TECHNICAL MEMORANDUM

80

A multi tasking numerical weather prediction model

J.K. Gibson

Research Department

October 1983

European Centre for Medium-Range Weather Forecasts
Europäisches Zentrum für mittelfristige Wettervorhersage
Centre européen pour les prévisions météorologiques à moyen

1. INTRODUCTION

One of the limiting factors in the operational production of weather forecasts using numerical prediction models is the time taken for the computer code to execute. Recent experience would suggest that a global prediction model is desirable for medium range weather forecasts for a period of 3 to 10 days. Current research work at ECMWF indicates that greater accuracy of prediction is possible using models of higher resolution, and using more sophisticated computational techniques. Since the current operational forecast fully utilises the computer resources and time available, further improvement is only possible given more powerful hardware and better techniques. To this end, ECMWF have agreed to purchase a Cray X-MP/22 computer together with a 128 M byte solid state storage device (SSD). This computer configuration is capable of allowing a single job to make simultaneous use of two central processing units. This paper describes the preparation and development of a preliminary multi-tasking version of ECMWF's numerical prediction model, designed to make use of the dual-processing facilities of the Cray X-MP.

2. TERMINOLOGY

2.1 Task

A task is a set of computer instructions which must be processed in sequential order. Within the context of this paper, a task consists of one or more subroutines, and constitutes an entity which can be scheduled for independent, or quasi-independent execution on a physical CPU along with other tasks. This definition of a task is consistent with the concept of "library tasks" used in some publications.

Each task has a unique identity. It may share data areas with other tasks. Quasi-independent execution implies that external communication and synchronisation with other tasks may be required.

2.2 Parallelism

Parallelism is the ability to process two or more software entities at the same time. Rigsbee (1983) has defined levels of parallelism of software processes as follows:-

LEVEL 1 Jobs; independent jobs, each has a CPU.

LEVEL 2 Job steps; related parts of the same job.

LEVEL 3 Routines.

LEVEL 4 Loops or blocks.

LEVEL 5 Statements.

2.3 Multi-Tasking

Multi-tasking is a mode of operation such that processor resources are shared among multiple quasi-independent tasks. It is usually (but not necessarily) associated with a multiprocessing environment (i.e. where more than one CPU is available). It enables parallelism to be exploited at level 3, and may be compared to multiprogramming, which exploits parallelism at level 1.

3. THE CRAY X-MP/22

3.1 Hardware Features

The Cray X-MP series of computers contain central processing units based on 16 gate array integrated circuits. The clock cycle time is 9.5 nanoseconds, and the memory bank cycle time is 38 nanoseconds. A Cray X-MP/22 contains 2 million 64 bit words of memory, and 2 CPU's. Four parallel memory access ports are available to each processor, giving a memory bandwidth eight times as large as the usable bandwidth of a Cray-1. Hardware improvements coupled with this higher bandwidth enable better instruction chaining than possible on a Cray-1, while larger instruction buffers reduce the frequency at which

instructions are fetched from memory. A CPU intercommunication section contains 3 clusters of shared registers, together with a shared real time clock.

The input/output support for ECMWF's Cray X-MP will consist of a 128 M byte solid state storage device (SSD), and 16 discs coupled via an input/output subsystem comprised of two processors linked by 8 M bytes of buffer memory. The SSD is connected to the Cray X-MP by a 1250 M byte/sec channel; it is thus capable of supporting input/output at such speed as to render the need for simultaneous input/output and overlapped CPU processing redundant.

3.2 Software Features

The Cray operating system (COS) supports a concept where tasks are scheduled in a manner similar to that normally used for job steps. Multi-tasking is supported by means of a set of library routines. These enable tasks to be initiated, locks to be set, events to be posted, tests to be made, and synchronisation to be achieved.

4. MULTI-TASKING USING THE CRAY X-MP

4.1 The Purpose of Multi-Tasking

Multi-tasking only makes sense if the resulting multiple tasks can be processed at higher speed than the corresponding sequence of single tasks. The benefit of multi-tasking may be expressed in terms of a speed-up factor:-

$$S = \frac{T(\text{single tasking})}{T(\text{multi-tasking})}$$

where T(single tasking) = time taken in single tasking mode

T(multi-tasking) = time taken in multi-tasking mode.

A value of S in excess of 1.0 will only be achieved if there is sufficient parallelism to utilise more than one CPU to perform the computations at a faster rate than could be acheived by a single processor, allowing for extra overheads necessarily incurred by the multi-tasking control mechanism.

It is only possible to realise the full benefit of multi-tasking when the multi-tasking job has sole command of the computer. Multi-tasking should only be used for time critical jobs which are likely to execute on the computer either in isolation, or at high priority.

4.2 Suitable Problems for Multi-Tasking

For a problem to be a suitable candidate for multi-tasking, certain basic requirements are important. First, the problem must be divisible into a set of tasks which will form a balanced load over the number of CPU's available. Secondly, tasks must be of sufficient length to justify the overheads of multi-tasking. Data required to be stored by tasks should be mutually distinct, and tasks should not alter data required as input to possibly simultaneous tasks. Computational independence and storage independence are problems which require careful and detailed attention. They are treated at length in the computer literature, and Rigsbee (1983) details aspects which are of particular relevence to multi-tasking on a Cray X-MP. For the purpose of this paper it is sufficient to state that routines comprising tasks should be capable of being re-entrant, should have no side effects, and should not re-define their inputs. Where critical regions exist, such as the updating of a global value within several tasks, such regions should be locked. Locking is a mechanism which restricts the execution of such critical regions to one such region at a time. The length and number of critical regions should be minimised to minimise overheads. All critical regions must be identified, otherwise results will be unpredictable.

5. THE ECMWF T63 SPECTRAL MODEL

5.1 The Single Tasked Version

A full description of the ECMWF spectral model will be contained in the forthcoming Model Manual. The system adopted for source code organisation is described in Gibson (1982), and algorithms used for memory management are listed in Gibson (1983). The following brief overview indicates, in simplified form, the general structure of the code.

The single tasked version was designed with multi-tasking in view, but without knowledge of the multi-tasking interface. In consequence, the principle specifications for the design were

a) modularity, with tree-like calling structures.

b) data separation, using a well managed system for addressing memory.

c) input/output functions separately coded in a separate library.

5.2 Organisation of the Computation

Figure 1 illustrates the partitioning of control for the two scan version of the model within the time stepping system. The forcast is advanced by a time interval of 20 minutes each time step. Thus 720 such steps are required to complete a ten day forecast. Two input/output scans are required, each involving a complete pass over a group of work files. Scan 2 (N2SC2) writes symmetric and antisymmetric Fourier components to work files, while scan 1 (NNSC1) reads these components, reads additional grid point values, performs computations in Fourier and grid point space, writes new grid point values, and alters spectral values utilizing increments computed in Fourier and grid point space. Between these data scanning routines the spectral computations which govern the dynamics of the model are completed.

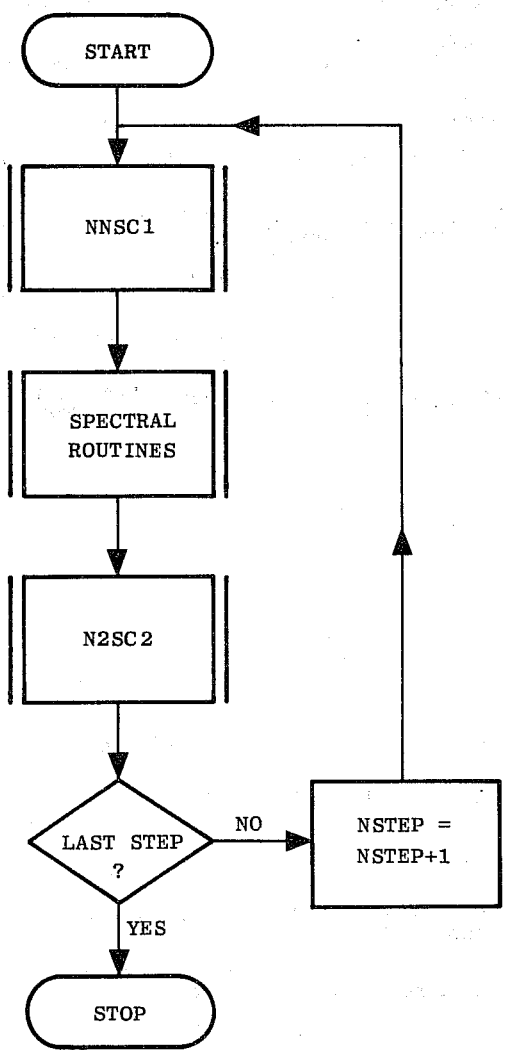

Fig. 1 ECMWF F/C - STEP CONTROL

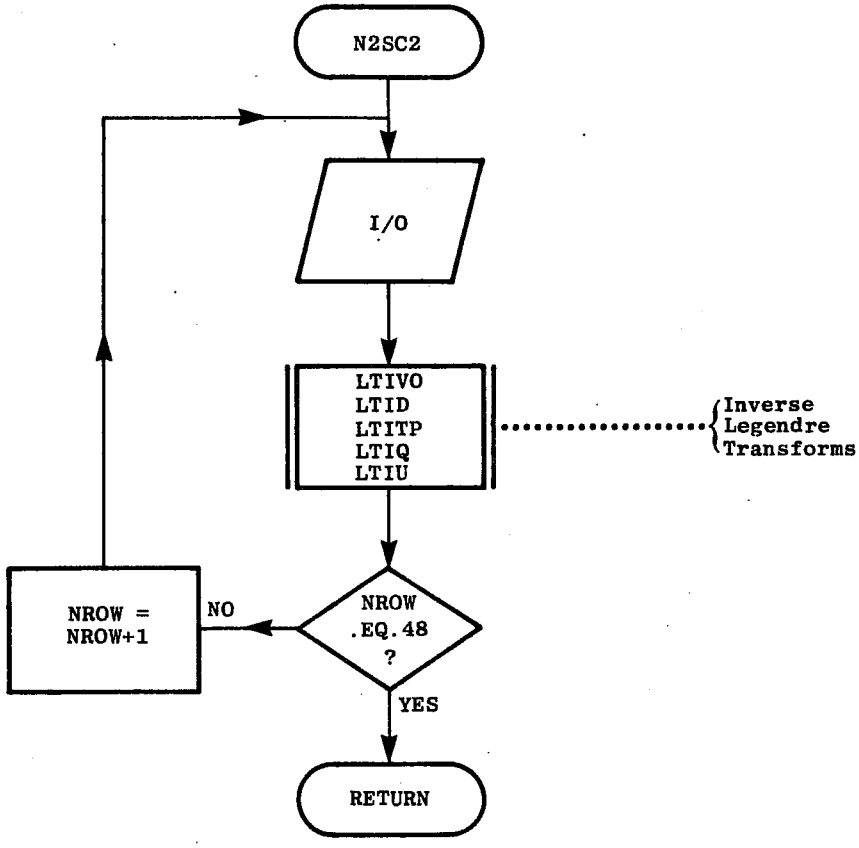

Fig. 2 ECMWF F/C - SCAN 2

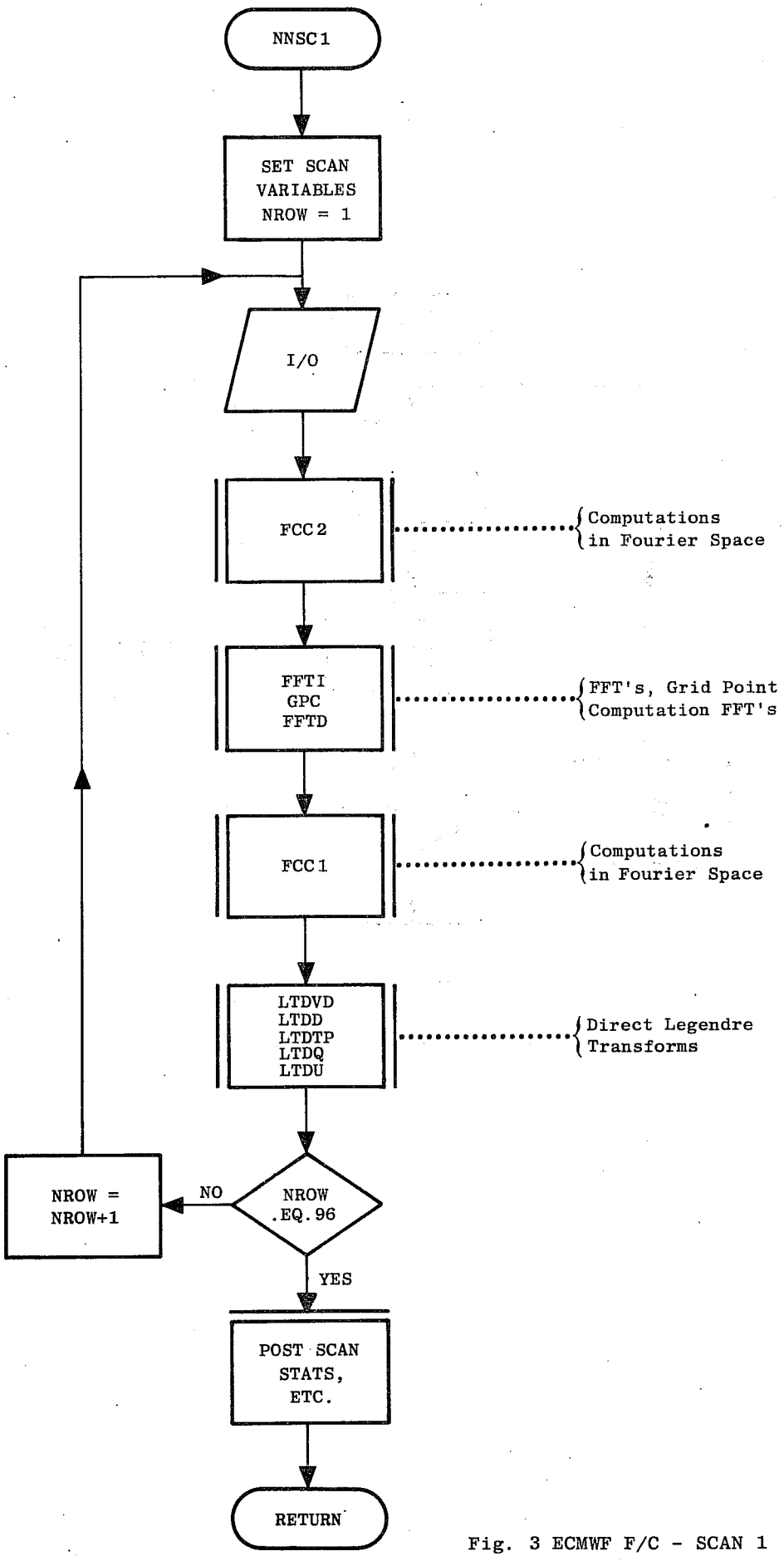

Fig. 3 ECMWF F/C - SCAN 1

Figure 2 illustrates scan 2 of the model. The computation takes place within five subroutines, each of which performs an inverse Legendre transform on one of the model variables.

Figure 3 illustrates scan 1 of the model. This involves computations in Fourier space; a transformation to grid point space using fast Fourier transforms; extensive grid point computations for the time filter, the dynamics and physics, and part of the semi-implicit time-stepping of the model; reverse Fourier transforms; further computation in Fourier space, and finally direct Legendre transforms from Fourier to spectral space. At the end of the scan statistics are computed and printed.

5.3 Addressing Variables

The code for the forecast model is written in FORTRAN. In general, the 1977 ANSI standard is followed closely, but Cray FORTRAN extensions are used to manage memory and address variables. The Cray POINTER statement allows dynamic addressing of array space by basing arrays concerned on addresses contained in associated POINTER variables. Use of this concept enabled a memory management package to be produced which

a) facilitates the naming of variables, and the splitting of buffers into array space.

b) supports a system of array recognition based on two attributes - a name and a code number. Arrays are located or allocated by supplying both a name and a code. Subsequent references within a subroutine are then by name only, in the normal FORTRAN manner. This system supports the ability to maintain several distinct data areas, each addressed by a common name, but distinguished by their different code numbers.

c) allows arrays to be allocated when required, and released when no longer needed. This uses the available space to maximum effect without the need to resolve complicated data mappings.

d) enables groups of arrays to be allocated contiguous space, to build buffer areas.

The call sequence for the most used routines of the memory manager illustrate these points:-

```
CALL ALLOCA(IPT,ILEN,'NAME',ICODE)
CALL ALLOCB(IPT,ILEN,'NAME',ICODE)
CALL LOCATE(IPT,'NAME',ICODE)
CALL UNLOC('NAME',ICODE)
```

where

ALLOCA	allocates space for an array.
ALLOCB	allocates space for an array such that successive calls to ALLOCB allocate contiguous space.
LOCATE	locates a previously allocated array.
UNLOC	releases space previously allocated.

and

IPT	is a POINTER variable in which is returned an address.
ILEN	is the array length required.
'NAME'	is the array name.
ICODE	is the code number used to qualify the array name.

5.4 Input/Output

If all of the intermediate results of the forecast model were to be retained in memory, a computer with 8 megawords of memory would be required. For horizontal resolutions higher than T63 a four-fold increase in memory would be required for a doubling of the resolution. In consequence, it is necessary to store intermediate values on work files, and to transfer data to and from memory as required using an input/output scheme.

Two types of work files are used - one containing symmetric and antisymmetric Fourier coefficients for variables associated with the dynamics of the model, the other containing grid point values used in the time-stepping and the

physics of the model. Since one set of symmetric and antisymmetric Fourier coefficients yields two latitude rows of grid point values (a northern hemisphere row, and a southern hemisphere row), a record structure based on latitude rows results in 96 grid point records and 48 Fourier coefficient records.

Scan 2 of the forecast time stepping procedure writes the 48 Fourier coefficient records.

In scan 1, each of these records is read in turn. First, a northern hemisphere row of grid points is processed. The corresponding record is read from the grid point work file; computations in grid point space are completed, then new values are written back to the grid point work file. Next, the corresponding southern hemisphere row is processed. Thus scan 1 requires 48 records of Fourier coefficients to be read, 96 grid point records to be read, and 96 grid point records to be written. This is carried out in a loop over the grid point records, Fourier coefficients being read on the first and each alternate pass through the loop.

For ease of restartability, and to allow the fastest available disc input/output scheme (which involves sequential rather than random input/output), two copies are maintained of each work file. Data is read from one file, and written back to the second file; at the end of a complete scan of a file, the file functions are swapped - the output file becomes the input file, and vice-versa.

Buffers for the data records are created within buffer defining routines, using the mechanisms described in 5.3 above. Two buffers are defined for each input/output process to enable asynchronous input/output requests to overlap the computational processes.

6. THE PRELIMINARY MULTI-TASKING T63 SPECTRAL MODEL

6.1 Introduction

The purpose of developing a preliminary multi-tasking version of the T63 spectral model was to examine the feasibility of multi-tasking, and to gain experience in multi-tasking with a view to continued development towards an efficient model. It was decided to approach the problem using a limited subset of the software features available, viz starting tasks, waiting for tasks to complete, and setting locks. This approach avoided constructs which would be difficult to test in a single processor environment, and reduced the planning excercise to

a) defining groups of routines that could excecute in parallel.

b) locating critical regions in such routines what would require locks to be set.

6.2 Organisation of the Computation

The first, obvious target for multi-tasking was the section in scan 1 dealing with the grid point computations. Provided sufficient grid point values could be retained in memory, it would be possible to compute a northern hemisphere row and a southern hemisphere row in parallel.

A second target concerned computations in Fourier space. In general these could be split into symmetric and antisymmetric parts, depending on the nature of the coefficients being updated. An exception to this broad division was found in the code relating to the updating of the symmetric and antisymmetric coefficients by contributions from northern and southern hemisphere grid point computations in routine SYM1 (called by FCC1 from scan 1). Splitting the computation into symmetric and antisymmetric parts was not simple, as both northern and southern rows make contributions to each. In consequence, the method chosen was to separate the real and

imaginary parts of the computations, since most variables were from the complex domain.

Figure 4 illustrates the modified structure for scan 2. The computation of the symmetric coefficients has been separated from that of the antisymmetric coefficients. Both computations may then be performed in parallel.

Figure 5 illustrates the modified structure of scan 1. It is augmented by Figure 6, which illustrates the division into tasks of FCC1. First, the initial computations to produce Fourier coefficients for northern and southern hemisphere latitude rows have been split into parallel tasks. Next, the grid point computations have also been split. FCC1 calls two routines, each of which can be regarded as suitable for parallel execution, but a synchronisation is required between them. Finally, the Legendre transformations may be executed in parallel.

6.3 Addressing Variables

The memory management package described in 5.3 above was modified to

a) enable arrays for local storage to be allocated from task dependent stacks.

b) include calls to set locks around critical regions where tables are updated.

This enables local work space arrays to be allocated to routines comprising tasks from different areas of memory for identical routines called within different tasks. Such routines use standard FORTRAN references to the arrays, once they are allocated.

All grid point arrays referenced in tasks where grid point computation is taking place for two rows in parallel are allocated memory manager codes which are modified by the task number. The root task is always task 0, while

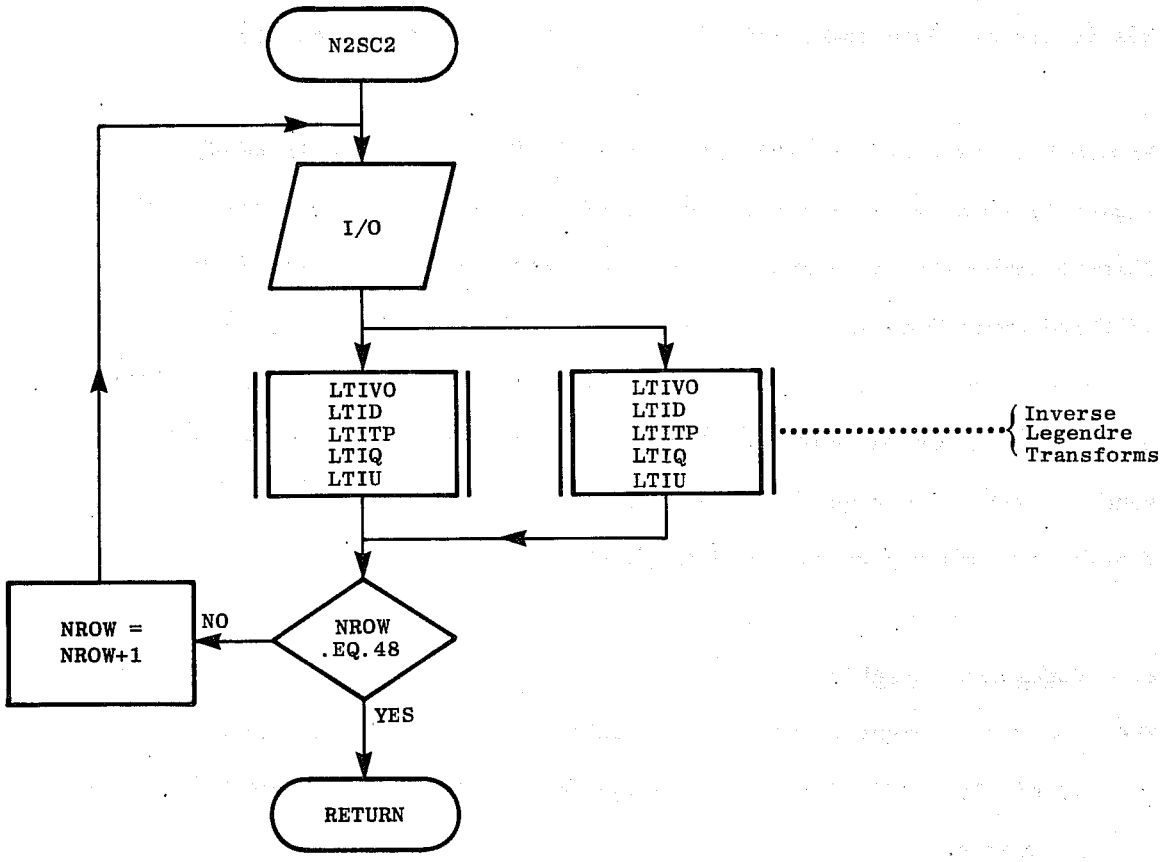

Fig. 4 ECMWF F/C - SCAN 2

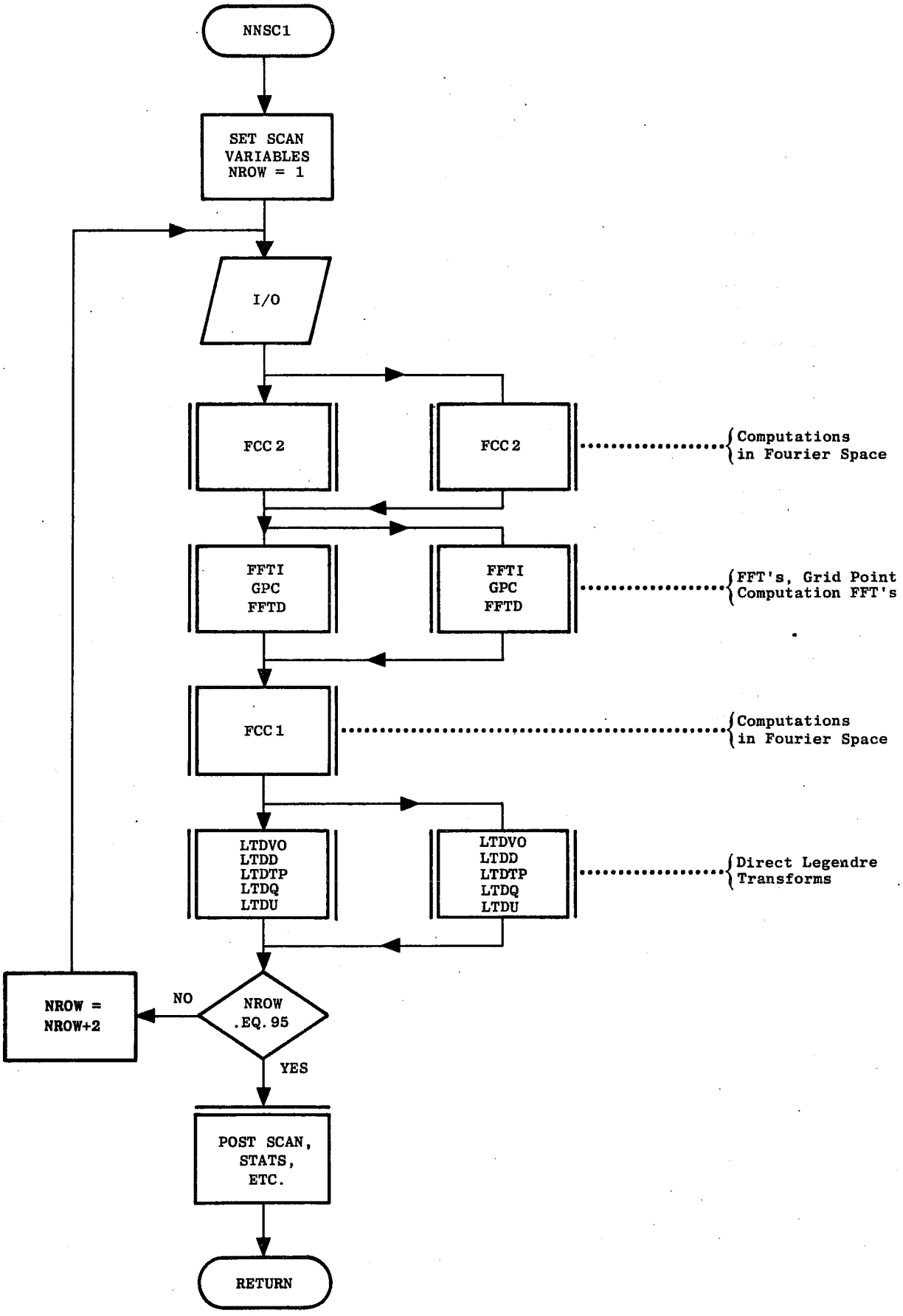

Fig. 5 ECMWF F/C - SCAN 1

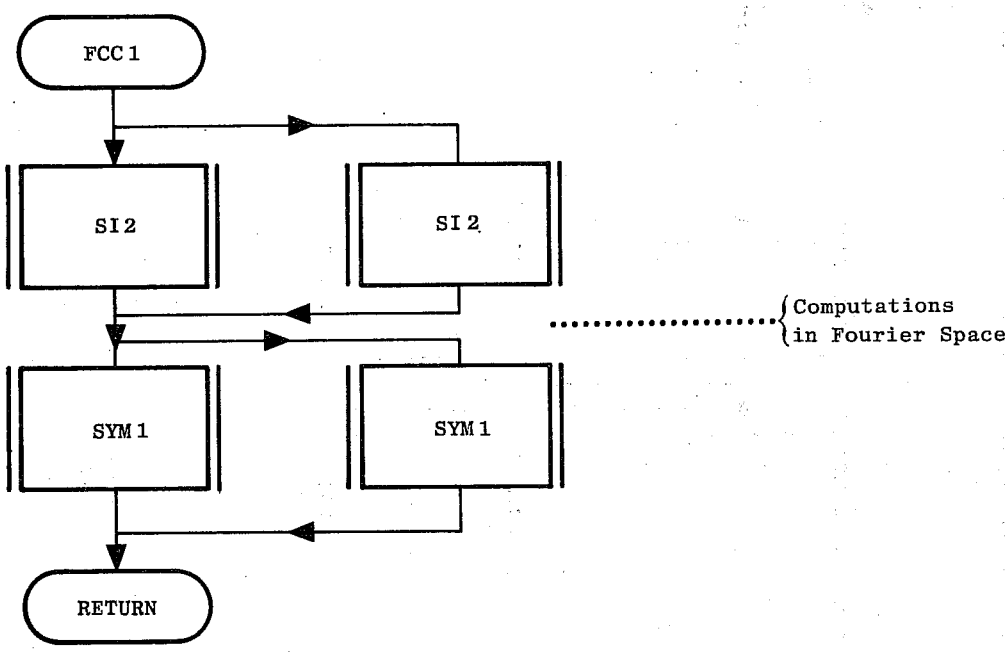

Fig. 6 ECMWF F/C - FCC 1

the spun-off task is task 1. If IGPTYPE is the code number usually given to grid point arrays, then a modified value of (IGPTYPE + task number) ensures that each task addresses appropriate grid point values.

It was necessary to check carefully for critical regions in the code. This was done by writing a cross-reference program to scan the source code. The Cray FORTRAN compiler, CFT, will produce an annotated cross-reference at the end of each routine indicating where variables are referenced, and which references involve storing re-defined values. By noting the calls to LOCKON and LOCKOFF within the source statement listings, it was possible to process these tables and construct new tables. If standard FORTRAN was used, only variables in COMMON blocks would be global, and these would be the only variables requiring attention. The POINTER extension of CFT enables additional variables to be global in scope, thus it was necessary to make provision for the inclusion of such additional variables, and to treat them in the same way as COMMON block variables. Due to the separation of variables brought about by the memory management strategy, only a few such additional variables were likely to be critical. The output from the cross-reference program indicates which potentially critical variables (COMMON block variables plus named additional variable) are stored, which are referenced, and in each case whether a lock is set. A table indicating the scope of all locks is also produced. This software tool proved invaluable in identifying sources of error during later development work.

6.4 Input/Output

To enable two rows of grid point values to be processed in parallel, two grid point records are read or written at a time. This would normally involve a doubling of the buffer space allocated to the grid point records. However, since the Cray X-MP target environment includes the provision of a solid state storage device, the input/output scheme was modified to support a

single buffered, synchronous, random access mode of operation. This results in a requirement for the same space for grid point buffers as the unmodified version, while savings in space required for Fourier coefficient buffers offsets the requirement for additional grid point arrays elsewhere in the code.

6.5 Results

The modified code was first tested exhaustively in single tasking mode on a Cray-1. It was then transported to Cray Research Incorporated, Mendotta Heights, Minneapolis and run on a Cray X-MP/22. Successful runs of a 10 day weather forecast were obtained in both single tasking and multi-tasking modes. Figure 7 indicates the timings obtained.

	CLOCK TIME	CPU TIME
SINGLE TASKING	2.00 HR	1.83 HR
MULTI-TASKING	1.25 HR	2.25 HR
SPEED UP	1.61	0.82

Figure 7 Multi-Tasking Results

Results from the multi-tasking version of the model were obtained which agreed with those obtained in single tasking mode, once all of the critical regions had been identified and locked.

7. CONCLUSIONS

7.1 Feasibility

The speed-up factor of 1.61, coupled with the ability to compute acceptable results indicate that multi-tasking is feasible for this application.

7.2 Efficiency

Only small areas of code were identified as critical and required locks to be set. Despite their small size, they were found to be extremely important in

obtaining correct results. Thus the main multi-tasking overhead consisted of calls to the task start and task wait routines. Scan 1 contained 48 x 5 calls to each of these routines per timestep, while scan 2 contained 48 calls. A 10 day forecast required 720 timesteps to be completed, giving a total of 207360 calls to each routine. Reduction of this overhead would increase efficiency and provide additional benefits from multi-tasking.

7.3 Continued Development

The preliminary multi-tasking version was deliberately split into more tasks than was necessary. This was done to aid development, and to enable sections of the code to be switched back to single tasking mode if necessary during initial multi-tasking tests on the Cray X-MP. With relatively little effort the 5 task start - task wait sequences in scan 1 could be reduced to 2. This would lead to a saving of half of the multi-tasking overhead, improving the speed-up factor to 1.8. One of the spectral routines takes sufficient time to benefit from splitting into two tasks. There are indications from ECMWF's factory trial tests that splitting the input/output over two tasks could be beneficial. Tests will be done to explore the possibility of synchronisation by posting events, which could prove less expensive than completing one task then setting off another. Finally, Cray Research have a good record for increasing the efficiency of their software over the course of time, and their multi-tasking software is relatively new. Taking all of these factors into account, it is anticipated that a final speed-up approaching 1.9 can be achieved.

References

Gibson, J.K. 1982: The DOCTOR system - a Documentary Oriented Programming System. ECMWF Tech.Memo. No.52, pp.17.

Gibson, J.K. 1983: A Simple Memory Manager. ECMWF Tech.Memo.No.73, pp.9.

Rigsbee, P. 1983: Multi tasking User's Guide. Cray Research Incorporated. (Pre-release version supplied by personal communication).