficient

data mapping strategies and algorithms, justifying the time spent on these issues.

6.3.3.2. Restrictions on model resolution

Since it is not possible for all three data representations to fully utilize the DAP array, only one can be made to fit. The choice depends on the amount of time spent in each space. From the work of Fishbourne (1980), most of the execution time is spent in the Legendre transforms. This would suggest matching the spectral coefficients to the size of the array or, as the Legendre transforms change total wavenumber n to latitude, the number of latitudes.

Consider the spectral coefficients first. The greatest space required for the spectral coefficients for triangular truncation is, using Eq.(5.2.2),

$$S = (M_u+1)(M_u+2) \tag{6.3.1}$$

where M_u is the truncation of U and V. Ignoring the data mapping but assuming real and imaginary coefficients are stored in the same matrix, write,

$$(M_u+1)(M_u+2) = r4096 \tag{6.3.2}$$

where r is any positive integer. For some values of r, this gives,

$$\begin{aligned} r &= 1, \quad M = 61 \\ r &= 2, \quad M = 88 \\ r &= 3, \quad M = 108 \\ r &= 4, \quad M = 125 \end{aligned} \tag{6.3.3}$$

where $M=M_u-1$. Table 9 shows the percentage of the DAP array that contains useful data in all three representations, for a resolution of T61. The real and imaginary Fourier coefficients are held in separate matrices. The gridpoint data is assumed mapped in a regular fashion. Using a sheet mapping for storing all the gridpoint values uses 75% of the matrices. Using a crinkled mapping this figure could be raised to 90% as the number of matrices required is reduced.

If the number of latitudes is matched to the DAP array dimension then,

(A) Matching spectral data to DAP array (M = 61)

Spectral data	- 95%	(1 matrix/variable)
Fourier data	- 73%	(4 matrices/variable)
Gridpoint data	- 75%	(6 matrices/variable)
Gridpoint data	- 90%	(5 matrices/variable)

(B) Matching number of latitudes (M = 42)

Spectral data	- 46%	(1 matrix/variable)
Fourier data	- 67%	(2 matrices/variable)
Gridpoint data	- 100%	(2 matrices/variable)

Table 9.

The percentage use of the DAP array for two resolutions for each of the data representations.

$$J = s\,64 \tag{6.3.4}$$

where s is a positive integer. Using Eq.(3.4.47) for triangular truncation gives,

$$M \leq (s128\text{-}1)/3 \tag{6.3.5}$$

For some values of s this gives,

$s = 1, \quad M = 42$
$s = 2, \quad M = 85$
$s = 3, \quad M = 127$

Table 9 also illustrates usage of the processor array for the T42 resolution.

From the values given in the table, it would seem that matching the spectral data gives the best aggregate efficiency over all representations. However, as the Legendre transforms will account for most of the CPU time of the model, it will be the data mappings in these stages that determine the performance of the model.

The allowable resolutions will clearly be reduced or increased for larger or smaller PE array sizes respectively. For the DAP-310 with 1024 processors, additional resolutions such as T21 and T30 could be efficiently used. Another opportunity would arise with multi-level models. The number of latitudes J in Eq.(6.3.4) could be reduced by a factor of two if two levels could be processed simultaneously.

For any model, the choice of resolution depends on the nature of the problem and perhaps more importantly the computing facilities available. Estimates of the DAP memory requirements and execution time showed that only low resolution barotropic spectral models would be practical on the DAPs at Edinburgh University, so a resolution of T42 was chosen. A triangular truncation was selected since it is preferred in meteorology and techniques to use it should be developed. The Legendre transform algorithms were developed in the previous chapter for this resolution.

The choice of resolution for spectral models implemented on vector computers also tends to be influenced by the architecture. Computers that have vector registers, such as the CRAY series, are most efficient when vector lengths are multiples of the number of vector registers. Another factor is that

the FFT algorithms used are commonly radix 2, where I must be 2^n, or where the number of points is a multiple of powers of prime numbers (Temperton, 1983). This is why spectral models are commonly designed to run with resolutions T21, T42, T63, T84 etc.

6.4. Fast Fourier transforms

6.4.1. Computational transforms

The spectral model requires inverse and direct Fourier transforms of the form,

$$F(\lambda_i,\mu_j,t) = \sum_{m=-M}^{M} F_m(\mu_j,t)\exp(im\lambda_i) \tag{6.4.1}$$

and,

$$F_m(\mu_j,t) = \frac{1}{I}\sum_{i=1}^{I} F(\lambda_i,\mu_j,t)\exp(-im\lambda_i) \tag{6.4.2}$$

where λ_i is given by Eq.(3.4.42). However, FFT routines are written to evaluate,

$$A_j = \sum_{n=0}^{N-1} a_n\exp(2\pi inj/N) \qquad j=0,\ldots N-1 \tag{6.4.3}$$

To use these routines, it is necessary to extend the range of zonal wavenumbers to $-I/2 \leq m \leq (I/2)-1$ and set the additional coefficients to zero. Following Orszag (1971), for the inverse transform Eq.(6.4.1), define $m'=m+I/2$. It follows that,

$$F(\lambda_i,\mu_j,t) = (-1)^j\sum_{m'=0}^{I-1} F_{m'}\exp(2\pi im'j/I) \tag{6.4.4}$$

where $m'=0,\ldots.I-1$. Thus the inverse transform requires the creation of the coefficients with m negative. The odd points of the resulting gridpoint field have to be negated. For the direct transform, one method would be the reverse of Eq.(6.4.4),

$$F_{m'}(\mu_j,t) = \frac{1}{I}\sum_{i=0}^{I-1} (-1)^j F(\lambda_i,\mu_j,t)\exp(-im'\lambda_i) \qquad (6.4.5)$$

such that the required coefficients for $0 \leq m \leq I/2-1$ are given by $m'=I/2,\ldots I-1$. Alternatively, Orszag (1971) notes that Eq.(6.4.2) is equivalent to the direct form of Eq.(6.4.3),

$$a_n = \frac{1}{N}\sum_{j=0}^{N-1} A_j \exp(-2\pi inj/N) \qquad (6.4.6)$$

for m and $n \geq 0$ but $< N/2$. By substituting a computational wavenumber of the form $n=m+I$ it can be shown that the coefficients for $I/2 \leq n < I$ are equivalent to the coefficients for $m<0$.

6.4.2. Complex transform

As the model has 128 points of longitude, the FFTs would be done by extending the wavenumbers to m=64 and setting these new coefficients to zero. The coefficients for $m<0$ would then have to be created. This is the approach used by Fishbourne (1980). However, since the gridpoint values are real and the Fourier coefficients therefore conjugate symmetric, the transforms can be done as complex ones on 64 points only. This reduction is well known and described, for example, in the appendix to Orszag (1971).

Assuming I to be a multiple of two, define,

$$B_m = F_m + F_{P+m} + i\exp[(2\pi im)/I](F_m - F_{P+m}) \qquad \text{for } m=0,\ldots\tfrac{1}{2}I-1 \qquad (6.4.7)$$

where $P=I/2$ and m refers to the computational wavenumber, dropping the prime for convenience. The inverse transform of B_m is then given by,

$$B_j = \sum_{m=0}^{P-1} B_m \exp(2\pi imj/P) \qquad (6.4.8)$$

from which it is straightforward to show that,

$$B_j = F'_{2j} + iF'_{2j+1} \qquad \text{for } j=0,\ldots P\text{-}1 \tag{6.4.9}$$

Hence the transform of B_m fully determines the transform of F_m. The even points are stored as the real part of B_j and the odd points as the imaginary part i.e.

$$B_j = F_{2j} - iF_{2j+1} \tag{6.4.10}$$

The direct transform of B_j uses Eq.(6.4.6). To recreate the F_m coefficients, Eq.(6.4.7) and the conjugate symmetry relation are used to give,

$$F_m = \tfrac{1}{4}[B_m + B^*_{P-m} - i\exp(-2\pi im/I)(B_m - B^*_{P-m})] \tag{6.4.11}$$

for m=0,...P-1. A similar expression can be obtained for the coefficients for m=P...I-1.

6.4.3. Implementation

The model stores Fourier coefficients in the range m'=I/2,....I-1. The above equations should be written to use only those coefficients stored by the model rather than creating the coefficients for m'=0,...I/2-1.

The conjugate symmetry relation (omitting the prime again) is,

$$F_m = F^*_{I-m} \tag{6.4.12}$$

and can be used to give the following relation for the inverse transform,

$$B_m = F^*_{I-m} + F_{P+m} + i\exp(2\pi im/I)(F^*_{I-m} - F_{P+m}) \qquad \text{for } m=0,\ldots\tfrac{1}{2}I\text{-}1 \tag{6.4.13}$$

from Eq.(6.4.7). Thus, all references to Fourier coefficients on the RHS are in the range $F_{I/2}$ to F_I and can use the model coefficients directly. At m=0, Eq.(6.4.13) reduces further to,

$$\begin{aligned} \Re e\{\, B_0 \,\} &= \Re e\{\, F_P \,\} \\ \Im m\{\, B_0 \,\} &= -\Re e\{\, F_P \,\} \end{aligned} \tag{6.4.14}$$

since F^*_I=F_0 and F_0=0 from the extension of the definition of the model wavenumber m, and F_P is real as this corresponds to the m=0 model zonal wavenumber.

The calculation of Eq.(6.4.13) is straightforward. First the coefficients F^*_{I-m} have to be formed by routing. If the mapping of the computational wavenumber indexing F_{P+m} is,

$$F_{P+m}: \quad (P+m) \rightarrow \{m+1\} \tag{6.4.15}$$

then to be able to add and subtract the F_{I-m} coefficients, they must also be mapped as Eq.(6.4.15). Adding $P-2m$ to Eq.(6.4.15) gives the mapping of F_{I-m} as,

$$F_{I-m}: \quad (I-m) \rightarrow \{65-m\}$$

before routing. A reversal of rows and a planar shift south are therefore required to align F_{I-m} and F_{P+m}. The planar shift ensures that for m=0 in Eq.(6.4.13), there is no contribution from the F^*_{I-m} term. The real and imaginary B_m coefficients can therefore each be computed by one DAP FORTRAN statement.

For the direct transform, Eq.(6.4.11) can be used directly as long as the sign of the odd gridpoint values is not changed i.e. the method of Orszag (1971) is used. For m=0, using Eq.(6.4.13) it can be shown that $B_0=B_P$ which gives,

$$F_P = \tfrac{1}{4}[B_0 + B_0^* + i(B_0 - B_0^*)]$$

Using Eq.(6.4.14), this reduces to,

$$\begin{aligned} \Re e\{F_P\} &= \tfrac{1}{2}[\Re e\{B_0\} - \Im m\{B_0\}] \\ \Im m\{F_P\} &= 0 \end{aligned} \tag{6.4.16}$$

The computation of F_m is done in the same way as for B_m with the same routing. However, there is a nonzero contribution from the routed B^*_{P-m} coefficient for $m=P$. Thus after routing, a vector assignment to set F_P is necessary.

To summarize, the inverse transform is computed in three stages. The first involves calculating the B_m coefficients and in doing so converting to the computational wavenumber. The second stage is the FFT itself for which the

Cooley and Tukey (1965) algorithm is used. Finally, the values at the odd gridpoints are negated.

Usually the final part of the Cooley-Tukey algorithm is a sorting step, as the transform leaves the data in a bit-reversed order (see chapter 2). However, since all the calculations in gridpoint space only involve values at each point, the mapping of gridpoint values along each latitude is irrelevant. All gridpoint data has to be transformed and so will use the same bit-reversed mapping. By eliminating the sorting step on the direct and inverse transform, a small saving in CPU time can be made. The omission of the sorting step might not be possible for a model that has physical parametrizations in which information is required from neighbouring gridpoints.

Since the data is in bit-reversed order along each latitude, the reverse of the Cooley-Tukey algorithm, the Gentleman and Sande (1966) FFT, has to be used for the direct transform. The direct transform consists of three steps; the sign change at odd numbered gridpoints, the FFT itself and the calculation of the Fourier coefficients as detailed above. Once the Fourier coefficients are obtained, coefficients for $m>42$ are set to zero.

The values at even points are stored in one matrix, the values at odd points in another. Changing the sign of odd values is therefore a single matrix operation with no masking.

6.5. Implementation of the model

6.5.1. Available parallelism

In spectral space, computations can at most proceed over all real and imaginary coefficients and all variables. Thus, calculations such as the time stepping and diffusion have,

$$3(M+1)(M+2) = 5676 \quad \text{for } M = 42$$

potential processes. The calculation of the wind components, considered in the next section, has less inherent processes as only two variables are involved. For the Legendre transforms, four variables could be transformed concurrently if the PE array was large enough as the inverse and direct Legendre transforms are required four times per timestep. The same applies to the

FFTs.

For gridpoint calculations, such as the evaluation of nonlinear products, there is again more parallelism available in the model than in the hardware. At T42, the grid size is an exact multiple of the DAP array size so there are no idle processors. All four gridpoint products could potentially be computed concurrently making the available processes eight times the number of available processors.

To conclude, although the potential parallelism depends on the computations at each stage and the space (spectral, Fourier or gridpoint) those computations are performed in, it is still greater than the parallelism available from the hardware.

6.5.2. Calculation of the velocity components

The gridpoint values of U and V have to be computed to evaluate the nonlinear terms in the tendency equations. Their spectral coefficients are evaluated first using Eq.(6.2.7) and then transformed. Storage of $U_{m,n}$ and $V_{m,n}$ at $n=M+1$ does not cause any problems since the extra coefficients are stored in the next column in the matrix.

To illustrate the computation of the coefficients, the equation to calculate $U_{m,n}$ is separated into real and imaginary parts,

$$U^r_{m,n} = \alpha_{m,n} D^i_{m,n} + \beta_{m,n+1}\xi^r_{m,n+1} - \beta_{m,n}\xi^r_{m,n-1} \qquad (6.5.1)$$

$$U^i_{m,n} = -\alpha_{m,n} D^r_{m,n} + \beta_{m,n+1}\xi^i_{m,n+1} - \beta_{m,n}\xi^i_{m,n-1}$$

where the superscripts r and i denote real and imaginary respectively. Eq.(6.5.1) implies that $\beta_{m,n}$ must be defined up to and including $n=M+2$.

In coding Eq.(6.5.1) in DAP FORTRAN, shift operations are used to correctly align the data. However, because the orientation of the n axis is different for the real and imaginary spectral coefficients, the shifts are applied in opposite directions for the calculation of the real and imaginary coefficients of the velocity components. Some repeated operations may be omitted by precomputing the product with $\alpha_{m,n}$. That is, set,

$$E^r_{m,n} = \alpha_{m,n} D^i_{m,n} \qquad (6.5.2)$$

$$E^i_{m,n} = -\alpha_{m,n} D^r_{m,n}$$

Substituting $E_{m,n}$ into Eq.(6.5.1) shows how the velocities are computed in the model.

These calculations could have been optimized further, by the additional preliminary calculations,

$$B^r_{m,n} = -\beta_{m,n+1}, \qquad B^i_{m,n} = -\beta_{m,n+1}$$

$$F^r_{m,n} = \xi^r_{m,n+1}, \qquad F^i_{m,n} = \xi^i_{m,n+1} \qquad (6.5.3)$$

$$G^r_{m,n} = \xi^r_{m,n-1}, \qquad G^i_{m,n} = \xi^i_{m,n-1}$$

Their real and imaginary parts have to be computed separately because the shifts to correctly align them are done in opposing directions. Once these variables are set, the evaluation of $U_{m,n}$ can be done by the single equation,

$$U_{m,n} = E_{m,n} + B_{m,n} F_{m,n} + \beta_{m,n} G_{m,n} \qquad (6.5.4)$$

This saves an addition, subtraction and two multiplications at the expense of some routing and assignments. The same approach can be applied to the calculation of $V_{m,n}$. Overall a saving of about 1.5msecs would have been achieved over the use of Eq.(6.5.2).

6.5.3. Spectral space calculations

The timestep calculations are completed in spectral space with the leapfrog step, time-filtering and the application of damping. The matrices of spectral data are then updated for the new time level.

The leapfrog step can be done in parallel over real and imaginary coefficients since they are stored in the same matrix. The constant $2\Delta t$ is broadcast to the tendencies. The time-filtering is done in the same way. In the model, the diffusion is applied to real and imaginary coefficients simultaneously, with one matrix multiply operation.

6.5.4. Model output

For the periodic output of model results (known as history output), the DAP Data eXpansion (DDX) software developed by the Edinburgh Regional Computing Centre was used. DDX provides a facility for synchronous or asynchronous I/O of a DAP FORTRAN COMMON block to or from EMAS disk files. The code is efficient and does not occupy much memory in the DAP. Transfer rates are typically 250–300Kbytes/sec, although it can vary significantly depending on the machine load.

A difficulty with DDX is that the COMMON block must be aligned to a 4K page boundary (recall that the DAP store also exists as host store). This can only be determined at runtime and so a second compilation is necessary once the appropriate offset has been found. The disk files are non-standard EMAS files and must be handled using special DDX commands only, which can convert them to standard EMAS files.

The prognostic spectral fields are output asynchronously at time t and $t-\Delta t$. The procedure is as follows.

1. Copy the fields into the output COMMON.
2. Normalize the fields.
3. Convert the data from DAP to host storage format.
4. Check that the previous write (if any) has finished.
5. Initiate the next asynchronous transfer.

The space required for the I/O COMMON is not substantial and can be accommodated within the available DAP store. The estimated time for the transfer is about 3secs, assuming a transfer rate of 250Kbytes/sec. With approximately 0.5 CPU seconds per timestep and output at intervals of more than 1 hour, there was no I/O overhead. This was confirmed by comparing the elapsed wall-clock time and the DAP CPU time of the job.

6.5.5. Programming environment

One important issue in the use of any fast computer is the programming environment; how easy the programming language is to use, what support is provided, what facilities or tools for debugging are available and so on. The DAPs at Edinburgh University, in the author's experience, offered a reasonable environment. Interactive use of the DAPs meant testing could be done quickly. User support was also good. A DAP subroutine library similar to the NAG library was provided. Submission of batch jobs was straightforward and a minimum of job control language (JCL) was required. Most jobs consisted of normal foreground operating system commands.

The main difficulties were encountered in the DAP FORTRAN language itself and the relationship between the DAP and the host. Although the array syntax of the language meant much neater and succinct code, useful FORTRAN77 constructs such as IF-THEN-ELSE were not available.

Initialization of data also required some thought. Whether data was initialized in BLOCKDATA subprograms in the DAP or host depended on which parts of the program the data was used and the loading sequence of the data blocks. The transfer of data between the DAP and the host also required some thought.

By far the biggest problem with the DAP was in debugging programs. As DAP FORTRAN has no WRITE statement, variables' values could not be obtained at strategic places in the code. The best method of obtaining diagnostics was by deliberately halting the program using the DAP FORTRAN ERROR statement (ICL, 1979). This had the same effect as a zero-divide say, and the values of all the variables in the calling subroutine and COMMON blocks were printed out. Debugging was generally time-consuming.

6.6. Storage requirements and performance

6.6.1. Storage requirements

The output produced by the DAP consolidator, which links the code to produce an executable binary, can be used to produce statistics on the use of the DAP store (Table 10). The largest requirement is for the program's COMMON blocks. The system and system workspace both require small amounts of

Storage area	Size (kbytes)	Number of planes occupied	Percentage of total
Program code	94.74	190	4.6%
System	14.0	28	0.68%
Workspace	18.0	36	0.88%
Stack	740.0	1480	36.1%
User COMMON	1181.0	2362	57.7%
Total	2047.74	4096	

Table 10.

Storage required for the DAP shallow-water model. Requirements shown are for the main sections of the program, as given by the DAP consolidator listing. Sizes are shown in kbytes and number of planes occupied.

memory. The remaining memory is used as stack, enough for the program's requirement. Unfortunately, the consolidator does not produce any statistics on the maximum stack used.

The requirements for the individual COMMON blocks can also be obtained from consolidator output and are shown in Table 11. The largest area is for the Legendre polynomials and their derivatives. The figures given in Table 11 are for the nonsymmetric Legendre transforms. For the symmetric versions, the space for the Legendre values is 704 planes, giving a total user space of 1658 planes (829kbytes). The Legendre data array now accounts for 42.5% of the total user data area instead of 60%. A significant decrease but still a large fraction. This is an inherent problem of spectral models. There is also a large amount of unused memory within these areas. For example, spectral space variables and constants use only 46% of their allocated space. Similarly, the Legendre polynomial data use 68.8%.

6.6.2. Performance

The CPU time per timestep for the shallow-water model was measured as 0.603secs. The variables are integrated sequentially so it is possible to run the model as a nondivergent model solving just the vorticity equation. The CPU time per timestep for the nondivergent configuration was measured as 0.305secs. The vorticity model has about half the cost of the shallow-water model.

In Table 12, timings of the main routines called in each timestep are presented, together with the routines that each one calls. The routines SC1 to SC4 are the four scans that compute the velocities and integrate the vorticity, divergence and geopotential respectively. The nondivergent model is obtained by calling SC1 and SC2 only. These four routines call other routines than those shown, for the calculation of gridpoint space products and to apply diffusion. However, these other routines were not timed as they contain only a few matrix operations and accurate estimates can be made.

The routines were timed by calling each one repeatedly within a DO loop. The CPU time of the DAP part of the program was reported by the operating system to the nearest whole second. Hence, each routine was called several thousand times to decrease the error in the timing. Each timing shown has a

Use	Size (kbytes)	Planes	Percentage of total
Main variables	128.0	256	10.9%
Auxiliary variables (velocity)	64.0	128	5.4%
Work COMMON	80.0	160	6.8%
Legendre values	704.0	1408	59.8%
Matrix constants	80.0	160	6.8%
Vector constants	12.0	24	1.0%
Scalar data	0.5	1	0.04%
Logical masks	3.0	6	0.25%
Global diagnostics	10.5	21	0.89%
Output COMMON	96.0	192	8.2%
Total	1178	2356	

Table 11.

Storage requirements for user data for the spectral model showing the size in kbytes, number of planes and percentage of total.

Routine name	Time per call (msec)	Main subroutines called
SC1	142.0	UV, ILEG(2), IFFT(2)
SC2	155.0	ILEG, IFFT, DFFT(2), DLEG, TSTEP, FU
SC3	133.0	DFFT, DLEG(2), TSTEP, FU
SC4	154.0	ILEG, IFFT, DFFT(2), DLEG, TSTEP
DFFT	15.4	GSFFTCOL, POSTDFFT
IFFT	14.6	PREIFFT, CTFFTCOL
POSTDFFT	3.79	NONE
PREIFFT	3.57	NONE
GSFFTCOL	11.6	NONE
CTFFTCOL	10.5	NONE
DLEG	56.7	NONE
ILEG	51.8	NONE
UV	5.94	NONE
TSTEP	0.362	NONE
FU	0.914	NONE

Table 12.

The execution times of the main subroutines of the spectral model and the main routines that each calls.

possible error of ±1 to the last digit. The cost of the DO loop is accounted for in these timings.

The time taken for each scan is roughly the same. Although the divergence equation involves more terms than the geopotential or vorticity equation, the routine SC3 takes less time than SC2 (vorticity) or SC4 (geopotential) because the nonlinear products involving U and V have already been computed by SC2 for the vorticity equation. The total time for the scans is less than the CPU time per model step. This can be attributed to additional computations taking place on entry and exit from the model and within the main time loop. However, the four scans account for 97% of the CPU time per step.

The spectral model of Fishbourne (1980) had a CPU time per step of 1.38secs, a factor of 2.3 more than the model of this chapter. This is due to several reasons, but mainly to more expensive transform routines. As Fishbourne did not store the real and imaginary parts of spectral variables in the same matrix, spectral space calculations took twice as long. The fastest version of Fishbourne's gridpoint shallow-water model took 0.05 CPU seconds per timestep for a 64x64 grid. This can be multiplied by a factor of 2 as the spectral model uses a 128x64 grid although, as pointed out by Jarraud and Simmons (1984), an equivalent gridpoint model would have a slightly larger grid than this because of the superior accuracy of the spectral technique. This means the spectral model of this chapter takes 6 times as long as an equivalent gridpoint model. The use of the symmetric Legendre transforms will improve this, but it does suggest that spectral models are not as suited to implementation on the DAP as gridpoint models.

The performance rates of the nonsymmetric inverse and direct Legendre transforms were given in chapter 5 as 5.7Mflops and 8.5Mflops respectively. The performance rates of the inverse and direct Fourier transforms (IFFT and DFFT in Table 12) are 15.9Mflops and 12.3Mflops respectively. These are better than the Legendre transforms because for most of the algorithm, all the processors are doing useful work. As the spectral transforms account for most of the computation in the model, an accurate performance rate for the model will be obtained by considering these routines. This gives a rate of 8.7Mflops, demonstrating the dominance of the performance of the Legendre transforms. This performance is 32% of the peak performance of the DAP

(27Mflops for 32-bit floating point addition) or 45% of the performance for an equal number of floating point additions and multiplications. For comparison, the shallow-water gridpoint models of Fishbourne (1980) achieved a performance of 10Mflops (see chapter 4).

6.6.2.1. Performance of the transforms

Table 12 shows that the spectral transforms account for 94.4% of the CPU time per model step. In each step there are four calls each to the inverse FFT, direct and inverse Legendre transforms and five calls to the direct FFT. The Legendre transforms alone account for 72.0% of the CPU time. The total time for the spectral transforms is 569.4msecs per step, with 434msecs for the Legendre transforms and 135.4msecs for the FFTs.

The spectral transforms of Fishbourne's (1980) model accounted for 99.2% of the CPU time per step, with the Legendre transforms using 82.7%. The time for his FFTs was 218.2msecs and 1150.4msecs for the Legendre transforms. The FFTs in Fishbourne's model are therefore 1.6 times more expensive. This is because he used 128 point real transforms rather than 64 point complex ones. His Legendre transforms are 2.7 times slower than the nonsymmetric routines developed for this model. There are a number of reasons for this. The Legendre polynomial values for Fishbourne's model are packed and unpacking incurs additional cost. This affects the inverse transform, for which the performance ratio is 1.9. In the Fishbourne model the Legendre polynomial derivatives are calculated for the direct transform at each step. The performance ratio for the direct transforms is 3.3. This illustrates the efficiency of the Legendre transform algorithms used for this model. The ratio of the Legendre transform time to that of the FFT is 3.2 for this model. This is typical for low resolution models, the T63 ECMWF model had a ratio of 4.

Now that the model timings are available, it is possible to estimate what the CPU time per step would be if the symmetric transforms were used. Recall that the measured time of the inverse symmetric Legendre transform was 32.8msecs, and the estimated attainable time for the symmetric direct Legendre transform was 41.5msecs. This means that the total time spent doing Legendre transforms decreases from 434msecs to 297.2msecs, a reduction by a factor of 1.46. Subtracting this decrease from the measured time of the model gives a time per step of 0.466secs. Of this, the Legendre

transforms take 64%, the FFTs 29% and the spectral transforms as a whole take 93%. The ratio of the CPU time for Fishbourne's Legendre transforms (that do not use the symmetry property) and the symmetric transforms is 3.9. Comparing the new model CPU time to an equivalent gridpoint model now gives a factor of 4.7.

6.6.2.2. Parallel processing performance

Aside from the CPU time per step and timings of individual subroutines, the program's parallel processing performance has to be examined i.e. how many of the processors are kept busy, what the inefficient parts of the program are, what amount of routing is necessary. As described in chapter 2, a useful performance measure for SIMD processor arrays is the weighted fraction of the PE array kept doing useful work during the program as defined by Eq.(2.6.3). This of course does not give a complete picture, a poor algorithm may still keep all processors busy. The timings of the routines are also important in this context. Comparisons with models on other architectures are interesting, but because of algorithm differences are of little value. A more relevant comparison would be between implementations on different SIMD machines, particularly with different array sizes where the percentage of the PE array kept busy may be less or more important. Comparison with the spectral model of Fishbourne (1980) has been useful.

By estimating the times for the routines called within each scan that were not timed, it is possible to calculate the percentages of the model CPU time during which computations take place in either spectral, Fourier or gridpoint space outside the spectral transforms, as shown in Table 13. To compute an average efficiency for the model, Eq.(2.6.3) is used by dividing the model CPU time per step into time slices for spectral space, Fourier space and gridpoint space calculations and the transforms. Using Table 13 and the efficiencies for the nonsymmetric Legendre transforms given in the previous chapter, an average efficiency of 63.5% is obtained. In other words, during the model run, on average two-thirds of the PE array is doing useful computation.

This percentage is strongly influenced by the efficiencies of the Legendre and Fourier transforms. So, although a mean efficiency of 63.5% may suggest scope for improvement, only improvements to the efficiency of the transforms will give significant increases in this mean value.

Representation	Percentage of time per step
Gridpoint	0.74%
Fourier	0.42%
Spectral	2.2%

Table 13.

The estimated percentage of the time per step that the data is in spectral, Fourier or gridpoint forms, excluding time spent during the spectral transforms.

It is possible to determine what fraction of the model time is spent routing and broadcasting data by noting the number of calls to the relevant DAP FORTRAN functions (MATR, MATC, REVR, REVC and the shift functions). Using the timings of these functions from chapter 2 the total time spent routing data is 52.77msecs per step with the nonsymmetric Legendre transforms and 58.21msecs with the symmetric transforms. If the time per step for the two cases is 603msecs and 466msecs, routing accounts for 8.8% and 12.5% of the total time per step respectively. Fishbourne (1980) reports a percentage of 11.6% for his model. Most of the routing takes place in the spectral transforms, with roughly equal routing in the FFTs as in the Legendre transforms. The increase in routing for the symmetric transforms is because the symmetric and antisymmetric coefficients are treated separately.

Routing can be regarded as an overhead for a parallel implementation of the model on the DAP. The routing overheads of this model are therefore not excessive. They are comparable to the multitasking overheads for the ECMWF spectral model i.e. 9% (Dent, 1988). The routing in the FFTs is 22% of the CPU time for this routine but the routing for the inverse and direct Legendre transforms is only 6.5% and 3.3% of the CPU time respectively, although each routine spends roughly similar times doing routing operations.

6.7. Legendre transforms at different resolutions

Using the timing estimates and the overheads for the symmetric Legendre transforms, it is possible to estimate the CPU time for the routines at the resolutions, T21, T62 and T85. To make these estimates, a sheet mapping is used for when the data array size is greater than the PE array size. Also, the same basic algorithm is used and no attempt is made to devise new, more efficient, algorithms.

At T21, there are 16 latitudes from the pole to the equator. If the Legendre data are separated into odd and even n about the centre column of the DAP array, the transforms are almost identical to those at T42. The difference is that the loop is now half the length. The estimated T21 inverse and direct transform times are 17.4msecs and 24.2msecs respectively. These times could possibly be improved since in theory four values of n could be computed simultaneously. However, more broadcasting and routing of data would be required, so it is not obvious how much improvement, if any, would occur.

At T62, the spectral coefficients of the velocities would not be able to use the same mapping as the rest of the spectral variables. The assumption is that the real and imaginary coefficients are stored separately for U and V only but this does not affect the time of the transforms. All the other spectral variables however, can use the same mapping as the T42 model.

For the Legendre coefficients, 48 latitudes of data must be stored. With odd and even n values separated as before they cannot be held together in the same matrix and as a consequence the odd and even n computation has to be done sequentially. This effectively doubles the code in the loop over n for the inverse and direct transforms. The Fourier coefficients would also need two matrices for both the real and imaginary values and therefore additional code is required when forming the symmetric and antisymmetric values. During the loop in the direct transform, as there are no free rows in the Legendre data array, the Fourier work matrix has to be updated on every pass. When all these changes to the T42 algorithms are made, the T62 inverse and direct transform estimated timings are found to be 81.0msecs and 112.9msecs respectively.

At a resolution of T85, 64 latitudes need to be stored. Separating odd and even n as before, the latitudes now use all the columns of the PE array unlike the T62 resolution. Additional array space is required to store all the coefficients. Partitioning the polynomials and their derivatives across four matrices would leave 41 rows free, so that the Fourier work matrices only have to be updated after 41 passes during the direct transform. For the inverse transform, if $n \leq 63$ the same code as the T62 case can be used in the loop. However, once $n>63$, the $P_{m,n}$ occupy additional matrices and extra code is required for the coefficients with $m>63$. Additional work arises at this resolution in both transforms as the space required for the Fourier and spectral coefficients is double the T62 case. All these modifications to the code give an estimated time for the inverse and direct transforms of 166.0msecs and 283.3msecs respectively. These estimates may be optimistic as the total storage area required for the Legendre data now exceeds the total DAP memory for the DAPs at Edinburgh University. Overheads from the I/O of Legendre values during the transforms may increase the estimated timings.

6.7.1. Comparison with serial routines

Serial versions of the Legendre transforms were written in the IMP80 programming language and timed at T21, T42, T62 and T85 on the AMDAHL 470/V8 computer running the EMAS-3 operating system at Edinburgh University. The code was optimized by the IMP compiler. Each routine was called four times in a loop and an average of the last three times was taken to avoid the dynamic loading overhead when the routine was first called.

Fig. 26 is a graph showing the variation in CPU time for the AMDAHL and DAP versions of the transforms. The serial versions clearly show the cubic variation of processor time with truncation. Between T21 and T42, the DAP times vary linearly with M, as the same T42 code is used but the loop length varies. Although the T42, T62 and T85 DAP times are joined by straight lines, times for intermediate resolutions would not lie on these lines. For example, at T43, extra space is required for the Legendre data so that the same data mapping as for T62 would have to be used and would mean a larger increase in time than depicted in the graphs. A more economical approach might be feasible using a vector array. The increased slope of the line joining the T42 and T62 times shows the additional cost as the odd and even n computations are no longer done in parallel. The increased slope between the T62 and T85 times is a consequence of the additional work required as the Legendre and spectral data use more space.

There is a clear performance gain at the higher resolutions for the DAP. For the inverse transform, the ratio of the DAP and serial times is 3.6 at T42 and 5.2 at T85. For the direct transform, the ratios are 5.6 and 8.8. The T21 resolution however, is too low to make effective use of the 64^2 PE array. The ratios are greater for the direct transform because the products involving the Legendre polynomials and their derivatives are computed in parallel in the DAP versions. This is shown by the ratio of the direct transform time to that of the inverse transform. For the serial versions the ratio is about 2.85 for all resolutions, whilst for the DAP versions it is 1.4, except at T85 when it becomes 1.7.

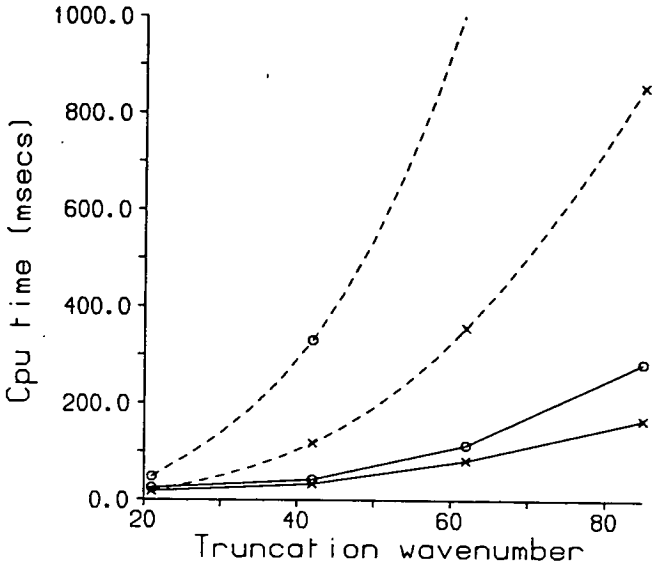


Figure 26. Variation of the CPU time with the truncation wavenumber for the Legendre transform algorithms. Solid curves represent the DAP routines, dashed curves the serial routines. Inverse transform times are shown with a cross, direct transform times with a circle. Taken from Carver (1988).

6.8. Model results

In this section, the model results are compared with the previously published results of Doron *et al.* (1974). They compare spectral and gridpoint models by using Rossby-Haurwitz waves as initial conditions. The Rossby-Haurwitz wave is an exact solution of the nondivergent barotropic equation but not for the divergent case. Zonal wavenumber 4 is stable and moves east at about 11° per day. However, zonal wavenumber 8 is unstable and breaks down within 5 days. Doron *et al.* use a rhomboidal spectral model and present results at two resolutions. The highest of these, R31, is comparable in resolution to the T42 model of this chapter.

6.8.1. Initial conditions

The initial streamfunction given by Doron *et al.* (1974) is,

$$\psi = -\omega\mu - \frac{K}{n(n+1)} P_{m,n}(\mu)\cos m\lambda \qquad (6.8.1)$$

with K as -0.8776 for both wavenumbers 4 and 8. The value of ω is chosen to give a super-rotation of $50 ms^{-1}$ at the equator. The fluid depth is 8km. The definition of $P_{m,n}$ used by Doron *et al.* for their spectral model differs from Eq.(3.4.18) in the normalization constant and to be consistent, for this model Eq.(6.8.1) becomes,

$$\psi = -\omega\mu - \frac{K}{n(n+1)\sqrt{2}} P_{m,n}\cos m\lambda \qquad (6.8.2)$$

To determine the initial vorticity coefficients start with the expansion,

$$\psi = \sum_{m=-M}^{M} \sum_{n=|m|}^{M} \psi_{m,n}(t) P_{m,n} e^{im\lambda} \qquad (6.8.3)$$

Since,

$$P_{0,1} = \sqrt{3}\,\mu$$

and assuming one nonzero coefficient, Eq.(6.8.3) becomes,

$$\psi = \sqrt{3}\mu\psi_{0,1} + P_{m,n}[(\psi_{m,n}e^{im\lambda})^* + \psi_{m,n}e^{im\lambda}] \qquad (6.8.4)$$

where the properties Eq.(3.4.19) and Eq.(3.4.27) have been used. Comparing Eq.(6.8.4) and Eq.(6.8.2) gives,

$$\begin{aligned} \psi_{0,1} &= -\omega / \sqrt{3} \ . \\ \mathcal{I}m\{\psi_{m,n}\} &= 0 \\ \mathcal{R}e\{\psi_{m,n}\} &= -K / [2\sqrt{2}\, n(n+1)] \end{aligned} \qquad (6.8.5)$$

Using Eq.(6.2.5) gives the initial vorticity coefficients as,

$$\begin{aligned} \xi_{0,1} &= 2\omega / \sqrt{3} \\ \xi_{m,n} &= K / 2\sqrt{2} = -0.31 \end{aligned} \qquad (6.8.6)$$

The geopotential coefficients are calculated from the divergence equation Eq.(6.2.10) by assuming the initial divergence and divergence tendency are zero.

6.8.2. Rossby wave results

The wavenumber 4 and 8 cases described by Doron *et al.* (1974) were run for the T42 model. A timestep of 5 minutes was used, whilst Doron *et al.* use a semi-implicit timescheme with a timestep of 30 minutes.

Contour maps of the total geopotential ($\bar{\phi}+\phi'$) are presented on a polar stereographic projection for the northern hemisphere. Graphs of the main spectral coefficients of vorticity are also presented. To calculate the magnitude of the waves, following Hoskins (1973), the spectral coefficients are written in amplitude-phase form,

$$\begin{aligned} F_{m,n} &= \tfrac{1}{2}A\exp(i\theta_{m,n}) \qquad \text{for } m \neq 0 \\ F_{0,n} &= A\exp(i\theta_{0,n}) \qquad \text{for } m = 0 \end{aligned} \qquad (6.8.7)$$

since there is no complex conjugate coefficient for m=0. This gives,

$$A = 2|F_{m,n}| \quad \text{for } m \neq 0$$
$$A = |F_{0,n}| \quad \text{for } m = 0 \qquad (6.8.8)$$

For comparison with the results of Doron *et al*, the amplitude A is normalized by dividing by the Earth's angular velocity. All amplitudes are multiplied by $\sqrt{2}$ so that values on the graphs may be compared to those of Doron *et al*.

6.8.2.1. Rossby wavenumber 4

In Fig. 27, the results from the R31 model of Doron *et al.* (1974) are reproduced for a zonal wavenumber 4 together with the T42 results. The treatment of the wave appears to be identical in both cases. A weak north-west/south-east tilt of the troughs and ridges is evident by day 3. This transfers momentum southwards weakening the zonal flow in mid-latitudes and produces a closed low. The tilting of the pattern weakens until by day 7 it has disappeared. The average eastward displacement in the T42 model over the 7 days is 11.2° per day.

Fig. 28, from Doron *et al*, shows some of the principal spectral components as they vary with time. In Fig. 29, the variation of the same spectral components for the T42 model are shown. Agreement is good but the curves are not identical. The T42 model decreases the amplitude of the (0,7) and (4,7) components slightly from day 6 to 7 whilst Fig. 28 does not appear to show this.

6.8.2.2. Rossby wavenumber 8

Fig. 30 shows the results from Doron *et al.* for a Rossby-Haurwitz zonal wavenumber 8 and the results from the T42 spectral model. As before, the integrations are in good agreement. The wave moves eastward at 30° per day in both models.

A tilt in the pattern again develops and persists, transferring energy from the main wave to the zonal flow. The breakdown of the wave at day 5 in the T42 model appears identical to the R31 model results.

The principal spectral components from Doron *et al.* are shown in Fig. 31, whilst those from the T42 model are shown in Fig. 32. The curves appear identical in both cases.

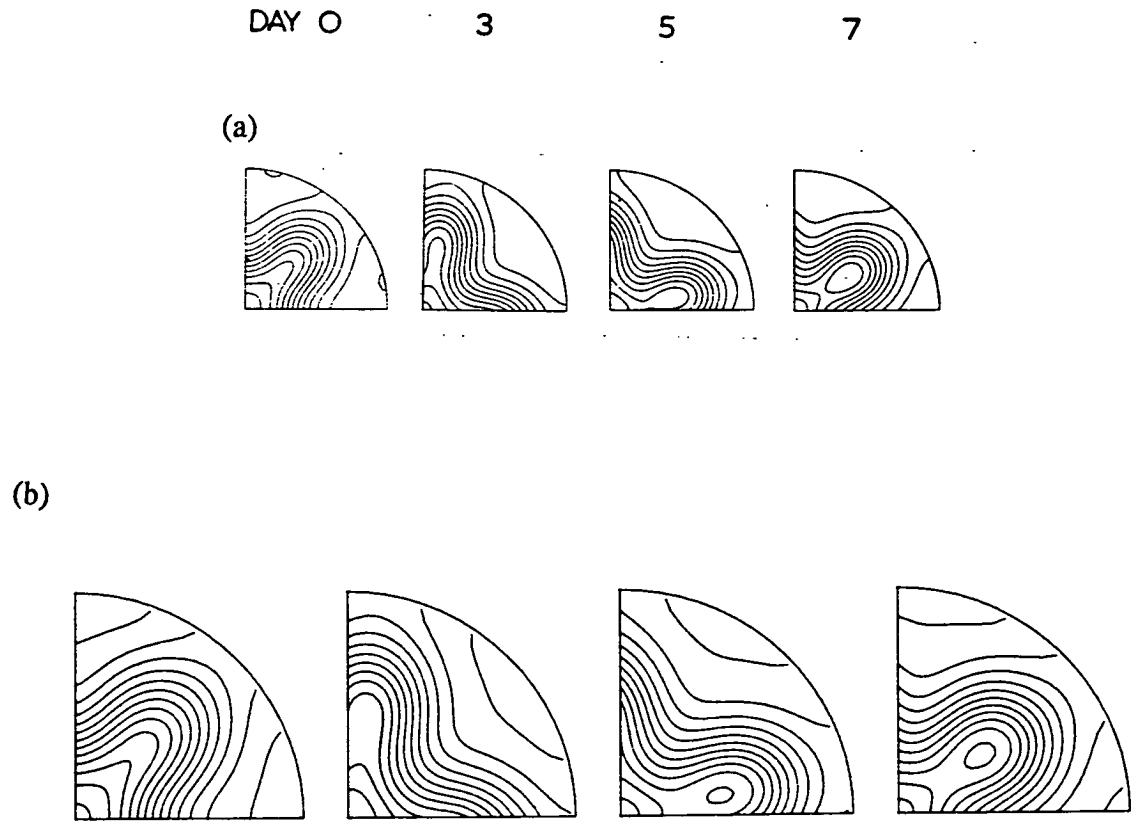


Figure 27. (a) Reproduced from Doron *et al.* (1974) showing polar stereographic projections of the height field at days 0, 3, 5 and 7 for a Rossby-Haurwitz wave of zonal wavenumber 4 for the high resolution spectral model, (b) The geopotential field for the T42 model.

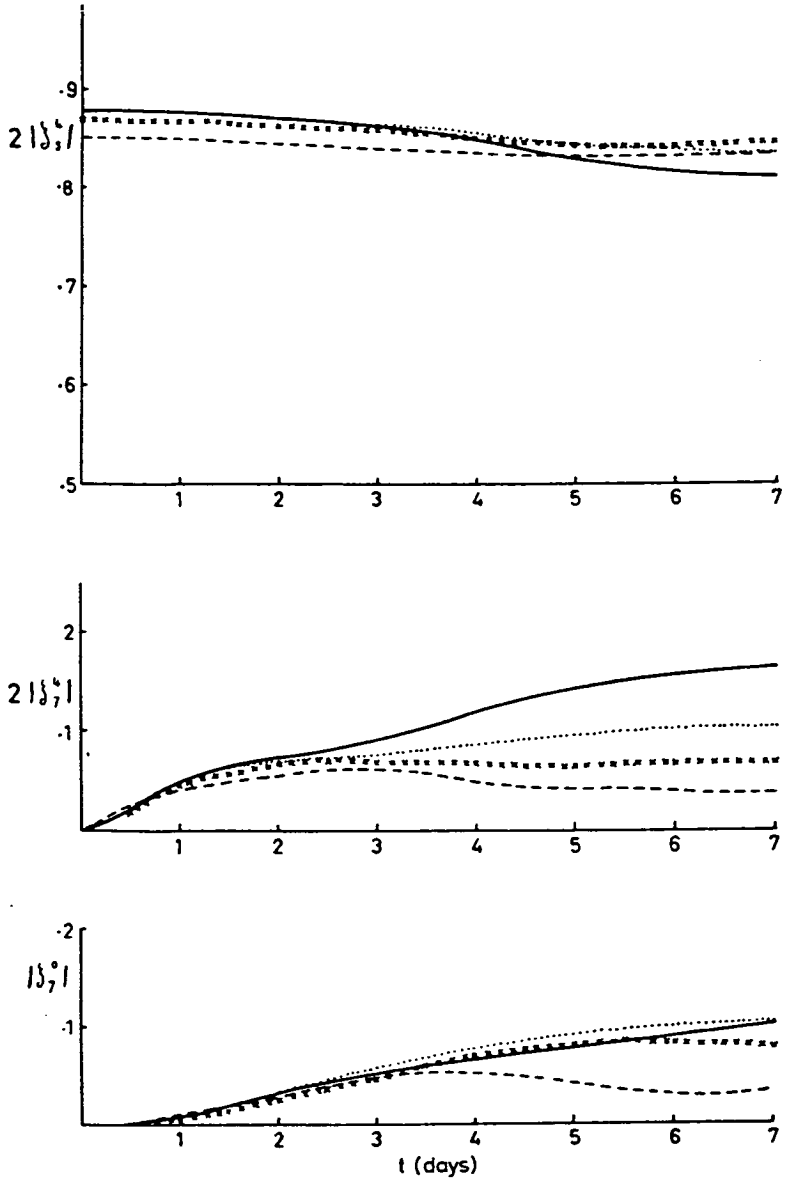


Figure 28. The variation in time of some principal spectral components of vorticity for the wavenumber 4 integrations of Doron *et al.* (1974). Solid line is the low resolution spectral model. Dotted line is the high resolution spectral model. Dashed line is the low resolution gridpoint model. Crossed line is the high resolution gridpoint model.

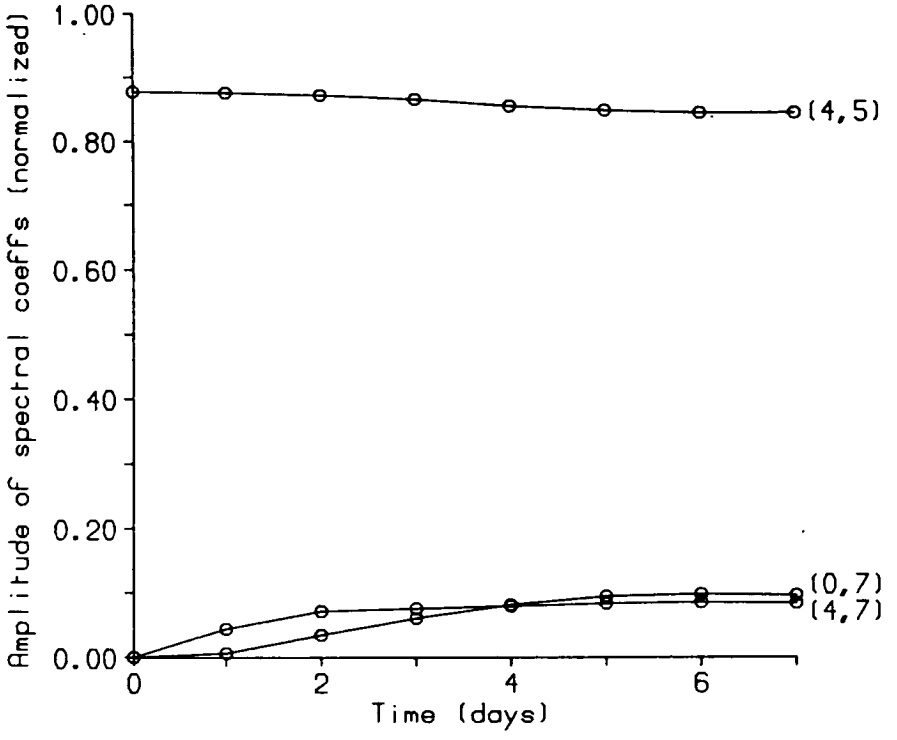


Figure 29. The variation in time of some principal spectral components of vorticity for the zonal wavenumber 4 integration with the T42 spectral model. Amplitudes are corrected for the different normalization of the Legendre polynomials to that of Doron *et al.* (1974). Points are joined by straight lines and circles indicate times at which output was available.

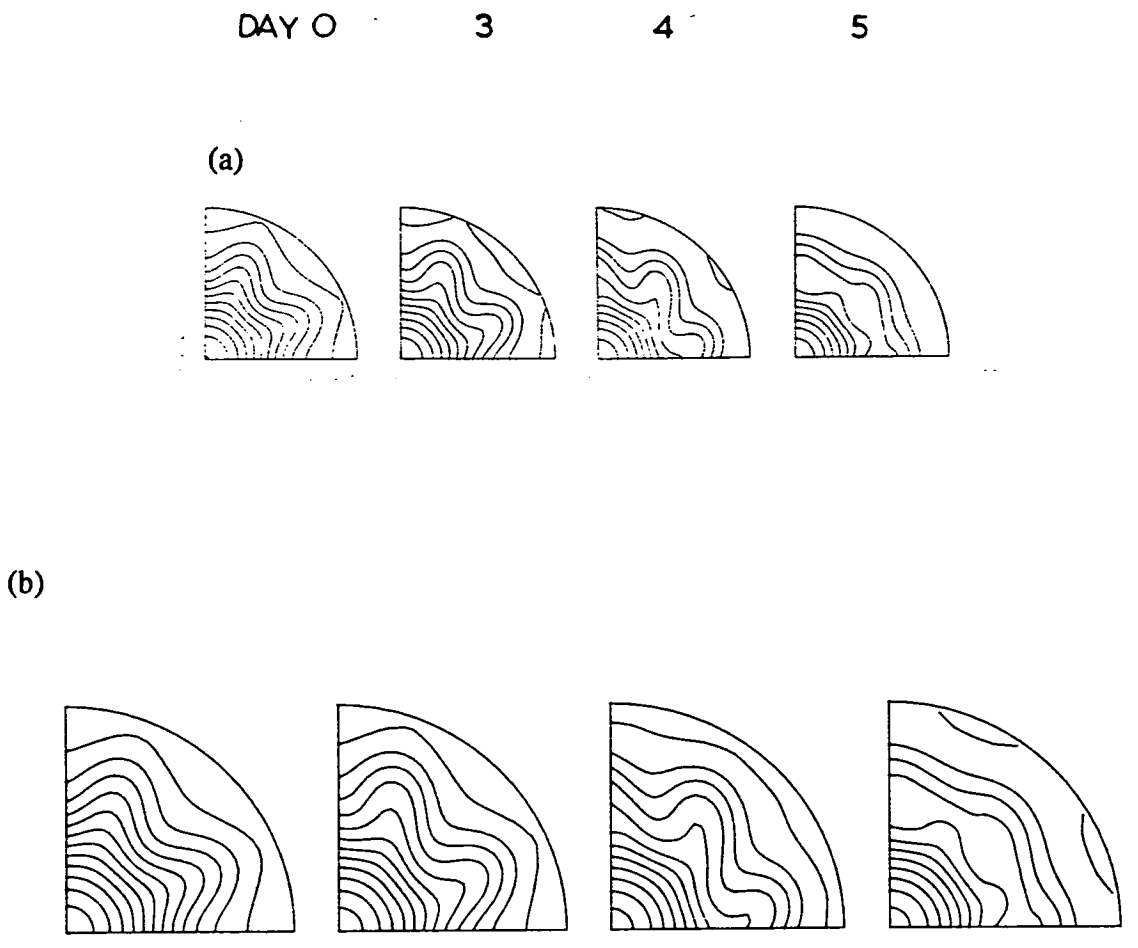


Figure 30. (a) Reproduced from Doron *et al.* (1974), showing polar-stereographic projections of the height field at days 0, 3, 4 and 5 for a Rossby-Haurwitz wave of zonal wavenumber 8 for the high resolution spectral model, (b) The geopotential field for the T42 spectral model.

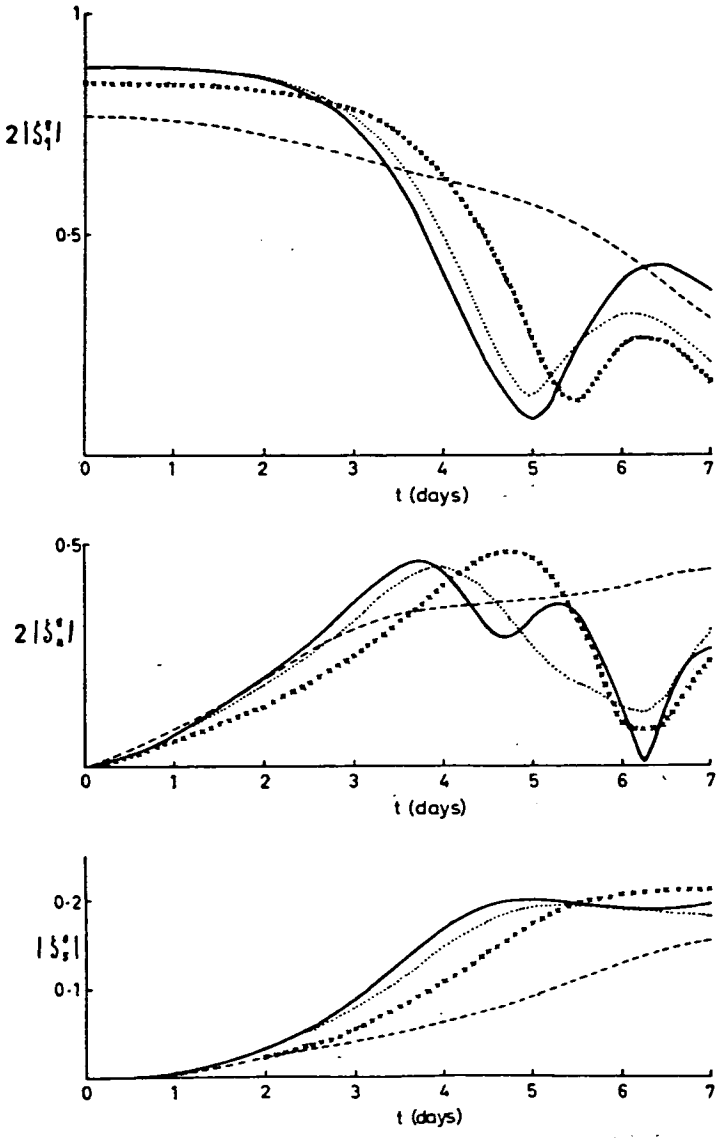


Figure 31. The variation in time of some principal spectral components of the vorticity in the zonal wavenumber 8 integration. Reproduced from Doron *et al.* (1974). Key as for Figure 28.

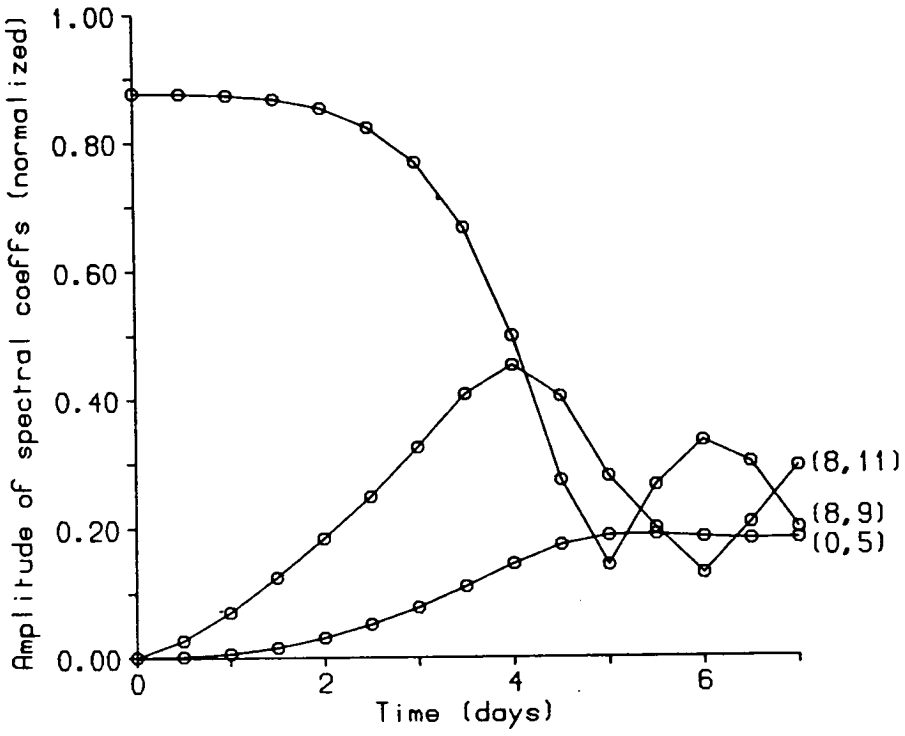


Figure 32. The variation in time of some principal spectral components of vorticity for the wavenumber 8 integration of the T42 model. Amplitudes are corrected for different normalization of Legendre polynomials to that of Doron *et al.* (1974). Circles represent times at which output was available and are joined by straight lines.

6.9. Conclusions

In this chapter a T42 spectral shallow-water model was designed and implemented on the ICL DAP. As the model execution time depends mainly on the transforms, further improvements to these routines would benefit the model most. The Fourier transforms could be improved as the number of gridpoints contain factors of four as described in Hockney and Jesshope (1981). A reduction in CPU time of 10% might result. For the Legendre transforms, no similar optimizations exist. Only minor optimizations would be possible since the major parts of the operations are tied to the storage format of the Legendre data. One optimization might be the use of lower precision arithmetic. This could perhaps be applied to all parts of the model, although it is not clear if it would be successful or possible for the transforms, particularly the Legendre transforms. Special summation algorithms might be required to preserve accuracy. For example, always summing from the highest total wavenumber to the lowest to accumulate the smallest contributions first. The use of lower precision arithmetic would require much study. Another optimization would involve writing the inverse Legendre transform algorithm in the DAP assembly language APAL to exploit the available processors, as described in chapter 5.

In concluding this chapter, the fundamental question to answer is whether the DAP is suited to spectral models. The answer is dependent on the problem to be implemented, for the DAPs at Edinburgh University will suit only medium resolution models with just a few levels. High resolution models require a large quantity of Legendre data and hence need a substantial amount of memory and the lack of a fast I/O facility would undoubtedly reduce the efficiency of the Legendre transforms. Only a few levels would be practical because of memory requirements. However, the addition of more levels adds another dimension to the number of potential processes and more efficient transform algorithms may result. This is an obvious next step to this work. The DAP is also suited to higher resolution models as evidenced by Fig. 26. The resolution of the model of this chapter is the lowest that would be worthwhile implementing on a DAP of this size, given the speedup achieved over the serial machine and the extra effort involved in designing the algorithms. One conclusion is that the DAP should have a higher ratio of memory to the number of processors and a fast I/O facility to be more suited to spectral models. The new DAP-310 satisfies both these requirements.

Larger array sizes on the other hand, will further restrict the choice of resolution for efficiency reasons and be more suited to proportionally higher resolutions.

A triangular truncation was used for this model, substituting a rhomboidal one would require some remapping of data but the Legendre transform algorithms would remain largely unchanged. It would be interesting to extend this work to consider what the best spectral truncations are for the DAP and the type of meteorological problems they would be used to study.

Finally, the Legendre transforms pose implementation problems whatever parallel computer is used. A comparison of techniques on different parallel computers is presented in chapter 8.

CHAPTER 7

A MESOSCALE FINITE ELEMENT MODEL ON THE ICL DAP

7.1. Introduction

In this chapter the implementation of a finite element meteorological model on the ICL DAP is described. Although the finite element technique is used infrequently in meteorology its use is growing. Recent studies using the finite element method for applications on the DAP have shown it to be well suited to the array processor architecture (Lai and Liddell, 1987a).

As was shown in chapter 3, the finite element technique involves different algorithms to those of gridpoint and spectral methods. Parallel algorithms for the solution of finite element systems of simultaneous equations is an area of current research. An aim of this chapter is to establish the algorithm best suited to finite element meteorological models. An obvious requirement for a time dependent problem is a fast algorithm. New parallel algorithms will also be required for evaluating the finite element matrices. Like the implementation of the spectral model, the DAP will affect the formulation of a finite element model and another aim of this chapter will be to study the differences between formulating the model on a serial and parallel machine.

The model had to be computationally cheap and simple to be implemented in a short time, but also be a realistic meteorological model so that typical problems in formulating finite element models for the DAP had to be solved. Since the spectral method is more accurate globally, the finite element method is usually used for limited area studies. The choice of model was the two-dimensional dry model of Orlanski and Ross (1977), hereafter referred to as OR77, which was used to simulate the development of a cold front.

The next section contains a description of the model and the following two sections describe the formulation of the finite element version. The model equations are then described followed by their implementation on the DAP. After describing the parallel algorithms to compute and solve the finite element matrices, an analysis of the model's storage requirements and performance is presented. Results from the model are then compared to those of Orlanski and Ross (1977). Finally, conclusions are presented and discussed.

7.2. Description of the model

7.2.1. Equations

The mesoscale model employs the deep anelastic equations formulated by Ogura and Phillips (1962) with a hydrostatic approximation. They are written in Cartesian coordinates (x,y,z) with the y-coordinate running parallel to the axis of the front. The numerical solution is assumed to be two-dimensional with all the variables independent of y. A vorticity, ξ, is used to represent the velocity field in the x-z plane. The potential temperature, θ, within the frontal system is determined by a y-velocity jet, v, initially in geostrophic balance with the temperature gradient across the front. A large scale geostrophic wind, $U_g(z)$, advects the front in the x-direction, where U_g is taken to be uniform in x. A horizontal temperature gradient in the y-direction is included for consistency with the large scale wind through the thermal wind relation. The Coriolis parameter, f, is assumed to be constant and all moist processes are neglected.

The prognostic variables are ξ, v and θ and their rates of change are given by,

$$\frac{\partial \xi}{\partial t} = -u\frac{\partial \xi}{\partial x} - \frac{w}{\alpha_0}\frac{\partial(\alpha_0 \xi)}{\partial z} + f\frac{\partial v}{\partial z} - \frac{g}{\theta_0}\frac{\partial \theta}{\partial x} + \frac{\partial}{\partial x}(K\nu_e\frac{\partial \xi}{\partial x}) + \frac{\partial}{\partial z}(\nu_e\frac{\partial \xi}{\partial z}) \qquad (7.2.1)$$

$$\frac{\partial \theta}{\partial t} = -u\frac{\partial \theta}{\partial x} - w\frac{\partial \theta}{\partial z} + \frac{f\theta_0}{g}v\frac{\partial U_g}{\partial z} + \frac{\partial}{\partial x}(K\kappa_e\frac{\partial \theta}{\partial x}) + \frac{\partial}{\partial z}(\kappa_e\frac{\partial \theta}{\partial z}) \qquad (7.2.2)$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - w\frac{\partial v}{\partial z} + f(U_g - u) + \frac{\partial}{\partial x}(K\nu_e\frac{\partial v}{\partial x}) + \frac{\partial}{\partial z}(\nu_e\frac{\partial v}{\partial z}) \qquad (7.2.3)$$

The specific volume is given by,

$$\alpha_0 \equiv 1 / \rho_0 = \alpha_s[1 - (gz/c_p\theta_0)]^{-c_v/R} \qquad (7.2.4)$$

with the surface specific volume, α_s, independent of x and θ_0 as the surface potential temperature at the right-hand boundary. A streamfunction, ψ, is defined so that the velocities u and w, in the x and z directions respectively, are given by,

$$\partial\psi/\partial z = u/\alpha_0, \qquad \partial\psi/\partial x = -w/\alpha_0 \tag{7.2.5}$$

The streamfunction is obtained from,

$$\xi = \frac{\partial}{\partial z}(\alpha_0 \frac{\partial \psi}{\partial z}) \tag{7.2.6}$$

The turbulent fluxes of momentum and heat are parametrized using an eddy viscosity parametrization developed by Orlanski and Ross (1973) and Orlanski *et al.* (1974) in which the eddy diffusivity takes the form,

$$\begin{aligned} \kappa_e &= \kappa_0 && , \Delta\theta \geq 0 \\ \kappa_e &= \kappa_0[\, 1 + C(\frac{g|\Delta\theta|(\Delta z)^3}{\theta_0 \kappa_0 \nu_0})^{1/3}] && , \Delta\theta < 0 \end{aligned} \tag{7.2.7}$$

where $\Delta\theta$ is the local vertical gridpoint difference in potential temperature. In the OR77 model, the parameters are set to,

$$\begin{aligned} &\nu_e = 0.7\kappa_e, \quad \nu_0 = 0.7\kappa_0, \\ &K = 1000, \\ &\kappa_0 = 5\mathrm{m}^2\mathrm{s}^{-1}, \quad C = 0.75. \end{aligned} \tag{7.2.8}$$

K is included in the momentum equations so that the horizontal eddy viscosity and heat diffusivity are 1000 times larger than their respective vertical values. κ_0 represents the background diffusivity.

7.2.2. Finite difference formulation

The finite difference formulation of the original model was that of Orlanski and Ross (1973) (based on that of Lipps, 1971). Centred space and time differences were used but with the diffusion terms lagged by one timestep. The variables were specified over a staggered grid with the advective terms written as Jacobians in ψ and formulated according to Arakawa (1966) and Lilly (1965), to minimize computational instability. The solution was time-smoothed every 30 timesteps to suppress the mode splitting of the leapfrog scheme used for the model.

The horizontal domain was 1500km with 76 equally spaced points giving a resolution of Δx=20km. The vertical spacing, however, was varied to give an

increased resolution near the ground with Δz=150m at the surface, 300m at 5km and 400m at 15km, the top of the model. There were 51 points in the vertical with a tropopause prescribed at 10km in the temperature profile.

7.2.3. Boundary conditions

The bottom boundary of the OR77 model consists of a level surface on which velocity slip (u and v nonzero) is permitted. The boundary condition for θ is,

$$\partial\theta/\partial z = 0 \quad \text{at} \quad z = 0 \tag{7.2.9}$$

In the interior of the model, v is roughly in geostrophic balance with the horizontal temperature gradient, so the thermal wind relation is assumed to hold at the surface, to give,

$$\frac{\partial v}{\partial z} = \frac{g}{f\theta_0}\frac{\partial\theta}{\partial x} \quad \text{at} \quad z = 0 \tag{7.2.10}$$

The cross-front circulation, represented by the streamfunction and vorticity, requires the use of the simple boundary condition,

$$\psi = 0 \quad \text{at} \quad z = 0 \tag{7.2.11}$$

Since the prognostic equation for vorticity involves diffusion terms, a boundary condition for vorticity is also required at the surface. OR77 used the condition,

$$\xi = 0 \quad \text{at} \quad z = 0 \tag{7.2.12}$$

The numerical domain of the model extends above the tropopause. Although intense perturbations may affect the lower stratosphere, the large static stability above the tropopause will weaken these perturbations. Therefore, rigid-lid boundary conditions were used at the top boundary. That is,

$$\psi = \psi(t{=}0) \tag{7.2.13}$$

and,

$$\frac{\partial \xi(t)}{\partial z} = \frac{\partial \xi(t=0)}{\partial z}, \qquad \frac{\partial v(t)}{\partial z} = \frac{\partial v(t=0)}{\partial z}, \qquad \frac{\partial \theta(t)}{\partial z} = \frac{\partial \theta(t=0)}{\partial z} \quad \text{at the top.} \tag{7.2.14}$$

Instead of rigid or periodic boundary conditions, open boundary conditions were used for the sides of the model to permit propagating waves to escape from the numerical domain. Orlanski (1976) discussed open boundaries in detail. His scheme is based on computing a phase velocity, which includes advection and wave propagation, in the neighbourhood of each boundary point and extrapolating to the boundary. The choice of whether a boundary point exhibits inflow or outflow behaviour is determined by the direction of this phase velocity, rather than the direction of the mean flow.

For any variable ϕ, at timestep $n+1$, for the boundary point $i=b$, its value is given by,

$$\phi_b^{n+1} = \left(\frac{1 - r}{1 + r}\right)\phi_b^{n-1} + \left(\frac{2r}{1 + r}\right)\phi_{b-1}^{n} \tag{7.2.15}$$

where $0 \leq r \leq 1$ and r is given by,

$$r = (\phi_{b-1}^{n-2} - \phi_{b-1}^{n}) \,/\, (\phi_{b-1}^{n} + \phi_{b-1}^{n-2} - 2\phi_{b-2}^{n-1}) \tag{7.2.16}$$

The values of r for the OR77 model variables are computed in the above manner except that for vorticity at both sides and temperature at the right side the values are fixed at 1, corresponding to an outflow velocity of $\Delta x/\Delta t$.

7.2.4. Initial conditions

Orlanski and Ross (1977) considered two types of initial conditions. The first was a surface jet only, using the following expression for the y-velocity field v,

$$v(x,z) = -\tfrac{1}{2}(x/x_0)\, V_m\{1 - \tanh[\beta(x+\alpha z-x_0)]\} \tag{7.2.17}$$

with x_0=500km, β=50km^{-1} and α=100. V_m is a jet intensity parameter. Eq.(7.2.17) produces a jet 4km deep and 600km wide with a maximum velocity at the surface.

The second type was a mid-tropospheric jet given by,

$$v(x,z) = -\tfrac{1}{2}(x/x_0)V_m\{1-\tanh[\beta(x+\alpha z-x_0)]\}+V_M\exp\{-R_j^{-2}[(z-z_j)^2+(\gamma(x-x_j))^2]\} \quad (7.2.18)$$

with $x_j=x_0$, z_j=4km, R_j=8/$\sqrt{2}$km and γ=0.03. This produces a weak surface jet as before but with a strong jet centred at 8km height and directed in the positive y direction.

The θ field is initialized so that the horizontal potential temperature gradient is in geostrophic balance with v from the thermal wind relation. The vertical variation of θ is prescribed on the rightmost boundary (the warm side of the front) with lapse rates of,

$$\begin{array}{ll} 2\times10^{-3}\ ^\circ\text{C/m} & \text{for } z < 2\text{km} \\ 4\times10^{-3}\ ^\circ\text{C/m} & \text{for } 2\text{km} \le z < 10\text{km} \\ 14.5\times10^{-3}\ ^\circ\text{C/m} & \text{for } z \ge 10\text{km} \end{array} \quad (7.2.19)$$

with θ_0 (not given by OR77) assumed to be 15°C, the mean sea-level temperature at mean sea-level pressure (McIntosh and Thom, 1981).

The initial vorticity field can be set using the large scale wind U_g. From Eq.(7.2.5) and Eq.(7.2.6) with $u=U_g(z)$,

$$\xi(t=0) = \partial U_g(z)/\partial z \quad (7.2.20)$$

7.3. Formulation of the DAP model

7.3.1. Model domain

The DAP implementation of the OR77 model has 64 points in the horizontal and 64 in the vertical. The horizontal grid spacing is 24km to give a domain width of 1512km. Thus, the resolution in the horizontal is less than for the original model but the superior accuracy of the finite element method should offset this.

For the vertical, the model may use either a constant spacing or the stretched vertical coordinate scheme of OR77. If a constant spacing is used, the increment Δz is 240m, giving a model top of 15.12km. With the stretched vertical coordinate, at the ground Δz=119m compared to 150m in OR77 and Δz=314m compared to 400m at the top of the model, with a model top of 14.85km. Thus, coupled with the use of the finite element method this model

has superior vertical resolution to the original model.

7.3.2. Method of solution

7.3.2.1. Choice of elements

As shown by Staniforth (1987), bilinear rectangular elements, defined by Eq.(3.5.9) and Eq.(3.5.6), are superior to triangular or higher order elements. As discussed in chapter 3, there are several advantages to using these elements. First, fourth order accuracy is obtained for the solution at the nodes of a regular grid. Second, the resulting system of equations can be solved as sets of one-dimensional tridiagonal problems. Last, for the calculation of a simple derivative, the two-dimensional calculation reduces to one dimension. Since efficient and economic solvers for tridiagonal systems exist for the DAP, the use of rectangular bilinear elements is made more attractive and are therefore used. The model equations may be solved as a succession of derivative and product computations in contrast to the more traditional technique where all terms are considered together and the resulting stiffness matrix assembled and solved.

7.3.2.2. Choice of grid

Williams (1981) showed that the elements must be staggered if the momentum form of the shallow-water equations is used, whereas no staggering is required for the vorticity-divergence form. If the nodal points for the free surface height are not staggered relative to those of the velocities, the energy in the small scales propagates in the wrong direction. Staniforth (1987) reported that unstaggered formulations also suffer from small scale noise. As vorticity is used in this model, an unstaggered grid is chosen in which values of all the variables are held at each node. A node is identified by integers i and j where x and z are given by,

$$x = i\Delta x, \qquad i = 0,\ldots. 63 \tag{7.3.1}$$

and,

$$z = \sum_{l=0}^{j} \Delta z_l, \qquad j = 0,\ldots. 63 \tag{7.3.2}$$

7.3.2.3. Approximation of variables

The approximation of the model variables with the basis functions Eq.(3.5.9) follows directly from chapter 3. From Eq.(3.5.11), any model variable f can be approximated by,

$$f(x,z,t) = \sum_{i=0}^{63} \sum_{j=0}^{63} f_{i,j}(t) a_i(x) b_j(z) \qquad (7.3.3)$$

where a_i and b_j are the piecewise linear functions in the x and z directions respectively. They are given by Eq.(3.5.6) except that the spacing in the x direction is constant. The nodal values at i=0, j=0, i=63 and j=63 for the appropriate variables are given by the boundary conditions in the previous section.

7.3.3. Calculation of velocities

7.3.3.1. Streamfunction

The first stage of a new timestep is to compute the streamfunction from the vorticity, using Eq.(7.2.6). The streamfunction and vorticity are expanded according to Eq.(7.3.3). The streamfunction's boundary conditions, Eq.(7.2.11) and Eq.(7.2.13), are essential conditions and must be explicitly satisfied.

The finite element method can be applied in the usual way. A truncation error analysis of the resulting scheme (assuming $\alpha_0(z)$=1 for all z) shows only first order accuracy on the irregular vertical spacing of the model (the horizontal mass matrix cancels from the LHS and RHS as for the first derivative case). However, Beland and Beaudoin (1985) have devised a scheme, outside the Galerkin finite element framework, for the second order equation,

$$d^2y/dx^2 = f$$

which is fourth order accurate on an irregular grid but remains as complex as the finite element method. Their method is applied to the streamfunction equation Eq.(7.2.6).

To simplify the derivation of the scheme initially, $\alpha_0(z)$=1 for all z is assumed so that the equation,

$$\partial^2\psi/\partial z^2 = \xi \tag{7.3.4}$$

is to be solved for ψ. The discretized form of this equation is assumed to be,

$$\frac{1}{h_i}(\psi_{i+1} - \psi_i) - \frac{d}{h_{i-1}}(\psi_i - \psi_{i-1}) = a\xi_{i-1} + b\xi_i + c\xi_{i+1} \tag{7.3.5}$$

where, for convenience, $h_i=\Delta z_i$. The coefficients a, b, c and d are found by requiring that Eq.(7.3.5) is exact for fourth order polynomials of ψ expanded about z_i and valid in the interval z_{i-1}, z_{i+1}. Beland and Beaudoin (1985) chose,

$$\psi = (z - z_i)^m, \qquad m = 1, 2, 3, 4 \tag{7.3.6}$$

which gives,

$$\xi = m(m - 1)(z - z_i)^{m-2} \tag{7.3.7}$$

Substituting these into Eq.(7.3.5) gives,

$$h_i^{m-1} + (-1)^m h_{i-1}^{m-1} d = m(m - 1)[a(-1)^{m-2}h_{i-1}^{m-2} + \delta_{m,2}b + ch_i^{m-2}] \tag{7.3.8}$$

This must hold for m=1, 2, 3 and 4. Substituting these values of m into Eq.(7.3.8) gives four equations that can be solved for a, b, c and d. The coefficients are found to be,

$$\begin{aligned} a_i &= \frac{h_i}{12}\left[1 + \frac{h_{i-1}^2 - h_i^2}{h_i h_{i-1}}\right] \\ b_i &= \frac{h_i+h_{i-1}}{12}\left[5 + \frac{(h_{i-1} - h_i)^2}{h_i h_{i-1}}\right] \\ c_i &= \frac{h_{i-1}}{12}\left[1 + \frac{h_i^2 - h_{i-1}^2}{h_i h_{i-1}}\right] \end{aligned} \tag{7.3.9}$$

with d=1 for all i to give,

$$\frac{1}{h_{i-1}}\psi_{i-1} - \left(\frac{1}{h_i} + \frac{1}{h_{i-1}}\right)\psi_i + \frac{1}{h_i}\psi_{i+1} = a_i\xi_{i-1} + b_i\xi_i + c_i\xi_{i+1} \tag{7.3.10}$$

The value of the coefficients given by Eq.(7.3.9) do not hold at the boundaries. However, since the values of ψ at the top and bottom of the model are known, no derivation of the coefficients at the boundaries is

required (although these coefficients are given by Beland and Beaudoin, 1985).

The specific volume must now be introduced and the streamfunction solved using Eq.(7.2.6) rather than Eq.(7.3.4). The specific volume only alters the LHS of Eq.(7.3.10). The LHS is the same scheme that the usual Galerkin procedure would give. Therefore, to solve Eq.(7.2.6), Eq.(7.3.10) is used but with the LHS derived from the finite element procedure for the second derivative term in Eq.(7.2.6).

Following the usual finite element procedure, the LHS is,

$$\text{LHS} = \int_0^X \int_0^Z \frac{\partial}{\partial z}(\alpha_0 \frac{\partial \psi}{\partial z}) a_k b_l \, dx dz \quad \text{for} \quad k = 0,...N, \; l = 1,...N\text{-}1 \qquad (7.3.11)$$

where x is the width and z the height of the domain. Integrating by parts gives,

$$\text{LHS} = \int_0^X \left\{ [\alpha_0 \frac{\partial \psi}{\partial z} b_l]_0^Z - \int_0^Z \alpha_0 \frac{\partial \psi}{\partial z} \frac{db_l}{dz} \, dz \right\} a_k \, dx$$

The integrated term is zero since b_l is 0 at z=0 and z for all l by definition. Substituting the expansion for ψ gives,

$$\text{LHS} = - \sum_i \sum_j \psi_{i,j} K^{\alpha}_{l,j} M_{k,i} \qquad (7.3.12)$$

where,

$$K^{\alpha}_{l,j} = \int_0^Z \alpha_0 \frac{db_j}{dz} \frac{db_l}{dz} \, dz \qquad (7.3.13)$$

$$M_{k,i} = \int_0^X a_k a_i \, dx$$

It can be shown that the horizontal mass matrix cancels with the same matrix on the RHS. The problem now is the evaluation of the integral for the stiffness matrix, $\underline{\underline{K}}^{\alpha}$. Only three values of j at l-1, l and l+1 give a nonzero

contribution to this integral,

$$K^{\alpha}_{l,l} = \frac{1}{h^2_{l-1}}\int_{z_{l-1}}^{z_l} \alpha_0 \, dz + \frac{1}{h^2_l}\int_{z_l}^{z_{l+1}} \alpha_0 \, dz$$

$$K^{\alpha}_{l,l-1} = -\frac{1}{h^2_{l-1}}\int_{z_{l-1}}^{z_l} \alpha_0 \, dz \qquad (7.3.14)$$

$$K^{\alpha}_{l,l+1} = -\frac{1}{h^2_l}\int_{z_l}^{z_{l+1}} \alpha_0 \, dz$$

using Eq.(3.5.17). These terms can be evaluated by the two integrals,

$$A_l = \frac{1}{h^2_{l-1}}\int_{z_{l-1}}^{z_l} \alpha_0 \, dz, \qquad B_l = \frac{1}{h^2_l}\int_{z_l}^{z_{l+1}} \alpha_0 \, dz \qquad (7.3.15)$$

These could be computed by numerical quadrature, but since α_0 is a known analytical function it is more accurate to evaluate them analytically. Substituting Eq.(7.2.4) into Eq.(7.3.15) and integrating gives,

$$A_l = \frac{1}{h^2_{l-1}}\left[-\frac{\alpha_s R}{\beta(R-c_v)}\right]\left[(1 - \beta z_l)^{\gamma} - (1 - \beta z_{l-1})^{\gamma}\right]$$

$$B_l = \frac{1}{h^2_l}\left[-\frac{\alpha_s R}{\beta(R-c_v)}\right]\left[(1 - \beta z_{l+1})^{\gamma} - (1 - \beta z_l)^{\gamma}\right] \qquad (7.3.16)$$

where,

$$\beta = g/c_p\theta_0, \qquad \gamma = 1 - (c_v/R) \qquad (7.3.17)$$

Substituting Eq.(7.3.15) into Eq.(7.3.14) and then expanding Eq.(7.3.12) gives the LHS of Eq.(7.3.10) as,

$$A_j\psi_{i,j-1} - (A_j + B_j)\psi_{i,j} + B_j\psi_{i,j+1} = a_j\xi_{i,j-1} + b_j\xi_{i,j} + c_j\xi_{i,j+1} \qquad (7.3.18)$$

where A_j and B_j are given by Eq.(7.3.16) and a_j, b_j and c_j are computed from Eq.(7.3.9). Since α_0 is constant with time, all these coefficients are computed once only at the beginning of the run and stored. Also, as Eq.(7.3.18) results

in tridiagonal matrices, it is as efficient as the normal Galerkin finite element approach.

7.3.3.2. Velocities

Once the streamfunction has been computed by Eq.(7.3.18), the next step is to compute the velocities u and w using Eq.(7.2.5). This is done by calculating,

$$u^{\alpha} = \partial\psi/\partial z, \qquad w^{\alpha} = -\partial\psi/\partial x \tag{7.3.19}$$

followed by,

$$u = \alpha_0 u^{\alpha}, \qquad w = \alpha_0 w^{\alpha} \tag{7.3.20}$$

It will be seen later that w^{α} is required for the calculation of one of the advective terms and so this result is stored for later use.

The procedure to calculate w^{α} is exactly that described in chapter 3 and the result can be written immediately using Eq.(3.5.27),

$$\underline{\underline{M}}\,\underline{w}^{\alpha} = -\underline{\underline{P}}_x\underline{\psi} \tag{7.3.21}$$

where $\underline{\underline{M}}$ is the horizontal mass matrix and $\underline{\underline{P}}_x$ is the projection operator for the first derivative in the x direction. The elements of these matrices are given by Eq.(3.5.18) and Eq.(3.5.19). In Eq.(7.3.21), it is assumed that the nodal index in the x direction, i, increases with the row index of the matrices $\underline{w}^{\alpha}$ and $\underline{\psi}$ and that the z direction nodal index, j, increases with the column index of the matrices. A similar matrix equation to Eq.(7.3.21) is used to calculate u^{α}. However, the solution is only first order accurate, because of the stretched vertical coordinate. As an alternative, the procedure of Beland and Beaudoin (1985) could have again been used to give a fourth order accurate solution to u^{α}.

The true velocities u and w are then calculated by using Eq.(7.3.20). Staniforth and Mitchell (1977) successfully used point collocation to compute products in which one function varies smoothly. Since the specific volume is a smoothly varying function, no aliasing will occur if u and w are computed by,

$$u_{i,j} = \alpha_0(z_j)u^{\alpha}_{i,j}, \qquad w_{i,j} = \alpha_0(z_j)w^{\alpha}_{i,j} \tag{7.3.22}$$

This is still fourth order accurate (Cullen, 1979).

7.3.4. Advection terms

The advection terms are of the form $w\partial\phi/\partial z$ and $u\partial\phi/\partial z$ where ϕ is either $(\alpha_0\xi)$, v or θ. The product $\alpha_0\xi$ is computed by point collocation as for the velocities.

As described in chapter 3, advection terms are computed more accurately by a two stage process (Cullen, 1979) in which the gradients are computed first, followed by the calculation of the product. For the first stage, exactly the same procedure as for the computation of the velocities is used so,

$$\underline{\underline{M}}\underline{\phi}_x = \underline{\underline{P}}_x\underline{\phi} \tag{7.3.23}$$

$$\underline{\underline{N}}(\underline{\phi}_z)^T = \underline{\underline{P}}_z\underline{\phi}^T$$

where $\underline{\underline{N}}$ is the vertical mass matrix and $\underline{\underline{P}}_z$ is the projection matrix for the z derivative.

The second stage is the evaluation of the product with the velocities, as described in chapter 3. The elements of the finite element product matrix are not given by Eq.(3.5.38) as a uniform spacing in both directions was assumed in chapter 3. The exact form will be shown later when the computation of the finite element matrices on the DAP is considered. The products will not be calculated to fourth order accuracy because of the stretched vertical spacing.

7.3.5. Diffusion terms

The diffusion terms must be integrated in time using either an implicit scheme or a forward scheme, as the leapfrog method is unsuitable. Since the eddy diffusivity coefficient κ_e varies with time, an implicit scheme would be costly. Instead an explicit scheme was used with the diffusion terms lagged by one step.

The diffusion terms take the general form,

$$\partial/\partial x(\kappa_e \partial\phi/\partial x), \quad \partial/\partial z(\kappa_e \partial\phi/\partial z) \tag{7.3.24}$$

where ϕ represents any of the model prognostic variables and any additional constants applied to κ_e are ignored. One approach is to solve these terms in 3 stages using the basic operations of differentiation and the calculation of the product of two functions. As the diffusion terms are evaluated using values at $t-\Delta t$, the first derivatives (all except $\partial\xi/\partial z$) could be stored from the previous step and used. However, this still requires three calls to the routine that solves the matrix equations. If the usual Galerkin procedure was applied to each term in Eq.(7.3.24) only two calls would be needed. Over one timestep, the extra cost would be significant as it is anticipated that the solution of the simultaneous equations will be the major portion of the CPU time per timestep. Furthermore the 3 stage process will not be fourth order accurate on the nonuniform grid due to the calculation of the product. It was therefore decided to use the usual Galerkin procedure directly.

Consider the term in x, the Galerkin method is applied to,

$$f = \partial/\partial x(\kappa_e \partial\phi/\partial x) \tag{7.3.25}$$

to give,

$$\int_0^X \int_0^Z f a_k b_l \, dx dz = \int_0^X \int_0^Z \frac{\partial}{\partial x}(\kappa_e \frac{\partial\phi}{\partial x}) a_k b_l \, dx dz \quad \text{for all } k, l \tag{7.3.26}$$

When the RHS is integrated by parts, it is convenient to set the integrated term to zero by setting the eddy diffusivity to zero at all the boundaries i.e.

$$\kappa_e(0,z) = \kappa_e(X,z) = \kappa_e(x,0) = \kappa_e(x,Z) = 0 \tag{7.3.27}$$

After substituting expansions for f, κ_e and ϕ, the LHS becomes,

$$\text{LHS} = \underline{\underline{M}}(\underline{\underline{N}}\,\underline{\underline{f}})^T \tag{7.3.28}$$

The RHS integral has not been seen before. It is,

$$F_{k,l} = \sum_i \sum_r \left\{ \sum_j \sum_s \kappa_{i,j}\phi_{r,s} \int_0^Z b_l b_s b_j \, dz \right\} \int_0^X \frac{da_k}{dx} \frac{da_r}{dx} a_i \, dx \tag{7.3.29}$$

where the subscript on κ_e has been omitted for clarity. The integral in curly brackets is the same as the integral in the calculation of the one-dimensional product Eq.(3.5.29). From Eq.(3.5.31),

$$\sum_j \sum_s \kappa_{i,j}\phi_{r,s} \int_0^z b_l b_s b_j \, dz = \tag{7.3.30}$$

$$(1/3)\Delta z_{l-1}\kappa_{i,l-\frac{1}{2}}\phi_{r,l-\frac{1}{2}} + (1/6)(\Delta z_{l-1} + \Delta z_l)\kappa_{i,l}\phi_{r,l} + (1/3)\Delta z_l \kappa_{i,l+\frac{1}{2}}\phi_{r,l+\frac{1}{2}}$$

where,

$$\phi_{r,l+\frac{1}{2}} = \tfrac{1}{2}(\phi_{r,l} + \phi_{r,l+1}) \tag{7.3.31}$$

Calculating Eq.(7.3.29), requires evaluating the integral,

$$I_{i,r,k} = \int_0^x a_i \frac{da_k}{dx}\frac{da_r}{dx} \, dx \tag{7.3.32}$$

This is simple to evaluate since the basis functions are only locally defined. Using Eq.(3.5.6) and Eq.(3.5.17) and evaluating the integral for the point $j=s=l$ gives,

$$\sum_i \sum_r \kappa_{i,l}\phi_{r,l} \int_0^x \frac{da_k}{dx}\frac{da_r}{dx} a_i \, dx = (1/\Delta x)(\kappa_{k-\frac{1}{2},l}\Delta_x\phi_{k-1,l} - \kappa_{k+\frac{1}{2},l}\Delta_x\phi_{k,l}) \tag{7.3.33}$$

where mid-point values at $k-\frac{1}{2}$ and $k+\frac{1}{2}$ are given by Eq.(7.3.31) and,

$$\Delta_x\phi_{k,l} = \phi_{k+1,l} - \phi_{k,l} \tag{7.3.34}$$

Applying Eq.(7.3.33) to all the terms in Eq.(7.3.30) gives the scheme for each point as,

$$\begin{aligned} F_{k,l} = (1/\Delta x)\{ & (1/3)\Delta z_{l-1}(\kappa_{k-\frac{1}{2},l-\frac{1}{2}}\Delta_x\phi_{k-1,l-\frac{1}{2}} - \kappa_{k+\frac{1}{2},l-\frac{1}{2}}\Delta_x\phi_{k,l-\frac{1}{2}}) \\ & + (1/6)(\Delta z_{l-1} + \Delta z_l)(\kappa_{k-\frac{1}{2},l}\Delta_x\phi_{k-1,l} - \kappa_{k+\frac{1}{2},l}\Delta_x\phi_{k,l}) \\ & + (1/3)\Delta z_l(\kappa_{k-\frac{1}{2},l+\frac{1}{2}}\Delta_x\phi_{k-1,l+\frac{1}{2}} - \kappa_{k+\frac{1}{2},l+\frac{1}{2}}\Delta_x\phi_{k,l+\frac{1}{2}}) \} \end{aligned} \tag{7.3.35}$$

The matrix equation is then,

$$\underline{\underline{M}}(\underline{\underline{N}}\,\underline{f}^{\mathrm{T}})^{\mathrm{T}} = -\underline{F} \tag{7.3.36}$$

For the derivative in z,

$$g = \frac{\partial}{\partial z}(\kappa_e \frac{\partial \phi}{\partial z}) \tag{7.3.37}$$

the same procedure gives the matrix equation,

$$\underline{\underline{M}}(\underline{\underline{N}}\,\underline{g}^{\mathrm{T}})^{\mathrm{T}} = -\underline{\underline{G}} \tag{7.3.38}$$

where elements of $\underline{\underline{G}}$ are given by,

$$\begin{aligned} G_{k,l} = {} & (\Delta x/3\Delta z_{l-1})(\kappa_{k-\frac{1}{2},l-\frac{1}{2}}\Delta_z\phi_{k-\frac{1}{2},l-1} + \kappa_{k,l-\frac{1}{2}}\Delta_z\phi_{k,l-1} + \kappa_{k+\frac{1}{2},l-\frac{1}{2}}\Delta_z\phi_{k+\frac{1}{2},l-\frac{1}{2}}) \\ & - (\Delta x/3\Delta z_l)(\kappa_{k-\frac{1}{2},l+\frac{1}{2}}\Delta_z\phi_{k-\frac{1}{2},l} + \kappa_{k,l+\frac{1}{2}}\Delta_z\phi_{k,l} + \kappa_{k+\frac{1}{2},l+\frac{1}{2}}\Delta_z\phi_{k+\frac{1}{2},l}) \end{aligned} \tag{7.3.39}$$

For the nonuniform grid used in this model, this scheme will only give first order accuracy. On a uniform grid, with a constant spacing in the x and z directions, this scheme is second order accurate (see Appendix B).

7.3.6. Time scheme

The time scheme for this model is the same as that used for the spectral model in the last chapter. A leapfrog scheme, Eq.(3.2.1) is used once the tendencies have been computed from the finite element equations. An Asselin (1972) time filter of the form Eq.(3.2.2) is used to control the computational mode of the leapfrog scheme. The first step is made using a forward or Euler step as in the spectral model.

The timestep for a one-dimensional model was limited by, Eq.(3.5.51),

$$\Delta t \leq 0.57\Delta x/c$$

With Δx=24km and c=300m/s (the fastest external gravity wave), $\Delta t \leq 45$ seconds. The timestep used by OR77 was one minute. Several runs of the DAP model were made in which the timestep was varied to test the timestep limit. It was found that a timestep of two minutes was always unstable.

However, a timestep of one minute was stable in some runs and a timestep of 30 seconds was always stable. A comparison of results after 20 hours from runs with timesteps of 30 and 15 seconds showed little differences. Thus, a timestep of 30 seconds was used.

The time filter coefficient was set to 0.01, twice that used for the spectral model. It was found that computational modes were reflected at the lateral boundaries and a filter coefficient of this value had some success in damping these modes. No other sensitivity tests were performed varying the time filter parameter in the model.

7.3.7. Initial conditions

The jet velocity field is set initially by either Eq.(7.2.17), the surface jet case, or by Eq.(7.2.18), the mid-tropospheric jet case. As discussed in chapter 3, these expressions can either be used directly to obtain the expansion coefficients, as the basis functions are interpolatory, or the Galerkin procedure can be applied (i.e. Eq.(3.5.68)) and the integrals evaluated numerically. Since the functions of v are smooth, the former method is chosen so that the nodal values $v_{i,j}$ are given by,

$$v_{i,j} = v(x_i, z_j) \tag{7.3.40}$$

To set θ, it is assumed to be in geostrophic balance with the v field through the thermal wind relation,

$$\frac{\partial \phi}{\partial x} = \frac{f}{g}\frac{\partial v}{\partial z} \tag{7.3.41}$$

with,

$$\phi = \log_e \theta \tag{7.3.42}$$

The first stage is to compute $\eta = \partial v / \partial z$ using the finite element method. The next step is to solve,

$$\frac{\partial \phi}{\partial x} = f\eta / g \tag{7.3.43}$$

The Galerkin procedure results in the matrix equation,

$$\underline{\underline{P}}_x \underline{\phi} = (f/g)\, \underline{\underline{M}}\, \underline{n} \tag{7.3.44}$$

The essential boundary condition is that ϕ is known at the right boundary from Eq.(7.2.19). Inclusion of this essential condition prevents $\underline{\underline{P}}_x$ from being singular and allows Eq.(7.3.44) to be solved. Once $\underline{\phi}$ is known, the nodal values of the potential temperature are computed from Eq.(7.3.42).

The nodal values of U_g are set as for u. Its vertical gradient (and hence the vorticity) is calculated using the finite element method.

7.4. Formulation of the boundary conditions

7.4.1. Lateral boundaries

Miller and Thorpe (1981) showed that the radiation boundary condition scheme of Orlanski (1976) is second order accurate and devised a scheme that was fourth order accurate. Their scheme is based on calculating an improved estimate for r. Using the present time-level, from Eq.(7.2.16), another calculation of r is,

$$r_1 = (\phi_{b-1}^{n-1} - \phi_{b-1}^{n+1}) \,/\, (\phi_{b-1}^{n+1} + \phi_{b-1}^{n-1} - 2\phi_{b-2}^{n}) \tag{7.4.1}$$

Similarly, another estimate could be calculated by using the boundary point itself,

$$r_2 = (\phi_b^{n-2} - \phi_b^{n}) \,/\, (\phi_b^{n} + \phi_b^{n-2} - 2\phi_{b-1}^{n-1}) \tag{7.4.2}$$

Miller and Thorpe (1981) showed that the combined scheme,

$$\hat{r} = r_1 + r_2 - r \tag{7.4.3}$$

with,

$$\phi_b^{n+1} = \left(\frac{1 - \hat{r}}{1 + \hat{r}}\right)\phi_b^{n-1} + \left(\frac{2\hat{r}}{1 + \hat{r}}\right)\phi_{b-1}^{n} \tag{7.4.4}$$

is fourth order accurate.

For a model based on finite differences, the implementation of the above scheme is straightforward. This is not the case for the finite element model in which the tendencies of the variables are computed using the finite element method. The usual procedure to implement the boundary conditions would be

to alter the equations for the boundary nodes in the finite element matrices. However, the Orlanski scheme gives,

$$\frac{\partial \phi_b}{\partial t} = -\frac{\hat{r}}{\Delta t}\left[\tfrac{1}{2}\left(\phi_b^{n+1} - \phi_b^{n-1}\right) - \phi_{b-1}^{n}\right] \tag{7.4.5}$$

for the tendency of the boundary node. This cannot be solved as it refers to a value at the next time-level. The time average is necessary in Eq.(7.4.5) to maintain stability in the presence of physical and computational modes. Without this average, large amplitude computational modes would be reflected off the boundaries.

To apply the Miller-Thorpe scheme, the usual Galerkin finite element framework is not adopted. Instead the tendencies are computed by the finite element method, ignoring the lateral boundary conditions. When the values at $t+\Delta t$ have been calculated, the ratios r, r_1 and r_2 are computed and the lateral boundary nodes are recomputed from Eq.(7.4.4).

It is to be expected that in using this scheme some noise will be reflected from the boundary as a wave passes through it. This is first because the usual finite element framework has been abandoned and the interior points are solved simultaneously with no knowledge of the correct boundary value (except from the previous timestep). Second, although the Miller-Thorpe scheme is fourth order accurate, it is based on a finite difference approximation and as described in chapter 3, the finite element method is more accurate than fourth order finite differences.

To examine the behaviour of this scheme, a finite element model of the one-dimensional linear advection equation,

$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial x} = 0 \tag{7.4.6}$$

was used. The velocity c is constant and hence the product with the spatial derivative was computed by point collocation rather than using the finite element approach. The spatial derivative is computed using the finite element approach. The linear advection equation was also used by Sundstrom and Elvius (1979) to study the behaviour of outflow boundary conditions for finite differences. A Gaussian profile is advected across the domain and out

through the boundary.

Fig. 33 shows results from this model for two cases. The lower curve shows the results when there are no boundary conditions applied to either end of the domain.[1] The top curve shows the results when the Miller-Thorpe scheme is used. No time filtering or diffusion is applied.

Differences between the two cases are apparent. However, in both cases, two gridlength noise is reflected off the boundary as the profile moves out of the domain. For the Miller-Thorpe scheme, the noise is a result of the sources of error described above. For the unconstrained solution, the noise results from the lesser accuracy of the solution at the boundary node; the usual truncation error analysis shows this is solved to first order accuracy only, compared to fourth order accuracy in the interior.

The noise propagates across the domain to the LHS. For the unconstrained solution, the short wave noise is reflected and the original profile reappears. This was also observed by Sundstrom and Elvius (1979). The similarity of this reflected profile to that of the initial shape indicates the accuracy of the finite element method. For the Miller-Thorpe scheme there is also evidence of a reflected profile. However, its amplitude, although significant, is greatly reduced.

It would seem that the Miller-Thorpe scheme is adequate but there is a clear need to remove or reduce the amplitude of any two gridlength noise in the domain to avoid further reflections at the opposite boundary. The Asselin timefilter was found to reduce the amplitude of these waves. The inclusion of diffusion in the model also helped to dampen any noise.

7.4.2. Top and bottom boundaries

The homogeneous boundary condition for the streamfunction at z=0, Eq.(7.2.11), and the inhomogeneous one at the model top, Eq.(7.2.13), are both essential conditions and are incorporated by modifying the equations for these points. The velocity components are computed subject to the boundary conditions implied by the streamfunction i.e. the vertical velocity is zero at the top and bottom of the model.

For the vorticity, the homogeneous condition Eq.(7.2.12) is an essential one

[1] This is therefore an ill-posed problem. Both boundary nodes are computed using a one-sided finite difference approximation to Eq. (7.4.6).

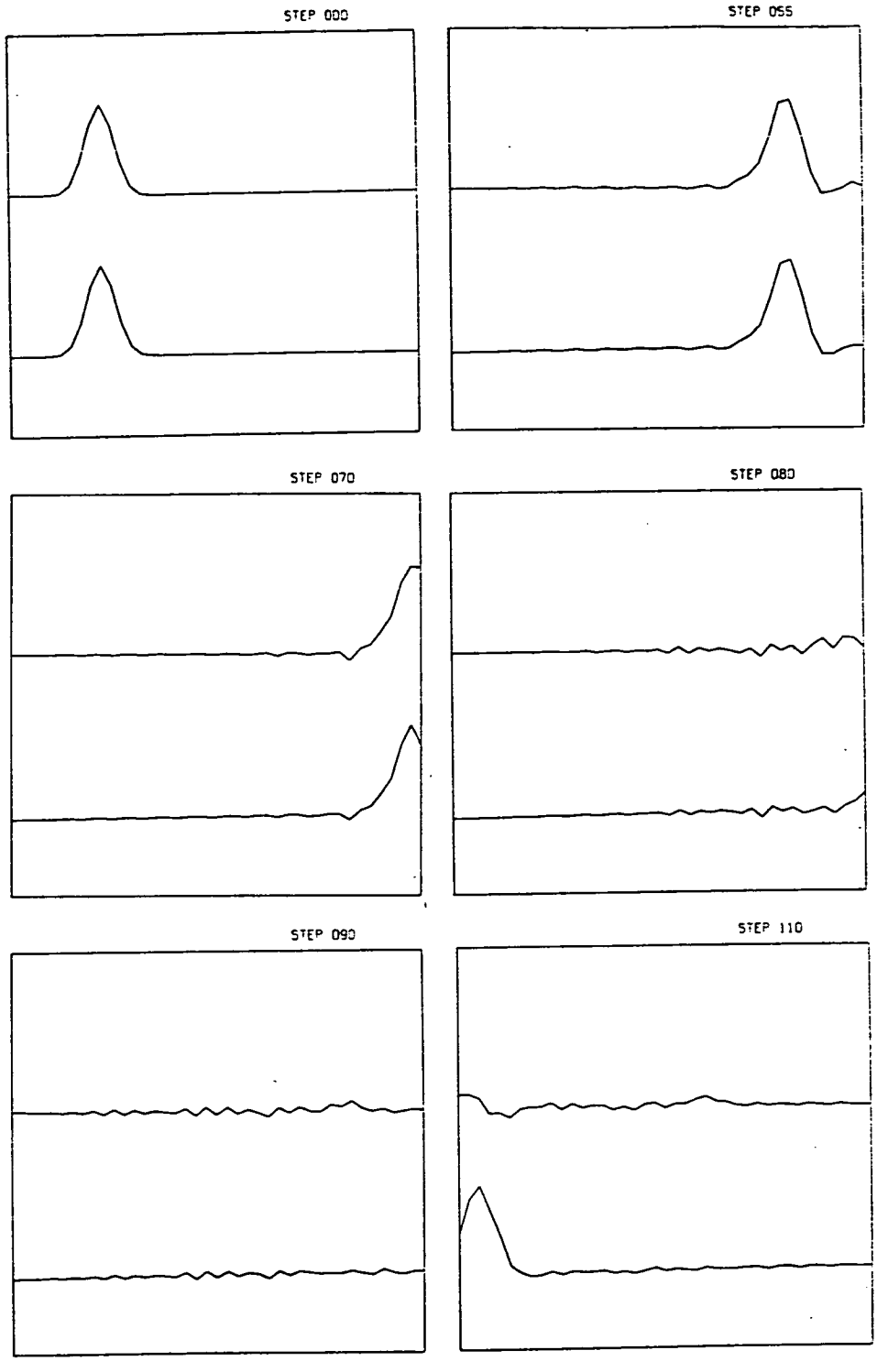


Figure 33. Behaviour of a finite element model of the one-dimensional advection equation when a Gaussian profile is advected out through the boundary. Bottom curve is when the boundary nodes are computed free from any boundary conditions. Top curve is when the Miller and Thorpe (1981) radiation scheme is used as described in the text.

and incorporated by modifying the first row of the matrices as the tendencies at the bottom must also be zero. Like, the other prognostic variables, to maintain a rigid top boundary, the vertical gradient of vorticity is held the same as its initial value. The usual method of including such inhomogeneous Neumann conditions is by modifying the Galerkin equation for second order terms; the diffusion term in this case. However, since this does not guarantee the condition will be satisfied (see the discussion of boundary conditions in chapter 3), it was decided to reformulate the boundary condition to ensure it is satisfied.

Starting with the equation,

$$\frac{\partial \xi}{\partial z} = k$$

where k represents the gradient at the initial time, the finite element method gives the following equation for the nodes at the model top,

$$\xi_{i,M} = \xi_{i,M-1} + \Delta z_{M-1} k \tag{7.4.7}$$

This is the same as the usual finite difference approximation and is first order accurate. A simple scheme to implement this would be the same as that used for the lateral boundaries, where the boundary condition is ignored when the nodes at the new time-level are calculated and the top level values recomputed using Eq.(7.4.7).

For the velocity, v, the top boundary condition is implemented as for the vorticity. The bottom boundary condition, Eq.(7.2.10), is enforced by using the same method. If the vertical gradient is approximated as for the top boundary, the condition becomes,

$$v_{i,0}^{n+1} = v_{i,1}^{n+1} - \Delta z_0 \left(\frac{g}{f \theta_0} \frac{\partial \theta_{i,0}^{n+1}}{\partial x} \right) \tag{7.4.8}$$

Therefore, the bottom boundary is recomputed after the new time-level values have been set.

For the potential temperature the top boundary condition is dealt with in the same way as for the other variables. The bottom boundary condition, Eq.(7.2.9), is a homogeneous natural condition and, as before, formulated as an

essential condition by writing,

$$\theta^{n+1}(x_i,z_0) = \theta^{n+1}(x_i,z_i) \qquad \text{for all } i \tag{7.4.9}$$

to ensure it is satisfied. At the end of each timestep, the bottom boundary values are overwritten with those of the next level.

7.5. Model equations

7.5.1. Vorticity

Multiplying Eq.(7.2.1) by the bilinear basis functions, integrating over the domain and substituting the expansions for the variables, yields the following matrix equation for the vorticity tendency,

$$\underline{\underline{M}}(\underline{\underline{N_e}}\dot{\underline{\xi}}^T)^T = -\underline{\underline{A}} - \underline{\underline{B}} + f\underline{\underline{M}}(\underline{\underline{P_z}}\underline{v}^T)^T - (g/\theta_0)[\underline{\underline{N}}(\underline{\underline{P_x}}\underline{\theta})^T]^T - 700\underline{\underline{F_\xi}} - 0.7\underline{\underline{G_\xi}} \tag{7.5.1}$$

The matrices $\underline{\underline{A}}$ and $\underline{\underline{B}}$ arise from the advective terms which are computed as described in a previous section. The matrices $\underline{\underline{F_\xi}}$ and $\underline{\underline{G_\xi}}$ result from the x and z diffusion terms. The elements of these matrices are given by Eq.(7.3.35) and Eq.(7.3.39) respectively. The subscript on the vertical mass matrix, $\underline{\underline{N_e}}$, indicates that an essential boundary condition at z=0 for ξ has to be satisfied.

The procedure to solve Eq.(7.5.1) is first to compute,

$$\underline{\underline{H}} = -(\underline{\underline{A}} + \underline{\underline{B}} + 700\underline{\underline{F_\xi}} + 0.7\underline{\underline{G_\xi}}) \tag{7.5.2}$$

Substituting this into Eq.(7.5.1) and multiplying both sides by the inverse of the horizontal mass matrix gives,

$$(\underline{\underline{N_e}}\dot{\underline{\xi}}^T)^T = \underline{\underline{M}}^{-1}\underline{\underline{H}} + f(\underline{\underline{P_z}}\underline{v}^T)^T - (g/\theta_0)\underline{\underline{M}}^{-1}\underline{\underline{P_x}}\underline{\theta}\underline{\underline{N}}^T \tag{7.5.3}$$

where the property $(\underline{\underline{A}}\underline{\underline{B}}^T)^T = \underline{\underline{B}}\underline{\underline{A}}^T$ has been used. The next stage is to solve,

$$\underline{\underline{K}} = \underline{\underline{M}}^{-1}\underline{\underline{H}} \tag{7.5.4}$$

and,

$$\underline{\theta_x} = \underline{\underline{M}}^{-1}\underline{\underline{P_x}}\underline{\theta} \tag{7.5.5}$$

Following this, compute,

$$\underline{\underline{R}}^T = \underline{\underline{K}} + f(\underline{\underline{P}}_z \underline{\underline{v}}^T)^T - (g/\theta_0)\underline{\underline{\theta}}_x \underline{\underline{N}}^T \tag{7.5.6}$$

so that Eq.(7.5.3) becomes,

$$\underline{\underline{N}}_e \dot{\underline{\underline{\xi}}}^T = \underline{\underline{R}} \tag{7.5.7}$$

The final stage is therefore to solve,

$$\dot{\underline{\underline{\xi}}} = (\underline{\underline{N}}_e^{-1} \underline{\underline{R}})^T \tag{7.5.8}$$

subject to Eq.(7.2.12). The tendency is then integrated forward in time and the remaining boundary conditions applied.

7.5.2. Potential temperature

Since U_g is constant, its vertical gradient is computed at the start of the run and stored. As this gradient is smooth, the product with the jet velocity is computed by point collocation,

$$A(x_i, z_j) = v(x_i, z_j) \frac{dU_g(z)}{dz} \frac{f\theta_0}{g} \tag{7.5.9}$$

Applying the finite element method to Eq.(7.2.2) gives the matrix equation,

$$\underline{\underline{M}}(\underline{\underline{N}}\dot{\underline{\underline{\theta}}}^T)^T = -\underline{\underline{B}} - \underline{\underline{C}} + \underline{\underline{M}}(\underline{\underline{N}}\underline{\underline{A}}^T)^T - 1000\underline{\underline{F}}_\theta - \underline{\underline{G}}_\theta \tag{7.5.10}$$

where $\underline{\underline{B}}$ and $\underline{\underline{C}}$ are the matrices for the advective terms, $\underline{\underline{F}}_\theta$ and $\underline{\underline{G}}_\theta$ are the diffusion matrices formed using the potential temperature and $\underline{\underline{A}}$ is given by Eq.(7.5.9).

The solution procedure follows from that used to solve for the vorticity tendency. First compute,

$$\underline{\underline{D}} = -(\underline{\underline{B}} + \underline{\underline{C}} + 1000\underline{\underline{F}}_\theta + \underline{\underline{G}}_\theta) \tag{7.5.11}$$

to give,

$$(\underline{\underline{N}}\dot{\underline{\underline{\theta}}}^T)^T = \underline{\underline{M}}^{-1}\underline{\underline{D}} + (\underline{\underline{N}}\underline{\underline{A}}^T)^T \tag{7.5.12}$$

Then solve,

$$\underline{\underline{E}}^T = \underline{\underline{M}}^{-1}\underline{\underline{D}} \tag{7.5.13}$$

which gives,

$$\underline{\underline{N}}\dot{\underline{\underline{\theta}}}^T = \underline{\underline{E}} + \underline{\underline{N}}\underline{\underline{A}}^T \tag{7.5.14}$$

Multiplying by the inverse of the vertical mass matrix,

$$\underline{\underline{H}}^T = \underline{\underline{N}}^{-1}\underline{\underline{E}} \tag{7.5.15}$$

the potential temperature tendency is given by,

$$\dot{\underline{\underline{\theta}}} = \underline{\underline{H}} + \underline{\underline{A}} \tag{7.5.16}$$

The tendency is then integrated forward in time and the boundary conditions applied.

7.5.3. Jet velocity

The ageostrophic component of the Coriolis force in Eq.(7.2.3) is computed by,

$$\hat{u}(x_i,z_j) = f[U_g(z_j) - u(x_i,z_j)] \tag{7.5.17}$$

Applying the finite element method to Eq.(7.2.3) gives the matrix equation,

$$\underline{\underline{M}}(\underline{\underline{N}}\dot{\underline{\underline{v}}}^T)^T = -\underline{\underline{A}} - \underline{\underline{B}} + \underline{\underline{M}}(\underline{\underline{N}}\hat{\underline{\underline{u}}}^T)^T - 700\underline{\underline{F}}_v - 0.7\underline{\underline{G}}_v \tag{7.5.18}$$

where $\underline{\underline{A}}$ and $\underline{\underline{B}}$ are the matrices resulting from the advection terms and $\underline{\underline{F}}_v$ and $\underline{\underline{G}}_v$ are the matrices arising from the diffusion terms. As before, first compute,

$$\underline{\underline{C}} = -(\underline{\underline{A}} + \underline{\underline{B}} + 700\underline{\underline{F}}_v + 0.7\underline{\underline{G}}_v) \tag{7.5.19}$$

Multiply Eq.(7.5.18) by the inverse of the horizontal mass matrix to give,

$$\underline{\underline{D}}^T = \underline{\underline{M}}^{-1}\underline{\underline{C}} \tag{7.5.20}$$

which implies,

$$\underline{\underline{N}}\dot{\underline{\underline{v}}}^T = \underline{\underline{D}} + \underline{\underline{N}}\hat{\underline{\underline{u}}}^T \tag{7.5.21}$$

To get the tendency, solve,

$$\underline{\underline{E}}^T = \underline{\underline{N}}^{-1}\underline{\underline{D}} \tag{7.5.22}$$

and then,

$$\underline{\underline{\dot{v}}}^T = \underline{\underline{E}} + \underline{\underline{\hat{u}}} \tag{7.5.23}$$

The tendency is then integrated forward in time and the boundary conditions applied.

7.6. Implementation on the DAP

7.6.1. Introduction

Unlike the spectral method, several research groups have studied the application of the DAP to finite element problems in engineering. The group at Liverpool University mainly studied the solution of two-dimensional elliptic problems, when the number of nodes was greater than the available processors (Wait and Martindale, 1985). The matrices were partitioned and their solution was by preconditioned conjugate gradient methods (Wait, 1988). The group at Hatfield Polytechnic studied the solution of steady two-dimensional problems formulated using a least squares approach (Dixon and Singh, 1984) and a variational approach (Dixon and Ducksbury, 1985). They also used a conjugate gradient method (Dixon *et al*, 1982) but modified so that the global stiffness matrix did not need to be assembled. The equations were solved on an element by element basis. As Lai and Liddell (1987a) point out, although this saves storage space more computing is required.

The group at the DAP Support Unit (DAPSU) reviewed existing studies (Lai and Liddell, 1987a) and studied the solution of problems by the conjugate gradient method (Lai and Liddell, 1987b). A parallel version of the SERC/NAG finite element library for the DAP has since been developed at DAPSU (Lai, 1989).

The finite element model of this chapter contrasts the above work in several ways. First, this problem is time dependent, putting more emphasis on fast solution methods and the efficient evaluation of the finite element matrices. Second, the adopted solution method decomposes the problem into sets of one-dimensional equations resulting in tridiagonal matrices rather than

the higher bandwidth matrices of the two-dimensional problems studied in the previously cited work. This means that the nodes of the finite element model are more appropriately mapped as a matrix on the DAP, rather than using vectors. Thirdly, it is not obvious that a conjugate gradient solver will be superior to existing tridiagonal solvers. Lastly, there is no assembly of the stiffness matrix for the RHS of the equations. Instead, each term is computed individually. This has several benefits. The first is that a set of kernel routines to compute spatial derivatives, products and diffusion terms can be designed and used as the basis for a model. Second, the work for the derivative terms is reduced by cancelling mass matrices (as seen in the previous section).

7.6.2. Data mapping

Two approaches have been used to map finite element grids onto the DAP. Lai and Liddell (1988) and Wait and Martindale (1985) use the 'long-vector' mode of the DAP, where elements are stored consecutively according to their element numbering. The group at Hatfield use an 'upper-leftmost' storage mapping in which the x and y coordinates of the nodes are mapped along the rows and columns of the DAP matrix.

An advantage of the long-vector storage format is that it easily handles arbitrary element shapes and irregular boundaries. Also, it permits a random allocation of elements to processors, useful for automatic mesh generation (Mouhas, 1987). For the model of this chapter, these issues are irrelevant. The choice of data mapping is effectively determined by the formulation of the finite element equations. These have been derived such that systems of simultaneous equations are solved along levels or vertical columns. Thus, the long-vector format is unsuitable because of the added cost of ordering the nodes for the solver. For the derivative, product and diffusion terms, an upper-leftmost mapping allows efficient calculation of the finite element matrices using shift operations on the processor array. Clearly, if the long-vector storage mode is used, the equations should be formulated using the two-dimensional basis functions and the nodes numbered using vectors.

A straightforward mapping of,

$$(x,z) = (i\Delta x, \sum_{k=0}^{j} \Delta z_k) \rightarrow \{ i+1, j+1 \} \qquad (7.6.1)$$

is therefore chosen for the model. Like the spectral model, the PE array size will influence the size of the domain if the best performance is to be achieved. The model domain was chosen such that there are the same number of nodes as processors, as this was a close match to the number of points used by the OR77 model. For grids larger than the array size both the crinkled (Wait, 1988) and sheet (Lai, 1989) mapping strategies have been successfully used for finite element problems. The mapping of data to the PE array is simpler than in the spectral model, as there is no change of representation space.

A problem with modelling on the DAP is that once the program is written the number of nodes and the resolution are fixed. Tests at higher resolutions than the working resolution are often desirable and usually straightforward with well coded models on serial or vector machines. Aside from this limitation, DAP FORTRAN makes coding the finite element model easy as the processor array has the equivalent structure to the nodal arrangement for the model. This removes the need for lookup tables to carry nodal information, as required by the long-vector approach (Lai and Liddell, 1987a). The same would be true for a finite difference model on a rectangular domain.

Since the model domain matches the size of the processor array, high efficiency can be expected during the calculation of the finite element matrices and their solution. Like the spectral model, there is a relationship between the efficiency of the algorithms and the choice of data mapping. For the calculation of the finite element matrices access to neighbouring nodes will be required. From Eq.(7.6.1), this can be achieved by using the shift functions. Only one mapping is optimum here, there is no choice as there was for the spectral model. This makes implementing the model quicker and easier.

The model equations are written in a form suitable for the DAP. The equations often use the transpose of matrices for which the DAP FORTRAN function TRAN can be used. Translation of the model equations to DAP FORTRAN is hence straightforward since there is a one-to-one correspondence between the matrices in the model equations and a matrix as defined by DAP FORTRAN. The model was therefore developed quickly with

concise code.

7.6.3. Model output

Like the spectral model, periodic history output of the model fields is possible. The DDX software was used as in the spectral model, following the same procedure. For more details, see the section on model output in the previous chapter.

7.6.4. Boundary conditions

As the boundary conditions apply only to a limited area of the domain, it is natural to use the DAP vector mode to calculate them rather than masked matrix operations which take more time. The nondimensional ratio, $\hat{r}$, used for the interpolation of the boundary nodes is computed in vector mode i.e. the value is calculated in parallel for all levels at the boundaries.

7.7. Efficient calculation of finite element matrices

7.7.1. Derivative terms

The finite element solution to a derivative of the form, $w=\partial u/\partial x$ is given by Eq.(3.5.20) for the interior nodes and,

$$(1/3)[\Delta x w_1 + \tfrac{1}{2}\Delta x w_2] = \tfrac{1}{2}(u_2 - u_1) \qquad (7.7.1)$$

$$(1/3)[\tfrac{1}{2}\Delta x w_{N-1} + \Delta x w_N] = \tfrac{1}{2}(u_N - u_{N-1})$$

for the boundary nodes. A DAP FORTRAN matrix function was written to evaluate the RHS of Eq.(3.5.20) and Eq.(7.7.1) (a matrix function is a DAP FORTRAN function that returns a matrix result). The difference is computed by shifting the matrix. The boundaries have to be calculated separately as different routing is required. The function was written to evaluate the matrix for the x derivative. A separate function could have been used for the z derivative but a matrix transpose on entry and exit allowed the same function to be used. This incurred added expense that can be estimated, since 5 derivatives in the z direction have to be computed. The additional cost is 2.2msecs per timestep.

7.7.2. Product terms

This section describes how the finite element matrix for the product of two functions is computed for the model. The general case of an irregular grid, where the spacing is variable in the x and z directions, is considered first. The semi-irregular grid of the model, with a constant spacing in the x direction, is then considered.

7.7.2.1. Irregular grid

To solve for the product of two functions, the matrix whose elements are given by,

$$F_{k,l} = \int_{x_{k-1}}^{x_{k+1}} \int_{z_{l-1}}^{z_{l+1}} uva_k b_l \, dx dz \tag{7.7.2}$$

must be evaluated, where u and v are any two-dimensional functions. If the usual Galerkin procedure is followed, this integral can be written in the form,

$$\begin{aligned} F_{k,l} = (1/36)\{ & \Delta x_{k-1}\Delta z_{l-1}[\, p_{k,l} + 2p_{k-\frac{1}{2},l} + 2p_{k,l-\frac{1}{2}} + 4p_{k-\frac{1}{2},l-\frac{1}{2}} \,] \\ & + \Delta x_k \Delta z_{l-1}[\, p_{k,l} + 2p_{k+\frac{1}{2},l} + 2p_{k,l-\frac{1}{2}} + 4p_{k+\frac{1}{2},l-\frac{1}{2}} \,] \\ & + \Delta x_{k-1}\Delta z_l[\, p_{k,l} + 2p_{k,l+\frac{1}{2}} + 2p_{k-\frac{1}{2},l} + 4p_{k-\frac{1}{2},l+\frac{1}{2}} \,] \\ & + \Delta x_k \Delta z_l[\, p_{k,l} + 2p_{k,l+\frac{1}{2}} + 2p_{k+\frac{1}{2},l} + 4p_{k+\frac{1}{2},l+\frac{1}{2}} \,] \,\} \end{aligned} \tag{7.7.3}$$

where p is defined by,

$$p_{k,l} = u_{k,l} v_{k,l} \tag{7.7.4}$$

$$p_{k+\frac{1}{2},l+\frac{1}{2}} = u_{k+\frac{1}{2},l+\frac{1}{2}} v_{k+\frac{1}{2},l+\frac{1}{2}}$$

and,

$$u_{k+\frac{1}{2},l} = \tfrac{1}{2}(u_{k,l} + u_{k+1,l}) \tag{7.7.5}$$

$$u_{k+\frac{1}{2},l+\frac{1}{2}} = \tfrac{1}{2}(u_{k+\frac{1}{2},l} + u_{k+\frac{1}{2},l+1})$$

The terms in Eq.(7.7.3) have been grouped for each grid rectangle. To evaluate

the integral, half-integer nodes are defined, as illustrated by Fig. 34.

To compute the integral efficiently, Staniforth and Mitchell (1978) advocated evaluating all contributions from a given grid rectangle to the neighbouring four nodes simultaneously, avoiding repeated calculations. This method would require evaluating the element matrix,

$$\underline{f}_e = \int_{x_k}^{x_{k+1}} \int_{z_l}^{z_{l+1}} uv a_i b_j \, dx dz \qquad \text{for } i=k,k+1;\ j=l,l+1 \tag{7.7.6}$$

so that $\underline{f}_e$ has four entries, one for each node. To calculate the entire matrix, the contributions from the element matrices are summed as,

$$\underline{\underline{F}} = \sum_e \underline{f}_e \tag{7.7.7}$$

This is the usual approach for engineering problems (Strang and Fix, 1973).

However, the four terms in Eq.(7.7.6) are the same as those for each grid element in Eq.(7.7.3), except for a different part of the grid. The only difference between DAP implementations of the two approaches is the routing required to align the products before the summation in Eq.(7.7.7), since on the DAP the products are evaluated for all the nodes simultaneously. The procedure to evaluate Eq.(7.7.3) is described in detail in Appendix C. A total of 11 multiplications and 17 additions in matrix mode are required.

7.7.2.2. Semi-irregular grid

If the constant spacing in the x direction is taken into account, the number of matrix operations can be reduced. For the irregular grid the boundary values are computed correctly using the equation for the interior nodal values by using planar shifts and setting unused parts of the matrices to zero. It is desirable to retain this for the semi-irregular grid of the model otherwise the added cost of vector operations to compute the boundary values is greater than the saving made in computing the interior.

The preserve the accuracy of Eq.(7.7.3) at the boundaries, two vectors, $\underline{\alpha}$ and $\underline{\beta}$ are introduced,

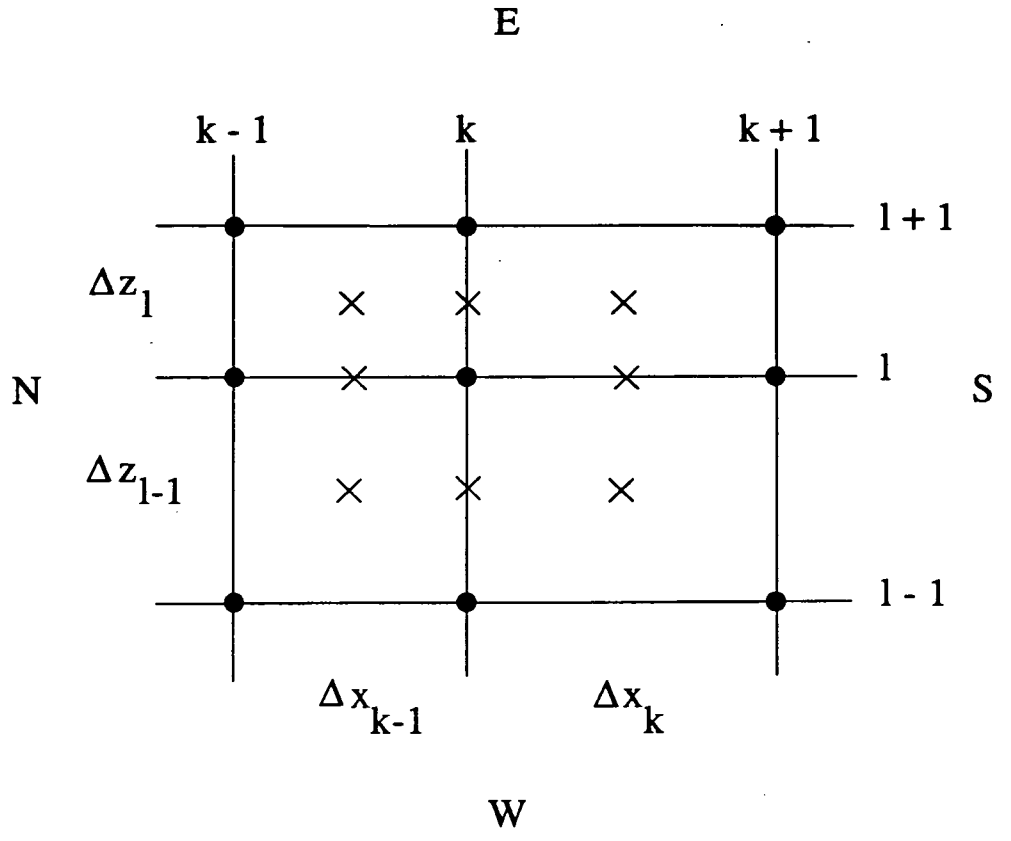


Figure 34. Illustration of the nodes and half-integer nodes used during the calculation of the integral over the product of two two-dimensional functions. Circles denote nodes at which elements are unity, crosses denote the additional half-integer nodes defined to calculate the product at node (k,1).

$$\underline{\alpha} = (\,0\;1\;1\;1\;\ldots\;1\;1\,)^T \tag{7.7.8}$$

$$\underline{\beta} = (\,1\;1\;1\;1\;\ldots\;1\;0\,)^T$$

such that the spacing in the x direction becomes,

$$\Delta x_{k-1} = \alpha_k \Delta x, \qquad \Delta x_k = \beta_k \Delta x \tag{7.7.9}$$

These are substituted into Eq.(7.7.3) to give,

$$\begin{aligned} F_{k,l} = (\Delta x \Delta z_{l-1}/36)\big[\, & (\alpha_k+\beta_k)p_{k,l} + 2\alpha_k p_{k-\frac{1}{2},l} + 2\beta_k p_{k+\frac{1}{2},l} \\ & + 2(\alpha_k+\beta_k)p_{k,l-\frac{1}{2}} + 4\alpha_k p_{k-\frac{1}{2},l-\frac{1}{2}} + 4\beta_k p_{k+\frac{1}{2},l-\frac{1}{2}} \,\big] \\ + (\Delta x \Delta z_l/36)\big[\, & (\alpha_k+\beta_k)p_{k,l} + 2\alpha_k p_{k-\frac{1}{2},l} + 2\beta_k p_{k+\frac{1}{2},l} \\ & + 2(\alpha_k+\beta_k)p_{k,l+\frac{1}{2}} + 4\alpha_k p_{k-\frac{1}{2},l+\frac{1}{2}} + 4\beta_k p_{k+\frac{1}{2},l+\frac{1}{2}} \,\big] \end{aligned} \tag{7.7.10}$$

where the definitions of Eq.(7.7.4) are assumed. All the products, p, multiplied solely by α_k will therefore disappear at k=0. However, since a planar shift south is used in the DAP FORTRAN code, these terms would be zero without α_k. Similarly for terms multiplied solely by β_k. Therefore, α and β can be removed from these terms. The remaining terms involve the sum of the vectors, written as,

$$\underline{\gamma} = \underline{\alpha} + \underline{\beta} = (\,1\;2\;2\;\ldots\;2\;2\;1\,)^T \tag{7.7.11}$$

The procedure to compute Eq.(7.7.10) for the model is given in Appendix C. The total operation count is 9 multiplications and 13 additions in matrix mode and 2 multiplications in vector mode. This is a reduction of 2 multiplications and 4 additions in matrix mode over the irregular case, achieved by computing the contribution from two grid rectangles along each level simultaneously. The boundary nodes are computed in parallel with the interior nodes. The saving in CPU time is 1.08msec i.e. 6.5msecs per timestep. For this particular calculation, the use of an irregular grid would involve only a small overhead.

7.7.2.3. Further improvements

Although the above algorithm was used for the DAP model, Staniforth and Beaudoin (1986) devised a more efficient method of evaluating the integral associated with a product. The changes to the DAP algorithm following their method are described in this section, although the algorithm was not coded.

Staniforth and Beaudoin (1986) considered the efficient evaluation of the integral Eq.(7.7.2) by 3 methods; substitution of the expansions of u and v as above, the use of Gaussian quadrature and the use of Simpson quadrature. They showed that Simpson quadrature gave a significantly lower operation count. For an irregular grid, they obtained an algorithm requiring 10 multiplications and 12 additions per node, about the same as the operation count for the DAP algorithm on the semi-irregular grid.

They stated that the algorithm obtained by substituting the expansions is computationally 4 times as expensive as the Simpson quadrature algorithm. This is incorrect. It is straightforward to show that the two methods are equivalent. Whilst it is true that to evaluate all the terms in Eq.(7.7.3) for each node would be expensive (which is how Staniforth and Beaudoin arrive at the factor of 4), repeated calculations are avoidable by computing intermediate results. The advantage to the Simpson quadrature method is that it leads directly to these intermediate results required for a reduction in the operation count. This is something that is difficult to see directly from Eq.(7.7.3).

To show the two methods are equivalent, consider the one-dimensional case for simplicity. The result is also true in two dimensions. The integral,

$$I_k = \int_0^X u(x)v(x)\phi_k(x)\,dx \tag{7.7.12}$$

where ϕ_k are the basis functions, becomes, after substituting the usual expansions for u and v, from Eq.(3.5.31),

$$I_k = (1/12)[\,\Delta x_k(u_k + u_{k+1})(v_k + v_{k+1}) + 2u_k v_k(\Delta x_k + \Delta x_{k-1}) + \Delta x_{k-1}(u_k + u_{k-1})(v_k + v_{k-1})\,] \tag{7.7.13}$$

Simpson's rule is,

$$\int_{x_k}^{x_{k+1}} f(x)\,dx = (\Delta x_k/6)[\, f(x_k) + 4f(x_{k+\frac{1}{2}}) + f(x_{k+1})\,] \tag{7.7.14}$$

where,

$$x_{k+\frac{1}{2}} = \tfrac{1}{2}(\, x_k + x_{k+1}\,) \tag{7.7.15}$$

Applying this to the evaluation of Eq.(7.7.12) gives,

$$I_k = (\Delta x_k/3)u_{k+\frac{1}{2}}v_{k+\frac{1}{2}} + (1/6)(\Delta x_k + \Delta x_{k-1})u_k v_k + (\Delta x_{k-1}/3)u_{k-\frac{1}{2}}v_{k-\frac{1}{2}} \tag{7.7.16}$$

where,

$$u_k = u(x_k), \qquad u_{k+\frac{1}{2}} = u(x_{k+\frac{1}{2}}) \tag{7.7.17}$$

Assuming a linear basis function gives,

$$u_{k+\frac{1}{2}} = \tfrac{1}{2}(u_k + u_{k+1}), \qquad v_{k+\frac{1}{2}} = \tfrac{1}{2}(v_k + v_{k+1}) \tag{7.7.18}$$

Substituting Eq.(7.7.18) into Eq.(7.7.16) gives Eq.(7.7.13).

The algorithm for an irregular grid, following the method of Staniforth and Beaudoin (1986), is given in Appendix C. The operation count is now 10 multiplications and 12 additions. The reduction by a multiplication and 5 additions over the previous algorithm for an irregular grid means a decrease in CPU time of 1msec or 18.5%. The algorithm is still accurate at the boundaries if unused elements of the matrices are set to zero and planar shifts are used.

To explain why this algorithm is more efficient, the integral Eq.(7.7.2) is partially evaluated and written as,

$$F_{k,l} = \frac{1}{3}\Delta z_{l-1} \int_0^x u(x,z_{l-\frac{1}{2}})v(x,z_{l-\frac{1}{2}})a_k \, dx$$

$$+ \frac{1}{6}(\Delta z_{l-1}+\Delta z_l) \int_0^x u(x,z_l)v(x,z_l)a_k \, dx \qquad (7.7.19)$$

$$+ \frac{1}{3}\Delta z_l \int_0^x u(x,z_{l+\frac{1}{2}})v(x,z_{l+\frac{1}{2}}) \, dx$$

If the first integral on the RHS is calculated for all nodes, this result can be used for the third integral. On the DAP this involves routing the result. The only other integral to evaluate is the second one. Hence, rather than computing contributions from adjacent grid rectangles, it is more efficient to evaluate the one-dimensional integrals along z_l and $z_{l-\frac{1}{2}}$ as the values of the integral for $z_{l-\frac{1}{2}}$ give those for $z_{l+\frac{1}{2}}$.

For the semi-irregular grid of the model, as before, the code should still compute the boundaries and the internal nodes concurrently. To ensure this, the vectors α_k and β_k are again used. When Eq.(7.7.9) is substituted into Eq.(7.7.19), after the integrals are evaluated by Simpson quadrature, it is possible to write,

$$F_{k,l} = G_{k,l} + H_{k,l} + H_{k,l-1} \qquad (7.7.20)$$

where,

$$G_{k,l} = (1/36)[\ \Delta x(\Delta z_l + \Delta z_{l-1})\gamma_k\ ]p_{k,l}$$

$$+ (1/72)[\ \Delta x(\Delta z_l + \Delta z_{l-1})\ ](\ p_{k+\frac{1}{2},l} + p_{k-\frac{1}{2},l}\ ) \qquad (7.7.21)$$

$$H_{k,l} = (1/72)[\ \Delta x\Delta z_l\gamma_k\ ]p_{k,l+\frac{1}{2}} + (1/144)[\ \Delta x\Delta z_l\ ](\ p_{k+\frac{1}{2},l+\frac{1}{2}} + p_{k-\frac{1}{2},l+\frac{1}{2}}\ )$$

The constants can be combined so that the product $\Delta z_l\gamma_k$ is a constant matrix. Use has again been made of the planar shift and unused elements are set to zero to avoid the need for the vector γ_k to be multiplied to the $p_{k+\frac{1}{2},l}$ and $p_{k+\frac{1}{2},l+\frac{1}{2}}$ products so that these constants can be formed.

The operation count for the semi-irregular grid algorithm is 8

multiplications and 12 additions. This is a saving of 0.5msecs or 12% over the irregular case using the Simpson quadrature approach. Compared to the original DAP algorithm on the semi-irregular grid of the previous section, the saving is 2 multiplications and 3 additions; 1msec or 21%. The overhead in using a fully irregular grid for the Simpson quadrature algorithms is small.

Much of this algorithm can be followed for an implementation on a serial or vector computer. However, the boundary nodes would have to be computed separately from the interior nodes for both the irregular and semi-irregular grids. The DAP algorithm exploits the hardware boundary conditions and computes the boundary and interior nodes simultaneously. The algorithm is formulated such that there is no overhead in calculating the values at the boundaries, as might at first be expected.

7.7.3. Diffusion terms

7.7.3.1. Calculation of the eddy diffusivity

Before discussing the evaluation of the integrals for the diffusion terms, the calculation of the eddy diffusivity, κ_e, used by the integrals is considered. This parameter is dependent on the local potential temperature difference as given by Eq.(7.2.7). In the model, the vertical potential temperature gradient is computed for the advection term. The potential temperature difference between two levels is set by,

$$\Delta\theta_l = \theta_z(k,l)\Delta z_{l-1} \tag{7.7.22}$$

where l=0,..63. Boundary conditions set θ_z to zero at the bottom boundary. The equation for the eddy diffusivity becomes,

$$\begin{aligned} \kappa_{ei,j} &= \kappa_0 && , \theta_z \geq 0 \\ \kappa_{ei,j} &= \kappa_0[\ 1 + D_j|\theta_{zi,j}|^{1/3}\ ] && , \theta_z < 0 \end{aligned} \tag{7.7.23}$$

where,

$$D_j = C(\ g\ /\ c\theta_0\kappa_0^2\ )^{1/3}\ (\Delta z_{j-1})^{4/3} \tag{7.7.24}$$

with D at j=0 equal to zero.

To compute κ_e at all the nodes, first set them equal to the background

field κ_0. Then assign a logical matrix such that elements are TRUE where θ_z is negative. If any of the elements of this matrix are TRUE, the correction Eq.(7.7.24) is added for which the logical matrix is used as a mask.

7.7.3.2. Semi-irregular grid

Consider the evaluation of the integral,

$$F_{k,l} = \int_{x_{k-1}}^{x_{k+1}} \int_{z_{l-1}}^{z_{l+1}} \frac{\partial}{\partial x}[\kappa_e(x,z)\frac{\partial \phi}{\partial x}]a_k b_l \, dx dz \tag{7.7.25}$$

arising from the diffusion term in the x direction. Constants have been ignored for clarity. Following the usual procedure, assuming an irregular grid gives,

$$\begin{aligned} F_{k,l} &= (\Delta z_{l-1}/6\Delta x_{k-1})(\, 2\kappa_{k-\frac{1}{2},l-\frac{1}{2}}\Delta_x\phi_{k-1,l-\frac{1}{2}} + \kappa_{k-\frac{1}{2},l}\Delta_x\phi_{k-1,l} \,) \\ &- (\Delta z_{l-1}/6\Delta x_k)(\, 2\kappa_{k+\frac{1}{2},l-\frac{1}{2}}\Delta_x\phi_{k,l-\frac{1}{2}} + \kappa_{k+\frac{1}{2},l}\Delta_x\phi_{k,l} \,) \\ &+ (\Delta z_l/6\Delta x_{k-1})(\, 2\kappa_{k-\frac{1}{2},l+\frac{1}{2}}\Delta_x\phi_{k-1,l+\frac{1}{2}} + \kappa_{k-\frac{1}{2},l}\Delta_x\phi_{k-1,l} \,) \\ &- (\Delta z_l/6\Delta x_k)(\, 2\kappa_{k+\frac{1}{2},l+\frac{1}{2}}\Delta_x\phi_{k,l+\frac{1}{2}} + \kappa_{k+\frac{1}{2},l}\Delta_x\phi_{k,l} \,) \end{aligned} \tag{7.7.26}$$

where the subscript has been omitted from κ_e for clarity. As for the algorithm for the product of two functions, the terms in the above equation are grouped according to their contribution from an individual element. Values at the half-integer nodes are defined as the mean of neighbouring nodal values as for the product term, the difference operator Δ_x is given by Eq.(7.3.34).

The algorithm for the irregular grid is not presented since the overhead to the semi-irregular grid is similar to that found for the evaluation of the product term matrix. To derive the algorithm for the semi-irregular grid case, the vectors $\underline{\alpha}$ and $\underline{\beta}$, given by Eq.(7.7.8), are again introduced to compute boundary and interior values simultaneously. Writing,

$$1 \, / \, \Delta x_{k-1} = \alpha_k \, / \, \Delta x, \qquad 1 \, / \, \Delta x_k = \beta_k \, / \, \Delta x \tag{7.7.27}$$

and defining,

$$p_{k,l} = 2\kappa_{k+\frac{1}{2},l+\frac{1}{2}}\Delta_x\phi_{k,l+\frac{1}{2}}$$

$$q_{k,l} = \kappa_{k+\frac{1}{2},l}\Delta_x\phi_{k,l} \qquad (7.7.28)$$

and substituting into Eq.(7.7.26) gives,

$$F_{k,l} = (\Delta z_{l-1}/6\Delta x)[\ \alpha_k(\ p_{k-1,l-1} + q_{k-1,l}\ ) - \beta_k(\ p_{k,l-1} + q_{k,l}\ )\ ] \qquad (7.7.29)$$

$$+ (\Delta z_l/6\Delta x)[\ \alpha_k(\ p_{k-1,l} + q_{k-1,l}\ ) - \beta_k(\ p_{k,l} + q_{k,l}\ )\ ]$$

The mapping expressions for p and q are,

$$p_{k,l}: \qquad (\ k,\ l\ ) \rightarrow \{\ k+1,\ l+1\ \} \quad k,l=0,..N\text{-}1 \qquad (7.7.30)$$

$$q_{k,l}: \qquad (\ k,\ l\ ) \rightarrow \{\ k+1,\ l+1\ \} \quad k=0,..N\text{-}1;\ l=0,..N$$

Using these expressions it can be seen that α_k is unnecessary since a planar shift south is required for the p and q coefficients anyway. The β_k term is also unnecessary as, from Eq.(7.7.30), p and q are not defined at $k=N$. So by setting unused elements of matrices to zero the need for the α and β vectors is removed.

If the differences,

$$u_{k,l} = p_{k-1,l} - p_{k,l} \qquad (7.7.31)$$

$$v_{k,l} = q_{k-1,l} - q_{k,l}$$

are computed then Eq.(7.7.29) is given by,

$$F_{k,l} = (\Delta z_{l-1}/6\Delta x)[\ u_{k,l-1} + v_{k,l}\ ] + (\Delta z_l/6\Delta x)[\ u_{k,l} + v_{k,l}\ ] \qquad (7.7.32)$$

The values of the eddy diffusivity at the half integer nodes are computed at the start of each timestep and stored. The total matrix operation count is 6 multiplications, 4 additions and 4 subtractions.

For the term in the z direction, the integral,

$$G_{k,l} = \int_{x_{k-1}}^{x_{k+1}} \int_{z_{l-1}}^{z_{l+1}} \frac{\partial}{\partial z}[\kappa_e \frac{\partial \phi}{\partial z}] a_k b_l \, dx dz \qquad (7.7.33)$$

has to be computed. The same procedure as before is followed; integrate by parts and assume an irregular grid to introduce the vectors $\underline{\alpha}$ and $\underline{\beta}$. This leads to the equation,

$$G_{k,l} = (\Delta x/6\Delta z_{l-1})[\, \alpha_k p_{k-1,l-1} + \beta_k p_{k,l-1} + \gamma_k q_{k,l-1} \,] \qquad (7.7.34)$$

$$- (\Delta x/6\Delta z_l)[\, \alpha_k p_{k-1,l} + \beta_k p_{k,l} + \gamma_k q_{k,l} \,]$$

where,

$$p_{k,l} = 2\kappa_{k+\frac{1}{2},l+\frac{1}{2}} \Delta_z \phi_{k+\frac{1}{2},l} \qquad (7.7.35)$$

$$q_{k,l} = \kappa_{k,l+\frac{1}{2}} \Delta_z \phi_{k,l}$$

analogous to Eq.(7.7.28) where γ_k is given by Eq.(7.7.11). At k=0, the first terms in the square brackets in Eq.(7.7.34) vanish as α_k=0. However, these terms would be zero anyway from the planar shift south applied to the matrix holding the $p_{k,l}$ coefficients. The same applies to the terms multiplied by β_k as $p_{k,l}$=0 at k=N by definition. Therefore, Eq.(7.7.34) becomes,

$$G_{k,l} = [\Delta x/6\Delta z_{l-1}] R_{k,l-1} - [\Delta x/6\Delta z_l] R_{k,l} \qquad (7.7.36)$$

where,

$$R_{k,l} = p_{k,l} + p_{k-1,l} + \gamma_k q_{k,l} \qquad (7.7.37)$$

The only difference from the calculation of the x derivative is the calculation of $R_{k,l}$ and $G_{k,l}$. The vector $\underline{\gamma}$ has to be retained to ensure the boundary and interior values are computed concurrently.

The operation count for this algorithm is 7 multiplications, 3 additions and 3 subtractions in matrix mode. The calculation of the z derivative term requires less time than that for the x derivative despite the matrix multiplication required for the vector $\underline{\gamma}$. Like the product term algorithm, efficient use of the DAP processor array is made.

The approach of Staniforth and Beaudoin (1986) can be used to derive a

more efficient algorithm for the diffusion terms. Using their approach, the contributions formed by integrals along each level and half integer level are evaluated rather than from each grid rectangle. The changes to the algorithm are not described in detail as it was not used for the model or coded in DAP FORTRAN. However, it is possible to show that for the x derivative term, 4 multiplications, 3 additions and 4 subtractions in matrix mode are required. This is a reduction of 2 multiplications and 1 addition, or 24% of the CPU time of the DAP algorithm described above.

7.8. Efficient solution of the finite element equations

The final aspect to the implementation of this model on the DAP is the solution of 64 sets of 64 tridiagonal simultaneous equations. Each timestep requires the solution of 15 such sets and so efficient algorithms are essential. The solution to the equation,

$$\underline{\underline{A}}\,\underline{x} = \underline{b} \tag{7.8.1}$$

is required where $\underline{\underline{A}}$ takes two forms. For the streamfunction equation, Eq.(7.3.18), the leading diagonal is the sum of the other diagonals so $\underline{\underline{A}}$ is not diagonally dominant. For all other equations, $\underline{\underline{A}}$ is either the horizontal or vertical mass matrix in which the ratio of the leading diagonal to the lower and upper diagonals is 1:4. This is approximate for the vertical mass matrix with a nonuniform vertical spacing. Hence these systems are diagonally dominant.

Hockney and Jesshope (1981) studied the performance of 3 algorithms; Gaussian elimination applied in parallel, a serial cyclic reduction algorithm applied in parallel and a parallel cyclic reduction algorithm applied in parallel. Their analysis showed that for the solution of m systems of n equations on the DAP, the parallel cyclic reduction algorithm is more efficient when mn is less than or equal to the number of processors, as for this model. As mn increases, the analysis showed that there is some value at which it is more efficient for each processor to solve a tridiagonal system using a serial algorithm. Wait (1988) and Lai and Liddell (1987a) also discussed methods for solving Eq.(7.8.1) on the DAP when mn is greater than the number of processors.

The cyclic reduction algorithm has been used for a tridiagonal solver

routine in the DAP subroutine library (DAPLIB). Generally however, it has been found that iterative methods are more suited to the DAP, mainly because parallel versions of these algorithms are more efficient (Lai and Liddell, 1987). The Jacobi method (described in chapter 2) is one example and this was used for another subroutine in DAPLIB (Bowgen, 1981a,b). Recently, the preconditioned conjugate gradient algorithm has received much attention and has been shown to be an efficient solution method for finite element problems on parallel machines (Wait, 1988; Lai and Liddell, 1987b; Adams, 1983). In the sections that follow, the performance of the DAPLIB routines is evaluated and a preconditioned conjugate gradient algorithm developed.

The timings of the routines in the following sections were made by calling them in a DO loop 10^4 times. The cost of the loop itself was measured and found to be 40msecs. Since the DAP CPU times were reported to the nearest second, this introduces a larger error than the DO loop. Therefore all times reported are given subject to an error of ±0.1msecs. In the following tests, $\underline{\underline{A}}$ is set equal to the horizontal mass matrix with Δx=1 for convenience. The matrix $\underline{b}$ was set to,

$$b(i,j) = \sin[\pi(j\text{-}1)/128]\ \cos[\pi(j\text{-}1)/64] \qquad \text{for all } i \text{ and } j \tag{7.8.2}$$

7.8.1. DAP library subroutines

The cyclic reduction algorithm used for the DAPLIB routine F04TRIDS64SQ is described in chapter 2 and by Whiteway (1979). It is a direct method and completes in $\log_2 n$ steps for a $n \times n$ system. When timed for the test problem, this routine took 26.6msecs.

The iterative solver F04ITTRIDS64SQ in the DAP library uses the hybrid Jacobi method, described in chapter 2 and Bowgen (1981), in which a single pass of the cyclic reduction algorithm is used before iterating using the Jacobi method. This routine was timed and converged in 20.1msecs after 11 iterations.

This routine was optimized by altering the code so that the convergence test was first done after the 9^{th} iteration. When timed, the optimized version converged after 11 iterations in a time of 18.5msecs. This is 30% faster than the cyclic reduction DAPLIB routine and therefore preferred for the model.

However, the streamfunction equation cannot be solved this way as it is not diagonally dominant, so there must be at least one use of the direct method solver.

7.8.2. Conjugate gradient algorithm

The review by Lai and Liddell (1987a) showed that the groups in the U.K. using the DAP for finite element problems have all concentrated on the conjugate gradient (CG) algorithm for their solution method. The DAP Support Unit at Queen Mary College employed the method in their finite element library (Davies, 1985). The CG algorithm is well suited to parallel implementation on the DAP.

In this section, the tridiagonal matrix equation Eq.(7.8.1) is solved using the CG algorithm. This is novel as engineering problems involve matrices with greater bandwidth and it is not clear how efficient the CG algorithm will be compared to the DAPLIB solvers timed in the previous section. The routine written for the model solves 64 sets of 64 equations each, all in parallel.

The method of conjugate gradients was first proposed by Hestenes and Stiefel (1952) for solving a set of simultaneous linear equations having a positive definite matrix of coefficients. It is a direct method as convergence is guaranteed in n steps for an $n \times n$ system. However, convergence usually occurs in less iterations. Golub and Van Loan (1983) give a detailed description of the CG algorithm.

The CG procedure given below is taken from Lai and Liddell (1987a) but written to take account of repeated calculations. Matrix and vector notation is omitted for clarity. The matrix $\underline{\underline{A}}$ and $\underline{b}$ are those of Eq.(7.8.1). The solution at each iteration is given by $\underline{x}_k$ and $\underline{r}_k$ and $\underline{p}_k$ are known as the residual and search direction respectively, whilst $\underline{\alpha}$ and $\underline{\beta}$ are vectors. The CG algorithm is,

$$
\begin{aligned}
& x_0 = 0, \quad r_0 = b, \quad p_0 = r_0, \\
& w = \surd{<b,b>}, \quad y = \surd{<r_0,r_0>} \\
& \text{For } k = 1 \text{ to } n \\
& \quad q = A p_{k-1} \\
& \quad \alpha = y \; / \; <p_{k-1},q> \\
& \quad x_k = x_{k-1} + \alpha p_{k-1} \\
& \quad r_k = r_{k-1} - \alpha q
\end{aligned}
\qquad (7.8.3)
$$

$y' = \langle r_k, r_k \rangle$
If $\surd(y') \; / \; w < \varepsilon$ stop
$\beta = y' \; / \; y$
$p_k = r_k + \beta p_{k-1}$
$y = y'$

The angle brackets denote the inner product and ε is the accuracy required for convergence. The calculation of the matrix product, $\underline{q} = \underline{\underline{A}}\,\underline{p}_{k-1}$, and the inner products can be computed efficiently using DAP FORTRAN as described by Lai and Liddell (1987b).

When timed for the test problem, the CG routine converged after 10 iterations in a time of 46.7msecs. So although fewer iterations than the DAPLIB iterative solver are required, the complexity of each iteration makes the algorithm too expensive.

7.8.2.1. Preconditioning

The convergence rate of the CG algorithm is determined by the condition number of $\underline{\underline{A}}$, $K(\underline{\underline{A}})$, (Adams, 1982). If $\underline{\underline{A}}$ is symmetric, its condition number is given by the ratio of the maximum and minimum eigenvalues (Golub and Van Loan, 1983).

The convergence rate can be improved by multiplying Eq.(7.8.1) by a preconditioning matrix $\underline{\underline{P}}^{-1}$ to give,

$$\underline{\underline{P}}^{-1}\underline{\underline{A}}\,\underline{x} = \underline{\underline{P}}^{-1}\underline{b} \tag{7.8.4}$$

such that the condition number of $\underline{\underline{P}}^{-1}\underline{\underline{A}}$ is less than that of $\underline{\underline{A}}$. This will be true if $\underline{\underline{P}}^{-1}$ is approximately equal to $\underline{\underline{A}}^{-1}$. The choice of $\underline{\underline{P}}$ is clearly important and many authors have put forward possible forms for $\underline{\underline{P}}^{-1}$ (e.g. Lai and Liddell, 1987a). Furthermore, the choice of $\underline{\underline{P}}$ partly depends on the problem to be solved and the machine in use.

7.8.2.2. Cyclic reduction preconditioner

Like the Jacobi iterative solver, the cyclic reduction algorithm can be applied to Eq.(7.8.1) before the CG algorithm is used. This is equivalent to multiplying by a preconditioning matrix as in Eq.(7.8.4), and solving the transformed system,

$$\underline{\underline{A}}'\underline{x} = \underline{b}' \qquad (7.8.5)$$

This will only improve convergence if $K(\underline{\underline{A}}') < K(\underline{\underline{A}})$. Schendel (1984) finds the eigenvalues of a matrix similar to $\underline{\underline{A}}$ and shows that the condition number depends on the diagonal dominance of $\underline{\underline{A}}$. As the cyclic reduction algorithm increases the diagonal dominance, it can therefore be used as a preconditioner.

A routine that used the cyclic reduction method once before applying the CG algorithm Eq.(7.8.3) was written. The use of the cyclic reduction method adds a slight overhead to the CG algorithm itself during the calculation of the matrix product $\underline{\underline{A}}\underline{p}$, as the upper and lower diagonals are further away from the main diagonal, increasing the routing for this operation.

When timed for the test problem, this routine took 5 iterations to converge in a time of 28.4msecs. This is an improvement by a factor of 1.7 but not enough to achieve a better performance than the DAPLIB routines. Timings of the routine were also made using several passes of the cyclic reduction method before using the CG algorithm. With 2 passes, the solution converged in a CPU time of 23.4msecs after 3 iterations. With 3 passes, the CPU time was 22.8msecs after 2 iterations. The CPU time is now less than the DAPLIB solver based on the cyclic reduction method.

As the cyclic reduction algorithm is used for tridiagonal systems, it has not been used to improve the convergence of the CG algorithm by any of the previously cited references. However, for this particular application, it is clearly effective. To improve on the time for the optimized DAPLIB Jacobi solver, the preconditioning applied to Eq.(7.8.1) must be computationally inexpensive and the CG algorithm must converge very rapidly.

7.8.2.3. m-step Jacobi preconditioner

It is possible to write the CG algorithm Eq.(7.8.3) to include a preconditioning step at each iteration (Golub and Van Loan, 1983). The algorithm becomes,

$$x_0 = 0, \quad r_0 = b, \quad w = \sqrt{<b,b>},$$
$$\text{Solve } Pz_0 = r_0, \quad y = <z_0,r_0>,$$
$$p_0 = z_0,$$

For $k = 1$ to n

$$
\begin{aligned}
&q = Ap_{k-1} \\
&\alpha = y \,/\, <p_{k-1},q> \\
&x_k = x_{k-1} + \alpha p_{k-1} \\
&r_k = r_{k-1} - \alpha q \\
&\text{If } \surd <r_k,r_k> \,/\, w < \varepsilon \quad \text{stop} \\
&\text{Solve } Pz_k = r_k \\
&y' = <z_k,r_k> \\
&\beta = y' \,/\, y \\
&p_k = z_k + \beta p_{k-1}
\end{aligned}
\tag{7.8.6}
$$

The convergence is accelerated by the solution of the matrix equation, $\underline{\underline{P}}\underline{z}=\underline{r}$ at each iteration, where $\underline{\underline{P}}$ is the preconditioning matrix.

Lai and Liddell (1987b) studied the implementation of the preconditioned conjugate gradient (PCG) algorithm Eq.(7.8.6) on the DAP for engineering problems with several preconditioners. They concluded that, for problems requiring one processor per node, the m-step Jacobi preconditioner is preferred. Adams (1982, 1983) studied in detail the conditions for m-step preconditioners to be applicable and effective in solving symmetric positive definite systems and showed that these conditions were met by the Jacobi scheme. Adams (1982) also studied the implementation of these algorithms on the MIMD finite element machine.

Using a m-step Jacobi preconditioner, the equation $\underline{\underline{P}}\underline{z}=\underline{r}$ is solved by m steps of the Jacobi scheme Eq.(2.6.13). To see the form of $\underline{\underline{P}}$ following Adams (1982), $\underline{\underline{A}}$ is written as,

$$\underline{\underline{A}} = \underline{\underline{I}} - \underline{\underline{Q}} \tag{7.8.7}$$

where $\underline{\underline{Q}}=-(\underline{\underline{L}}+\underline{\underline{U}})$ from Eq.(2.6.13). Applied to the solution of the equation $\underline{\underline{A}}\underline{z}=\underline{r}$, this gives,

$$\underline{z}^{(m)} = \underline{\underline{Q}}\underline{z}^{(m-1)} + \underline{r} \tag{7.8.8}$$

Eq.(7.8.8) can be written as,

$$\underline{z}^{(m)} = \underline{\underline{Q}}^{m}\underline{z}^{(0)} + (\underline{\underline{Q}}^{m-1} + \underline{\underline{Q}}^{m-2} + \ldots + \underline{\underline{Q}} + \underline{\underline{I}})\underline{r} \tag{7.8.9}$$

If the initial guess, $\underline{z}^{(0)}$, is chosen to be zero, Eq.(7.8.9) becomes,

$$(\underline{\underline{Q}}^{m-1} + \ldots + \underline{\underline{Q}} + \underline{\underline{I}})^{-1}\underline{z}^{(m)} = \underline{r} \tag{7.8.10}$$

so that the preconditioning matrix is,

$$\underline{\underline{P}} = (\underline{\underline{Q}}^{m-1} + \ldots + \underline{\underline{I}})^{-1} \tag{7.8.11}$$

Adams (1982) shows this result holds for a general class of iterative schemes based on a splitting of $\underline{\underline{A}}$.

The convergence rate will be improved if $K(\underline{\underline{P}}^{-1}\underline{\underline{A}}) < K(\underline{\underline{A}})$. It can be seen from Eq.(7.8.10) that as m increases, $\underline{z}^{(m)}$ tends to $\underline{z}$ and $\underline{\underline{P}}$ tends to $\underline{\underline{A}}$. As $\underline{\underline{P}}$ becomes a better approximation to $\underline{\underline{A}}$ so $K(\underline{\underline{P}}^{-1}\underline{\underline{A}})$ will reduce as desired.

There will be some value of m however, for which the increase in the cost of the preconditioner offsets the reduction in CPU time from the increased convergence. This value of m will depend on the preconditioner in use but also on the computer. For example, the relative cost of the arithmetic to the cost of the communication.

Dubois *et al.* (1979) proved that the m-step preconditioner can reduce the number of iterations needed by a 1-step PCG algorithm by a factor of m at most. In practice, this theoretical limit may not be reached. The extent to which it is true depends on the distribution of the eigenvalues of $\underline{\underline{A}}$. However, the implication is that m-step preconditioning will be most effective when m is small.

For the tridiagonal system Eq.(7.8.1), it is only worthwhile considering simple iterative procedures as preconditioners if the time for the optimized DAPLIB routine is to be improved upon. Therefore the Jacobi scheme is used, following the recommendations of Lai and Liddell (1987b). To implement the preconditioner, the algorithm Eq.(7.8.6) is used with,

$$Pz_k = r_k$$

replaced by,

$$\begin{aligned} &z_k^{(0)} = 0 \\ &\text{For } l = 1, m \\ &z_k^{(m)} = Qz_k^{(m-1)} + r_k \\ &z_k = z_k^{(m)} \end{aligned} \tag{7.8.12}$$

A routine using the algorithm Eq.(7.8.6) with the m-step preconditioner, Eq.(7.8.12), was coded and timed for different values of m. The results are presented in Table 14.

The optimum value of m is 7, although this gives a time greater than both the DAPLIB solvers and also the CG algorithm with the cyclic reduction preconditioner. As expected, the greatest improvement is for small values of m. For example, with only one preconditioning step, the number of iterations required reduces from 10 to 6. The CPU time however only reduces by 20%. This is because the preconditioner increases the cost of each CG iteration, whereas the cyclic reduction algorithm was used only to precondition the matrix equation before the CG algorithm and did not alter the CG algorithm itself. The use of the cyclic reduction method to precondition the equations before the CG algorithm is clearly more effective.

7.8.2.4. Combined preconditioner

Since the cyclic reduction preconditioner is applied before the CG algorithm Eq.(7.8.3) is used, it is possible to combine the cyclic reduction preconditioner with the m-step Jacobi PCG algorithm Eq.(7.8.6). Both methods have been seen to improve the convergence rate and their combination is straightforward. The introduction of the cyclic reduction method adds an overhead to the initial calculations and the PCG iteration since, as explained previously, the amount of routing required for the calculation of $\underline{\underline{A}}\underline{p}$ increases.

A routine combining these two techniques was written and timed. The cyclic reduction preconditioner was used once. The results are presented in Table 15. As expected, for m=1 to 7, the use of the cyclic reduction method reduces the number of iterations for convergence. Again, the effect is most noticeable for small values of m, with a factor of 2 decrease in the number of iterations. The CPU times are all less than the cyclic reduction method DAPLIB routine (the time for m=5 is the same). Also, there are now two values of m for which the CPU time is below that of the optimized iterative DAPLIB solver (18.5msecs). In these cases the algorithm halts at the first convergence test. This is a relatively large reduction in the CPU time compared with two iterations as the code to compute a new search vector p_k and solve $Mz_k=r_k$ is not executed. Using this algorithm with m=7 instead of the optimized DAPLIB solver would mean a saving of 6.7% in the CPU time of the model.

Number of preconditioning steps	Iterations to convergence	Time (msecs)
1	6	38.9
2	5	38.1
3	4	35.1
4	3	30.0
5	3	33.2
6	3	36.4
7	2	27.0
8	2	29.2
9	2	31.3
10	2	33.4

Table 14.

The CPU time for a conjugate gradient algorithm using a m-step Jacobi preconditioner.

Number of preconditioning steps	Iterations to convergence	Time (msecs)
1	3	24.2
2	2	20.1
3	2	22.3
4	2	24.4
5	2	26.6
6	1	17.2
7	1	18.2

Table 15.

The timings of a m-step Jacobi preconditioned conjugate gradient algorithm. One pass of the cyclic reduction is used to precondition the equations initially.

It was previously found that several uses of the cyclic reduction method continued to reduce the iterations required for convergence. Therefore the algorithm was modified to include two and three passes of the cyclic reduction algorithm. The timings are presented in Table 16 and Table 17.

Table 16 shows that with two passes of the cyclic reduction algorithm, one iteration of the PCG algorithm can be achieved with only 2 preconditioning steps. The CPU time shows a slight improvement over the best time reported in Table 15. For the tridiagonal systems of this model, to be competitive with the DAPLIB optimized solver, the PCG algorithm must clearly converge in one iteration. When the cyclic reduction method was applied a third time, Table 16 shows the routine becomes more costly than with 2 passes. The cyclic reduction method is more efficient as a preconditioner because it gives a greater reduction in the CPU time for less additional work.

It should be noted that this PCG algorithm is specific to tridiagonal matrix problems. Furthermore, the use of the Jacobi preconditioner implies that for best performance the equation $\underline{\underline{A}}$ should be diagonally dominant. The streamfunction would therefore have to be solved using the cyclic reduction DAPLIB routine. The use of the cyclic reduction algorithm to precondition the equations is only possible with tridiagonal systems. It is not useful for matrices with a greater bandwidth.

Finally, the convergence rate of the CG or PCG algorithm might be improved by using the value of the variable to be solved at the previous timestep if it is available. The algorithms Eq.(7.8.3) and Eq.(7.8.6) have to be modified to do this. The initial assignments, $x_0=0$ and $r_0=b$ are replaced by $x_0=x^{(t-1)}$ and $r_0=b-Ax_0$. The technique would only have been possible for a small number of the matrix equations solved in the model since not all results at the previous timestep were available.

7.9. Storage requirements and performance

7.9.1. Storage requirements

The output from the DAP consolidator can be used to produce statistics on the use of the DAP store by the model (Table 18). The model's COMMON blocks take up a third of the available DAP store, so the model fits easily into the

Number of preconditioning steps	Iterations to convergence	Time (msecs)
1	2	21.9
2	1	16.8
3	1	17.8

Table 16.

As Table 15 but with two passes of the cyclic reduction method initially.

Number of preconditioning steps	Iterations to convergence	Time (msecs)
1	1	19.6

Table 17.

As Table 15 but with three passes of the cyclic reduction method initially.

Storage area	Size (kbytes)	Number of planes occupied	Percentage of total
Program code	69.69	140	3.4%
System	12.0	24	0.6%
Workspace	18.0	36	0.9%
Stack	1308.0	2616	63.9%
User COMMON	640.0	1280	31.2%
Total	2047.69	4096	

Table 18.

The storage required for the DAP finite element model. Values given are for the main sections of the binary file, as given by the DAP consolidator listing.

DAP.

The consolidator output also gives the space required for the model's individual COMMON blocks, presented in Table 19. The largest fraction of the user COMMON data is used for the matrices that hold the three diagonals for the x and z mass matrices and the streamfunction equation. For efficiency in the tridiagonal solvers, each diagonal is replicated in every column of its matrix. If it was necessary to economize on use of memory, these diagonals would be stored in vectors and broadcast across the PE array as required. The matrix constants would then occupy 19 planes instead of 577. However, the CPU time per model step would increase.

The finite element method is more suited to machines with a limited memory (2Mb for the DAPs at Edinburgh University) and a slow I/O than the spectral method, as there are no large arrays required. With about half the DAP store still available, there is more space to increase the number of nodes in the finite element model than there was for the spectral model.

7.9.2. Model timings

In this section, the CPU time per model step is presented for each different algorithm used to solve the matrix equations. These timings were performed by running the model for 1000 steps, using a timestep of 30 seconds. The mid-tropospheric jet and surface jet initial conditions were used to see if there were any differences caused by the initial conditions.

The results for the mid-tropospheric jet case are presented in Table 20. Those for the surface jet case are shown in Table 21. The DAP time for each run was approximately 6 minutes and was reported to the nearest second. Thus the timings presented can be considered accurate to 1msec. The tables show the algorithm used to solve each of the 3 types of matrix equation; x direction, z direction and the streamfunction. Also presented with each algorithm is the average number of iterations required for convergence, as monitored by the model.

There are several interesting points arising from these results. The unoptimized DAPLIB Jacobi routine takes less iterations for convergence, on average, than in the test case. The average number of iterations when the optimized version of this routine is used, however, is greater, although the

Use of store	Size (kbytes)	Planes	Percentage of total
Main variables	96.0	192	15.1%
Auxiliary variables	96.0	192	15.1%
Work COMMON	54.0	108	8.5%
Matrix constants	288.5	577	45.3%
Vector constants	4.5	9	0.7%
Scalar data	0.5	1	0.08%
Logical masks	0.5	1	0.08%
Output COMMON	96 (+1)	192 (+2)	15.2%
Total	637.0	1274	

Table 19.

The storage requirements for user data for the finite element model. The output COMMON block includes the size of the dummy COMMON (in brackets) required to page align the output COMMON. This offset varies with program size.

Streamfunction		x direction		z direction		Time per model step (msecs)
Algorithm	Iterations	Algorithm	Iterations	Algorithm	Iterations	
1	6.0	1	6.0	1	6.0	493
1	6.0	2	9.677	2	9.823	376
1	6.0	3	10.411	2	9.814	373
1	6.0	3	10.399	3	9.970	364
1	6.0	4	1.194	4	1.614	398
4	44.0	4	1.200	4	1.429	----
1	6.0	5	1.000	5	1.000	372

Table 20.

The CPU time per model step for various choices of algorithms to solve the three types of matrix equations indicated. Initial conditions are the mid-tropospheric jet case. The key for the algorithms is:

(1) DAPLIB cyclic reduction algorithm.

(2) DAPLIB hybrid Jacobi iterative algorithm.

(3) Optimized DAPLIB Jacobi iterative algorithm.

(4) 2-step preconditioned conjugate gradient algorithm.

(5) 3-step preconditioned conjugate gradient algorithm.

Streamfunction		x direction		z direction		Time per model step (msecs)
Algorithm	Iterations	Algorithm	Iterations	Algorithm	Iterations	
1	6.0	2	10.158	2	9.889	382
1	6.0	3	10.470	3	10.142	366
1	6.0	5	1.0	5	1.0	372

Table 21.

The CPU time per model step for various choices of algorithms to solve the three types of matrix equations indicated. Initial conditions are the surface jet only case. Key to the algorithms is the same as Table 20.

CPU time per model step is still less. The reason for the average being greater is because the minimum iterations for this routine is 9, the number of iterations performed before the convergence test is carried out. However, the standard DAPLIB routine computes an estimate of the minimum number of iterations required. If the optimized routine has a higher average, some matrix equations take fewer than 9 iterations to converge. If the model was run on the DAP again, an iterative routine with a test for convergence after every step should be used, with the model monitoring the maximum and minimum number of iterations required for all equations. From this, the routine could be optimized by adjusting the minimum number of iterations as necessary. The average number of iterations for convergence for the solution of the z direction equations shows much less increase than for the x direction equations when the optimized routine is used. This is because the nonuniform vertical spacing of the model gives a tridiagonal matrix in which the ratio of the main diagonal to the upper and lower diagonals is different from the ratio for the uniform spacing in the x direction. The implication is therefore, that convergence in 8 iterations or less occurs less frequently.

An interesting feature of these results is that when one type of equation is solved using a different algorithm, the average number of iterations changes for the other type of equations for which the algorithm is not changed. There are several examples of this in Table 20. When the x direction equations are solved using the optimized Jacobi routine instead of the standard DAPLIB routine, the average number of iterations for the z direction equations decreases. Similarly when the streamfunction equation is solved using the PCG algorithm, the averages for the x and z direction equations change. In all cases, the changes are slight but show the effect of the precision of the solution on the model fields. The results from these runs were compared. Occasional differences in the placement of contours drawn in some fields were noted but there were no significant differences seen on runs of 20 hours duration.

When the 2-step PCG algorithm was used, the average number of iterations for convergence for the x and z direction equations was greater than one, implying that sometimes more than one iteration is required (Table 20). When this occurs, a new search vector is computed. This is costly and Table 20 shows that the 2-step PCG routine performs worse than the standard

DAPLIB routines. It can also be seen that the average number of iterations for the z direction equations is greater than for the x direction equations, as expected. Thus, it becomes necessary to use a 3-step PCG routine. When tested previously, this still gave a faster time than the optimized DAPLIB Jacobi routine. From Table 20, this is no longer the case, as fewer iterations are required in the model using the optimized DAPLIB Jacobi routine than when tested. However, the 3-step PCG algorithm always requires one iteration. It is puzzling however, that whilst the number of iterations required for convergence in the DAPLIB Jacobi routines shows a decrease from the test case, the 2-step PCG algorithm, which uses a Jacobi preconditioner, shows an increase. No explanation can be offered for this. The model needs to be run on the DAP again to identify the equations for which a different number of iterations to the test case is required. The use of the 3-step instead of the 2-step PCG algorithm reduces the model CPU time by 6.3%.

The performance of the PCG algorithm when applied to the streamfunction equation is also shown in Table 20. There is a large increase in the number of iterations required as the mass matrix is not diagonally dominant. Only the cyclic reduction algorithm is suitable for this equation.

For the algorithms used here, the optimized DAPLIB iterative solver gives the best performance. This is contrary to the test results given previously. However, both the DAPLIB and PCG routines could be optimized further by examining the iterations required for convergence for every equation solved. It may be possible to reduce the computation in each routine for some of these equations. However, the gains in performance are likely to be small as are the relative performances of one routine to the other. The optimized DAPLIB routine is preferred for its performance and algorithmic simplicity.

Table 21 shows that the performance of the optimized and unoptimized Jacobi solvers and the CPU time per model step does depend on the initial conditions. When the unoptimized DAPLIB Jacobi routine is used, there is an increase in the average number of iterations for the x and z direction equations, increasing the CPU time per model step of 1.6% compared to the mid-tropospheric jet initial conditions. Without more detailed statistics from the model, it is impossible to say which equations are causing these increases.

7.9.3. Performance

The timings of the subroutines in the model are given in Table 22. Each model timestep is divided into four scans similar to the spectral model. The first scan computes diagnostic variables, such as the velocity components and eddy diffusivity, for that timestep. The other scans integrate forward in time the vorticity, potential temperature and jet velocity, in that order. By computing each variable in turn, workspace requirements are minimized. It was difficult to time the four scans accurately outside the model because the behaviour of the iterative solvers is data dependent. The values given in Table 22 for SCAN1, SCAN2, SCAN3 and SCAN4 are therefore estimates, calculated using measured timings of the routines called by the four scan routines. These estimates assume the use of the 3-step PCG routine which always executes in the same time.

The other routines in Table 22 were timed outside the model. Each was called in a DO loop and timed over a large number of calls (in the range 5000-500000). The timings presented allow for the cost of the DO loop but because the system reported the DAP CPU time to the nearest second, each time is subject to a possible error of ±1 in its last digit.

The solution of the matrix equations takes the greatest percentage of the CPU time per model step. The percentage of the model CPU time spent solving the matrix equations is shown in Table 23. There are 15 matrix equations solved each timestep. The average CPU time per call for the 3-step PCG and cyclic reduction DAPLIB routines are taken from the timings presented in the previous section. The average CPU time per call for the Jacobi routines was estimated from the cost of 1 iteration and the average number of iterations given in Table 20. As the streamfunction equation is always solved using the same algorithm, the percentages are shown with and without this calculation taken into account. The lower percentage is therefore the relative cost of the solution of the x and z direction equations only.

The solution of the matrix equations takes 74% to 81% of the CPU time of the model. The solution of the x and z direction equations alone takes 67% to 68% (excluding the figure for the cyclic reduction algorithm).

The relative costs of solving the matrix equations for the surface jet initial conditions run are the same as for the mid-tropospheric jet initial conditions

Routine name	Time per call (msec)	Main subroutines called
SCAN1	59.55*	M3XM, SOLVER(3), DEVRHS(2)
SCAN2	97.98*	SOLVER(4), DEVRHS(3), PRODRHS(2), DIFFX, DIFFZ, TIMESTEP, TIMEFILTER, VORBNDRY
SCAN3	97.78*	SOLVER(4), DEVRHS(2), PRODRHS(2), DIFFX, DIFFZ, TIMESTEP, TIMEFILTER, THETABNDRY
SCAN4	99.92*	SOLVER(4), DEVRHS(2), PRODRHS(2), DIFFX, DIFFZ, TIMESTEP, TIMEFILTER, YVELBNDRY
DEVRHS	0.724	NONE
PRODRHS	5.82	NONE
DIFFX	3.35	NONE
DIFFZ	3.18	NONE
M3XM	1.39	NONE
THETABNDRY	3.20	RATIO
VORBNDRY	1.45	NONE
YVELBNDRY	5.09	RATIO(2)
RATIO	1.86	NONE
TIMESTEP	0.362	NONE
TIMEFILTER	0.914	NONE

Table 22.

The execution times of the main subroutines of the finite element model and the subroutines that each calls.

* indicates these times are estimates, based on the time for the 3-step preconditioned conjugate gradient algorithm.

Algorithm	Average time per call (msecs)	Time per step (msecs)	Percentage of time per step	
			Including solution of streamfunction	Excluding solution of streamfunction
1	26.6	493	80.9%	75.5%
2	18.4	376	75.6%	68.3%
3	17.4	364	74.2%	66.9%
5	17.8	372	74.1%	67.0%

Table 23.

The percentage of the CPU time per model step that is spent solving the matrix equations. Percentages are shown for different choices of algorithm. The streamfunction equation is always solved using the same algorithm, percentages with and without this taken into account are shown. See Table 20 for the key describing the algorithms. Results are for the mid-tropospheric jet case.

and are therefore not shown. These results show that optimization efforts will give the best results if applied to the tridiagonal solvers.

Section 7 of this chapter described improvements to the algorithms that compute the finite element matrices for the product and diffusion terms. Using the timings for the routines PRODRHS, DIFFX and DIFFZ given in Table 22 and using a CPU time per model step of 364msecs (assuming the optimized DAPLIB Jacobi routine is used) the calculation of the finite element matrices for the product and diffusion terms taken 15% of the CPU time per model step. It was estimated in section 7 that the CPU time for the calculation of the finite element matrix for the product (PRODRHS) could be improved by 21%. For the x derivative diffusion term the improvement was estimated at 24%. The improvement for the z derivative diffusion term was not estimated but is assumed to be the same as the x derivative term. The reduction in the CPU time per model step if these improvements were made would be 12.0msecs, a decrease of 3.3%.

It is interesting to see the relative cost of computing the boundary conditions, which are computed in vector mode. Adding up the cost of the routines THETABNDRY, VORBNDRY and YVELBNDRY, assuming a CPU time per model step of 364msecs, these routines take 2.7% of the model CPU time.

7.9.4. Parallel processing performance

As for the spectral model, it is necessary to study the finite element model from a parallel processing point of view. It is obvious that this model makes more efficient use of the DAP PE array than the spectral model. The available parallelism does not alter during a model timestep in the same way that it does in the spectral model, as there are no transformations from gridpoint to spectral space. In the finite element model, as each processor is assigned a node, there is no further parallelism to be gained from calculations on the grid. However, several terms could be computed simultaneously on a larger DAP. For example, all the variables could be integrated in time simultaneously or derivatives could be computed simultaneously. Thus, by exploiting the same operations applied to different variables there are more potential processes than available processors. However, certain points in the program have a parallelism the same as the number of nodes (e.g. calculation of the eddy diffusivity coefficients) although these take only a small fraction of the

model time. A MIMD machine would be needed for further parallelism to be achieved.

The movement of data between processors can be considered as an overhead for any program on the DAP. By noting the time for the broadcasting and routing of data in the program this overhead can be computed. The total time per step for data routing is 15.0msecs, assuming the use of the optimized DAPLIB Jacobi routine, giving an overhead of 4.1%. This is much less than the value obtained for the spectral model. Data routing is clearly not an expensive overhead for this model. Almost 90% of the cost comes from the solver routines.

Using Eq.(2.6.3), the efficiency of the algorithms to compute the finite element matrices for the product and diffusion terms is 98.4% and 98.9% respectively. Their performance rates are 15.2Mflops and 16.9Mflops respectively. The efficiency of the algorithm to compute the finite element matrix for the derivative term is 98.4% and its performance rate is 11.3Mflops. For the tridiagonal solvers, the cyclic reduction DAPLIB routine has an efficiency of 86.4% and a performance rate of 9.7Mflops. The optimized Jacobi DAPLIB routine's efficiency and performance rate is 98.2% and 12.9Mflops.

The calculation of the finite element matrices and the solution of the tridiagonal equations together account for 90% of the model CPU time. Considering only the performance of these routines would therefore give a good estimate of the performance of the model. The efficiency of the model can therefore be calculated as 97% from Eq.(2.6.3). The performance rate is 13.1Mflops.

This model is therefore very efficient as almost all of the PE array is performing useful work during each step. However, it should not be inferred from this that a finite element model would be preferred to a spectral model for global modelling. The choice of grid on the sphere might result in a mapping of nodal data onto the processor array that leaves some processors unused. The triangular elements on an icosahedron in the global finite element model of Cullen and Hall (1979) might be one example.

7.10. Model results

The model of OR77 was used to study the quasi-steady circulation that develops within the initial frontal system. In this section, the results from this model are compared to two of the cases discussed by OR77.

7.10.1. Surface jet case

The first solution discussed by OR77 is a surface jet configuration representing cold and warm air masses separated by a frontal system with a jet whose maximum occurs at the surface. The jet profile is set using Eq.(7.2.17) with V_M=45ms^{-1}. The geostrophic wind, $U_g(z)$, is set according to,

$$U_g = 8.92\alpha_0(z) - 5.61$$

such that the front is advected to the right in the figures to be presented. OR77 refer to this case as SJ3(45).

Fig. 35 is reproduced from OR77 and shows the evolution of the cross-front flow field in the x-z plane for the SJ3 case. The frames to the left show composite contour diagrams of the potential temperature (solid lines) and the perturbation streamfunction (dashed lines) at 3, 11.43 and 19.43 hours into the run. The perturbation streamfunction, ψ', is defined by,

$$\psi' = \psi - \psi_g \tag{7.10.1}$$

where the geostrophic streamfunction, ψ_g, is given by the solution of,

$$U_g(z) = \alpha_0 d\psi_g/dz \tag{7.10.2}$$

The frames to the right of Fig. 35 show the jet velocity, v (solid lines) and the total streamfunction, ψ (dashed lines).

Circulation cells develop in the perturbation streamfunction by 3 hours. The vorticity within these cells is negative on the warm side of the front and positive on the cold side. These cells intensify until a quasi-equilibrium is achieved in which the cells coexist with the moving frontal system. As shown by the streamfunction, on the right of Fig. 35, the effect of this vorticity above the front is to cause air parcels to sink as they pass over the frontal surface. In contrast, the positive vorticity on the cold side of the front reduces the cross-front advection of cold air parcels close to the surface. This localized

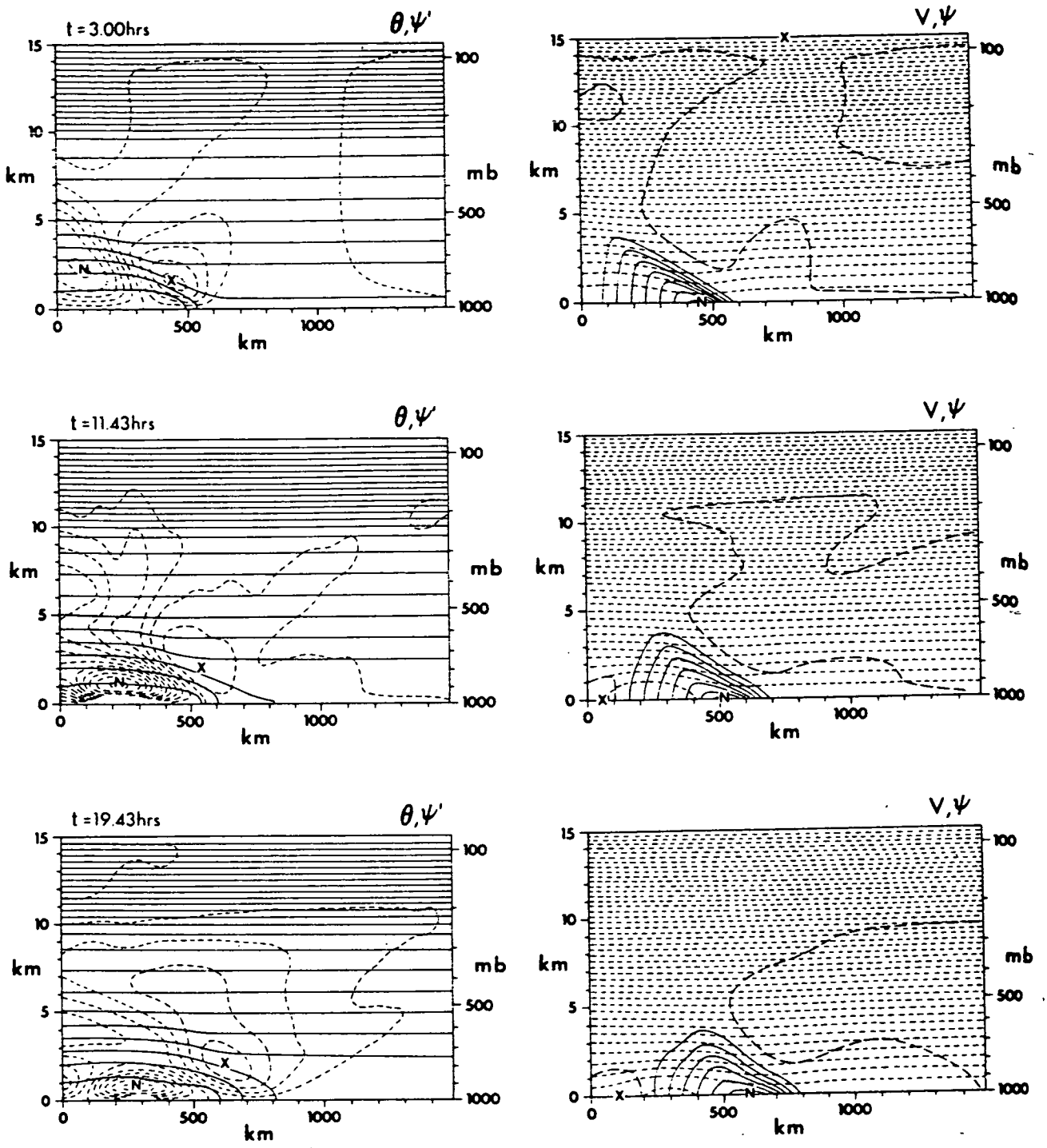


Figure 35. Reproduced from Orlanski and Ross (1977). Time sequence from surface jet case SJ3(45). Left frames show potential temperature (solid contours) and perturbation streamfunction (dashed contours) with X and N denoting locations of maximum and minimum streamfunction. Right frames show the jet velocity (solid contours with long dashed contours for zero lines) and total streamfunction (short dashed contours) with X and N denoting locations of maximum and minimum velocity. Contour intervals are: $\Delta\theta=5^{\circ}C$, $\Delta\psi'=250kg(ms)^{-1}$, $\Delta v=5ms^{-1}$, $\Delta\psi=2000kg(ms)^{-1}$.

blocking produces the dome in the streamfunction and potential temperature contours in Fig. 35.

Fig. 36 shows the potential temperature and perturbation streamfunction at 3, 12 and 20 hours from a run of the finite element model with the SJ3 initial conditions. The perturbation streamfunction shows the same early development of two circulation cells. The negative cell develops to the same dimensions as in the OR77 run and to about the same strength. However, the positive cell on the warm side of the front is clearly much too strong and its centre is lower than in Fig. 35. This means the model is generating more negative vorticity ahead of the front than in the run of OR77.

The potential temperature fields are in good agreement with those in Fig. 35. The dome shape of the potential temperature contours can be seen but it appears that the front is not advected as far in this model.

Fig. 37 shows the streamfunction and the jet velocity at the same times as in Fig. 36. The streamfunction shows the dome-like contours behind the front, the same as in OR77. However, there is also a feature associated with the nose of the front, at 700km at 12 hours. This is undoubtedly associated with the more intense circulation ahead of the front in this case.

The evolution of the jet velocity field is in broad agreement with that shown in Fig. 35. The flattening of the contours is well represented although the advection of the jet is some 100km less than in the OR77 case after 20 hours. However, the warm side of the jet shows different development. In Fig. 37 the jet has extended much further into the warmer air than in Fig. 35 and the depth of the jet is shallower. Forward of the jet minimum, a nose develops where the vertical gradient of v is negative over the first 200m, before becoming positive. Comparison with Fig. 36 shows this change in gradient is correlated with the horizontal gradient in the potential temperature at the surface. This implies the bottom boundary condition for the jet velocity is incorrect and this was confirmed when inspection of the program code revealed a programming error. The bottom boundary of the jet velocity is set to be in geostrophic balance according to Eq.(7.2.10), the values at the bottom level are set by Eq.(7.4.8). Unfortunately, in the model the θ_x term was added rather than subtracted. From Eq.(7.2.1), the primary generation of vorticity, at least initially, results from the deviation from thermal wind balance of the jet

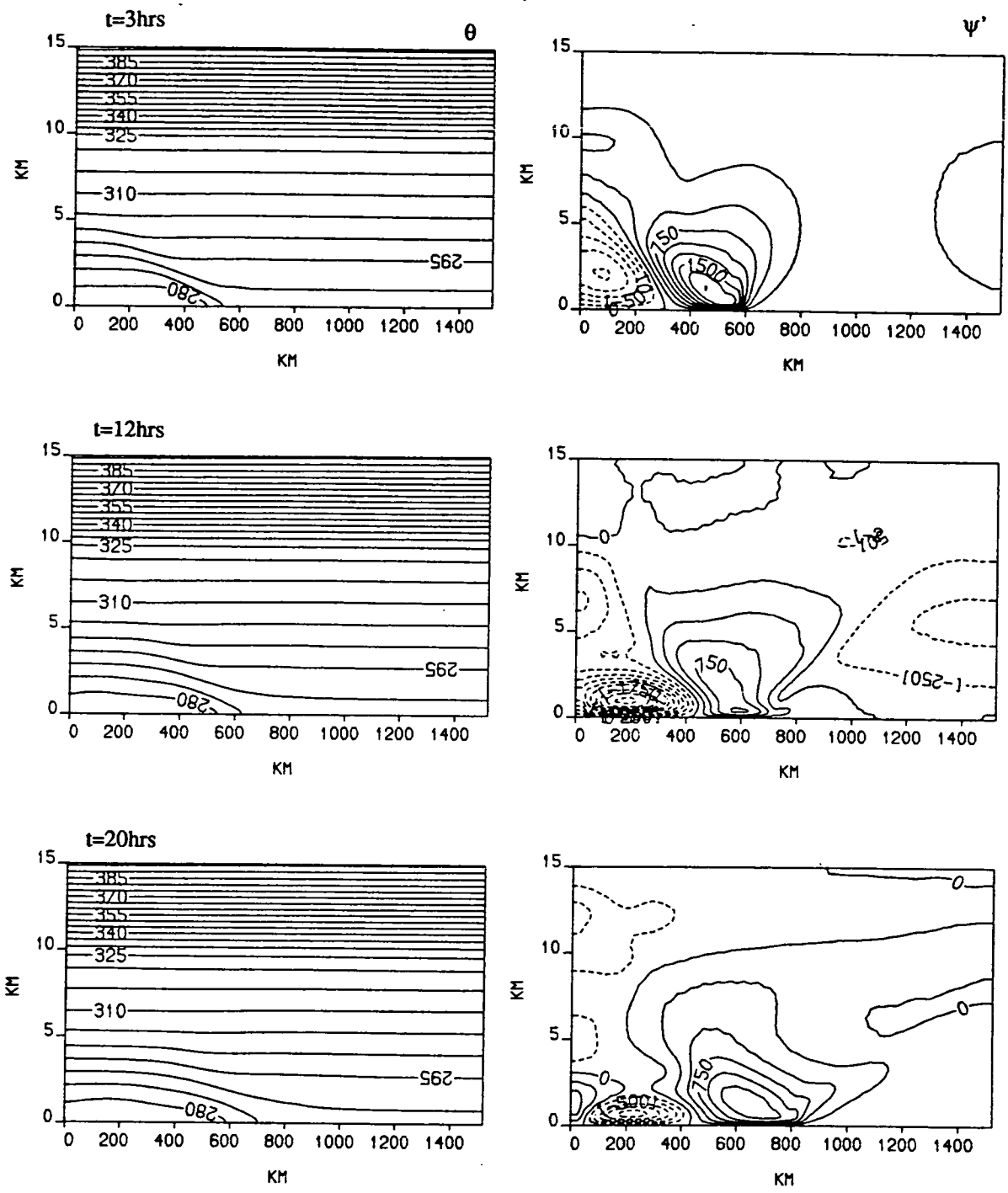


Figure 36. Results from the finite element model run of the SJ3(45) case presented by Orlanski and Ross (1977). Shown are the potential temperature and perturbation streamfunction fields at 3, 12 and 20 hours. Contour intervals are the same as in Figure 35.

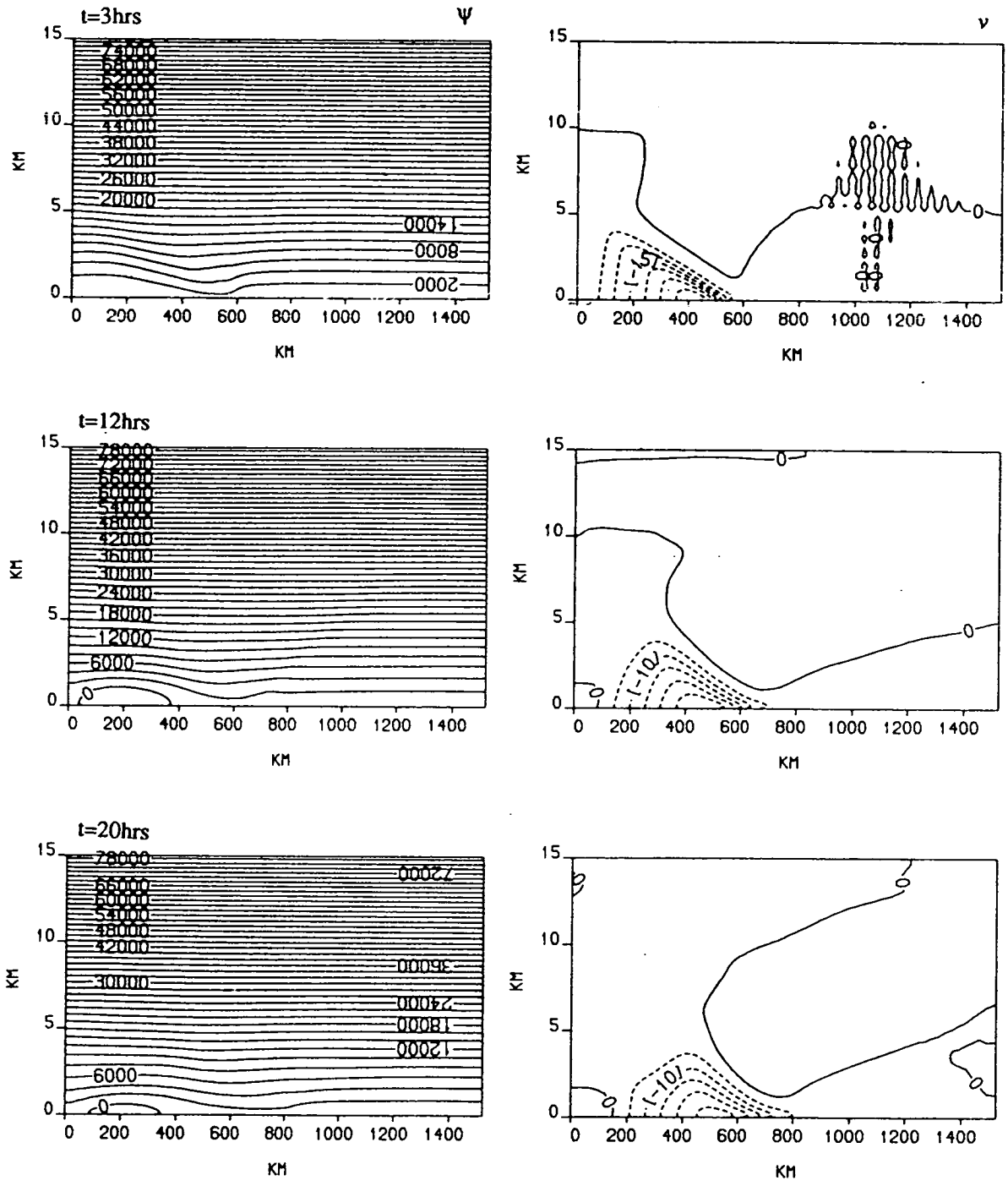


Figure 37. As Figure 36 but showing the total streamfunction and the jet velocity.

velocity and the potential temperature fields. Since the erroneous boundary condition ensures this is always the case at the front at the surface, more negative vorticity is generated in this region than should be. This is clearly apparent if the vorticity fields are examined and accounts for the stronger circulation in the perturbation streamfunction figures and the other discrepancies noted above. It is not clear to what extent the potential temperature fields are affected by this error. Unfortunately the Edinburgh DAPs were no longer available for this case to be rerun with the correct boundary condition.

7.10.2. Mid-tropospheric jet case

OR77 also presented results for a more realistic mid-tropospheric jet run. The initial jet velocity field is given by Eq.(7.2.18). The jet parameter, V_M, is set to 30ms^{-1} and the geostrophic wind field is set to,

$$U_g = 2 + 3\tanh(z/5000)$$

OR77 refer to this run as MTJ2(30).

Fig. 38 is reproduced from OR77 and shows the results for the MTJ2 case. The format of this diagram is the same as for the surface jet case. The fields are shown at times of 3, 10.87 and 14.86 hours. The MTJ2 initial conditions have comparable mean frontal wind shear in the jet velocity to the SJ3 case (which produces a similar horizontal gradient of potential temperature due to the initial geostrophic balance) and in the vertical shear in the geostrophic wind field at the front. Even though these shears are comparable, the circulation that develops is much stronger than in the surface jet run. A negative cell develops on the warm side of the front i.e. positive vorticity rather than the negative vorticity of the SJ3 case. The circulation in the colder air is much weaker by comparison. As a consequence, this circulation produces a blocking effect near the surface ahead or downwind of the surface front, thereby lifting air parcels ahead. This is seen clearly in the streamfunction shown in Fig. 38. The jet velocity profile remains largely unchanged except for the tilting produced by the vertical shear of the advecting wind.

Fig. 39 shows the streamfunction and potential temperature fields at 3, 11 and 15 hours from a run of the finite element model. The initial development

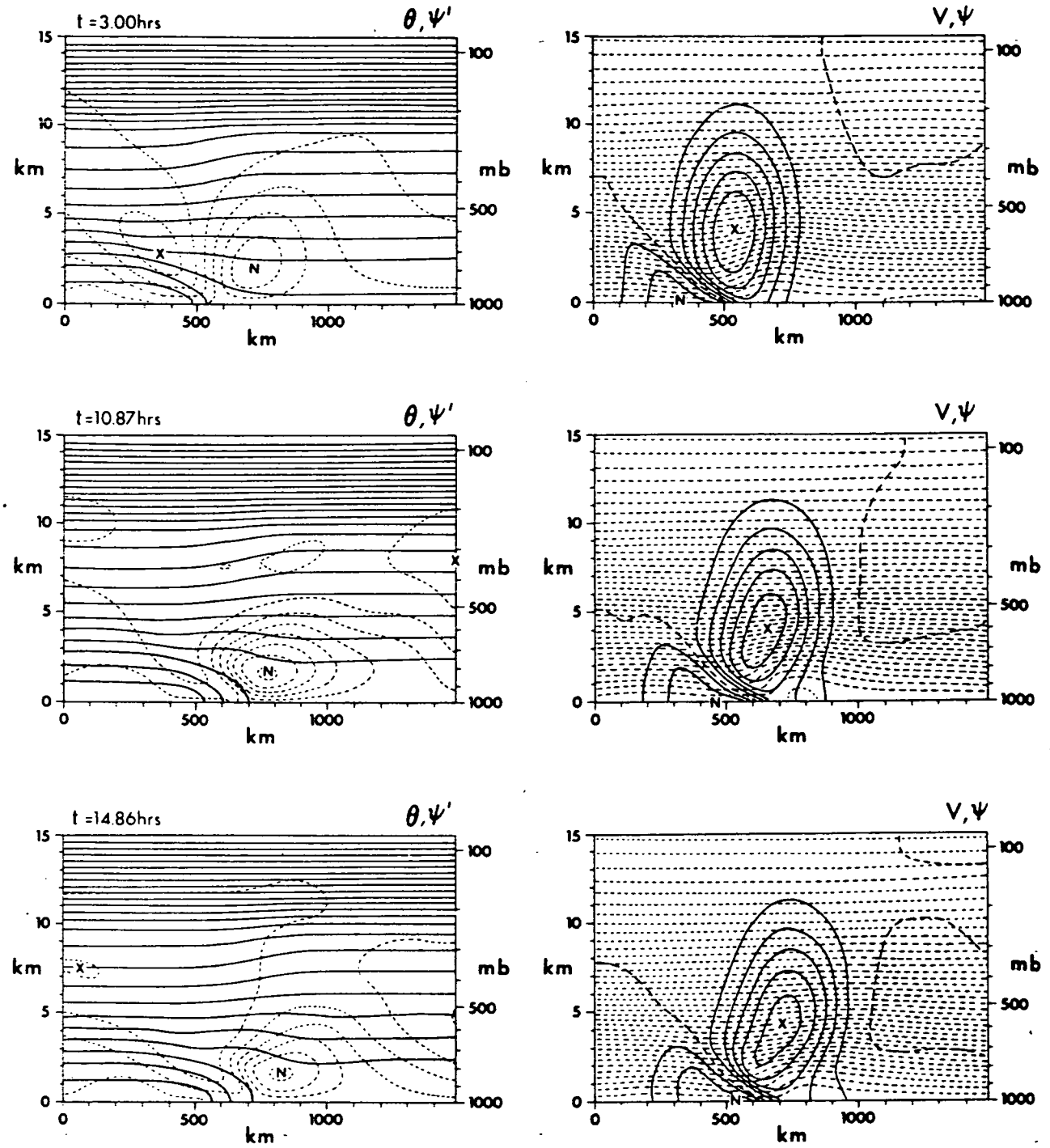


Figure 38. Reproduced from Orlanski and Ross (1977). Time sequence from the mid-tropospheric case MTJ2(30). Format is the same as Figure 35. Contour intervals are: $\Delta\theta=5^{\circ}C$, $\Delta\psi'=500kg(ms)^{-1}$, $\Delta v=5ms^{-1}$, $\Delta\psi=1000kg(ms)^{-1}$.

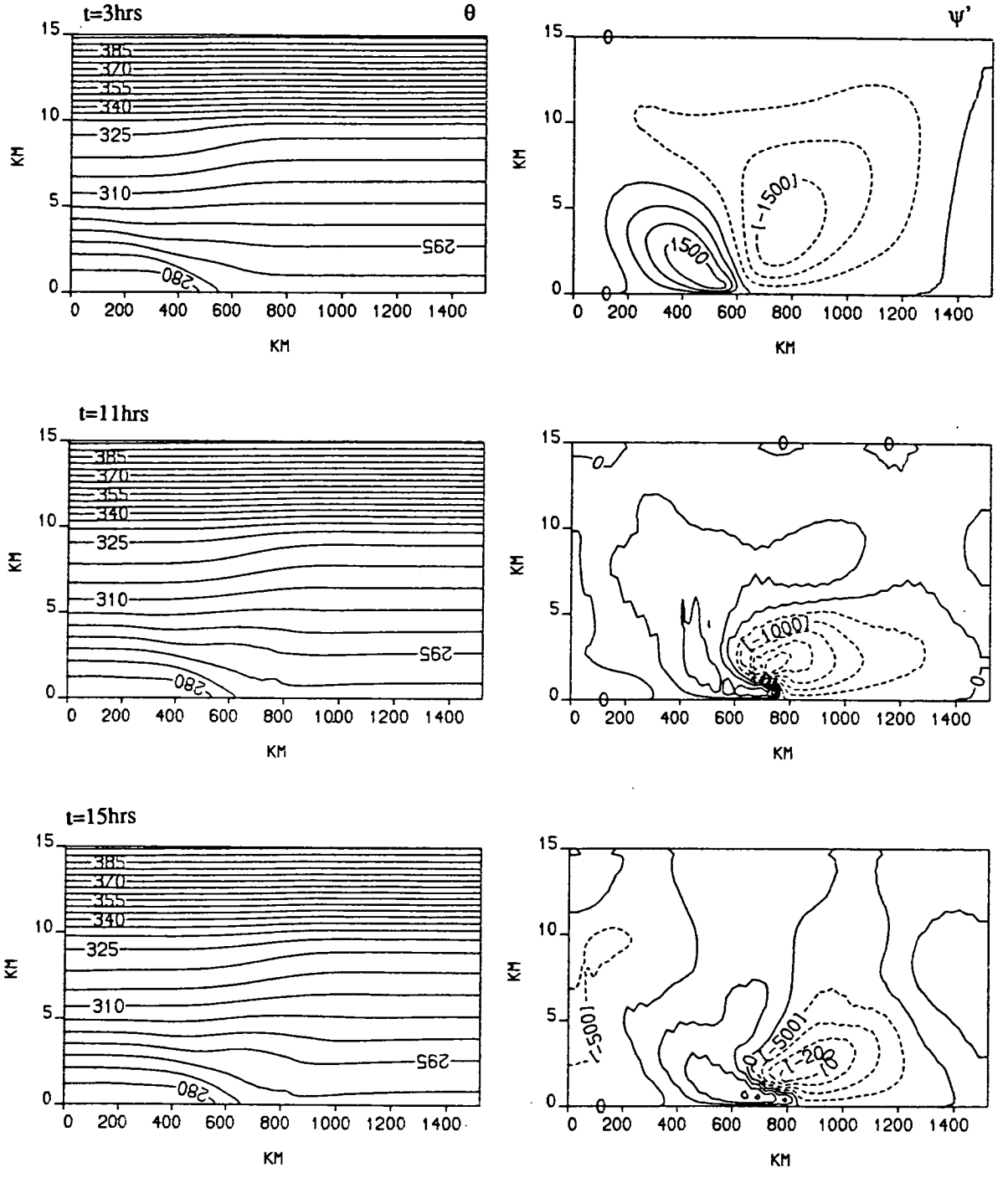


Figure 39. Results from the finite element model for the MTJ2(30) case of Orlanski and Ross (1977). Shown are the potential temperature and perturbation streamfunction at 3, 11 and 15 hours. Contour intervals are the same as Figure 38.

in the perturbation streamfunction is very similar to the OR77 results. However, the erroneous boundary condition produces negative vorticity at the surface at the front whilst, as in the OR77 run, positive vorticity is generated aloft. The result is that the positive perturbation streamfunction cell extends into the warmer air, as clearly seen in Fig. 39, producing an intense gradient at the interface of the negative vorticity near the surface and the positive vorticity aloft resulting in a strong circulation and numerical noise in this region. The effect of the erroneous boundary condition is more noticeable than in the SJ3 case. The potential temperature field also shows the effect of the strong circulation at the nose of the front.

The streamfunction and the jet velocity fields are shown in Fig. 40 for the MTJ2 case. The streamfunction shows the lifting ahead of the front but also the effect of the erroneous boundary condition with a small cell at a height of 1km. The jet velocity field reproduces the general features of the OR77 solution. The change in sign of the vertical gradient of v at the surface is again apparent.

To summarize, a programming error in applying the bottom boundary condition for the jet velocity has meant it has not been possible to make anything other than a qualitative comparison of results from the two models. The error has a significant effect on the circulations seen in the perturbation streamfunction, particularly for the MTJ2 case. However, the model appears to produce the general development described by OR77, so that devoid of the erroneous boundary condition the model should reproduce the results of OR77 well.

7.11. Discussion and conclusions

In this chapter, the implementation of the two-dimensional frontal model of Orlanski and Ross (1977) on the ICL DAP was described. The model was formulated using finite elements rather than finite differences as in the original model to test the suitability of this technique to the DAP. The streamfunction was solved outside the usual Galerkin framework to achieve a higher accuracy. The boundary conditions of this model were also not formulated in the way usual for finite elements. The lateral boundaries were computed using a fourth order accurate scheme rather than the second order scheme of the original model.

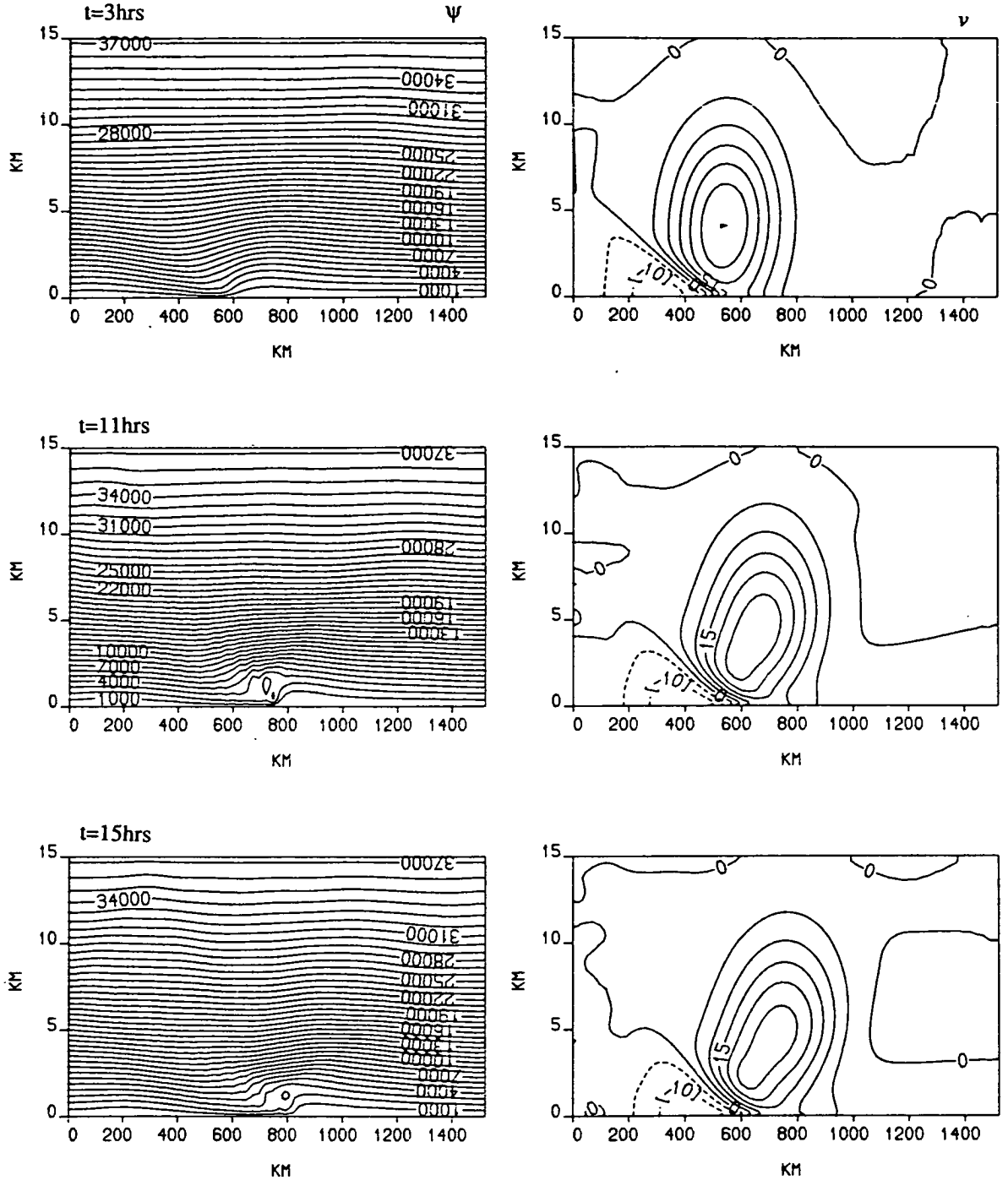


Figure 40. As Figure 39 but showing the total streamfunction and the jet velocity.

A detailed description of the solution procedure was given in which the equations were written using matrices. This made implementation on the DAP convenient as each processor was assigned an element of these matrices. The finite element method would usually use two-dimensional basis functions for a two-dimensional problem, but the equations of this model used basis functions written as the product of two one-dimensional basis functions so that the equations could be solved as a succession of one-dimensional problems, as advocated by Staniforth (1987). This had the advantage that tridiagonal matrix problems, instead of higher bandwidth matrix problems, had to be solved.

Parallel algorithms to compute the finite element matrices for the derivative, product and diffusion terms were derived. These were optimized for the semi-irregular grid of the model. They were also formulated to ensure that the boundary nodes, which required a slightly different calculation, were computed in parallel with the interior nodes. As the hardware boundary conditions of the DAP were used, this approach would not be possible if implementing these algorithms on a serial or MIMD machine (which did not have equivalent hardware conditions).

It was found that the calculation of these matrices took a 15% of the model CPU time per step. Most of the time, about 75%, was spent solving the matrix equations. Two DAP subroutine library routines were timed for a test problem and several conjugate gradient algorithms were designed and tested. A preconditioned conjugate gradient algorithm was developed which, for the test problem, had a superior performance to the DAP library routines and an optimized version of the Jacobi DAP library routine. However, when used in the model, these results were reversed and the optimized Jacobi routine gave the best performance. To be competitive, the conjugate gradient routine used preconditioning such that only one iteration was required. The use of the cyclic reduction algorithm was found to be successful at reducing the number of iterations to convergence. In this respect, the preconditioned conjugate gradient algorithm was specialized to this model. These solution methods would not be used on conventional architectures where methods such as Gaussian elimination are more appropriate.

A programming error prevented an accurate comparison of results from this model to those of Orlanski and Ross (1977). There is a general agreement

between the results however. Two-gridlength noise was apparent in the solutions presented for this model. Undoubtedly, much of this was a result of the programming error but there was some evidence of false reflections off the lateral boundaries. Further tests would be needed to confirm this.

The finite element method is suited to the DAP; a conclusion supported by other authors (Lai and Liddell, 1987a). The spectral method by comparison is not so well suited.[2] The matrix notation for the equations allowed their easy translation to DAP FORTRAN; the low overhead for routing data in the model supports this. Near optimum use of the DAP processors is made during a model step in contrast to the lower efficiency of the spectral model. The storage requirements of the finite element model are also much lower than the spectral model. In particular, the main storage requirements, the mass matrix components, increase linearly with increased resolution. However, the Legendre polynomial storage requirement in the spectral model increases quadratically with resolution. Overall, the development effort and time for this model was considerably less than that of the spectral model.

The finite element method is best suited to certain types of problems, for example limited area modelling, so it is not possible to recommend the use of the finite element technique over the spectral method for a global modelling problem. However, given a choice of architectures on which to implement the problem, one could comment on the preferred architecture.

Further optimization of the model is possible. It was described how the algorithms to compute the finite element matrices for the product and diffusion terms could be improved. The optimized Jacobi DAPLIB routine could be tuned to give a better performance. Lower precision arithmetic would also reduce the execution time. The best results would be obtained if the Jacobi routine used lower precision. However, it is not clear if this would be possible and further work would be needed to clarify this.

It has been shown that the minimum parallelism available in a SIMD sense is the number of nodes of the finite element grid. A MIMD approach would yield further parallelism. Therefore, a larger processor array would require the same number of nodes as processors for efficiency. This could be achieved with a three-dimensional model, but at this stage it is not obvious how best to implement a three-dimensional finite element model. Once again, the lack

[2] As the finite element and spectral models used different grids, this conclusion is based on the domain to which each method is commonly applied. However, limited area spectral models can use a Fourier basis function in both horizontal directions. Such a model would therefore Fourier transform only and would be more suited to the DAP for this reason.

variable size arrays in DAP FORTRAN is seen as a disadvantage for meteorological modelling.

CHAPTER 8

METEOROLOGICAL ALGORITHMS ON OTHER PARALLEL COMPUTERS

8.1. Introduction

It would not be prudent to discuss the application of the DAP to meteorological modelling without an understanding of the issues involved in using other parallel computer architectures. Several authors have recognized that a simple meteorological benchmark would give an indication of the likely performance on a particular computer. A finite difference model has therefore been implemented on different computers and this work is reviewed in the following section.

As the spectral transforms account for a large proportion of the execution time of spectral models, they are an obvious benchmark. FFTs are often one of the first algorithms applied to any new computer, in which case the Legendre transforms become an appropriate benchmark. The design of the Legendre transform algorithms is discussed in section 3. The issues in implementing spectral models on parallel computers are discussed in section 4.

In the summer of 1986, the author was employed by the ECMWF for 3 months to assist in the development of an enhanced multitasking strategy for their spectral forecast model. This work is described in the penultimate section of this chapter. In the final section, the multiprocessing strategy of the Meteorological Office's finite difference model is described, to contrast the scheme used by the ECMWF.

8.2. A finite difference meteorological benchmark

A number of authors used the finite difference shallow-water model in Cartesian coordinates described by Sadourney (1975b), as a meteorological benchmark for supercomputers and parallel computers. Hoffmann *et al.* (1988) ran the model on the CRAY-1, CYBER-205 and CRAY X-MP computers. Ikeda (1988) benchmarked the model on the Fujitsu VP-400. Tanqueray and Snelling (1988) applied the model to a 16-node FPS T-series machine. McBryan (1988) ran the model on the Connection Machine (CM-2), whilst Fishbourne (1980) used the model in spherical coordinates on the ICL DAP.

Table 24 presents a compilation of the results from these benchmarks. The

Computer	Grid size	Number of processors	Average performance (Mflops)	Percentage of peak speed
ICL DAP	64x64	4096	10	38%
Connection Machine (CM2)	512x1024	8192	214	5%
CRAY-1	64x64	1	61	38%
CYBER 205	64x64	1	39	20%
CYBER 205	64x64	1	110 (optimized)	55%
FPS T-series	64x64	16	13	7%
FPS T-series	256x256	16	41	21%
CRAY X-MP	64x64	1	98	47%
CRAY X-MP	64x64	2	168 (tasks)	40%
CRAY X-MP	64x64	2	188 (events)	45%
CRAY X-MP	512x512	1	148	70%
CRAY X-MP	512x512	4	560	67%
FUJITSU VP-400	64x64	1	379	33%
FUJITSU VP-400	256x256	1	567	50%

Table 24.

Performance of the shallow-water finite difference benchmark model for various computers. See text for additional comments.

performance figures given are average Mflop rates. Unfortunately, the authors did not all use the same grid size or the same degree of optimization so precise comparisons are difficult.

All the benchmarks were made with the same basic code (given by Hoffmann *et al*, 1988), with modifications only for language differences or machine specific optimizations, except Fishbourne's (1980) model. This was a global model and included Fourier filtering near the poles.

The CM-2 implementation was written in parallel LISP rather than FORTRAN. Although a powerful machine in theory, in practice, communication overheads and an inability to maintain the pipeline for the floating point processor yielded a poor average performance relative to the peak performance. However, for a three dimensional primitive equation model on the same number of gridpoints (a 128x128x32 grid), Pozo and MacDonald (1989) obtained an average performance of 1.1Gflops, 28% of the peak performance. It is not clear from the papers why the two models gave such different performances, but a likely explanation is that the three dimensional model had longer vectors. McBryan (1988) pointed out that there is a serious performance decrease, by a factor of 2.4, for grids that are not a power of 2 in each direction. This is because of the extra communications required for the periodic boundary conditions. It can be concluded that careful programming and a problem with a large degree of parallelism, many times the number of processors, are required in order to obtain a reasonable efficiency with the CM-2.

The CRAY-1, CYBER-205 and CRAY X-MP implementations used arrays declared as (65,65) to decrease memory bank conflicts. The CRAY-1 and CRAY X-MP versions used no other optimizations other than that performed by the compiler. The CYBER-205 version was written to give longer vectors than the CRAY version, to overcome the longer start up time of the vector pipeline on this machine. Another CYBER-205 version was written with the loops replaced by special vector subroutines. This optimized version (see Table 24) improved the performance by almost a factor of 3.

For the CRAY X-MP multitasking runs with 2 processors, two versions were written by Hoffmann *et al.* (1988). The first explicitly created processes for each piece of work and synchronized by waiting for processes to finish. The

other version created the processes once only and synchronized using signals or CRAY EVENTs. The better performance of the EVENT version illustrates the overhead of repeated process creation.

The program for the T-series was optimized for the vector processor at each node by subroutine calls. The programming strategy overlapped communications with computation. A one dimensional torus topology was used. The results show a poor performance relative to the peak performance of the machine. As Tanqueray and Snelling (1988) pointed out, this was due to the imbalance between the communication time and computation time, the improved performance of the 256^2 grid model illustrates this. Tett *et al.* (1988) implemented a spherical coordinate, finite difference shallow-water model on the Edinburgh Concurrent Supercomputer. The finite difference scheme was based on that used for the Meteorological Office forecast model. Unfortunately, they did not give a performance figure for their model.

The performance of the model on the VP-400 is impressive compared to the CRAY X-MP, as the machine has only one processor. Ikeda (1988) does not indicate if any optimizations were performed, but the figures from Table 24 suggest some scope for improvement.

Overall, Table 24 shows that performance figures between a third to a half of the peak performance of the pipelined machines were achieved for a 64^2 grid. Larger grids resulted in longer vectors and gave performance rates closer to the asymptotic limit. It is important to realize that this was achieved with standard, sequential FORTRAN, a vectorizing compiler and a modest amount of additional code for multiprocessing. The situation seems to be different on the distributed memory machines such as the SIMD CM-2 and MIMD FPS T-series. It appears more difficult to achieve rates close to the peak theoretical performance. For the DAP, the SIMD nature and simple interprocessor connections remove communication overheads to a large extent and a percentage performance comparable to that on the pipelined machines was achieved. Since most forecast models are now spectral, it would be interesting to compare the performance of a spectral benchmark on different machines.

8.3. Legendre transforms

8.3.1. Use as a benchmark

As mentioned in the introduction, the implementation of the Legendre transforms is a key issue in the design of spectral models on parallel computers. There are a number of reasons why these transforms would be useful as a benchmark for spectral models, as described below. There have been several recent studies of the Legendre transforms on parallel computers, such as Snelling (1988b), Hoffman and Nehrkorn (1989) and chapter 5 of this thesis, illustrating this point.

It has already been shown that the Legendre transforms account for the major computational part of a single-level spectral model. Even with multi-level forecast models they still account for a significant proportion. For example, Dent (1988) reported a figure of 22% for the T106 spectral model of ECMWF. Furthermore, the computation in the Legendre transforms varies as the cube of the truncation and therefore represents the fastest growing computational load.

The Legendre transform algorithms require a globally available array to hold the spectral data. This has an effect on the algorithm when implemented on a shared, hierarchical or distributed memory machine. It should be noted that the spectral data are the only data in a spectral model that are required to be globally available to all processors. The consequences of this are that synchronization and communication between processors are necessary for machines without a shared memory.

8.3.2. Algorithms

In this section, the algorithms used for the Legendre transforms are reviewed. A multi-level model is assumed.

In order to use a high resolution with a limited memory, it is usually necessary to design a spectral model such that only part of the data are held in central storage. As the calculations in gridpoint and Fourier space are independent of latitude, the usual approach is to scan through the latitudes, storing gridpoint and Fourier data for one latitude in memory at a time. As the summation for the direct Legendre transform is across latitudes, it implies

that each pass through the latitude loop adds a contribution to the spectral data. This approach is used at ECMWF (Dent, 1988).

However, Hoffman and Nehrkorn (1989) suggested an alternative scanning approach. They used latitude scanning for all gridpoint and Fourier space calculations, but looped over the zonal wavenumber m for the Legendre transforms and spectral space calculations. This has a number of advantages for a multiprocessing algorithm, which are discussed in the next section. A disadvantage is the additional memory that the wavenumber scanning approach requires. To store the Fourier coefficients of a variable for one latitude and one level requires $2(m+1)$ words. In contrast, $3m+1$ words would be needed per zonal wavenumber.

The shape of the retained modes in a triangular truncation means that the spectral coefficients are usually stored in one dimensional arrays. They can be stored in a column-wise or diagonal-wise format, as depicted in Fig. 16. These two formats result in two formulations of the Legendre transform algorithm, register-to-memory and memory-to-memory approaches respectively. The architecture of the target machine determines the best algorithm to use. That is, the register-to-memory approach is best suited to a machine with vector registers, such as the CRAY X-MP, whereas the memory-to-memory approach is best suited to a computer like the ETA-10 where the pipelines are fed directly from memory.

To illustrate the difference between the two approaches, consider the inverse Legendre transform,

$$F_m(\mu,z) = \sum_{n=|m|}^{M} F_{m,n}(z) P_{m,n}(\mu)$$

where z represents any vertical coordinate. Assuming a column-wise storage format, in FORTRAN this might be written as,

```
      DO 100 J=1, NLAT
      DO 100 M=1, (MT+1)
      DO 100 N=M, (MT+1)
      DO 100 L=1, NLEV2
100   FM(L,M,J) = FM(L,M,J) + FMN(L,M,N) * PMN(M,N,J)
```

where NLAT is the number of latitudes, MT is the truncation wavenumber (M=m+1) and NLEV2 is twice the number of levels. The real and imaginary components account for this factor of 2. The vector length is therefore NLEV2 but could be made longer if all the variables were stored together. As the Fourier coefficients depend only on m and not n, during the loop over N the Fourier data vector FM, over NLEV2, is invariant and can remain in the vector register whilst the sum over n is accumulated. Memory references are therefore minimized. In this example, the spectral coefficients are stored two-dimensionally for clarity.

Now suppose a diagonal-wise approach was used. The code becomes,

```
      DO 100 J=1, NLAT
      DO 100 N=1, (MT+1)
      DO 100 M=1, (MT-N+2)
      DO 100 L=1, NLEV2
100   FM(L,M,J) = FM(L,M,J) + FMN(L,M,M+N-1) * PMN(M,M+N-1,J)
```

The spectral coefficients are now accessed, in order, along diagonals. As the loops over M and N have been interchanged, the vector FM holding the Fourier coefficients over all levels and real and imaginary components has to be continually updated from memory. In practice, the transforms would use the symmetry property of the Legendre polynomials but the approach is the same.

The wavenumber scanning method, discussed previously, can be used with the register-to-memory algorithm since the loops over latitude and zonal wavenumber may be interchanged. However, this is not possible for the memory-to-memory algorithm. The wavenumber scanning method might therefore be better suited to computers with vector registers.

It is obviously important, when devising algorithms for a pipelined machine, to try to achieve vectors which are as long as possible. The vector length in the two cases above is relatively short, twice the number of levels, although this is used to advantage in the register-to-memory algorithm. Current forecast models typically have about 20 levels. The performance from a vector of 40 elements can be estimated from the equation relating the parameters, r_∞ and $n_{\frac{1}{2}}$. From Hockney (1988), the fraction r of the asymptotic

performance that would be achieved using a vector length n is given by,

$$r = 1 \; / \; (1 + n_{\frac{1}{2}}/n) \qquad (8.3.1)$$

Hockney (1988) gives $n_{\frac{1}{2}}$ as 60 for the CRAY X-MP for the type of calculation used in the inverse Legendre transform (known as the CYBER 205 triad), whilst Mozdzynski (1988) gives a figure of 53 for the ETA-10 (the CRAY X-MP $n_{\frac{1}{2}}$ should be regarded as approximate as it was measured for memory-to-memory operations). Therefore, with a vector length of about 40, only 40-45% of the asymptotic processing rate of the pipeline would be achieved. If several variables, say 4, could be processed in one vector, about 70% of the maximum rate could be obtained. Snelling (personal communication) increased the vector length by replicating the Legendre polynomials over the levels so that the vector length was over all levels, real and imaginary parts and the zonal wavenumber. However, when tried on the ETA-10, the scatter operation required was not available in the hardware and the cost of the additional code for this operation resulted in only minor improvements in execution time.

- The inverse Legendre transform can be written as a matrix times a vector for each m and latitude. Optimized matrix algebra routines often exist in the libraries provided with pipelined computers. The direct Legendre transform can also be written as a matrix times vector for every m and n. Therefore this approach could not be used with the latitude scanning model, only a wavenumber scanning model or one in which all data could be held in memory.

8.4. Parallel implementation of spectral models

Spectral models have so far been applied to pipelined machines with shared or hierarchical memories and a modest number of processors (10 or less). In this section, the techniques for applying spectral models to MIMD shared and hierarchical memory machines are reviewed, with some comment on implementing them on distributed memory machines.

8.4.1. Multiprocessing

For a computer with a small number of processors, the obvious way to spread the workload across the processors is to assign a subset of the latitudes to each processor for the latitude scanning approach, or a subset of latitudes and wavenumbers for the latitude-wavenumber scanning approach. This is efficient if the work per processor is the same. For a model with no physical parametrizations, Hoffman and Nehrkorn (1989) found that load balancing problems arose when the number of latitudes or wavenumbers was not an exact multiple of the number of processors. For a model that includes physical parametrizations, load imbalances also occur due to lack of convective activity in the polar regions (Dent, 1988). In these cases, dynamic scheduling improves the load balance.

An important difference between gridpoint and spectral models is that a synchronization point is required every timestep in spectral models. This is because a new step cannot commence until all contributions to the spectral coefficients during the direct Legendre transform have been accumulated. For a gridpoint model however, there is no such requirement. As pointed out by White and Wiley (1988), sub-domains of the grid can be integrated forward in time independently, although the lack of boundary information limits the maximum difference in timesteps between two adjacent regions. The latitude scanning approach requires a minimum of one synchronization, at the end of the latitude loop. The latitude-wavenumber scanning approach however requires two synchronization points, one at the end of each loop. This need not result in a worse performance if the processes are suitably scheduled (Hoffman and Nehrkorn, 1989) and has the advantage that spectral space calculations can be multiprocessed with the transforms. For a latitude scanning approach, spectral space calculations have to be performed on one processor, or they can be microtasked or multitasked if the grain allows.

8.4.2. Data management and communication

The requirement that the spectral data are available to all processors is easily satisfied by shared memory MIMD computers (sMIMD). The spectral data are only read during the inverse Legendre transform so no problems arise. However, as the direct Legendre transform updates the spectral data, this forms a critical region if the latitude scanning approach is used. The

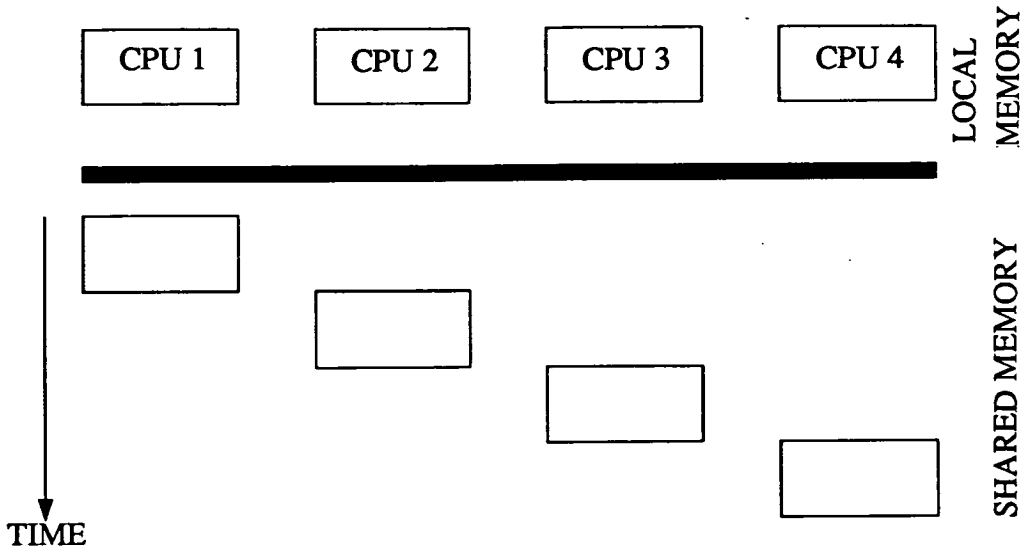
updating procedure must be protected to ensure that two or more processors cannot update the same data at the same time. The other concern is that results must be reproducible.

With the latitude scanning scheme, reproducibility can be ensured by using signals to force processes to increment the spectral data in a set order. This incurs an overhead because processes may not be at the point where they are ready to receive the signal before it is sent. Dent (1988) reported that these delays accounted for 1% of the total wall-clock execution time of the ECMWF model. However, reproducibility is assured if the Legendre transforms are multiprocessed over the zonal wavenumber m. This is because the summations over total wavenumber and latitude in the inverse and direct transforms respectively, are contained entirely within each process (Hoffman and Nehrkorn, 1989).

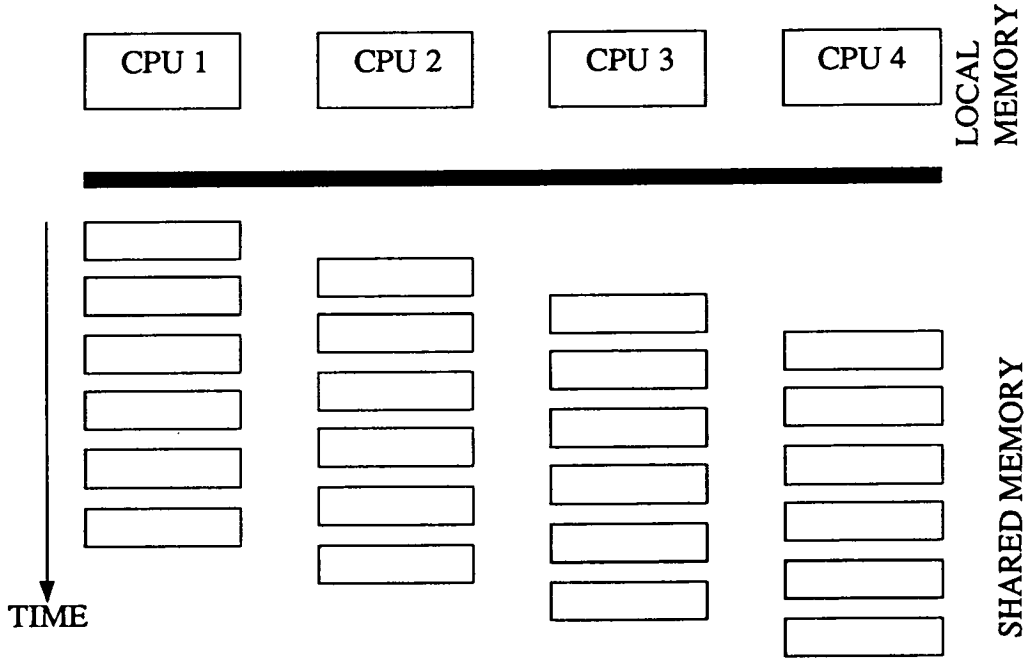
When a hierarchical memory MIMD machine is used (hMIMD), the appropriate technique depends on the size of the local memory attached to each processor i.e. there may not be enough memory available to store all of the spectral coefficients. Snelling (1988b), Hoffmann and Snelling (1988) and Mozdzynski (1988) applied the Legendre transform algorithms to the ETA-10.

The ideal case is when there is sufficient local memory to hold a complete copy of the spectral coefficients. The complexity of the multiprocessing strategy is then contained in the method used to perform the associative reduction of the data at each timestep. A sequential method would be where the first process copies its spectral data (for the latitudes it processed) to shared memory. All other processes then copy, update and copy the data back to the shared memory in turn. If the spectral data are split into a number of segments, the second process can read, update and write the first segment whilst the first process writes the second segment in parallel and so on. These two methods are illustrated in Fig. 41. Whilst the parallel method is more efficient, increasing the number of segments improves the parallelism but increases the overhead from the I/O operations. Mozdzynski (1988) presents the optimum number of segments for the ETA-10. In this approach, each processor must read and write the entire spectral data once.

If the local memory is not sufficient to contain all the spectral data, it has to be split into several parts. Each part is copied to local memory and



(a) Serial reduction of spectral data for the direct Legendre transform.



(b) Parallel reduction of spectral data.

Figure 41. Redrawn from Mozdzynski (1988).

updated in turn. A natural partitioning of the spectral data is into symmetric and antisymmetric parts and by variables. If this does not reduce the memory requirements enough, the spectral coefficients can be separated by zonal wavenumber m. Hoffmann and Snelling (1988) describe the results of simulating such algorithms, where the spectral coefficients are separated by zonal wavenumber into 8 and 16 parts (for a T63 resolution), on a 4 processor CRAY X-MP. They measured a factor of 2 reduction in the speedup obtained for the 8 and 16 part algorithms over the case where all the spectral data resides in memory, because of the additional costs involved.

A latitude-wavenumber scanning method can also be applied to a hMIMD computer. Data movement is required at both the necessary synchronization points in this method. After the latitude loop, the processors must gather the wavenumbers at the other latitudes they require, whilst after the wavenumber loop they gather the latitudes at the other zonal wavenumbers. The key differences between the two scanning approaches, are that no spectral data are transferred and the communication necessary is to exchange data and not as part of the summation for the direct Legendre transform. All data transfers to and from the shared memory can therefore proceed in parallel. Hoffman and Nehrkorn (1989) use a set of shared, protected variables to indicate when data are available to be copied to a processor's local memory. Although transfer of data to and from shared memory is required at two distinct points in the program, there are less data to be transferred at each point, given enough processors. Suppose there are p processors and each processor is assigned $(3M+1)/2p$ latitudes and $(M+1)/p$ zonal wavenumbers, where M is the truncation wavenumber. At the end of the latitude loop, each processor has to write its Fourier data to shared memory. This is $(3M+1)(M+1)/p$ words per variable per level. It must also read the Fourier data it needs for the wavenumber scan, $(3M+1)(M+1)(p-1)/p^2$ words per variable per level. The number of words, W, transferred per processor in the latitude-wavenumber scanning approach at the end of each loop is thus,

$$W = (3M + 1)(M + 1)(2p - 1) \; / \; p^2 \qquad (8.4.1)$$

For a latitude scanning approach, each processor must read and write the entire spectral data, $2(M+1)(M+2)$ words per level per variable. Equating the two gives p=3. Therefore, when 3 or more processors are being used, less words are transferred per processor using latitude-wavenumber scanning than

for latitude scanning.

An important consequence of looping over m for the spectral calculations, is that each processor does not need to hold all the spectral data, only the coefficients for the wavenumbers assigned to it. The space required decreases with an increasing number of processors.

Static scheduling is advantageous for latitude-wavenumber scanning because certain constant data and data at previous timelevels can be fixed to a processor. With dynamic scheduling, these data, some of which are spectral, must also be communicated to processors, increasing the amount of communication (Hoffman and Nehrkorn, 1989). To overcome load balancing problems, polar latitudes could be paired with equatorial latitudes and, for a triangular truncation, low wavenumbers with high wavenumbers, to give processors a similar workload.

8.4.3. Processor arrays

The transputer array is an example of a distributed memory MIMD computer (dMIMD). To obtain optimum performance from dMIMD architectures, communication between processors must be kept to a minimum and localized. In this respect, the latitude-wavenumber scanning scheme is the only one suitable for dMIMD architectures. Coupled with a static scheduling scheme, communication (between processors now rather than to and from a shared memory) could be kept to a minimum. As Eq.(8.4.1) varies as $1/p$, communication requirements per processor scale well with the number of processors.

An important question to address in implementing spectral models on dMIMD computers is the optimum topology. It may be the case that different topologies give the best performance for different operations. For example, gridpoint calculations are independent of their neighbours so the topology is irrelevant. However, some calculations couple the vertical levels so each processor should process all levels for one gridpoint. This is an important difference between gridpoint and spectral models on dMIMD machines since the topology for a gridpoint model is determined by the nearest neighbour interactions from the finite difference equations (Tett *et al*, 1988). The topology for a spectral model would therefore be determined by the need for

efficient FFT and Legendre transform algorithms.

Hoffman and Nehrkorn (1989) suggested a ring of hypercubes as a topology for spectral models. Along each latitude, the gridpoints are connected as a hypercube, allowing the FFTs on each latitude to be conducted efficiently. Assuming a static scheduling scheme, the Fourier coefficients at the end of the latitude and wavenumber loops would be sent around the ring. Although the total amount of data travelling around the ring is more than if the spectral coefficients were sent, each processor only writes and reads a small fraction rather than the whole spectral array. Furthermore, a static scheduling scheme would mean each processor could direct its data to those processors that require it. This would be impossible with a dynamic scheduling scheme.

A dMIMD implementation would suffer from some of the same inefficiencies as the DAP implementation. That is, as not all of the Fourier wavenumbers are retained, some processors would be left idle. However, it might be possible to keep these processors busy by assigning several levels from a neighbouring process. Hoffman and Nehrkorn (1989) suggest an alternative method where 2 or more gridpoints or wavenumbers are assigned to a processor. This is useful because it reduces the communications required. The ideal case is when each processor contains all the gridpoints for one latitude so that the FFT is performed using a sequential algorithm and only the communication of Fourier data at the end of the loops remains.

8.5. The ECMWF MIMD spectral model

In this section, the ECMWF forecast model is used as an example of the development of a MIMD model on a sMIMD computer. The contribution, by the author, to the development of an enhanced multitasking strategy is also described. D.Dent and D.Snelling were supervisors for this work.

The development of the multiprocessing aspects of the ECMWF model is documented by Gibson (1985), Dent (1988) and Dent and O'Neill (1988). It currently runs on the CRAY X-MP/48 at ECMWF.

8.5.1. Overview

The first version of the spectral forecast model had a resolution of T63 and 16 levels and was run on a CRAY-1A. In 1985, a CRAY X-MP/22 was purchased and the model multitasked to allow a T106 resolution. Then in 1986, with the arrival of the CRAY X-MP/48 at ECMWF, further development allowed the model to utilize 4 processors and the vertical resolution was increased to 19 levels.

8.5.1.1. Structure

The ECMWF model uses the latitude scanning approach described previously. To eliminate the need to store spectral coefficients at two time-levels, the model is organized into two latitude scans, as illustrated in Fig. 42. The model resolution used for forecasting requires gridpoint and Fourier data to be held on backing store in workfiles. During the latitude loops, these data are brought into memory one latitude at a time.

At the start of scan 1, the variables are in Fourier representation. The work in scan 1 consists of; input of Fourier coefficients, inverse FFTs, gridpoint space calculations, direct FFTs and direct Legendre transforms. There are also some computations performed in Fourier space, notably concerning the semi-implicit time scheme. The time-stepping is completed in spectral space and diffusion applied. Scan 2 consists of the inverse Legendre transforms and the output of the Fourier coefficients to the work files, ready for the start of scan 1 at the next timestep.

8.5.1.2. Data and I/O

The model uses workfiles for the Fourier data, Legendre coefficients and the gridpoint data. At the start of scan 1, a northern hemisphere and southern hemisphere latitude row of Fourier coefficients are read in. These are necessary to construct the symmetric and antisymmetric coefficients for the Legendre transforms. This implies that the length of the latitude loop is over half the model latitudes and each pass processes a northern row and its corresponding southern row.

Before the calculations in gridpoint space, gridpoint data are read from the gridpoint workfiles. After the gridpoint calculations on each latitude row are

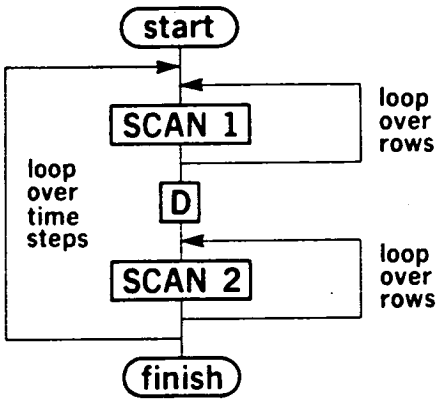


Figure 42. Scan structure of the ECMWF spectral model. From Dent (1988).

completed, the gridpoint data are written back to these files.

The latitude loop in scan 2 is also over half the model latitudes. At the start of each pass, the required Legendre polynomials are read from workfiles. At the end, the Fourier data for the northern and southern rows are output to the Fourier workfiles.

All these workfiles are normally stored in the Solid-state Storage Device (SSD). As the transfer rate between SSD and main memory is fast, all I/O is performed synchronously i.e. the model waits until the I/O has completed. Dent (1988) reported that the penalty for synchronous I/O is about 3% of the execution time. All I/O takes place to and from buffers declared within the model. If the SSD is not used, the model stores the workfiles on disk with all I/O double buffered. This means that twice as many I/O buffers are declared in the model and all I/O is asynchronous. That is, whilst computation is proceeding on the data in one buffer, the data for the next latitude is being read into (or written from) the other buffer. Using the SSD therefore reduces memory requirements and simplifies the code.

8.5.2. Static scheduling schemes

8.5.2.1. Original approach

As the model was first multitasked on the CRAY X-MP/22, memory use was the dominant concern. A multitasking strategy was therefore chosen that provided efficient computation with minimum memory requirement. The approach chosen split scan 1 into two pairs of processes after the Fourier data had been read in. The first pair of processes includes all the work between the Fourier space calculations inclusive. One process uses the northern hemisphere row whilst the other process uses the corresponding southern hemisphere row. The second process pair compute the direct Legendre transform. The Legendre transform separates naturally into a process to update the symmetric components of the spectral data and another to update the antisymmetric components. Since the processes therefore update separate halves of the spectral data, there is no danger of writing to the same memory simultaneously and hence no critical region. However, there must be a synchronization point between the pairs of processes as both northern and southern rows contribute to the symmetric and antisymmetric Fourier

coefficients.

In scan 2, another pair of processes compute the inverse Legendre transform for the north and south latitude rows of Fourier coefficients. The diffusion calculations between the two scans have a sufficiently large grain that they can also be multitasked. This multitasking structure for two processors is illustrated in Fig. 43.

This scheme can be extended to any even number of processors by replicating the multitasking strategy described above. Each pair of processors is given a pair of north and south latitudes to process. The Fourier coefficients for each pair of processors are read in by the odd numbered processors (starting from 1) before the process to compute the southern row is created. A static scheduling scheme is used where the rows are assigned to processors at compile time rather than run time. This multitasking procedure is illustrated schematically in Fig. 44. Scan 2 can be multitasked over 4 or more processors in the same way.

However, with 4 or more processors, the updating of the spectral coefficients during the direct Legendre transform becomes a critical region as there are now two processors trying to update the symmetric and antisymmetric parts. The ECMWF model uses two mechanisms to prevent this happening. The first protects the critical region of code using CRAY LOCKs. A process that enters the critical region sets the LOCK variable and unsets it when completed. Any process that finds the LOCK set when it reaches the critical region must wait until it is released. Unfortunately, with 4 or more processors the order in which the contributions for symmetric and antisymmetric parts are added to the spectral data is indeterminate. The second method of protecting this critical region is through the use of CRAY EVENTs. This facility is used to control the order of processors incrementing the spectral data to ensure reproducibility.

Whilst each process must be synchronized in every pass of the latitude loop, the process pair for a northern and southern row contain a latitude loop and do not need to be synchronized with other process pairs until they have finished all their rows (see Fig. 44). However, although the amount of work in the dynamics part of the model is the same for each latitude, this is not true in the physical parametrization part due to the variation in convective activity.

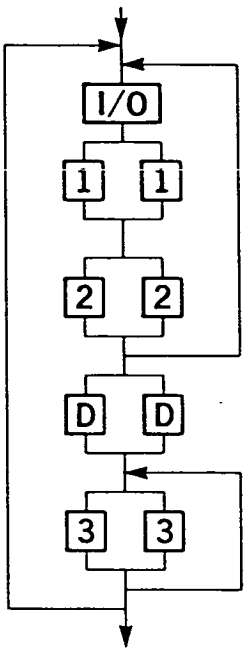


Figure 43. Original multitasking structure of the ECMWF model using 2 processors. From Dent (1988).

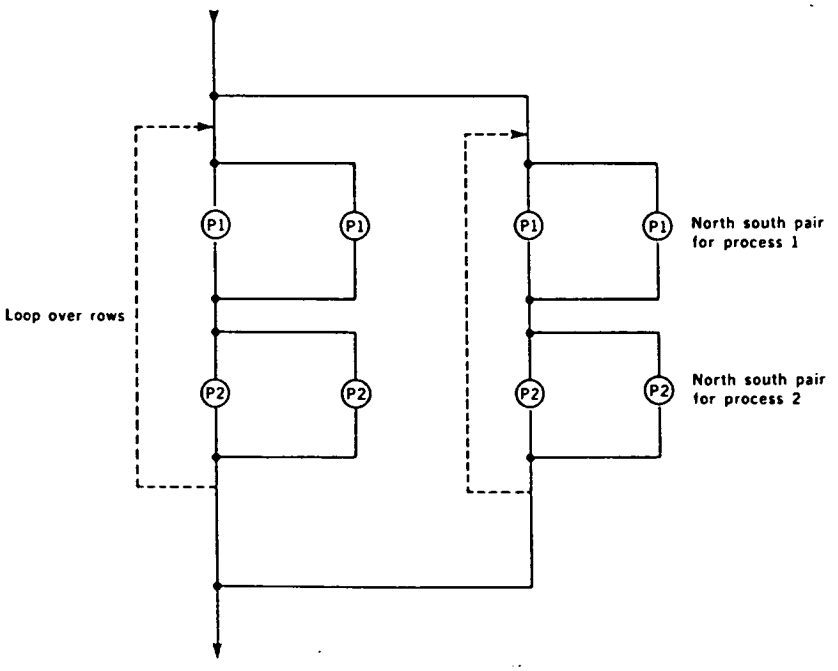


Figure 44. Schematic illustration of the multitasking strategy for scan 1 of the ECMWF model, using 4 processors. From Dent (1988).

The graph in Fig. 45 shows a typical variation in the time for process 1 and clearly shows the effect of the decreasing occurrence of convection moving from the tropics to polar regions. The use of EVENTs to ensure reproducibility and other LOCKs in the code add to the imbalance between the workload of each processor. Fig. 46 shows how the execution time is broken down. The out-of-balance percentage is due to the variation in time resulting from the convection parametrization. The multitasking overheads (MT overheads) are due to the cost of creation and synchronization of processes and the use of LOCKs.

8.5.2.2. Enhanced approach

In order to reduce the amount of wasted CPU time indicated in Fig. 46, the static scheduling strategy was enhanced. Using the extra memory available on the CRAY X-MP/48 it was possible to implement a different multitasking strategy which retained the pairing of north and south rows but computed them sequentially (north followed by south) instead of in parallel as before. In this scheme, each process does the same operations but on different latitude pairs, which allows the model to run on a computer with N processors rather than $2N$ processors.

Twice as much memory is required in this scheme to store the Fourier coefficients, since every process now reads a pair of Fourier latitudes. The main advantage however, is that the delays caused by synchronizing process 1 in every latitude loop pass have been removed. Also, as no processes are created during the latitude loop the overhead is reduced.

The only synchronization point now required is after the processes have completed all their latitudes. Imbalances from convection still arise however. Scan 2 remains unchanged. This enhanced scheme was implemented by David Snelling as a model option.

8.5.3. Dynamic scheduling

In order to reduce the imbalance in the workload between the processes, a dynamic scheduling approach was implemented, in which the latitude row pairs are allocated to the processes at run time. This technique uses a counter which each process accesses and increments to determine which row pair to use next. Processes working on polar latitudes will access this counter more

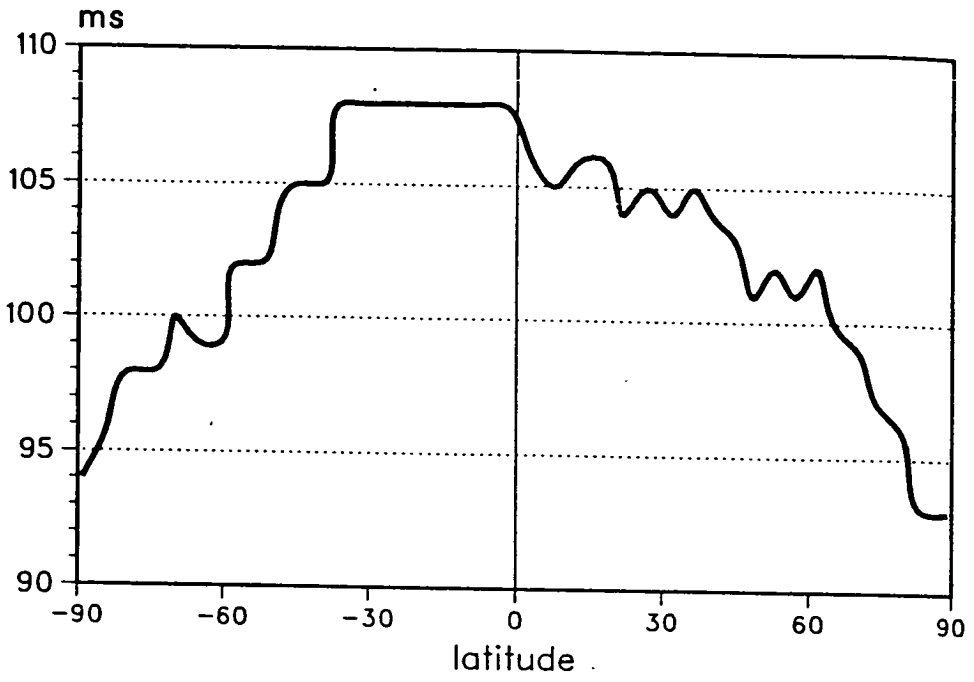


Figure 45. Graph showing the measured time of process 1 against latitude for a timestep from a forecast model run. From Dent (1988).

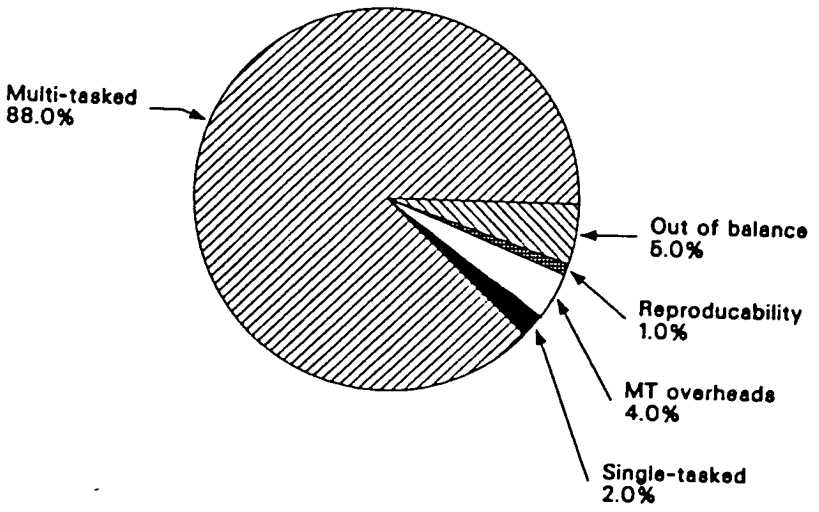


Figure 46. The multitasking efficiency of the ECMWF model using the static multitasking scheme. From Dent (1988).

often for work. For this scheme to be effective, the variation in the grain size must be significant compared to the grain size itself. Although an additional LOCK is required to protect the use of the latitude counter, the time spent in this section of code is negligible. Whilst the updating of the counter must obviously be protected from simultaneous execution by processes, reading the counter is also a critical region of code, otherwise several processes may begin using the same latitude. Identifying critical regions of code which do not involve writing to shared data can be difficult and time consuming.

The dynamic scheduling approach was implemented, as a model option, to both the original multitasking strategy (north and south rows parallel) and the enhanced strategy (north and south rows sequential). All the work was done by this author with assistance on certain details by Messrs. Dent and Snelling. Changes to the control flow of scans 1 and 2 (ignoring the I/O aspect of the model for now) were straightforward and mainly involved adding code to implement the latitude counter. The dynamic scheduling strategy is most effective when used with the north-south sequential multitasking scheme. The combined scheme became known as the DSC scheme (for dynamic scheduling). Fig. 47 illustrates the control flow for scan 1 using the DSC scheme and should be compared to Fig. 44.

The inefficiencies introduced by enforcing reproducibility become larger with the north-south sequential approach. This is because, as each processor now updates both symmetric and antisymmetric parts of the spectral data, only one processor at a time can be executing the direct Legendre transform compared to two in the original scheme. The wasted CPU time can be minimized by dividing the spectral array into sections so that waiting processes can commence as soon as the active process finishes updating each portion. More recently, Dent and O'Neill (1988) have successfully used microtasking in the direct Legendre transform to speed the work at loop level for one process over the idle processors.

The performance of the DSC model is presented by Dent (1988) and reproduced in Fig. 48. The out-of-balance overhead has been reduced from 5% to 1% and the multitasking overhead to 3%. This produces an increase in the portion of time spent multitasking to 92%. The multitasking speedup (defined as the ratio of the wall-clock time on one processor to the time on 4 processors) of the original static scheduling strategy was 3.6. With the DSC

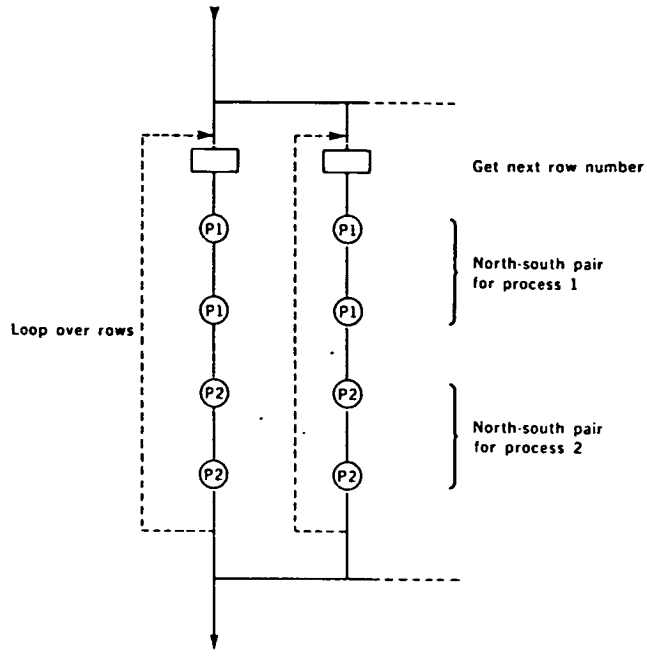


Figure 47. Dynamic multitasking strategy for scan1 using the north-south sequential approach. From Dent (1988).

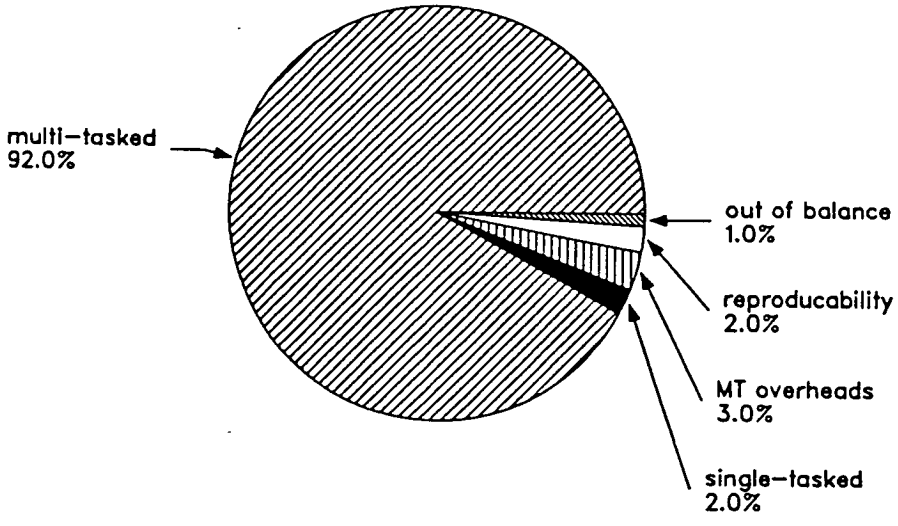


Figure 48. Multitasking efficiency for the dynamic scheduling north-south sequential (DSC) multitasking strategy. From Dent (1988).

strategy, the speedup increased to 3.7.

In retrospect, the imbalances caused by convection could be reduced by a careful ordering of latitudes. Following analysis of process 1 times, such as shown in Fig. 45, throughout the seasons, latitudes could be assigned to processors (in a list) in such a way as to give a better balance of the workload. Whilst this method does not have the flexibility of dynamic scheduling, it would greatly simplify the changes required to the I/O structure and control routines of the model.

8.5.4. Dynamic I/O scheme

In the I/O routines for the original multitasking scheme the I/O buffers were assigned to particular rows. The routines were passed the row number to be read in and if results needed to be written, calculated the row number waiting in the buffer. However, with a dynamic scheduling scheme this is no longer possible. First, the previous row processed, waiting to be written out, may have no relationship to the row the processor is about to use. Second, the assignment of a buffer and a subset of rows to a processor is no longer valid since different processors might be given rows that would involve the use of the same buffer. It became necessary therefore to completely rewrite the I/O code of the model and design a dynamic I/O system which would allocate rows to buffers when they became free. This new I/O system was designed and implemented by the author. A brief description of the dynamic I/O system is given by Dent (1988). A more detailed description of the design and implementation of the system is given by Carver (1986).

8.5.4.1. Requirements

The new I/O system had to support the existing model options and those introduced by the new multitasking schemes. Specifically, the code had to support single and multitasking, static and dynamic scheduling, north-south parallel and sequential multitasking, single buffering (SSD workfiles) and double buffering (disk workfiles). Other existing options in the original I/O code also had to be supported. These are detailed in Carver (1986).

As the code had to be completely rewritten, at the suggestion of David Dent, the opportunity was taken to introduce new model options and take advantage of new operating system facilities. An additional requirement of the

new I/O system was for latitudes to be retained in memory until all processors indicated they were no longer needed. This was to allow parametrization schemes to be developed that could access gridpoints on neighbouring latitudes, something not possible with the original code.

At the time of the design of the new system, a new version of the CRAY operating system (COS 1.15) was due for release which would allow asynchronous queued I/O (ASIO). This would achieve savings by queuing transfer requests and avoid operating system overheads for each individual request. ASIO is particularly attractive for random I/O requests to SSD when several requests may be outstanding at any one time. A further design aim was therefore to make best use of this facility, assuming its existence. The results presented in Fig. 48 had to simulate the ASIO facility using existing I/O routines as the new operating system was not available at that time.

8.5.4.2. Design and implementation

To allow all the options described above to be supported, a sophisticated and flexible design was required. The dynamic allocation of buffers to rows implied that a method of flagging buffers no longer needed had to be developed. This was done by providing each buffer with a counter for each process, that was incremented by that process when it finished with its row. Free buffers were detected by summing these counters and comparing the result with the known total number of times each latitude was required. Each process had a separate counter to avoid the need for a LOCK.

To use the ASIO facility as efficiently as possible the I/O queue had to be prevented from emptying. Given that finished buffers could be detected, it was decided to design the I/O system to 'lookahead' and start any read transfers that had yet to be requested by processes. A set of rules was therefore developed to decide the next row to be assigned to a free buffer. These rules had to support the functional requirements described in the previous section and are described in detail by Carver (1986).

Although the row required by a process was passed to the I/O routines, it was realized that the lookahead rules developed also applied to this row so that all free buffers could be treated using the same logic, simplifying the code. It was impossible to tell the order in which the rows for the free

buffers would be requested by other processes, but the transfer request for the calling process was always initiated first.

As the master I/O routines were called from within the multitasked code, a LOCK was used to ensure only one process at a time executed these routines. However, this did not cause much inefficiency since the executing process detected the free buffers of the waiting processes and queued their transfer requests. The processes delayed by the LOCK therefore normally found no work to do when it was released, except wait for completion of their I/O which was done outside the LOCK.

As the lookahead rules anticipated the next rows required by the processes, double buffering was achieved simply by doubling the number of available buffers (although the lookahead rules change if the row pairs are processed sequentially). The provision of additional latitude rows was achieved in the same way, although the number of buffers required varied. The minimum number of gridpoint buffers applied when processors were working on consecutive rows. The additional buffers are required at the ends of the ranges of northern and southern rows. On the other hand, if the latitudes the processes were using were well separated, each process required additional buffers for the latitudes north and south of its row. This is the maximum number of buffers required.

When dynamic scheduling is used and a number of buffers less than the maximum is provided, it is possible that, if the processes separate sufficiently, one or more rows required by the calling process are not yet available. In this situation it must wait for the other processes to catch up sufficiently, so that they will free buffers and start the I/O to read the rows required by the delayed process into these buffers. There is thus a trade-off between the number of buffers that can be provided and the efficiency of the dynamic scheduling. If the minimum number is provided and the I/O single buffered, the dynamic scheduling will be constrained to operate in a static scheduling manner. If double buffering is used, the extra buffers will be used to satisfy requests for additional rows instead, so that true double buffering will fail, although this will just introduce delays in waiting for some I/O requests to complete.

The dynamic I/O system is therefore very flexible. The I/O system also

functions correctly when the number of buffers provided is not an integer multiple of the number of processes, so that as much of the available memory as possible can be used.

8.6. The Meteorological Office MIMD finite difference model

The Meteorological Office recently purchased a 4 processor ETA-10 to replace their single processor CYBER-205. At the time of writing, plans to adapt the Meteorological Office finite difference forecast model for multiprocessing were well advanced (Mozdzynski, 1988; Dickinson, personal communication). These plans are briefly described here to contrast the multitasking strategy of the spectral model at ECMWF.

The basic multitasking approach is to divide the model latitudes into 4 sections. Each section overlaps adjoining sections by several latitudes. This has to be done because 3 adjustment steps for the gravity wave terms are made each timestep (Gadd, 1985), resulting in invalid values for the boundary latitudes. The correct boundary values are then read at the start of the new timestep from the appropriate adjoining process. In this static scheduling approach, a barrier is used to synchronize all processors to ensure they have all written their edge data to the shared memory. Thus the data transfers for each process are a read and write of the boundary latitudes, compared to the ECMWF spectral model in which the entire spectral data is transferred.

Load imbalances due to convection still arise. However, the Fourier filtering that takes place on polar latitudes acts to even the workload. It is also straightforward to alter the number of latitudes in each section to provide a better balance of the workload based on an analysis of the average time spent processing each latitude.

The I/O in this static scheduled scheme is synchronous. That is, a processor writes its boundary data after it has finished processing its latitudes. Also computation cannot begin until boundary data have been read in. This overhead can be avoided by using a dynamic multitasking scheme (Mozdzynski, 1988). The dynamic scheme works by partitioning the latitudes into smaller segments so that each processor will process several segments. The number of segments each processor is given depends on the length of time spent processing the segments. The I/O overhead is minimized by

allowing the data transfers to be asynchronous. Whilst each processor is computing a segment, the next segment for processing can be read in at the same time as the previous segment is written out. There is clearly much more data transferred using this approach as all the latitudes are read once and written once to shared memory per timestep. However, by assigning groups of segments to processors (static scheduling), the total amount of data transferred reduces to boundary data only, as before, but the I/O remains asynchronous.

CHAPTER 9

CONCLUSIONS

This thesis has presented a study of the application of meteorological modelling to the ICL DAP and other parallel computers. Chapter 2 reviewed some current and future parallel computers, including the DAP. Parallel programming languages and the facilities they offer were also described. Software tools to assist in the development of models on parallel computers will be invaluable since most meteorological models are still written in serial FORTRAN. FORTRAN 8X will undoubtedly be a step in the right direction with its support for SIMD parallelism, making SIMD architectures such as the DAP potentially available to more users. Chapter 2 also reviewed the main issues in programming parallel computers and the differences between SIMD and MIMD programming, illustrating the added complexities of the latter.

Chapter 3 reviewed the meteorological modelling techniques that were applied to the DAP in later chapters. The gridpoint, spectral and finite element methods were reviewed and shown to contrast in the issues that each presents for implementation on the DAP.

In chapter 4, related research work by other authors was reviewed. They concentrated mainly on gridpoint models (e.g. Hunt, 1974b), although Fishbourne (1980) also studied the implementation of a spectral model on the DAP. For global gridpoint models, the choice of grid and how it is mapped to the DAP is very important. Hunt (1974b) applied the Meteorological Office's global forecast model at that time to the DAP. The model grid had a varying number of longitude points on each latitude. Mapping the grid to the PEs was not straightforward, the finite difference calculations and hence the DAP FORTRAN were more complicated than with the latitude-longitude grid in the gridpoint model of Fishbourne (1980). However, Fishbourne (1980) found that the Fourier filtering required at polar latitudes for a latitude-longitude grid was costly. The studies described in chapter 4 all showed the gridpoint method to be well suited to the DAP. The local nature of the finite difference calculations resulted in models that used the DAP array efficiently and with low routing overheads.

In contrast, the T42 spectral model of Fishbourne (1980), also described in chapter 4, was seen to be inefficient on the DAP. The CPU time was 10 times

that of an equivalent gridpoint model. As the Legendre transforms accounted for 83% of the execution time, substantial improvements in the CPU time of this model required significantly improved Legendre transform algorithms.

In chapter 5, new parallel Legendre transform algorithms for the DAP were developed. Different storage arrangements of the Legendre polynomials and their derivatives were considered and it was found that the best algorithms resulted from data mappings that made the most efficient use of processor memory, but did not involve packing the data.

Legendre transform algorithms that did not use the symmetry property of the Legendre polynomials were developed first. Compared to the algorithms used by Fishbourne (1980), the inverse Legendre transform was faster by a factor of 1.9, the direct transform was faster by a factor of 3.3. The inverse transform has a number of inefficiencies related to the triangular nature of the spectral truncation. It was pointed out that improvements can be made by programming the algorithm in the DAP assembly language to take advantage of the idle processors. In comparison, the direct transform made efficient use of the DAP array by computing the two products in the transform concurrently. This was possible because of the efficient way in which the Legendre polynomials and their derivatives were stored in the same array.

The algorithms and data mappings were then altered to enable the use of the symmetry property of the Legendre polynomials. This resulted in a speedup of 1.6 for the inverse transform and 1.4 for the direct transform over the original nonsymmetric algorithms, as well as halving the storage requirements of the Legendre data. Again, programming the symmetric inverse transform in the DAP assembly language would improve the algorithm.

The relationships between the spectral and Legendre data mappings and the efficiency of the algorithms were pointed out. That is, at each stage of the algorithm, certain restrictions on the data mapping applied if that stage was to be efficient. The mappings of the real spectral coefficients and the Legendre data were shown to be important to the execution time of the routines, whereas the imaginary coefficients' mapping had a negligible effect. The development of these algorithms illustrates the strong relationship between the storage mapping and the efficiency of the algorithm for the DAP.

Chapter 6 described a T42 spectral shallow-water model implemented on the DAP, using the Legendre transform algorithms developed in chapter 5. The CPU time per step of this model was a factor of 2.3 less than that of the T42 spectral shallow-water model of Fishbourne (1980). This was mainly due to the superior Legendre transform algorithms but the FFTs of the model were also faster. This makes the spectral method appear to be more suited to the DAP than Fishbourne's results implied.

The lack of a fast I/O facility and the limited amount of processor memory were put forward as the main reasons for the DAPs at Edinburgh University being generally unsuitable for multi-level and high resolution spectral models, even though the results of chapter 6 showed that the DAP is potentially suited to high resolution models. The latest generation DAP from AMT has both these requirements.

In chapter 7, a finite element model was implemented on the DAP to study the suitability of this technique, in contrast with the gridpoint and spectral methods. The finite element model was shown to be efficient in its use of processors and storage, more so than the spectral model. On average, the finite element model kept 97% of the processors busy doing useful work compared to 64% for the spectral model. Therefore its performance rate of 13.1Mflops was more than the 8.7Mflops for the spectral model. The gridpoint shallow-water models of Fishbourne (1980) had a performance rate of 10Mflops.

Parallel algorithms to construct the matrices arising from the finite element approximations were derived. The most efficient method of solving the system of simultaneous equations was also studied. The hybrid Jacobi iterative routine in the DAP subroutine library was found to give the best performance, although a preconditioned conjugate gradient algorithm was developed that took almost the same CPU time. The finite element model was designed so that tridiagonal matrix equations resulted. This was important as it allowed the use of existing, efficient algorithms to solve them. Based on the efficient use of resources, it was concluded that the finite element method is better suited to the DAP than the spectral method, although each is applied to different types of meteorological modelling problems.

A serious problem with DAP FORTRAN is that it is impossible to program a

model which can be run at different resolutions, because of the constraint on the array size. As AMT have recently announced their intention to support FORTRAN 8X (chapter 2) this objection to the use of the DAP for meteorological modelling is removed. However, the size of the DAP processor array will still influence the possible efficient resolutions as discussed in chapters 6 and 7.

The DAP exploited the geometric parallelism in the models. That is, each processor represented a gridpoint, spectral coefficient or finite element node. The size of the processor array therefore had a strong influence on the domain size of the models implemented on the DAP. However, the parallelism available in meteorological models in a geometric sense is generally more than can be exploited by the hardware, especially for multi-level models. There is also further SIMD parallelism available from identical calculations between variables. This leads to the question of whether meteorological modelling is better suited to SIMD or MIMD architectures, as much SIMD parallelism is available in the models and MIMD programming is more complex. Chapter 8 reviewed the application of meteorological models to other parallel computers and discussed why the parametrization of physical processes in multi-level models requires MIMD architectures for optimum efficiency. However, the work undertaken with the ECMWF spectral model illustrates the degree of sophistication required in programming MIMD computers to obtain this efficiency. The effect of conditional operations in the parametrizations on the performance of a model on the DAP depends on the fraction of time spent processing with a reduced number of processors, as discussed in chapter 2.

It is easier to conclude the architectures not suited to meteorological modelling rather than those that are. Communication and synchronization requirements of the models lead to the conclusion that their performance on distributed memory SIMD or MIMD computers, whose processing speed is unmatched by interprocessor communication speed, is poor relative to the peak performance of the computer. The same might be true for hierarchical memory machines where communication takes place via a shared memory. On the DAP, arithmetic is slower than the transfer of data between processors. This is not to say that modelling cannot be done on these machines, but for a time critical program such as a forecast model, a machine with a

proportionally greater processing speed (or number of processors) would be required. A heterogeneous architecture consisting of SIMD processors and MIMD processors might be more suitable. The dynamics calculations in the models would be done on the SIMD part whilst the physics would be done on the MIMD part. As far as the DAP is concerned, for the latest generation from AMT and taking into account future compilers, the degree of SIMD parallelism available in meteorological models means that DAP is a suitable computer architecture. This thesis has shown that the gridpoint and finite element methods are both able to make efficient use of the DAP. Whilst the spectral method is less suited by comparison, its performance is not such as to dismiss the DAP architecture, bearing in mind that spectral models warrant detailed study when implemented on other parallel computers, as described in chapter 8.

There are several ways in which this research could continue. First, the inverse Legendre transform should be written in the DAP assembly language to determine the performance benefits. The next stage would be the development of a multi-level spectral model. This would allow the effect of the conditional operations in the physical parametrizations to be properly assessed. The additional levels would also allow the study of whether the additional parallelism could be exploited by the Legendre transform algorithms in the spectral model. Another research area would be the use of block floating point or reduced precision arithmetic in gridpoint, spectral and finite element meteorological models. Finally, in chapter 8 it was suggested that the spectral transforms (FFTs and Legendre transforms) could be used as a benchmark for spectral models. For a finite element model, the solution of tridiagonal simultaneous equations would perhaps be a suitable benchmark. It would be worthwhile implementing these benchmarks on different parallel computers to enable further conclusions to be made about the suitability of different parallel architectures to meteorological modelling.

Acknowledgements

First and foremost, I would like to express my thanks to my supervisor, Dr. Charles Duncan, for his constant encouragement, enthusiasm and constructive advice of this work and for proofreading this thesis. I would also like to thank the other members of the Department of Meteorology at Edinburgh University for making my stay there so enjoyable.

Thanks are also due to Charles Duncan, Edinburgh University, SERC and ECMWF for allowing me to spend 3 months working at ECMWF. At ECMWF I would like to thank David Dent and David Snelling (now at Leicester University) for their help and useful conversations.

Financial support from SERC and GEC is acknowledged.

Last, but by no means least, I would like to express my deepest thanks to my wife for her constant support and encouragement whilst this thesis was being prepared. I would also like to thank her for proofreading this thesis and her assistance in typing the text and preparing some of the figures and tables.

Appendix A. Code examples from the spectral model

Appendix A.I. Inverse Legendre transform algorithms

This section presents the DAP FORTRAN used for estimating the execution times of the inverse Legendre transform algorithms, derived from the Legendre polynomial mappings. The generic function ROUTE in the code below represents any of the suitable routing operations discussed in chapter 5. All variables are assumed to be set to zero before use.

Latitude vertical:

```
      ZREAL(REAL_MASK) = FMN(,)
      ZIMAG(REAL_MASK) = ROUTE( FMN(,) )
      DO 100 J = 1, 64
         FMR(,J) = SUMC( ZREAL * PMN(,,J) )
         FMI(,J) = SUMC( ZIMAG * PMN(,,J) )
100   CONTINUE
```

m vertical:

```
      ZREAL(REAL_MASK) = FMN(,)
      ZIMAG(REAL_MASK) = ROUTE( FMN(,) )
      DO 100 M = 1, 43
         FMR(,M) = SUMC( MATR( ZREAL(,M) ) * PMN(,,M) )
         FMI(,M) = SUMC( MATR( ZIMAG(,M) ) * PMN(,,M) )
100   CONTINUE
```

n vertical:

```
      ZREAL(REAL_MASK) = FMN(,)
      ZIMAG(REAL_MASK) = ROUTE( FMN(,) )
      DO 100 N = 1, 43
         FMR(,) = FMR(,) + MATC( ZREAL(,N) ) * PMN(,,N)
         FMI(,) = FMI(,) + MATC( ZIMAG(,N) ) * PMN(,,N)
100   CONTINUE
```

Appendix A.II. Direct Legendre transform algorithms

This section presents the DAP FORTRAN used for estimating the execution times of the direct Legendre transform algorithms derived in chapter 5.

Latitude vertical:

```
      AMR(,) = AMR(,) * MATR( HJ() )
      AMI(,) = AMI(,) * MATR( HJ() )
      BMR(,) = BMR(,) * MATR( HJ() )
      BMI(,) = BMI(,) * MATR( HJ() )
      DO 100 J = 1, 64
         F(PMN_MASK) = MATC( AMI(,J) )
         F(DQ_MASK)  = ROUTE( MATC( BMR(,J) ) )
         FMNR(,)     = FMNR(,) + F(,) * PMN(,,J)
         F(PMN_MASK) = MATC( AMR(,,J) )
         F(DQ_MASK)  = ROUTE( MATC( BMI(,J) ) )
         FMNI(,)     = FMNI(,) + F(,) * PMN(,,J)
  100 CONTINUE
      FMN(REAL_MASK) = ROUTE( FMNR ) + FMNR
      FMN(IMAG_MASK) = ROUTE( FMNI ) - FMNI
```

m vertical:

```
      MP1 = 44
      AMR(,) = AMR(,) * MATR( HJ() )
      AMI(,) = AMI(,) * MATR( HJ() )
      BMR(,) = BMR(,) * MATR( HJ() )
      BMI(,) = BMI(,) * MATR( HJ() )
      DO 100 M = 1, 43
         F(PMN_MASK) = MATR( AMI(M,) )
         F(DQ_MASK)  = ROUTE( MATR( BMR(MP1-M,) ) )
         FMNR(M,)    = SUMC( F(,) * PMN(,,M) )
         F(PMN_MASK) = MATR( AMR(M,) )
         F(DQ_MASK)  = ROUTE( MATR( BMI(MP1-M,) ) )
         FMNI(M,)    = SUMC( F(,) * PMN(,,M) )
  100 CONTINUE
      FMN(REAL_MASK) = ROUTE( FMNR ) + FMNR
      FMN(IMAG_MASK) = ROUTE( FMNI ) - FMNI
```

n vertical:

```
      AMR(,) = AMR(,) * MATR( HJ() )
      AMI(,) = AMI(,) * MATR( HJ() )
      BMR(,) = BMR(,) * MATR( HJ() )
      BMI(,) = BMI(,) * MATR( HJ() )
```

```
      FR(,)  = AMI(,)
      FR(DL_MASK) = ROUTE( BMR(,) )
      FI(,)  = AMR(,)
      FI(DL_MASK) = ROUTE( BMI(,) )
      DO 100 N = 1, 43
         FMNR(,N) = SUMC( FR(,) * PMN(,,N) )
         FMNI(,N) = SUMC( FI(,) * PMN(,,N) )
         IF ( N.NE.21 ) GO TO 5
            FR(DL1_MASK) = AMI(,)
            FI(DL1_MASK) = AMR(,)
5        IF ( N.NE.42 ) GO TO 100
            FR(43,) = AMI(43,)
            FI(43,) = AMR(43,)
100   CONTINUE
      FMN(REAL_MASK) = ROUTE( FMNR(,) ) + FMNR(,)
      FMN(IMAG_MASK) = ROUTE( FMNI(,) ) - FMNI(,)
```

Appendix A.III. Symmetric Legendre transform algorithms

This section presents the DAP FORTRAN for the inverse and direct Legendre transforms using the symmetry property of the Legendre polynomials. This was used for estimating and measuring the execution time of the algorithms.

Inverse Legendre transform:

```
      AMWORK1(REAL_MASK) = AMFMN(,)
      AMWORK2(REAL_MASK) = REVC( REVR( AMFMN(,) ) )
      DO 100 N = 1, 23
         N2 = 2 * N
         N2P1 = N2 + 1
         AM(,)           = MATC( AMWORK1(,N2) )
         AM( ALTC(32) ) = MATC( AMWORK1(,N2P1) )
         AMFM(,,1)       = AMFM(,,1) + AM(,) * PMN(,,N)
         AM(,)           = MATC( AMWORK2(,N2) )
         AM( ALTC(32) ) = MATC( AMWORK2(,N2P1) )
         AMFM(,,2)       = AMFM(,,2) + AM(,) * PMN(,,N)
100   CONTINUE
      AM(,)           = REVC( AMFM(,,1) )
      AM( ALTC(32) ) = -AM(,)
      AMFM(,,1)       = AMFM(,,1) + AM(,)
      AMFM( ALTC(32).AND..NOT.ALTR(1),1 ) = -AMFM(,,1)
      AM(,)           = REVC( AMFM(,,2) )
      AM( ALTC(32) ) = -AM(,)
      AMFM(,,2)       = AMFM(,,2) + AM(,)
      AMFM( ALTC(32).AND..NOT.ALTR(1),2 ) = -AMFM(,,2)
```

Direct Legendre transform:

```
      AM(,)              = REVC( AMBM(,,1) )
      AMBM( LMEAST,1 )   = -AMBM(,)
      AM( LMWEST )       = -AM(,)
      AMBM(,,1)          = AM(,) + AMBM(,,1)
      AM(,)              = REVC( AMBM(,,2) )
      AMBM( LMEAST,2 )   = -AMBM(,,2)
      AM( LMWEST )       = -AM(,)
      AMBM(,,2)          = AM(,) + AMBM(,,2)
      AM(,)              = REVC( AMAM(,,1) )
      AMAM( LMEAST,1 )   = -AMAM(,,1)
      AM( LMWEST )       = -AM(,)
      AMAM(,,1)          = AM(,) + AMAM(,,1)
      AM(,)              = REVC( AMAM(,,2) )
      AMAM( LMEAST,2 )   = -AMAM(,,2)
      AM( LMWEST )       = -AM(,)
      AMAM(,,2)          = AM(,) + AMAM(,,2)
      AM(,,1)            = REVR( AMAM(,,1) )
      AM(,,2)            = REVR( AMAM(,,2) )
      AMFMR(,)           = AMAM(,,1)
      AMFMI(,)           = AMAM(,,2)
      AMFMR( LMDL21 )    = AMBM(,,2)
      AMFMI( LMDL21 )    = AMBM(,,1)
      DO 100 N = 1, 10
         N2 = 2 * N
         N2P1 = N2 + 1
         AM(,) = AMFMR(,) * PMN(,,N)
         CALL SUM2C( AM, AMWR(,N2), AMWR(,N2P1) )
         AM(,) = AMFMI(,) * PMN(,,N)
         CALL SUM2C( AM, AMWI(,N2), AMWI(,N2P1) )
  100 CONTINUE
      AMFMR( LMDL41 ) = AMBM(,,2)
      AMFMI( LMDL41 ) = AMBM(,,1)
      DO 200 N = 11, 20
         N2 = 2 * N
         N2P1 = N2 + 1
         AM(,) = AMFMR(,) * PMN(,,N)
         CALL SUM2C( AM, AMWR(,N2), AMWR(,N2P1) )
         AM(,) = AMFMI(,) * PMN(,,N)
         CALL SUM2C( AM, AMWI(,N2), AMWI(,N2P1) )
  200 CONTINUE
      AMFMR(42,) = AMBM(42,,2)
      AMFMR(43,) = AMBM(43,,2)
      AMFMI(42,) = AMBM(42,,1)
      AMFMI(43,) = AMBM(43,,1)
      DO 300 N = 21, 22
         N2 = 2 * N
         N2P1 = N2 + 1
         AM(,) = AMFMR(,) * PMN(,,N)
         CALL SUM2C( AM, AMWR(,N2), AMWR(,N2P1) )
         AM(,) = AMFMI(,) * PMN(,,N)
         CALL SUM2C( AM, AMWI(,N2), AMWI(,N2P1) )
  300 CONTINUE
```

```
AMFMN( REAL_MASK ) = AMWR(,) + REVC( REVR( SHEP( AMWR,19 ) ) )
AMFMN( IMAG_MASK ) = SHEP( AMWI,19 ) - REVC( REVR( AMWI ) )
```

Appendix B. Accuracy of the finite element scheme for the diffusion term

This appendix presents the proof of the accuracy of the finite element scheme used for the diffusion term in the finite element model of chapter 7, on a grid with constant spacing in the x and z directions. The method of Cullen (1976) is used.

Consider the equation,

$$w = Lu \tag{1}$$

where L is the differential operator acting on u. Define a projection, p, such that pu and pw are the representations of u and w on the finite element grid i.e. the numbers held in the computer. If the finite element form of L is written as $\tilde{L}$ then, as shown by Cullen (1976), the spatial error E is given by,

$$E = (\tilde{L}p - pL)u \tag{2}$$

This is the difference between the finite element solution and the analytical solution Eq.(1) projected onto the finite element grid.

To simplify the analysis, consider the one-dimensional case only on a uniform grid,

$$f = \frac{\partial}{\partial y}(\kappa_e \frac{\partial u}{\partial y}) \tag{3}$$

where u and κ_e are assumed to be of the form,

$$u = e^{i\theta y} \qquad \kappa_e = e^{i\psi y} \tag{4}$$

and y is a normalized coordinate with $\theta = k\Delta x$, $\psi = l\Delta x$ with $\Delta x = 1$.

The projection of u and κ_e onto the finite element grid is given by,

$$pu = \sum_i \alpha(\theta)\exp(i\theta y_i)\phi_i$$

$$p\kappa_e = \sum_i \alpha(\psi)\exp(i\psi y_i)\phi_i \tag{5}$$

with ϕ_i as the piecewise linear basis functions. The projection factor α, is given by Eq.(3.5.24). The analytical solution to Eq.(3) using Eq.(4) is given by,

$$f = -\theta(\theta+\psi)\exp[i(\theta+\psi)y] \tag{6}$$

The computer representation of this is,

$$pf = -\sum_i \theta(\theta+\psi)\alpha(\theta+\psi)\exp[i(\theta+\psi)y_i]\phi_i \tag{7}$$

The finite element scheme for Eq.(3) follows from the two-dimensional case and is,

$$(1/3)(f_{i-1} + 4f_i + f_{i+1}) = (\kappa_{i+1} + \kappa_i)(u_{i+1} - u_i) - (\kappa_i + \kappa_{i-1})(u_i - u_{i-1}) \tag{8}$$

If the numerical solution of f is written as,

$$\tilde{f} \equiv \tilde{L}pu = \sum_i \beta(\theta,\psi)\alpha(\theta)\alpha(\psi)\exp[i(\theta+\psi)y_i]\phi_i \tag{9}$$

and Eq.(5) is substituted into Eq.(8) to solve for β, then,

$$\beta(\theta,\psi) = \frac{-3[\,1 - \cos\theta + \cos\psi - \cos(\theta+\psi)]}{2 + \cos(\theta+\psi)} \tag{10}$$

From Eq.(2), the error is therefore,

$$E = \sum_i [\alpha(\theta)\alpha(\psi)\beta(\theta,\psi) + \theta(\theta+\psi)\alpha(\theta+\psi)]\,\exp[i(\theta+\psi)y_i]\,\phi_i \tag{11}$$

The error of the finite element scheme is therefore proportional to,

$$\frac{\alpha(\theta)\alpha(\psi)\beta(\theta,\psi)}{\theta(\theta+\psi)\alpha(\theta+\psi)} + 1 \tag{12}$$

at each node. Expanding this as θ and ψ tend to 0 and using the result from Cullen (1976) that,

$$\alpha(k) \simeq 1 + k^2/12 + \ldots\ldots \qquad \text{as } k \to 0$$

the leading error term is found to be $|\theta\psi|/6$. Thus, on a regular grid the scheme is second order accurate.

Appendix C. Calculation of finite element matrices

This appendix details the algorithms used to calculate the finite element matrix for the product term. The procedure for an irregular grid, semi-irregular grid and an improved algorithm following Staniforth and Beaudoin (1986) are presented.

Appendix C.I. Product term on an irregular grid

Consider the implementation of Eq.(7.7.3) on the DAP. The first stage is the calculation of the products on all the nodes. By redefining the half-integer values as,

$$\bar{u}_{k+\frac{1}{2},l} = u_{k,l} + u_{k+1,l} \tag{13}$$

$$v_{k+\frac{1}{2},l} = \tfrac{1}{2}\left(v_{k,l} + v_{k+1,l} \right)$$

and the products as,

$$\bar{p}_{k+\frac{1}{2},l} = \bar{u}_{k+\frac{1}{2},l} v_{k+\frac{1}{2},l} \tag{14}$$

then,

$$2p_{k+\frac{1}{2},l} = \bar{p}_{k+\frac{1}{2},l}$$

$$4p_{k+\frac{1}{2},l+\frac{1}{2}} = \bar{p}_{k+\frac{1}{2},l} \tag{15}$$

which can be used to remove multiplications in the calculation of the half-integer products and the multiplication by 2 and 4 in Eq.(7.7.3).

As the DAP array matches the model domain, matrix notation can be used to express the operations on the nodes. The multiplications are written explicitly to show that they are DAP FORTRAN multiplications and not matrix multiplications in the mathematical sense.

To compute the products, the half-integer values are first formed by,

$$\underline{\bar{u}}_x = \underline{u} + \mathrm{SHNP}(\underline{u})$$

$$\underline{v}_x = 0.5 * \left(\underline{v} + \mathrm{SHNP}(\underline{v}) \right)$$

$$\underline{\underline{\bar{u}}}_z = \underline{\underline{u}} + \text{SHWP}(\underline{\underline{u}})$$

$$\underline{\underline{v}}_z = 0.5 * (\underline{\underline{v}} + \text{SHWP}(\underline{\underline{v}}))$$

where the shift functions of DAP FORTRAN are written explicitly where they are required. Unused elements of the matrices are set to zero,

$$\bar{u}_x(64,i) = 0 \qquad \text{for } i = 1,...64$$

$$\bar{u}_z(i,64) = 0$$

$$v_x(64,i) = 0$$

$$v_z(i,64) = 0$$

before the remaining half-integer values are computed by,

$$\underline{\underline{\bar{u}}}_{xz} = \underline{\underline{\bar{u}}}_x + \text{SHWP}(\underline{\underline{\bar{u}}}_x)$$

$$\underline{\underline{v}}_{xz} = 0.5 * (\underline{\underline{v}}_x + \text{SHWP}(\underline{\underline{v}}_x))$$

$$\bar{u}_{xz}(i,64) = v_{xz}(i,64) = 0 \qquad \text{for } i = 1,...64$$

The products are computed by,

$$\underline{\underline{p}} = \underline{\underline{u}} * \underline{\underline{v}}$$

$$\underline{\underline{\bar{p}}}_x = \underline{\underline{\bar{u}}}_x * \underline{\underline{v}}_x$$

$$\underline{\underline{\bar{p}}}_z = \underline{\underline{\bar{u}}}_z * \underline{\underline{v}}_z$$

$$\underline{\underline{\bar{p}}}_{xz} = \underline{\underline{\bar{u}}}_{xz} * \underline{\underline{v}}_{xz}$$

The mapping expressions for these products are,

$$\bar{p}_{k,l} : \qquad (k, l) \rightarrow \{ k+1, l+1 \} \qquad \text{for } k, l = 0,....63$$

$$\bar{p}_{k+\frac{1}{2},l} : \qquad (k+\tfrac{1}{2}, l) \rightarrow \{ k+1, l+1 \}$$

$$\bar{p}_{k,l+\frac{1}{2}} : \quad (k, l+\tfrac{1}{2}) \rightarrow \{ k+1, l+1 \} \tag{16}$$

$$\bar{p}_{k+\frac{1}{2},l+\frac{1}{2}} : \quad (k+\tfrac{1}{2}, l+\tfrac{1}{2}) \rightarrow \{ k+1, l+1 \}$$

and the constant matrix, $\underline{\underline{A}}$, is set by,

$$A_{i,j} = \Delta x_i \Delta z_j \; / \; 36 \tag{17}$$

for which the mapping expression is,

$$A : \quad (k, l) \rightarrow \{ k+1, l+1 \} \tag{18}$$

Using these mapping expressions it is straightforward to write the operations for the contributions from each element to include the correct shifts. First, some intermediate results are formed. The overbars are dropped for convenience. The operations required are,

$$\begin{aligned} \underline{\underline{B}} &= \underline{\underline{p}} + \underline{\underline{p}}_x \\ \underline{\underline{C}} &= \underline{\underline{p}} + \mathrm{SHSP}(\underline{\underline{p}}_x) \\ \underline{\underline{D}} &= \underline{\underline{p}}_z + \underline{\underline{p}}_{xz} \\ \underline{\underline{E}} &= \underline{\underline{p}}_z + \mathrm{SHSP}(\underline{\underline{p}}_{xz}) \end{aligned} \tag{19}$$

so that the elemental contributions are,

$$\begin{aligned} \underline{\underline{F}}_1 &= \mathrm{SHSP}(\mathrm{SHEP}(\underline{\underline{A}})) * [\, \underline{\underline{C}} + \mathrm{SHEP}(\underline{\underline{E}}) \,] \\ \underline{\underline{F}}_2 &= \mathrm{SHEP}(\underline{\underline{A}}) * [\, \underline{\underline{B}} + \mathrm{SHEP}(\underline{\underline{D}}) \,] \\ \underline{\underline{F}}_3 &= \mathrm{SHSP}(\underline{\underline{A}}) * [\, \underline{\underline{C}} + \underline{\underline{E}} \,] \\ \underline{\underline{F}}_4 &= \underline{\underline{A}} * [\, \underline{\underline{B}} + \underline{\underline{D}} \,] \end{aligned} \tag{20}$$

The matrix for the integral is then formed by,

$$\underline{\underline{F}} = \sum_i \underline{\underline{F}}_i, \qquad i = 1, 2, 3, 4 \tag{21}$$

The total operation count is therefore eleven multiplications and seventeen

additions in matrix mode.

Appendix C.II. Product term on a semi-irregular grid

Consider the implementation of Eq.(7.7.10) on the DAP. It is a computational advantage to redefine the products, p, to be,

$$\bar{p}_{k+\frac{1}{2},l} = (u_{k,l} + u_{k+1,l})(v_{k,l} + v_{k+1,l})$$

$$\bar{p}_{k+\frac{1}{2},l+\frac{1}{2}} = (u_{k,l} + u_{k+1,l} + u_{k,l+1} + u_{k+1,l+1})(v_{k,l} + v_{k+1,l} + v_{k,l+1} + v_{k+1,l+1}) \quad (22)$$

similar to the irregular grid, so that,

$$\bar{p}_{k+\frac{1}{2},l} = 4p_{k+\frac{1}{2},l} \quad (23)$$

$$\bar{p}_{k+\frac{1}{2},l+\frac{1}{2}} = 16p_{k+\frac{1}{2},l+\frac{1}{2}}$$

where the products, p, are given by Eq.(7.7.4). Substituting into Eq.(7.7.10) then gives,

$$\begin{aligned} F_{k,l} = (\Delta x \Delta z_{l-1}/36)[\ & \gamma_k \bar{p}_{k,l} + \tfrac{1}{2}\gamma_k \bar{p}_{k,l-\frac{1}{2}} + \tfrac{1}{2}\bar{p}_{k-\frac{1}{2},l} \\ & + \tfrac{1}{2}\bar{p}_{k+\frac{1}{2},l} + \tfrac{1}{4}\bar{p}_{k-\frac{1}{2},l-\frac{1}{2}} + \tfrac{1}{4}\bar{p}_{k+\frac{1}{2},l-\frac{1}{2}}] \\ + (\Delta x \Delta z_l/36)[\ & \gamma_k \bar{p}_{k,l} + \tfrac{1}{2}\gamma_k \bar{p}_{k,l+\frac{1}{2}} + \tfrac{1}{2}\bar{p}_{k-\frac{1}{2},l} \\ & + \tfrac{1}{2}\bar{p}_{k+\frac{1}{2},l} + \tfrac{1}{4}\bar{p}_{k-\frac{1}{2},l+\frac{1}{2}} + \tfrac{1}{4}\bar{p}_{k+\frac{1}{2},l+\frac{1}{2}}] \end{aligned} \quad (24)$$

The first stage in calculating Eq.(24) is again the calculation of the products, except that there are no multiplications by $\frac{1}{2}$ necessary, saving 3 matrix multiplies. The next stage is to evaluate the contributions from each grid element. However, since there is a regular spacing in the x direction, the contribution from the two elements at the same level can be evaluated together. That is, first compute,

$$\underline{\underline{B}} = \text{MATC}(\underline{\underline{\gamma}}) * \underline{\underline{\bar{p}}} + 0.5 * (\underline{\underline{\bar{p}}}_x + \text{SHSP}(\underline{\underline{\bar{p}}}_x)) \quad (25)$$

$$\underline{\underline{C}} = \text{MATC}(\underline{\underline{\gamma}}') * \underline{\underline{\bar{p}}}_z + 0.25 * (\underline{\underline{\bar{p}}}_{xz} + \text{SHSP}(\underline{\underline{\bar{p}}}_{xz}))$$

where the vector $\underline{Y}'$ is defined by,

$$\underline{Y}' = \tfrac{1}{2}\underline{Y} \tag{26}$$

Then evaluate the matrix by,

$$\underline{\underline{F}} = \text{SHEP}(\underline{\underline{A}}) * (\underline{\underline{B}} + \text{SHEP}(\underline{\underline{C}})) + \underline{\underline{A}} * (\underline{\underline{B}} + \underline{\underline{C}}) \tag{27}$$

The total operation count is now 10 multiplications and 13 additions in matrix mode.

The above procedure can be optimized further as the vector $\underline{Y}'$ is not equal to 1 only at k=0 and 63. Thus, the factor of 0.5 at these points can be performed by two multiplications in vector mode, replacing the matrix multiply and the MATC call. The saving in time is small at 0.18msecs but there is also no need to store the vector $\underline{Y}'$.

Nearly all of the DAP array is used during these calculations. The calculation of the half-integer nodes uses all the array. The calculation of the products uses all but 1 row or column. For the products at the nodes at the mid-point of each rectangle, 1 row and column are unused. As the resulting matrix $\underline{\underline{F}}$ in Eq.(7.7.10) has an entry for each node, all the processors are performing useful work for Eq.(25) and Eq.(27).

Appendix C.III. Further improvements

Following Staniforth and Beaudoin (1986), the DAP algorithm for an irregular grid is as follows. First compute the half-integer sums,

$$\begin{aligned}
\underline{\underline{\bar{u}}}_x &= \underline{\underline{u}} + \text{SHNP}(\underline{\underline{u}}) \\
\underline{\underline{\bar{v}}}_x &= \underline{\underline{v}} + \text{SHNP}(\underline{\underline{v}}) \\
\underline{\underline{\bar{u}}}_z &= \underline{\underline{u}} + \text{SHWP}(\underline{\underline{u}}) \\
\underline{\underline{\bar{v}}}_z &= \underline{\underline{v}} + \text{SHWP}(\underline{\underline{v}}) \\
\underline{\underline{\bar{u}}}_{xz} &= \underline{\underline{\bar{u}}}_x + \text{SHWP}(\underline{\underline{\bar{u}}}_x) \\
\underline{\underline{\bar{v}}}_{xz} &= \underline{\underline{\bar{v}}}_x + \text{SHWP}(\underline{\underline{\bar{v}}}_x)
\end{aligned} \tag{28}$$

Unused elements of these matrices are set to zero as before. The next stage is the computation of the products, different from before,

$$\underline{\underline{p}} = \underline{\underline{u}} * \underline{\underline{v}} * \text{MATC}[\ (\underline{\Delta x} + \text{SHRP}(\underline{\Delta x}))/6\ ]$$

$$\underline{\underline{\bar{p}}}_z = \underline{\underline{\bar{u}}}_z * \underline{\underline{\bar{v}}}_z * \text{MATC}[\ (\underline{\Delta x} + \text{SHRP}(\underline{\Delta x}))/6\ ]$$

$$\underline{\underline{\bar{p}}}_x = \underline{\underline{\bar{u}}}_x * \underline{\underline{\bar{v}}}_x * \text{MATC}[\ \underline{\Delta x}/12\ ] \qquad (29)$$

$$\underline{\underline{\bar{p}}}_{xz} = \underline{\underline{\bar{u}}}_{xz} * \underline{\underline{\bar{v}}}_{xz} * \text{MATC}[\ \underline{\Delta x}/12\ ]$$

Then some intermediate results are calculated by,

$$\underline{\underline{A}} = \text{MATR}[\ \underline{\Delta z}/12\ ] * (\ \underline{\underline{\bar{p}}}_{xz} + \text{SHSP}(\underline{\underline{\bar{p}}}_{xz} + \underline{\underline{\bar{p}}}_z\ )\ ) \qquad (30)$$

$$\underline{\underline{B}} = \underline{\underline{p}} + \underline{\underline{\bar{p}}}_x + \text{SHSP}(\underline{\underline{\bar{p}}}_x)$$

The final matrix is given by,

$$\underline{\underline{F}} = \underline{\underline{A}} + \text{SHEP}(\underline{\underline{A}}) + \text{MATR}[\ (\underline{\Delta z} + \text{SHRP}(\underline{\Delta z})/6\ ] * \underline{\underline{B}} \qquad (31)$$

The terms in square brackets represent values that can be precomputed and stored. The operation count is now 10 multiplications and 12 additions.

For a semi-irregular grid, the required changes to the algorithm steps in Eq.(29), Eq.(30) and Eq.(31) are straightforward. New constants are defined and two multiplications are removed.

Parallel Computing 8 (1988) 121–126
North-Holland

A spectral meteorological model on the ICL DAP

Glenn CARVER

Department of Meteorology, The University, Edinburgh, United Kingdom EH9 3JZ

Abstract. The key points to the implementation of a meteorological spectral model on the ICL Distributed Array Processor (DAP) are presented. Spectral models involve transforming the variables between spectral and gridpoint space and these spectral transforms comprise of Legendre and fast Fourier transforms.

The storage format of the data is discussed and the algorithms used for the Legendre transforms presented. Timings of these algorithms are compared with those from a serial machine.

Keywords. DAP, meteorology, spectral, Legendre transforms.

1. Introduction

The ICL DAP is a SIMD machine [7] comprising a 64 by 64 array of 1-bit processing elements (PEs) and 2 Mbytes of memory. Whilst this architecture seems ideally suited to finite difference techniques [2,8], the spectral method [3,5] is preferred in Meteorology for modelling the atmosphere over the globe for its superior accuracy. In this method, the model variables are expanded as a truncated set of spherical harmonics defined by

$$Y_{m,n}(\lambda, \mu) = e^{im\lambda} P_{m,n}(\mu)$$

where m is the number of waves around a latitude circle, n is the total wavenumber, λ is the longitude and $\mu = \sin\theta$ where θ is the latitude. The $P_{m,n}$ are normalised Legendre polynomials.

The most common form of truncation is known as triangular truncation because of the shape of the retained modes in spectral or wavenumber space. Figure 1 illustrates this for a triangular truncation at wavenumber 5, denoted by T5.

The expansion coefficients of each variable are functions of time only. The model equations are written so that these expansion coefficients are integrated forward in time. However, it has been found more efficient to evaluate non-linear terms by first transforming to a collocation grid in physical space to form the products and then transforming back to spectral space. These

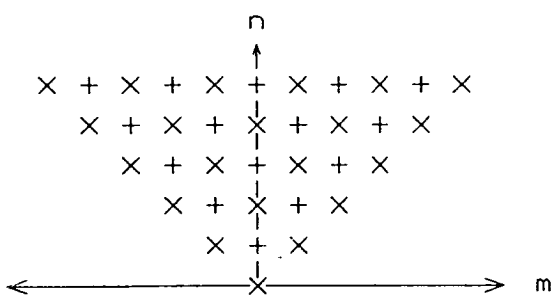


Fig. 1. Spectral coefficients for triangular truncation at wavenumber 5 (T5). Crosses denote coefficients where $|m| + n$ is even, pluses where $|m| + n$ is odd.

spectral transforms consist of Legendre and fast Fourier transforms (FFTs) and form a large part of the overall model calculations.

To study the suitability of the DAP architecture to spectral meteorological models, a T42 spectral shallow-water model [3], where the atmosphere is represented as a single layer of fluid, was implemented on the ICL DAP. For this model, the spectral transforms account for more than 90% of the computing time. The Legendre transforms alone account for 74%. Thus efficient implementation of the spectral transforms is essential if the method is to be successful on this type of architecture. The implementation of the FFT follows that in [6] and is not discussed here.

2. Legendre transforms

The Legendre transforms use the property

$$P_{m,n}(-\mu_j) = (-1)^{|m|+n} P_{m,n}(\mu_j)$$

in order to reduce the amount of computation and the storage required for the Legendre polynomials and their derivatives. The model latitudes are denoted by μ_j, where j is a positive integer.

The inverse Legendre transform involves first computing

$$\begin{aligned} A_m &= \sum_{n=|m|}^{M} F_{m,n}(t) P_{m,n}(\mu_j) \quad \text{for } |m|+n \text{ even}, \\ B_m &= \sum_{n=|m|}^{M} F_{m,n}(t) P_{m,n}(\mu_j) \quad \text{for } |m|+n \text{ odd} \end{aligned} \tag{1}$$

where $F_{m,n}$ are the spectral coefficients and M the truncation wavenumber. The symmetric (A_m) and antisymmetric (B_m) Fourier coefficients are then combined to give the final Fourier coefficients by

$$F_m(\mu_j) = A_m(\mu_j) + B_m(\mu_j), \qquad F_m(-\mu_j) = A_m(\mu_j) - B_m(\mu_j). \tag{2}$$

The symmetric and antisymmetric spectral coefficients are illustrated in Fig. 1.

The direct Legendre transform required by the model takes the general form

- $|m|+n$ even:

$$F_{m,n}(t) = \sum_{j=1}^{J/2} \left(G_m(\mu_j) - G_m(-\mu_j)\right) \frac{\mathrm{d}P_{m,n}}{\mathrm{d}\mu} + \mathrm{i}\left(H_m(\mu_j) + H_m(-\mu_j)\right) P_{m,n}, \tag{3}$$

- $|m|+n$ odd:

$$F_{m,n}(t) = \sum_{j=1}^{J/2} \left(G_m(\mu_j) + G_m(-\mu_j)\right) \frac{\mathrm{d}P_{m,n}}{\mathrm{d}\mu} + \mathrm{i}\left(H_m(\mu_j) - H_m(-\mu_j)\right) P_{m,n}$$

where J is the number of model latitudes (64 at T42 resolution) and G_m and H_m are the Fourier coefficients obtained from non-linear products evaluated on the grid in physical space. The weights associated with this Gaussian quadrature equation are assumed to have been incorporated into the Fourier coefficients. It is clearly a computational advantage to store the Legendre polynomials and their derivatives.

3. Storage of data

The different representations of a variable during the spectral transforms, spectral, Fourier and gridpoint, have differing degrees of freedom [5]. The storage requirements are such that it is not possible, in general, to achieve optimum use of the DAP store for the three representations. Whilst it is possible to utilise 100% of the PE array in gridpoint space for the T42 model, the spectral coefficients only occupy 46% of their matrix. On the other hand, a higher resolution of T62 would achieve 98% for the spectral coefficients but 75% for the gridpoint data (although a simple packing scheme could increase this to 90%).

The percentage use of the PE array by the spectral coefficients could be increased by packing them. For example, several variables could be held in one matrix using the long vector storage format [4]. However, this introduces additional overhead in unpacking and in any case the storage for the spectral coefficients is small compared to that of the Legendre polynomials.

3.1. Storage of the spectral coefficients

The spectral coefficients are stored in the DAP in the same way that they appear in Fig. 1. Since the model variables are real, only positive values of m need to be stored as the negative values are their complex conjugate. The imaginary parts are stored in the same matrix as the real parts where their mapping is obtained for efficiency by builtin DAP FORTRAN functions. A matrix transpose or the reversal of rows and columns is suitable, the latter is used in the model. This storage format can be expressed as

$$\begin{aligned} &F_{m,n} \text{ real:} && \{m,\ n\} \to (m+1,\ n+2), \\ &F_{m,n} \text{ imaginary:} && \{m,\ n\} \to (64-m,\ 63-n). \end{aligned} \tag{4}$$

This mapping expression is read as, for example, the $(m,\ n)$th real coefficient is found at the element in row $m+1$, column $n+2$ of the DAP matrix.

In order to achieve vectorisation of the Legendre transforms for the implementation of a spectral model on a vector machine, it has been found necessary to store the spectral coefficients in a 'diagonal-wise' manner [1], where coefficients along diagonals of $m+n$ odd or even are stored consecutively. A similar approach could be used for the DAP implementation where diagonals are stored down columns of the PE array. This could be expressed as

$$\begin{aligned} &F_{m,n} \text{ real:} && \{m,\ n\} \to (m+1,\ n-m+1), \\ &F_{m,n} \text{ imaginary:} && \{m,\ n\} \to (64-m,\ 64-n+m). \end{aligned}$$

A different storage format for the $P_{m,n}$ and their derivatives is necessary in this case but the code for the Legendre transforms remains the same apart from minor changes to some routing operations.

3.2. Storage of the Legendre polynomials and derivatives

As with the spectral coefficients, the $P_{m,n}$ are not packed in order to avoid routing overheads, which means the amount of storage required is considerable. The most efficient mapping was found to be when the wavenumber m and latitude μ are mapped horizontally on the DAP store and total wavenumber n increases vertically down the store. The derivatives $\mathrm{d}P_{m,n}/\mathrm{d}\mu$ can then be stored in the same storage area 'upside down' in much the same way as the imaginary parts of the spectral coefficients are stored in relation to the real parts. One benefit of storing the derivatives in this way is that the products involving the polynomial values and their derivatives in the direct transform (3) can then be computed in parallel.

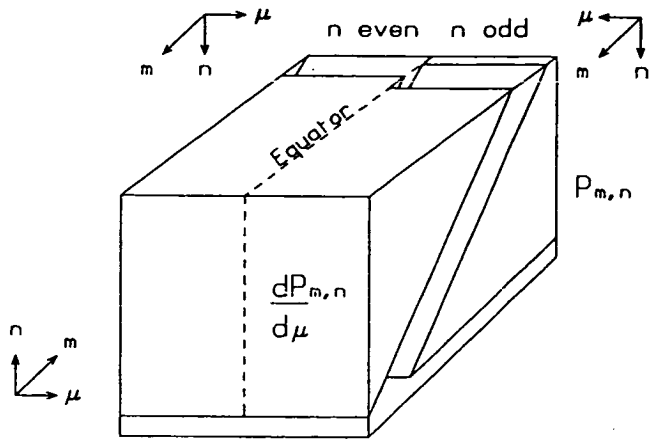


Fig. 2. Schematic illustration of the storage format for the Legendre polynomials and their derivatives.

The T42 resolution allows the symmetric and antisymmetric parts of the transforms to be computed in parallel. To do this, the storage format of the polynomial values best suited to (4), is to store all even values of n in one half of the array and all the odd values in the other half. This mapping can be written as

$$P_{m,n}\ (n \text{ even}): \quad \{m, n, \mu_j\} \to (m+1,\ j,\ \tfrac{1}{2}n+1),$$
$$P_{m,n}\ (n \text{ odd}): \quad \{m, n, \mu_j\} \to (m+1,\ 65-j,\ \tfrac{1}{2}(n+1))$$
$$\text{for } j = 1, \ldots, 32.$$

A similar expression occurs for the derivatives. Figure 2 illustrates this storage arrangement.

The total number of planes occupied is then 704 (assuming 32-bit precision) or 17% of the total store of the DAP. Within this area of memory the $P_{m,n}$ and the $\mathrm{d}P_{m,n}/\mathrm{d}\mu$ values together occupy 68%.

4. Legendre transform algorithms

4.1. Inverse transform

Having defined the storage arrangement for the relevant data, the procedure to compute (1) and (2) is straightforward. The algorithm is parallel in m, μ and symmetric and antisymmetric parts and sequential in n. The loop over n executes $(M+2)/2$ times and on each pass a column vector of spectral coefficients of n odd and even is broadcast and multiplied to the Legendre polynomial array. The symmetric and antisymmetric parts of the Fourier coefficients are then combined after the loop by simple routing.

It can be easily seen that this algorithm is not efficient. At worst ($n = 0$), only 1.5% of the PE array is performing useful work, whilst at best ($n = 42$) 67% is in use. Simple packing strategies can be used to improve on this but at the expense of being able to compute the products in (3) in parallel so that there is no net gain for the model.

4.2. Direct transform

The direct Legendre transform algorithm has additional parallelism to the inverse transform in that the products involving $P_{m,n}$ and $\mathrm{d}P_{m,n}/\mathrm{d}\mu$ are computed in parallel. Each pass of the

loop over n involves multiplying the symmetric and antisymmetric parts of the Fourier coefficients to the $P_{m,n}$ array and summing along each half row to give a vector for n odd and even. The two products are combined at the end of the loop to give the final spectral coefficients.

Unlike the inverse transform, the efficiency of this algorithm stays constant with 68% of the PE array performing useful work. The summation along rows was done in the model by a cascade sum routine written in DAP FORTRAN. However, the CPU time presented in the next section for the direct transform is corrected from that measured by assuming the use of an assembler written routine which takes the same time as the DAP FORTRAN builtin function that sums across all columns to give a vector result [4].

5. Results

Figure 3 shows the CPU times of the Legendre transforms measured for the T42 resolution and estimated at other resolutions. Also shown are the times for serial versions of the transforms, measured on a AMDAHL 470/V8 computer.

The DAP times for the inverse transform show the linear dependency on truncation, the increased slope between T42 and T62 occurs because the symmetric and antisymmetric components can no longer be computed in parallel. At resolutions greater than T62, additional work is required as the spectral and Legendre polynomial values require additional matrices of storage. By contrast, the serial algorithms show the cubic variation with truncation. Clearly, the DAP performance improves with the higher truncation; the ratio between the CPU times is 3.6 at T42 and 5.2 at T85. The T21 resolution is too low to make effective use of the DAP architecture.

Figure 3 also shows the times for the direct Legendre transform. The difference between the DAP and serial times is now much greater due to the additional parallelism exploited in the DAP algorithm. The ratio of these times is 5.6 at T42 and 8.8 at T85.

Whilst the ratio of the direct transform time to that of the inverse transform is about 2.8 for the serial routines, by contrast the DAP algorithms show a much smaller ratio of about 1.4

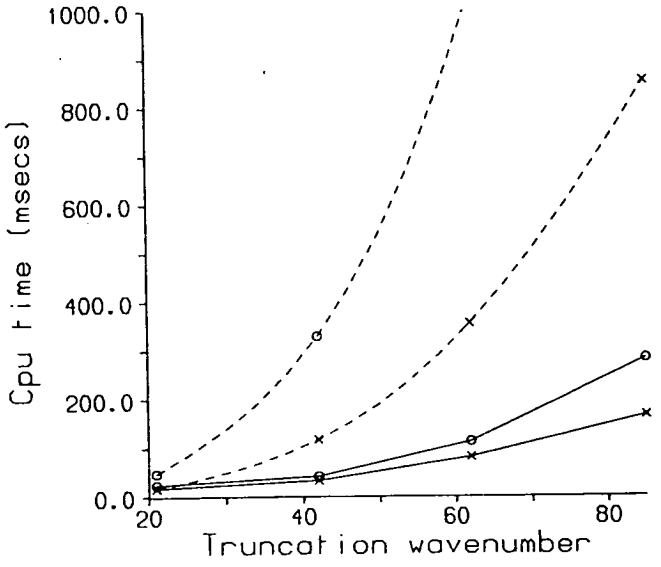


Fig. 3. Variation of CPU time with truncation for the Legendre transform algorithms. Solid curves represent the DAP routines, dashed curves the serial routines. Inverse transform times are shown with a cross, direct transform times with a circle.

(except at T85 where it is 1.7). The T85 DAP times are only rough guides as this resolution would require I/O during the transforms as the required space for the Legendre polynomials would exceed that available.

6. Discussion

The CPU time per step of this model is approximately 4 times that of the finite difference shallow-water model in [2]. However, equivalent spectral and finite difference models on serial and vector computers tend to require approximately the same computing time. From the above results this would suggest the use of higher resolutions for greater efficiency, or perhaps multilevel models where the additional available parallelism might lead to more efficient transform algorithms.

Acknowledgments

This paper forms a part of the author's Ph.D. thesis, for which financial support from SERC is acknowledged.

References

[1] A.P.M. Baede, M. Jarraud and U. Cubasch, Adiabatic formulation and organisation of ECMWF's spectral model, ECWMF Technical Report No. 15, European Centre for Medium Range Weather Forecasts, Reading, United Kingdom, 1979.
[2] J. Fishbourne, Unfinished Ph.D. thesis, Computer Science Department, Reading University, United Kingdom.
[3] G.J. Haltiner and R.T. Williams, *Numerical Prediction and Dynamic Meteorology* (Wiley, New York, 1980).
[4] I.C.L., DAP: FORTRAN language reference manual, ICL Technical Publication TP6918, 1979.
[5] M. Jarraud and A.J. Simmons, The spectral technique, *Proc. Seminar on Numerical Methods for Weather Prediction*, Vol. 2 (European Centre for Medium Range Weather Forecasts, Reading, 1984).
[6] C.R. Jesshope, The implementation of fast radix-2 transforms on array processors, *IEEE Trans. Comput.* **29** (1980) 20–27.
[7] S.F. Reddaway, DAP—a distributed array processor, *Proc. 1st IEEE/ACM Annual Symposium on Computer Architecture* (1973).
[8] S.F. Reddaway, D.J. Hunt and D. Parkinson, Study of a meteorological operational suite, ICL RADC Document CM.52, 1976.

References

Abramowitz M., Stegun I.A. 1965 *Handbook of mathematical functions with formulas, graphs and mathematical tables,* New York Dover Publications, 1046pp.

Adams L.M. 1982 Iterative algorithms for large sparse linear systems on parallel computers, NASA contractor report 166027, available from the DAP Support Unit, Doc. 8.34, Queen Mary College.

Adams L.M. 1983 m-step preconditioned conjugate gradient methods, NASA contractor report 172130, available from the DAP Support Unit, Doc. 8.32, Queen Mary College.

Amdahl G. 1967 The validity of the single processor approach to achieving large scale computing capabilities, *AFIRS Conf. Proc.*, SJCC, 3, 483-485.

AMT 1989 FORTRAN for the DAP, Centre for Parallel Computing, Newsletter No.2, Queen Mary College, London, 7-11.

Arakawa A. 1966 Computational design for long-term numerical integration of the equations of motion, *J. Comput. Phys.*, **1**, 119-143.

Arakawa A., Lamb V.R. 1977 Computational design of the basic dynamical processes of the UCLA general circulation model, *Meth. in Comp. Phys.*, **17**, 174-267.

Asselin R. 1972 Frequency filter for time integrations, *Mon. Wea. Rev.*, **100**, 487-490.

Axelrod T.S. 1986 Effects of synchronisation barriers on multiprocessor performance, *Parallel Computing*, **3**, 129–140.

Baede A.P.M., Dent D., Hollingsworth A. 1976 The effect of arithmetic precision on some meteorological integrations, *ECMWF Tech. Rep. No.2*, 22pp.

Baede A.P.M., Jarraud M., Cubasch U. 1979 Adiabatic formulation and organisation of ECMWF's spectral model, *ECMWF Tech. Rep. No.15*, 39pp.

Baer F. 1972 An alternate scale representation of atmospheric energy spectra, *J. Atmos. Sci.*, **29**, 649–664.

Baer F., Platzman G.W. 1961 A procedure for numerical integration of the spectral vorticity equation, *J. Meteor.*, **18**, 393–410.

Bates J.R. 1984 An efficient semi-Lagrangian and alternating direction implicit method for integrating the shallow water equations, *Mon. Wea. Rev.*, **112**, 2033–2047.

Bates J.R., McDonald A. 1982 Multiply upstream, semi-Lagrangian advective schemes: analysis and application to a multi-level primitive equation model, *Mon. Wea. Rev.*, **110**, 1831–1842.

Bates J.R., McDonald A. 1987 Improving the estimate of the departure point position in a two time level semi-Lagrangian and semi-implicit model, *Mon. Wea. Rev.*, **115**, 781–794.

Beland M., Beaudoin C. 1983 A global spectral model with a finite element formulation for the vertical discretization: Adiabatic formulation, *Mon. Wea. Rev.*, **113**, 1910–1919.

Bengtsson L. 1988 Computer requirements for atmospheric modelling, *Multiprocessing in meteorological models*, edited by Hoffmann and Snelling, Springer-Verlag, 438pp.

Blinova E.N. 1942 Hydrodynamic theory of pressure and temperature waves and centres of action of the atmosphere, (Translated from Russian, 1944), Regional control offices, Second Weather Region, Pattersonfield, U.S.A.

Boas M.L. 1966 *Mathematical methods in the physical sciences*, John Wiley and Sons, London, 778pp.

de Boor C. 1974 *Mathematical aspects of finite elements in partial differential equations*, Academic Press, New York.

Bourke W. 1972 An efficient, one-level, primitive equation spectral model, *Mon. Wea. Rev.*, **100**, 683–689.

Bourke W. 1974 A multi-level spectral model. I: Formulation and hemispheric integrations, *Mon. Wea. Rev.*, **102**, 687–701.

Bourke W., McAvaney B., Puri K., Thurling R. 1977 Global modelling of atmospheric flow by spectral methods, in: *Meth. in Comp. Phys.*, **17**, Academic Press.

Bowgen G. 1981a Iterative solution of tridiagonal linear equations, *DAP Support Unit Tech. Doc. No.2-9,* Queen Mary College.

Bowgen G. 1981b Iterative solution of partial differential equations, *DAP Support Unit Lecture Note 3,* Queen Mary College.

Brown M.W. 1986 Integrating Distributed Array Processing into EMAS 2900, *Software - practise and experience,* **16**, 517–529.

Bryan K. 1966 A scheme for numerical integration of the equations of motion on an irregular grid free of nonlinear instability, *Mon. Wea. Rev.*, **94**, 38–40.

Burridge D.M., Haseler J. 1977 A model for medium range weather forecasts – adiabatic formulation, *ECMWF Tech. Rep. No. 4,* Reading, U.K.

Burridge D.M., Steppler J., Strufing R. 1986 Finite element schemes for the vertical discretization of the ECMWF forecast model using linear elements, *ECMWF Tech. Rep. 54,* Reading, 51pp.

Carpenter K.M. 1981 The accuracy of Gadd's modified Lax-Wendroff algorithm for advection, *Q. J. R. Meteorol. Soc.*, **107**, 467–470.

Carver G. 1986 A note on a dynamic I/O system for dynamic balancing of the ECMWF forecast model, End of contract report (unpublished), ECMWF, August, 22pp.

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

Chalmers C.B., Kenway R.D., Roweth D. 1987 Algorithms for calculating quark propogators on large lattices, *J. Comp. Phys.*, **70**, 500–520.

Chang H.R., Shirer H.N. 1985 Compact spatial differencing techniques in numerical modelling, *Mon. Wea. Rev.*, **113**, 409–423.

Chang L.P., Takle E.S., Sani R.L. 1982 Development of a two-dimensional finite element PBL model and two preliminary model applications, *Mon. Wea. Rev.*, **110**, 2025–2037.

Charney J.G., Fjortoft R., von Neumann J. 1950 Numerical integration of the barotropic vorticity equation, *Tellus*, **2**, 237–254.

Cohn S.E., Dee D., Isaacson E., Marchesin D., Zwas G. 1985 A fully implicit scheme for the barotropic primitive equations, *Mon. Wea. Rev.*, **113**, 436–448.

Collins W.G. 1983 An accurate variation of the two-step Lax–Wendroff integration of horizontal advection, *Q. J. R. Meteorol. Soc.*, **109**, 255–261.

Conner J.J., Brebbia C.A. 1976 *Finite element techniques for fluid flow*, Newnes-Butterworths, 310pp.

Cooley J.W., Tukey J.W. 1965 An algorithm for the machine calculation of complex Fourier series, *Math. Comp.*, **19**, 297–301.

Corby G.A., Gilchrist A., Newson R.L. 1972 A general circulation model of the atmosphere suitable for long period integrations, *Q. J. R. Meteorol. Soc.*, **98**, 809-832.

Corby G.A., Gilchrist A., Rowntree P.R. 1977 U.K. Meteorological Office five-level general circulation model, *Meth. in Comp. Phys.*, **17**, 67-110.

Cote J., Beland M., Staniforth A.N. 1983 Stability of vertical discretization schemes for semi-implicit primitive equation models: Theory and application, *Mon. Wea. Rev.*, **111**, 1189-1207.

Courant R., Hilbert D. 1953 *Methods of Mathematical Physics, Vol I,* Interscience publishers Ltd, London, 561pp.

CRAY 1986 CRAY Programmer's Library Reference Manual, SR-0113, Cray Research Inc., USA.

Cullen M.J.P. 1973 A simple finite element method for meteorological problems, *J. Inst. Math. Applications*, **11**, 15-31.

Cullen M.J.P. 1974a A finite element method for a non-linear initial value problem, *J. Inst. Math. Applications*, **13**, 233-247.

Cullen M.J.P. 1974b Integration of the primitive equations on a sphere using the finite element method, *Q. J. R. Meteorol. Soc.*, **100**, 555-562.

Cullen M.J.P. 1976 On the use of artificial smoothing in Galerkin and finite difference solutions of the primitive equations, *Q. J. R. Meteorol. Soc.*, **102**, 77-93.

Cullen M.J.P. 1979 The finite element method,
Numerical methods used in atmospheric science,
vol 2, GARP Publications Series No.17, WMO.

Cullen M.J.P. 1983 Current progress and prospects in numerical techniques for weather prediction models,
J. Comput. Phys., **50**, 1-37.

Cullen M.J.P., Hall C.D. 1979 Forecasting and general circulation results from finite element models,
Q. J. R. Meteorol. Soc., **105**, 571-592.

Daley R. 1980 The development of efficient time integration schemes using model normal modes,
Mon. Wea. Rev., **108**, 100-110.

Daley R.W., Bourassa Y. 1978 Rhomboidal versus triangular spherical harmonic truncation: some verification statistics,
Atmosphere, **16**, 187-196.

Davies S. 1985 Notes on a DAP finite element library,
DAP Support Unit Doc.2-87, Queen Mary College.

Dent D. 1988 The ECMWF model: Past, present and future,
Multiprocessing in meteorological models,
eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Dent D., O'Neill M. 1988 Microtasking as a complement to macrotasking,
Parallel Computing, **8**, 149-154.

Deque M., Cariolle D. 1986 Some destabilizing properties of the Asselin time filter,
Mon. Wea. Rev., **114**, 880-884.

Dey C.H. 1969 A note on forecasting with the Kurihara grid, *Mon. Wea. Rev.*, **97**, 597-601.

Dixon L.C.W., Ducksbury P.G., Singh P. 1982 A parallel version of the conjugate gradient algorithm for finite element problems, Hatfield Poly., Numerical Optimisation Centre, Tech. Rep. No. 132.

Dixon L.C.W., Singh P. 1984 The finite element method for Navier-Stokes equations: A least squares approach, Hatfield Poly., Numerical Optimisation Centre, Tech. Rep. No. 142.

Dixon L.C.W., Ducksbury P.G. 1985 Finite element optimisation on the DAP, *Comp. Phys. Commun.*, **37**, 187-193.

Dongarra J.J. 1985 Performance of various computers using standard linear equations software in a FORTRAN environment, Argonne National Lab., Tech. Memo. No.23, Argonne, Illinois, USA.

Doron E., Hollingsworth A., Hoskins B.J., Simmons A.J. 1974 A comparison of grid-point and spectral methods in a meteorological problem, *Q. J. R. Meteorol. Soc.*, **100**, 371-383.

Dubois P., Greenbaum A., Rodrique G. 1979 Approximating the inverse of a matrix for use in iterative algorithms on vector processors, *Computing*, **22**, 257-268.

Ducksbury P.G. 1983 Experience solving non-linear PDE's by finite element optimisation on the DAP. I: The least squares conjugate gradient solution of Morgan's problem, Hatfield Poly., Numerical Optimisation Centre, Tech. Rep. No. 136.

Ducksbury P.G. 1986 *Parallel Array Processing,* Ellis Horwood Series in Electrical and Electronic Engineering, 112pp.

Duller A.W.G., Paddon D.J. 1984 Processor arrays and the finite element method, *Int. Conf. 'Parallel Computing 83',* Elsevier Science Publishers B.V. (North Holland), 131–136.

Dutton J.A. 1986 *The ceaseless wind,* Dover Publications Inc., 617pp.

ECMWF 1985 ECMWF Forecast model – Adiabatic part, *ECMWF Research Manual 2,* ECMWF, Reading.

Eliasen E., Machenhauer B., Rasmussen E. 1970 On a numerical method for integration of the hydrodynamical equations with a spectral representation of the horizontal fields, *Institut for Teoretisk Meteorologi,* Rep. No.2, University of Copenhagen.

Ellsaesser H.W. 1966 Evaluation of spectral versus grid methods of hemispheric numerical weather prediction, *J. Appl. Meteor.,* **5**, 246–262.

Ettinger J.E. 1987 Distributed Array Processors, *Infotech State of the Art Report,* **15:4**, ed. C.R.Jesshope, Pergamon Infotech Ltd, 19–36.

Evans D.J., Shanehchi J., Barlow R.H. 1984 Implementation of the conjugate gradient and Lanczos algorithms for large sparse banded matrices on the ICL DAP, *Int. Conf. 'Parallel Computing 83',* Elsevier Science Publishers B.V. (North Holland), 131–136.

Feo J.T. 1988 An analysis of the computational and parallel complexity of the Livermore loops, *Parallel Computing,* **7**, 163–185.

Fishbourne J. 1980 Unfinished Ph.D. thesis,
Comp. Sci. Dept., Reading University.

Flanders P.M., Hunt D.J., Reddaway S.F., Parkinson D. 1977 Efficient high speed computing with the Distributed Array Processor,
High speed computer and algorithm organisation,
Academic Press, 113–128.

Flanders P.M. 1982 A unified approach to a class of data movements on an array processor,
IEEE Trans. on computers, **C–31**, 809–819.

Flynn M.J. 1972 Some computer organisations and their effectiveness,
IEEE Trans. on computers, **C–21**, 948–960.

Gadd A.J. 1978a A numerical advection scheme with small phase speed errors,
Q. J. R. Meteorol. Soc., **104**, 583–594.

Gadd A.J. 1978b A split explicit integration scheme for numerical weather prediction,
Q. J. R. Meteorol. Soc., **104**, 569–582.

Gadd A.J. 1980 Two refinements of the split explicit integration scheme,
Q. J. R. Meteorol. Soc., **106**, 215–220.

Gadd A.J. 1985 The 15 level weather prediction model,
Met. Mag., **114**, 222–226.

Galerkin B. 1915 Rods and plates. Series occurring in various questions concerning the elastic equilibrium of rods and plates,
Vestnik Inzhenerov, **19**, 897–908.

Gallopoulos E.J. 1984 The massively parallel processor for problems in fluid dynamics, *Proc. of Vector and Parallel Processors in Comp. Sci.*, Oxford, England.

Gates W.L., Riegel C.A. 1962 A study of numerical errors in the integration of barotropic flow on a spherical grid, *J. Geophys. Res.*, **67**, 773-784.

Gates W.L., Batten E.S., Kahle A.B., Nelson A.B. 1971 A documentation of the Mintz-Arakawa two-level general atmospheric circulation model, *Advanced Research Projects Agency*, Report R-877-ARPA, Rand Corporation, Santa Monica, California.

Gentleman W.M., Sande G. 1966 Fast Fourier transforms - for fun and profit, *1966 Fall Joint Computer Conference*, AFIPS Proc., **29**, 563-578.

Gibson J.K. 1985 A production multitasking numerical weather prediction model, *Comp. Phys. Commun.*, **37**, 317-327.

Golub G.H., Van Loan C.F. 1983 *Matrix computations*, North Oxford, 476pp.

Gordon T., Stern W. 1974 Spectral modelling at GFDL, GARP WGNE Report, No. 7, 46-82.

Grimmer M., Shaw D.B. 1967 Energy conserving integrations of the primitive equations on the sphere, *Q. J. R. Meteorol. Soc.*, **93**, 337-349.

Gustafsson B. 1971 An alternating direction implicit method for solving the shallow-water equations, *J. Comput. Phys.*, **7**, 239-254.

Haltiner G.J., Williams R.T. 1980 *Numerical prediction and dynamic meteorology,* John Wiley and Sons, New York, 477pp.

Hestenes M.R., Stiefel E. 1952 Methods of conjugate gradients for solving linear systems, *J. Res. N. B. S.*, **49**, 409.

Hoare C.A.R. 1978 Communicating sequential processes, *Communications of the ACM,* **21**, 666–677.

Hobson E.W. 1931 *The theory of spherical and ellipsoidal harmonics,* Cambridge Univ. Press, 500pp.

Hockney R.W. 1977 Supercomputer architecture, *Infotech State of the Art Conf.: future systems,* Chairman F.Sumner, Maidenhead Infotech Intl, 65–93.

Hockney R.W. 1985 MIMD computing in the USA – 1984, *Parallel Computing,* 2, 119–136.

Hockney R.W. 1988 $(r_\infty, n_{\frac{1}{2}}, s_{\frac{1}{2}})$ Measurements on the 2-CPU CRAY X-MP, *Multiprocessing in Meteorological Models,* eds. Hoffmann and Snelling, Springer-Verlag, 67–88.

Hockney R.W., Jesshope C.R. 1981 *Parallel computers: Architecture, programming and algorithms,* Adam Hilger Ltd, Bristol, 423pp.

Hockney R.W., Snelling D.F. 1984 Characterizing MIMD computers:e.g. the Denelcor HEP, *Int. Conf. 'Parallel Computing 83',* Elsevier Science Publishers B.V. (North-Holland), 521–526.

Hoffman R.N., Nehrkorn T. 1989 Multiprocessing algorithms for global spectral numerical weather prediction, *Proc. of Int. Conf. on Supercomputing,* Crete, Greece, June 5-9.

Hoffmann G.-R., Snelling D.F. 1988 A comparative study of the ECMWF weather model on several multiprocessor architectures, *Multiprocessing in meteorological models,* eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Hoffmann G.-R., Swarztrauber P.N., Sweet R.A. 1988 Aspects of using multiprocessors for meteorological modelling, *Multiprocessing in meteorological models,* eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Holloway J.L., Manabe S. 1971 Simulation of climate by a global general circulation model: I. Hydrological cycle and heat balance, *Mon. Wea. Rev.*, 99, 335-370.

Holloway J.L., Spelman M.J., Manabe S. 1973 Latitude-longitude grid suitable for numerical time integration of a global atmospheric model, *Mon. Wea. Rev.*, **101**, 69-78.

Hoskins B.J. 1973 Stability of the Rossby-Haurwitz wave, *Q. J. R. Meteorol. Soc.*, **99**, 723-745.

Hoskins B.J., Simmons A.J. 1975 A multi-layer spectral model and the semi-implicit method, *Q. J. R. Meteorol. Soc.*, **101**, 637-655.

Hsiung C.C. 1988 The myth of performance for parallel machines, *Multiprocessing in Meteorological Modelling,* eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Hunt D.J. 1974a A proposed implementation of a meteorological initialization program on the DAP, *ICL Research and Advanced Development Centre,* Doc. No. CM24, 15pp.

Hunt D.J. 1974b A proposed implementation of the Meteorological Office global circulation model on the DAP, *ICL Research and Advanced Development Centre,* Doc. No. CM29, 13pp.

Hunt D.J. 1981 A study of finite element analysis on DAP, *ICL Research and Advanced Development Centre,* Doc. No. CM22.

Hunt D.J., Webb S.J., Wilson A. 1980 Application of a parallel processor to the solution of finite difference problems, DAP Support Unit, Doc. 4.8, Queen Mary College.

ICL 1979 DAP:FORTRAN language reference manual, *ICL Tech. Publ. TP6918.*

Ikeda M. 1988 Multitasking with a memory heirarchy, *Multiprocessing in meteorological models,* eds. Hoffmann & Snelling, Springer-Verlag, 438pp.

Jarraud M., Girard C. 1983 An extensive quasi-operational comparison between a spectral and a gridpoint model, *ECMWF seminar proceedings,* 'Numerical methods for weather prediction', Reading.

Jarraud M., Simmons A.J. 1983 Adiabatic formulation of models: the spectral technique., *ECMWF seminar proceedings,* 'Numerical methods for weather prediction', Reading.

Jesshope C.R. 1980 The implementation of fast radix-2 transforms on array processors,
IEEE Trans. Comput., **C-29**, 20-27.

Jesshope C.R. 1987 Parallel processing,
Infotech State of the Art Report, **15:4**,
ed. C.R.Jesshope, Pergamon Infotech Ltd, 336pp.

Jordan H.F. 1987 The Force,
The characteristics of parallel algorithms,
MIT Press, 395-436.

Kasahara A. 1977 Numerical integration of the global barotropic primitive equations with Hough harmonic expansions,
J. Atmos. Sci., **34**, 687-701.

Kasahara A. 1978 Further studies on a spectral model of the global barotropic primitive equations with Hough harmonic expansions,
J. Atmos. Sci., **35**, 2043-2051.

Kasahara A., Washington W.M. 1971 General circulation experiments with a six-layer NCAR model including orography, cloudiness and surface temperature calculations,
J. Atmos. Sci., **28**, 657-701.

Klappholz D., Kong X., Parle H.C., Stein K. 1987 Refined languages: an evolutionary approach to the use of sequential languages for programming parallel (MIMD) machines,
Infotech State of the Art Report, **15:4**,
ed. C.R.Jesshope, Pergamon Infotech Ltd, 59-70.

Krylov V.I. 1962 *Approximate calculation of integrals*,
MacMillan, New York, 357pp.

Kuehn J.T., Siegel H.J. 1985 Extensions to the C programming language for SIMD/MIMD parallelism, *Proc. IEEE Intl. Conf. Parallel Processing*, 232–235.

Kurihara Y. 1965 Numerical integration of the primitive equations on a spherical grid, *Mon. Wea. Rev.*, **93**, 399–415.

Kurihara Y., Tuleya R.E. 1974 Comments "On the importance of precision for short-range forecasting and climate simulation", *J. Appl. Met.*, **13**, 601–602.

Lai C.H. 1989 The DAP finite element library, Centre for Parallel Computing, Doc 5A.31, Queen Mary College, London.

Lai C.H., Liddell H.M. 1987a A review of parallel finite element methods on the DAP, *Appl. Math. Modelling*, **11**, 330–340.

Lai C.H., Liddell H.M. 1987b Preconditioned conjugate gradient methods on the DAP, *Math. of Finite Elements and Appl.*, **IV**, 145–156, ed. J.R.Whiteman, Academic Press.

Lai C.H., Liddell H.M. 1988 Finite elements using long vectors of the DAP, *Parallel Computing*, **8**, 351–362.

Larson J.L. 1988 Practical concerns in multitasking on the CRAY X-MP, *Multiprocessing in Meteorological Modelling*, eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Leasure B. 1988 An approach to automatic parallel processing, *Multiprocessing in meteorological models*, eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Lilly D.K. 1965 On the computational stability of numerical solutions of time dependent non-linear geophysical fluid dynamics problems, *Mon. Wea. Rev.*, **93**, 11-26.

Lipps F.B. 1971 Two dimensional numerical experiments in thermal convection with vertical shear, *J. Atmos. Sci.*, **28(1)**, 3-19.

Machenhauer B. 1979 The spectral method, in: *Num. Meth. used in Atm. models*, vol 2, GARP Publications Series No.17, 124-277.

Machenhauer B., Rasmussen E. 1972 On the integration of the spectral hydrodynamical equations by a transform method, *Institut for Teoretisk Meteorologi*, Rep. No.3, University of Copenhagen.

Mailhot J., Benoit R. 1982 A finite element model of the atmospheric boundary layer suitable for use with numerical weather prediction models, *J. Atmos. Sci.*, **39**, 2249-2266.

Mandl F. 1957 *Quantum Mechanics*, Butterworths, London, 267pp.

Marchuk G.I. 1974 *Numerical methods in weather prediction*, Academic Press, New York, 277pp.

Matsuno T. 1966 Numerical integrations of the primitive equations by a simulated backward difference method, *J. Meteor. Soc. Japan*, Ser. 2, **44**, 76-84.

McBryan O.A. 1988 New architectures: Performance highlights and new algorithms, *Parallel Computing*, **7**, 477-499.

McDonald A. 1986 A semi-Lagrangian and semi-implicit two time level integration scheme, *Mon. Wea. Rev.*, **114**, 824-830.

Mcintosh D.H., Thom A.S. 1981 *Essentials of meteorology*, Taylor and Francis Ltd, London, 240pp.

Merilees P.E. 1973 An alternative scheme for the summation of a series of spherical harmonics, *J. Appl. Meteor.*, **12**, 224-227.

Merilees P.E., Ducharme P., Jacques G. 1977 Experiments with a polar filter and a one-dimensional semi-implicit algorithm, *Atmosphere*, **15**, 19-32.

Mesinger F., Arakawa A. 1976 Numerical methods used in atmospheric models, *GARP Publ. Ser. No. 17*, Vol I, WMO-ICSU.

Miller M.J., Thorpe A.J. 1981 Radiation conditions for the lateral boundaries of limited-area numerical models, *Q. J. R. Meteorol. Soc.*, **107**, 615-628.

Mitchell A.R., Wait R. 1977 *The finite element method in partial differential equations*, John Wiley and Sons, 198pp.

Miyakoda K. 1973 Cumulative results of testing a meteorological mathematical model. The description of the model, *Proc. Roy. Irish Academy*, **73A**, 99.

Mouhas C. 1987 Automatic mesh generation on the DAP, Centre for Parallel Computing, Doc. 5A.13, Queen Mary College, London.

Mozdzynski G. 1988 Multitasking on the ETA10, *Proc. of Workshop on Multiprocessing in meteorological models*, ECMWF, Dec. 1988.

Navon I.M., Riphagen H.A. 1979 An implicit compact fourth-order algorithm for solving the shallow-water equations in conservation law form, *Mon. Wea. Rev.*, **107**, 1107-1127.

Ogura Y., Phillips N.A. 1962 Scale analysis of deep and shallow convection in the atmosphere, *J. Atmos. Sci.*, **19**, 173-179.

Orlanski I. 1976 A simple boundary condition for unbounded hyperbolic flows, *J. Comput. Phys.*, **21**, 251-269.

Orlanski I., Ross B.B. 1973 Numerical simulation of the generation and breaking of internal gravity waves, *J. Geophys. Res.*, **78**, 8808-8826.

Orlanski I., Ross B.B., Polinsky L.J. 1974 Diurnal variation of the planetary boundary layer in a mesoscale model, *J. Atmos. Sci.*, **31**, 965-989.

Orlanski I., Ross B.B. 1977 The circulation associated with a cold front, Part I: dry case, *J. Atmos. Sci.*, **34**, 1619-1633.

Orszag S.A. 1970 Transform method for calculation of vector coupled sums: Application to the spectral form of the vorticity equation, *J. Atmos. Sci.*, **27**, 890-895.

Orszag S.A. 1971 Numerical simulation of incompressible flows within simple boundaries. I. Galerkin (spectral) representations, *Studies in Appl. Math.*, Vol L, No.4, 293-327.

Orszag S.A. 1974 Fourier series on spheres,
Mon. Wea. Rev., **102**, 56-75.

Owen A. 1982 A note on using finite differences on the ICL Distributed Array Processor,
Appl. Math. Modelling, **6**, 394-396.

Perrott R.H., 1987 The design and implementation of a Pascal based language for array processor architectures,
J. of Parallel and Distrib. Computing, **4**, 266-287.

Phillips N.A. 1959 Numerical integration of the primitive equations on the hemisphere,
Mon. Wea. Rev., **87**, 333-345.

Pielke R.A. 1984 *Mesoscale meteorological modelling*,
Academic Press, 612pp.

Pozo R., MacDonald A.E. 1989 Performance characteristics of scientific computation on the Connection Machine,
Dept. of Comp. Sci. Rep. CU-CS-440-89,
Colorado Univ., Boulder, USA.

Reddaway S.F. 1973 DAP - a distributed array processor,
1st Annual Sym. on Comput. Architecture,
IEEE/ACM, Florida.

Reddaway S.F., Hunt D.J., Parkinson D. 1976 Study of a meteorological operational suite,
ICL Research and Advanced Development Centre,
Doc. No. CM52, 15pp.

Reid J.K., Wilson A. 1986 The array features in FORTRAN 8X with examples of their use,
Comp. Phys. Commun., **37**, 125-132.

Ritchie H. 1985 Application of a semi-Lagrangian integration scheme to a moisture equation in a regional forecast model, *Mon. Wea. Rev.*, **113**, 424-435.

Ritchie H. 1987 Semi-Lagrangian advection on a Gaussian grid, *Mon. Wea. Rev.*, **115**, 608-619.

Ritchie H. 1988 Application of the semi-Lagrangian method to a spectral model of the shallow water equations, *Mon. Wea. Rev.*, **116**, 1587-1598.

Robert A.J. 1966 The integration of a low order spectral form of the primitive meteorological equations, *J. Meteor. Soc. Japan*, Ser. 2, **44**, 237-245.

Robert A.J. 1969 The integration of a spectral model of the atmosphere by the implicit method, *Proc. WMO/IUGG Symp. on Numerical Weather Prediction*, Tokyo, Japan Meteor. Agency, VII-19-VII-24.

Robert A.J. 1981 A stable numerical integration scheme for the primitive meteorological equations, *Atmos. Ocean*, **19**, 35-46.

Robert A.J. 1982 A semi-Lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations, *J. Meteor. Soc. Japan*, **60**, 319-325.

Robert A.J., Henderson J., Turnbull C. 1972 An implicit time integration scheme for baroclinic models of the atmosphere, *Mon. Wea. Rev.*, **100**, 329-335.

Robert A.J., Yee T.L., Ritchie H. 1985 A semi-Lagrangian and semi-implicit numerical integration scheme for multilevel atmospheric models, *Mon. Wea. Rev.*, **113**, 388-394.

Ross B.B., Orlanski I. 1978 The circulation associated with a cold front. Part 2: Moist case, *J. Atmos. Sci.*, **35**, 445-465.

Sadourney R. 1972 Conservative finite difference approximations of the primitive equations on quasi-uniform spherical grids, *Mon. Wea. Rev.*, **100**, 136-144.

Sadourney R. 1975a Compressible model flows on the sphere, *J. Atmos. Sci.*, **32**, 2103-2110.

Sadourney R. 1975b The dynamics of finite difference models of the shallow water equations, *J. Atmos. Sci.*, **32**, 680-689.

Sadourney R., Arakawa A., Mintz Y. 1968 Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere, *Mon. Wea. Rev.*, **96**, 351-356.

Sakellarides G. 1984 Some properties of the Asselin time filter in the presence of friction, *ECMWF Tech. Memo. No. 89*, Reading, U.K., 20pp.

Sankar-Rao M., Unscheid L. 1969 Tests of the effect of grid resolution in a global prediction model, *Mon. Wea. Rev.*, **97**, 659-664.

Sawyer J.S. 1963 A semi-Lagrangian method of solving the vorticity advection equation, *Tellus*, **15**, 336-342.

Schendel U. 1984 *Introduction to numerical methods for parallel computers,* Ellis Horwood Ltd, 151pp.

Schlesinger R.E., Uccellini L.W., Johnson D.R. 1983 The effects of the Asselin time filter on numerical solutions of the linearised shallow water equations, *Mon. Wea. Rev.*, **111**, 455–467.

Searle J.W., Davies D.R. 1975 A note on the effect of gridpoint numerical errors on the large-scale flow characteristics of a four-level model of the atmosphere, *Tellus*, **27**, 155–156.

Silberman I. 1954 Planetary waves in the atmosphere, *J. Meteor.*, **11**, 27–34.

Simmons A.J., Hoskins B.J. 1975 A comparison of spectral and finite difference simulations of a growing baroclinic wave, *Q. J. R. Meteorol. Soc.*, **101**, 551–565.

Simmons A.J., Hoskins B.J., Burridge D. 1978 Stability of the semi-implicit method of time integration, *Mon. Wea. Rev.*, **106**, 405–415.

Snelling D.F. 1988a Standard FORTRAN77 as a parallel language, *Parallel Computing*, **8**, 409–414.

Snelling D.F. 1988b A high resolution parallel Legendre transform algorithm, *Lecture Notes in Comp. Sci.*, **297**, 'Supercomputing', Springer-Verlag, 854–862.

Snelling D.F. 1988c Tools for assessing multiprocessing, *Multiprocessing in Meteorological Modelling*, eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Snelling D.F., Hoffmann G.-R. 1988 A comparative study of libraries for parallel processing, *Parallel Computing*, **8**, 255-266.

Staniforth A.N. 1987 Review: Formulating efficient finite-element codes for flows in regular domains, *Int. J. Num. Meth. in Fluids*, **7**, 1-16.

Staniforth A.N., Daley R.W. 1977 A finite element formulation for the vertical discretization of sigma coordinate primitive equation models, *Mon. Wea. Rev.*, **105**, 1108-1118.

Staniforth A.N., Mitchell H.L. 1977 A semi-implicit finite element barotropic model, *Mon. Wea. Rev.*, **105**, 154-169.

Staniforth A.N., Mitchell H.L. 1978 A variable resolution finite element technique for regional forecasting with the primitive equations, *Mon. Wea. Rev.*, **106**, 439-447.

Staniforth A.N., Daley R.W. 1979 A baroclinic finite element model for regional forecasting with the primitive equations, *Mon. Wea. Rev.*, **107**, 107-121.

Staniforth A.N., Pudykiewicz J. 1985 Reply to comments on and addenda to "Some properties and comparative performance of the semi-Lagrangian method of Robert in the solution of the advective-diffusion equation", *Atmos. Ocean*, **23**, 195-200.

Staniforth A.N., Beaudoin C. 1986 On the efficient evaluation of certain integrals in the Galerkin finite element method, *Int. J. Num. Meth. in Fluids*, **6**, 317-324.

Staniforth A.N., Temperton C. 1986 Semi-implicit semi-Lagrangian integration schemes for a barotropic finite element regional model, *Mon. Wea. Rev.*, **114**, 2078-2090.

Stephens P.D., Yarwood J.K. 1986 Providing multi-user access to Distributed Array Processors, *Software - practise and experience*, **16**, 531-539.

Steppler J. 1986 Finite element schemes for the vertical discretization of the ECMWF forecast model using quadratic and cubic elements, *ECMWF Tech. Rep.55*, Reading, UK, 59pp.

Stoker J.J., Isaacson E. 1975 *Final Report 1*, Courant Institute of Mathematical Sciences, IMM 407, New York Univ., 73pp.

Strang G., Fix G.J. 1983 *An analysis of the finite element method*, Prentice-Hall, 306pp.

Suarez M.J. 1988 Atmospheric modelling on a SIMD computer, *Multiprocessing in Meteorological Modelling*, eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Sundstrom A., Elvius T. 1979 Computational problems related to limited area modelling, *Numerical methods used in atmospheric science*, Vol II, GARP Publ. Series No. 17, WMO.

Swartz B., Wendroff B. 1974 The relative efficiency of finite difference and finite element methods. I. Hyperbolic problems and splines, *SIAM J. Numer. Anal.*, **11**, 979-993.

Tanguay M., Robert A.J. 1986 Elimination of the Helmholtz equation associated with the semi-implicit scheme in a gridpoint model of the shallow-water equations, *Mon. Wea. Rev.*, **114**, 2154-2162.

Tanqueray D.A., Snelling D.F. 1988 A distributed memory implementation of the shallow water equations, *Proc. ECMWF Workshop on Multiprocessors in meteorological models*, ECMWF, Reading.

Temperton C. 1977 Normal modes of a barotropic version of the ECMWF gridpoint model, *ECMWF Research Dept. Internal Rep. 12*, Reading, 39pp.

Temperton C. 1983 Fast mixed-radix real Fourier transforms, *ECMWF Tech. Memo. No.71*, Reading, 18pp.

Temperton C., Staniforth A.N. 1987 An efficient two time level semi-Lagrangian semi-implicit integration scheme, *Q. J. R. Meteorol. Soc.*, **113**, 1025-1039.

Tett S.F.B., Harwood R.S., Kenway R.D. 1988 Implementation of atmospheric models on large multi-processor surfaces, *ECMWF Workshop on Multiprocessing in Meteorological models*, December.

Umscheid L., Sankar-Rao M. 1971 Further tests of a grid system for global numerical prediction, *Mon. Wea. Rev.*, **99**, 686-690.

Umscheid L., Bannon P.R. 1977 A comparison of three global grids used in numerical prediction models, *Mon. Wea. Rev.*, **105**, 618-635.

Wait R. 1986 The solution of finite element equations on the DAP,
Proc. of Vector and Parallel Processing,
Leon, Norway, June.

Wait R. 1988 Partitioning and preconditioning of finite element matrices on the DAP,
Parallel Computing, **8**, 275-284.

Wait R., Martindale I. 1985 Finite elements on the DAP,
The math. of finite elements and applications,
V, 113-122, ed. J.R.Whiteman', Academic Press.

Wang H.H., Halpern P., Douglas J., Duport T. 1972 Numerical solutions of the one-dimensional primitive equations using Galerkin approximations with localised basis functions,
Mon. Wea. Rev., **100**, 738-746.

Washington W.M., Kasahara A. 1970 A January simulation experiment with the two-layer version of the NCAR global circulation model,
Mon. Wea. Rev., **98**, 559-580.

White P.W., Wiley R.L. 1988 U.K. Meteorological Office's plans for using multiprocessor systems,
Multiprocessing in Meteorological Modelling,
eds. Hoffmann and Snelling, Springer-Verlag, 438pp.

Whiteway J. 1979 A parallel algorithm for solving tridiagonal systems,
DAP Newsletter No. 3, DAP Support Unit,
Queen Mary College, London.

Wiin-Nielson A. 1959 On the applications of trajectory methods in numerical forecasting,
Tellus, **11**, 180-196.

Williams R.T. 1981 On the formulation of finite element prediction models,
Mon. Wea. Rev., **109**, 463-466.

Williams R.T., Zienkiewicz O.C. 1981 Improved finite element forms for the shallow-water wave equations, *Int. J. Num. Meth. Fluids*, **1**, 81-97.

Williamson D.L. 1968 Integration of the barotropic vorticity equation on a spherical geodesic grid, *Tellus*, **20**, 642-653.

Williamson D.L. 1970 Integration of the primitive barotropic model over a spherical geodesic grid, *Mon. Wea. Rev.*, **98**, 512-520.

Williamson D.L. 1971 A comparison of first and second order difference approximations over a spherical geodesic grid, *J. Comput. Phys.*, **7**, 301-309.

Williamson D.L. 1976 Linear stability of finite difference approximations on a uniform latitude longitude grid with Fourier filtering, *Mon. Wea. Rev.*, **104**, 31-41.

Williamson D.L. 1979 Difference approximations for fluid flow on a sphere, *Numerical methods used in atmospheric models*, Vol II, GARP Publ. Series No. 17, WMO, 53-123.

Williamson D.L., Browning G.L. 1973 Comparison of grids and difference approximations for numerical weather prediction over a sphere, *J. Appl. Meteor.*, **12**, 264-274.

Williamson D.L., Washington W.M. 1973 On the importance of precision for short-range forecasting and climate simulation, *J. Appl. Met.*, **12**, 1254-1258.

Zienkiewicz O.C. 1971 *The finite element method in engineering science*, McGrawHill, 521pp.