Aspects of Using Multiprocessors for Meteorological Modelling

by

G-R. Hoffmann,[1] P.N. Swarztrauber,[2] and R.A. Sweet[3]

1 INTRODUCTION

Since the introduction of computers for scientific research, meteorological modelling has always been one of the first disciplines to use the most advanced computers available at any given time. Today, meteorological institutions throughout the world operate class VI supercomputers like the CYBER 205, CRAY-1, or CRAY X-MP. Average processing speeds in excess of 200 Mflops (10^6 floating operations per second) are achievable for global weather prediction mo&els with a horizontal resolution of about 100 km in grid-point space. As the results of the operational meteorological models are only useful during a very short time--the forecast for today is history tomorrow--the demands for greater processing power of computers are obvious, especially when one considers that a doubling in horizontal resolution extends the forecast range but in turn increases the computing requirements by the factor $8 = 2^3$. The next logical step envisaged by meteorologists for the end of the decade is to double the model resolution without changing the elapsed time for the computation. Therefore, a computer capable of delivering around 1600 Mflops on average and in excess of 3000 Mflops as peak rate is required. This estimate of the computing speed assumes that the complexity of the calculations does not increase as well.

---

1 European Centre for Medium Range Weather Forecasts (ECMWF), Shinfield Park, Reading, Berks., RG2 9AX, England

2 National Center for Atmospheric Research (NCAR), Boulder, CO. 80307, USA

3 National Bureau of Standards (NBS), Boulder, CO. 80303, USA

Judging from the current trend in technology and assuming that no major breakthrough in physics like the step from valves to transistors occurs, it can be safely postulated that any future computer capable of achieving a performance in excess of 2.5 Gflops (10^9 floating operations per second) will employ multiple vector processors. Depending on the architecture used, the number of processors may vary from as few as possibly four to as many as thousands. However, currently known projects to build computers of the required performance seem to assume that the use of a large number of processors is still extremely difficult and only in the research phase, while the commercially undertaken studies for class VII computers only use between four and 256 processors.

In the following, the case of a very large number of processors will not be discussed: instead, emphasis will be given to three commercial developments which are based on currently available class VI machines, i.e., the CDC CYBER 205, the CRAY X-MP, and the Denelcor HEP-1. Rather detailed descriptions of the architecture of these machines may be found in many places, including [5,6], and they are therefore not repeated here. Reference will be made to special features only when we consider it necessary for explaining the expected structures of the follow-on products. However, we would like to stress that all qualitative or quantitative statements about the successors to the three computers discussed here are based on extrapolation only and certainly do not imply any commitment from the manufacturers. We also wish to note that we do not intend the exclusion of any class VII supercomputer project to reflect in any way on the validity of that project.

After discussing some general aspects of using multiprocessor systems and investigating the expected architecture of three future systems, we will consider algorithmic structures, and highlight possible problem areas. We will give the solution of a simple fluid dynamic model, and describe the experience of implementing it on the CRAY-1, the CYBER 205, and the HEP-1. We will detail the changes required by using multiprocessor configurations on the CRAY X-MP/2200 and the HEP-1. We will also show the influence of multiprocessor architecture for some algorithms commonly used in weather modelling.

2 ASPECTS OF MULTIPROCESSING

In the following, we make some remarks applicable to all multiprocessing environments. We emphasis aspects which are usually not relevant for SISD or SIMD architectures, but which affect users of multiprocessor systems (MIMD).

2.1 Parallelisation of code

If a program runs for T units of time on a single processor and is modified to use p processors in parallel ($p \geq 1$), then the percentage S of the total time T can be found during which the program will use only one processor. Using this figure, the theoretical maximum speedup SP of the program for using p processors can be expressed as:

$$SP = \frac{1}{S + \frac{(1-S)}{p}}$$

Fig. 1 shows the curves for p = 16, 12, 8, and 4.

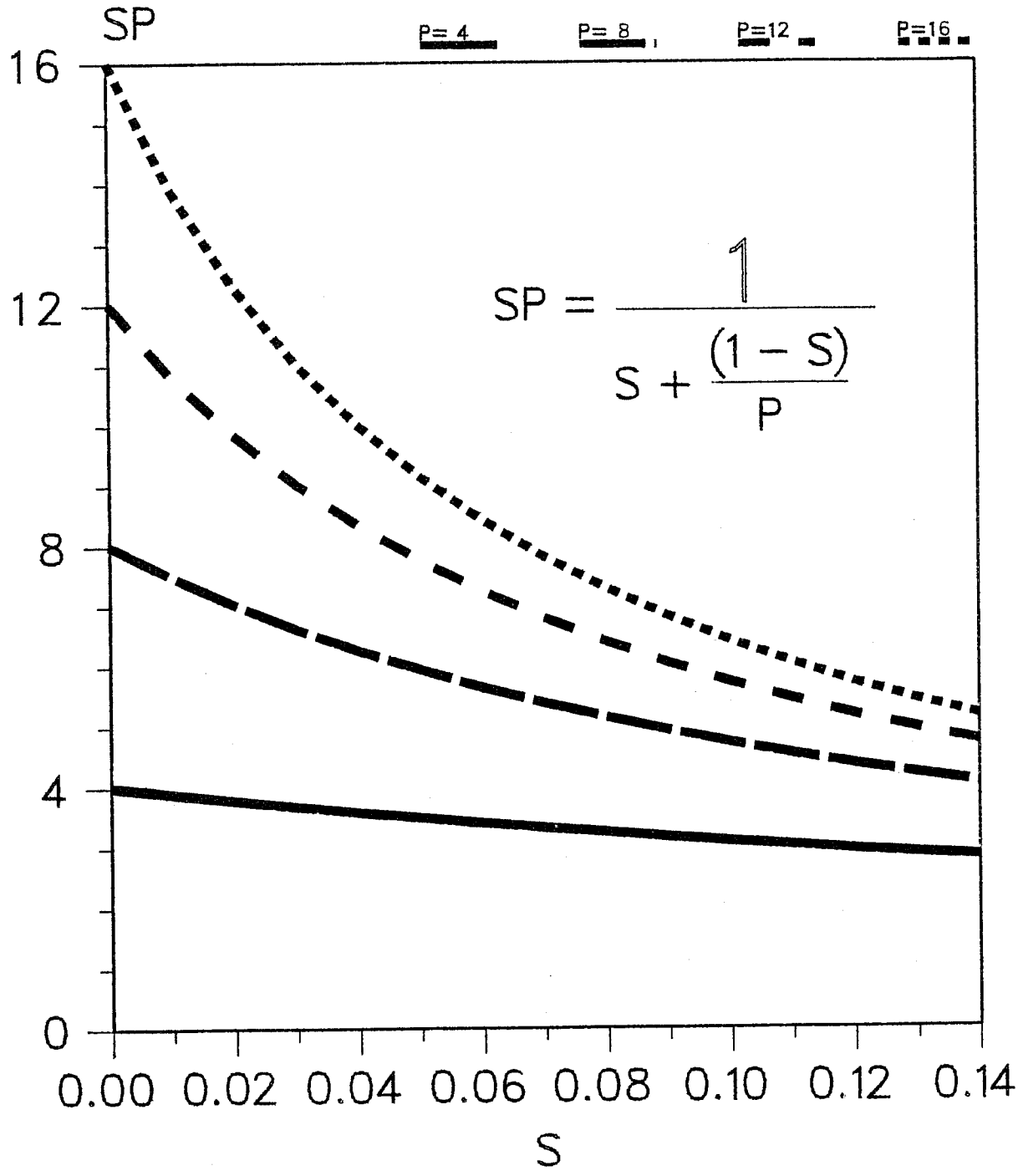


Fig. 1: Speedup when using p processors for varying percentages of sequential code (p = 16, 12, 8, 4)

In order to achieve an economically feasible CPU utilisation of at least 75%, S must be less than 0.022; i.e., the sequential code must take less than 2.2% of the total time T. If overheads are included in the calculation, the figure has to further decrease in order to achieve the required CPU utilisation; hence it becomes clear that either the code has to contain very few sequential passages or the configuration has to have sufficient resources to sustain a reasonable multiprogramming factor. For operational weather forecasting the turnaround time for a program plays the central role, and hence, the demand for code parallelisation becomes paramount. Indeed it may be easier to develop parallel algorithms for the sequential passages than to develop faster scalar processors.

2.2 Program decomposition

A program which uses multiple processors in parallel has to be split into processes which may be distributed across the processors. How a sequential program is decomposed into a number of parallel processes depends on the algorithm used and on the scheduling technique chosen. The algorithms and their inherent parallelism are described later. For scheduling, there exist two basic strategies: one assumes that the work of the program has been split into separate tasks which have been assigned statically to processes, while the other requires a queue of work units waiting to be processed and processes picking up work when they have finished with their previous tasks. The first strategy works best with a limited number of processes and a problem which can easily be split into identical work units.

If there is a large number of processes with different computational requirements that must all be completed before the solution of the problem can proceed, then some of the processors become idle, which decreases the

parallelisation of the code and thus the performance of the system. In these cases, it may be advisable to split the solution of the problem into a larger number of smaller work units to put these into a queue, and then to use processes which in a loop dequeue work units and process them. The number of processes in this case equals the number of processors. The granularity discussed below has to be considered as well. In addition, the decomposition of the program may interfere with its vectorising features; in particular, the average vector length may be reduced. In these circumstances, a detailed analysis of the effects of the decomposition is required and may lead to the choice of a different algorithm which allows a more suitable partitioning of the data.

2.3 Code granularity

In this paper we define the granularity G of a process as the time, in machine cycles, during which the process can continue without external or internal interruption. Usually an interruption is caused by the need to wait for another process to finish. There will be some overhead when the process begins and when it reaches the end of its uninterrupted spell. If we call the overhead in machine cycles at the beginning O_B and at the end O_E, if we assume $S = 0.02$, and if we take into consideration that O_B and O_E are, in most cases, spent in sequential processing mode, then it follows from the discussion in Section 2.1 that $G \geq 49(O_B + O_E)$; i.e., G must be about two orders of magnitude greater than $(O_B + O_E)$. If operating system calls are involved in either O_B or O_E, then the size of $O_B + O_E$ is easily $O(10^3)$, which implies that G must be $O(10^5)$.

2.4 Memory requirements

Each process active in a processor must have sufficient memory available to keep it busy during its granularity G. Subject to the algorithms used, this condition may imply that some hundreds of thousands of memory cells are to be accessible without interruption. As weather models seem to be very data-intensive, we assume in the following that a process of granularity G will use O(G) memory cells during its computation. If the memory available to each process is not big enough to hold all the data for the problem to be solved, then sufficient space must be available for a read, write and active process. In this case, however, the granularity of the active process has to increase substantially because the time in machine cycles for a read/write process involving 10^5 memory cells is $O(10^5)$ to $O(10^7)$ depending on the storage medium used, and this time influences the size of $(O_B + O_E)$. Taking into consideration that the data requirements for high-resolution weather models are on the order of 10^8 memory cells, then we can deduce that the minimum memory configuration accessible per process should be on the order of 10^6 to 10^8. In addition, one can easily see that the read/write process must be able to transfer one memory cell per machine cycle because of the granularity consideration. As future supercomputers will have cycle times between 1 and 4 nanoseconds with 64 bits per memory cell, the I/O device has to support transfer rates of 16 to 64 Gbits per second. This requirement can currently only be met with solid-state memory devices. The memory bandwidth in this case must be sufficient to allow at least three simultaneous accesses to memory per machine cycle and per processor.

2.5 Synchronisation tools

The tools provided by the manufacturer of a multiprocessor supercomputer for the mutual exclusion and synchronisation of programs must fit into the framework of program granularity. The method of implementing these tools directly influences the granularity of the programs using them, because O_B and O_E are determined by the time it takes to synchronise two tasks or to use locks. Which particular method, e.g., semaphores, monitors, etc., is chosen by a manufacturer to allow synchronisation by user programs affects programming ease for the user but does not have any other impact.

2.6 Process communication

It is mainly the algorithm which determines how often and in which way the processes of a program running in a multiprocessor environment have to communicate. As is shown later, there often exists a wide spectrum of algorithms with different attributes to solve a particular problem. The required granularity of the processes and the memory architecture of the target machine determine the right choice.

For current configurations with relatively few processors this problem is only occasionally evident as memory bank conflicts or processor synchronisation. However, as the number of processors increases, the communication time between processors or between processors and memory could dominate the computational time. This remains a concern even for computations that are 100% parallel.

It is clear that future computers can be designed with a very large number of processors. What is not so clear is how to interconnect and control the processors. The goal of the interconnection scheme is to minimize communication time. For a particular problem one can design a special-purpose computer with interconnections that match the computational communication paths. However, communication algorithms and processor interconnection are not well defined for a general-purpose computer.

Future general-purpose multiprocessors should be configured in such a way as to guarantee parallel nonblocking communication. Without this capability a multiprocessor is in effect reduced to a serial computer for problems in which any part of the communication is blocked.

3 FUTURE SYSTEMS

A number of computer manufacturers are currently engaged in designing supercomputers with performance in excess of 1 Gflops. In the following, we attempt to examine the proposed architecture of three of these machines. The already available class VI machines of the same manufacturers provide the basis for the expected structures.

3.1 CYBER 205 successor

As a spin-off company of Control Data Corporation, ETA Systems has got the brief to develop a successor to the CYBER 205. This machine is expected to become available in 1986 and is called ETA^{10}. Its performance will lie in the region of 10 Gflops and will be achieved by liquid-nitrogen-cooled multiple processors. Each of the processors will be similar to the CYBER 205; in particular, the dual pipeline structure and the paged memory are expected to be continued. All processors will have local memories, but will also share a big global memory and some kind of synchronisation buffer. The expected architecture of the machine is depicted in Fig. 2.

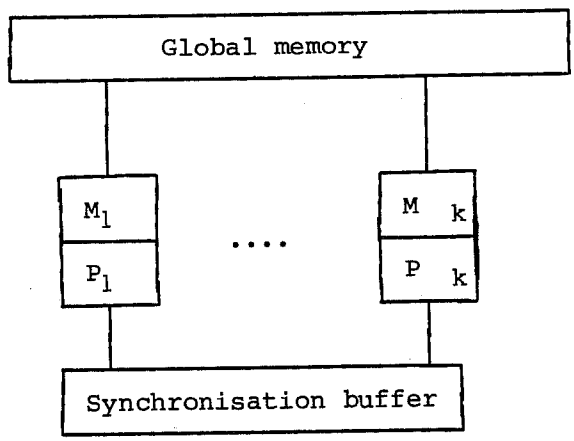


Fig. 2: Expected structure of CYBER 205 successor with k ($k \geq 1$) processors

We can be assume that the memory sizes both--local and global--will be sufficiently large to fulfill the requirements outlined in Section 2.4 above. However, because of the memory hierarchy, read and write processes will have to be provided, possibly by using built-in paging routines. The optimum way to use such an architecture seems to be the following:

(i) Acquire exclusive access to a set of pages in global memory which require changing.

(ii) Move data to local memory.

(iii) Process data.

(iv) Move processed data back to global memory.

(v) Continue with (i) until algorithm finishes.

Of course, steps (ii) to (iv) can be carried out in parallel if sufficient local memory is available. Synchronisation will be required for step (i) only, but may be quite time-consuming if the order of access to pages is not well planned in advance and the termination of a number of processors has to be awaited. In addition, the choice of the most suitable algorithm will have to be considered very carefully, because data changed within one process should occupy as few pages as possible in order to minimise the set of pages being exclusively kept by any one process. Because of the parallel read/write processes, the granularity of the active process will have to be rather large. This, however, will probably be advisable anyway, because the best vector performance of the machine can be expected with long vectors, as is the case with the CYBER 205.

3.2 CRAY X-MP successor

It can be expected that the development which led to a four-processor CRAY X-MP after progressing from a CRAY-1 to a two-processor CRAY X-MP will continue for some time in the future. The architecture of such a multiprocessor CRAY X-MP can be seen in Fig. 3.

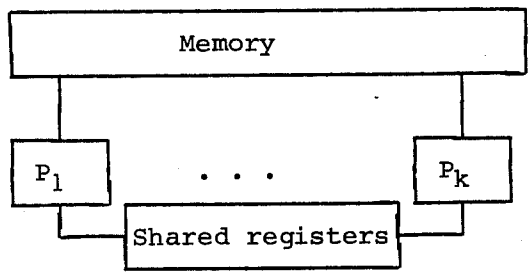


Fig. 3: Expected structure of CRAY X-MP successor with k ($k \geq 1$) processors

The main difference from the CYBER 205 successor will be the absence of local memory if one does not count the CRAY machine registers as such. Furthermore, the current addressing scheme of the CRAY prevents very large memories, which necessitates the addition of a large solid-state storage device (SSD) to the configuration. The SSD acts as a fast I/O device with a transfer speed meeting the requirements outlined in Section 2.4 above, but nevertheless requiring read/write processes. In addition, k processors will simultaneously access the global memory within vector instructions. The demands on the bandwidth of the memory and the requirement to avoid memory bank conflicts will be major problems. It is conceivable that the memory access patterns of different processes will have to be coordinated in order to avoid excessive memory wait times. A detailed study of the memory structure of the CRAY X-MP can found in [3].

Another area of concern will be the absence of any individual memory protection for the processes of one program. All the memory will be accessible to any process and the only safeguard against improper access will be faultless programming. The reason for this rather unsatisfactory behaviour of the CRAY X-MP is the fact that only one pair of base address and field length is used for all processes of one program. Experience has already shown that debugging a multiprocessor version of such a program becomes an extremely tedious task.

The granularity of the processes can vary widely without affecting performance because it is possible either to use high-level synchronisation tools or to access the shared registers directly, resulting in a very short synchronisation time.

3.3 HEP-1 successor

The main development of the HEP-1 system can be expected to be an increase in performance of each processing element (PEM). This may be achieved by using more advanced technology and possibly by adding some kind of vector capabilities. The structure of the new system can be seen in Fig. 4.

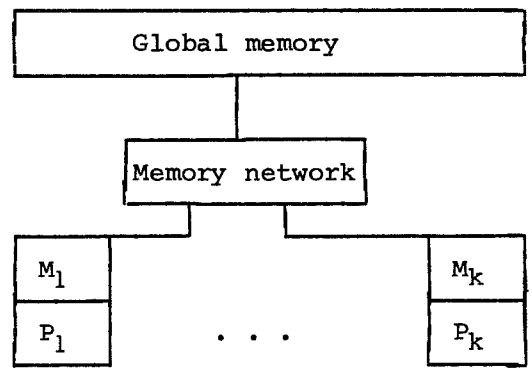


Fig. 4: Expected structure of HEP-1 successor with k(k ≥ 1) PEMs

One of the major differences between the HEP architecture and those of the previously discussed systems is the existence of an instruction pipeline within each PEM which contains instructions for all the active processes. Each PEM, therefore, is able to be looked upon as a multiprocessor system of its own. From the example discussed later it follows that currently about 20 processes are required to use a single PEM efficiently for a weather model. The local memory of a PEM has therefore to be large enough to allow sufficient active processes. For the successor of the HEP-1, the number of active processes per PEM may even be larger because the accesses to global memory may become a bottleneck if the speed of the memory network is not increased in line with the speed of the individual PEMs, especially if vector capabilities are added. We expect that in excess of 200 separate processes will be required to achieve the envisaged performance of the HEP system for weather modelling applications. Since the synchronisation tools are easy to use and applicable without large overheads, processes of relatively small granularity may be used. However, the vector capabilities may enforce a lower limit on the granularity. Nevertheless, the large number of active processes will probably require the self-scheduling programming technique outlined in Section 2.2. The algorithms for implementation on the HEP system will have to support such a scheme without too much overhead.

4 THE SHALLOW-WATER MODEL

In this section we describe a simple atmospheric dynamics model based on what are known as the shallow-water equations. The model and the computational methods are given in detail in [8]. The solutions to these equations are intended to provide rough estimates of the performance of more complex atmospheric models and to identify areas in which future computer designs might be improved in order to enhance performance. The shallow-water equations are accepted as a primitive but useful model of the dynamics of the atmosphere, particularly since we are concerned, not so much with the physical validity of the model, as with having the computations somewhat representative of those found in atmospheric models. The solution of this simple model adapts well to current supercomputer designs, which is not too surprising since the designers of these computers are well aware of the importance of computers to atmospheric science.

The equations will be solved in Cartesian coordinates on the rectancular domain $a \leq x \leq b$ and $c \leq y \leq d$. The shallow-water equations without the Coriolis term can be written in the form

$$\frac{\partial u}{\partial t} - \zeta v + \frac{\partial H}{\partial x} = 0 \qquad (4.1)$$

$$\frac{\partial v}{\partial t} + \zeta u + \frac{\partial H}{\partial y} = 0 \qquad (4.2)$$

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x}(Pu) + \frac{\partial}{\partial y}(Pu) = 0 \qquad (4.3)$$

where u and v are velocities in the x and y directions, respectively; P is density or pressure; and ζ is vorticity, given by

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \qquad (4.4)$$

A quantity H which is related to the height field is defined as

$$H = P + \tfrac{1}{2}(u^2 + v^2) \qquad (4.5)$$

If, in addition, we define the mass fluxes

$$U = Pu \tag{4.6}$$

$$V = Pv \tag{4.7}$$

and potential velocity

$$Z = \frac{\zeta}{P} \tag{4.8}$$

then (4.1), (4.2), and (4.3) take the form

$$\frac{\partial u}{\partial t} - ZV + \frac{\partial H}{\partial x} = 0 \tag{4.9}$$

$$\frac{\partial v}{\partial t} + ZU + \frac{\partial H}{\partial y} = 0 \tag{4.10}$$

$$\frac{\partial P}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \tag{4.11}$$

In order to facilitate presentation and implementation, we specify periodic boundary conditions, in which the variables u, v, U, V, and P satisfy $f(x+b,y) = f(x+a,y)$ and $f(x,y+d) = f(x,y+c)$. We wish to obtain an approximate solution to these equations on the rectangle $a<x<b$ and $c<y<d$. To this end we select integers M and N and define the grid

$$x_i = i\Delta x+a \qquad i = 0,\tfrac{1}{2},1,..M+1$$

$$y_j = j\Delta y+b \qquad j = 0,\tfrac{1}{2},1,..N+1$$

where $\Delta x=(b-a)/(M+1)$ and $\Delta y=(d-c)/(N+1)$. A staggered grid is used, with the positions of the dependent variables given in Fig. 5 below.

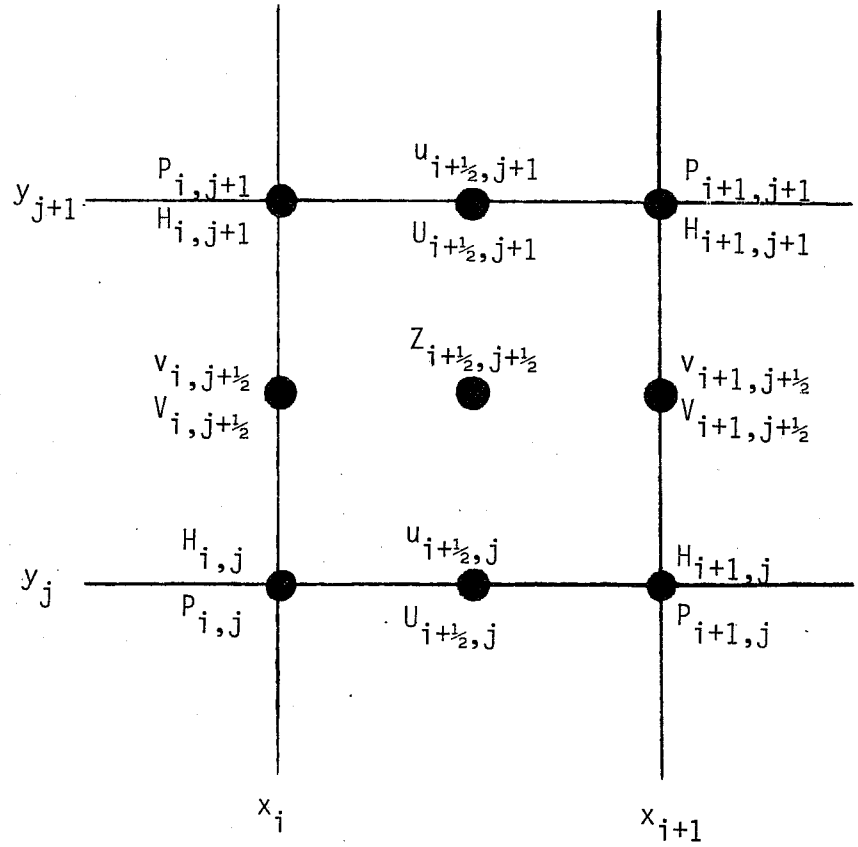


Figure 5

The variables P and H have integer subscripts, Z has half subscripts, u has half and integer subscripts, and v has integer and half subscripts. In order to obtain an approximate solution of (4.9), (4.10) and (4.11) in the form of a tabulation, the dependent variables are required to satisfy the following finite-difference approximations of (4.5) through (4.11):

$$U_{i+\frac{1}{2},j} = \tfrac{1}{2}(P_{i+1,j}+P_{i,j})u_{i+\frac{1}{2},j} \tag{4.12}$$

$$V_{i,j+\frac{1}{2}} = \tfrac{1}{2}(P_{i,j+1}+ P_{i,j})v_{i,j+\frac{1}{2}} \tag{4.13}$$

$$Z_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{\left[(v_{i+1,j+\frac{1}{2}}-v_{i,j+\frac{1}{2}})/\Delta x-(u_{i+\frac{1}{2},j+1}-u_{i+\frac{1}{2},j})/\Delta y\right]}{1/4(P_{i,j}+ P_{i+1,j}+ P_{i+1,j+1}+ P_{i,j+1})} \tag{4.14}$$

$$H_{i,j} = P_{i,j}+ \tfrac{1}{2}\left[\frac{(u^2_{i+\frac{1}{2},j}+ u^2_{i-\frac{1}{2},j})}{2} + \frac{(v^2_{i,j+\frac{1}{2}}+ v^2_{i,j-\frac{1}{2}})}{2}\right] \tag{4.15}$$

$$\begin{aligned}\frac{u^{(n+1)}_{i+\frac{1}{2},j} - u^{(n-1)}_{i+\frac{1}{2},j}}{2\Delta t} &= \tfrac{1}{2}(Z_{i+\frac{1}{2},j+\frac{1}{2}}+ Z_{i+\frac{1}{2},j-\frac{1}{2}})x\\ &\quad 1/4(V_{i+1,j+\frac{1}{2}}+ V_{i,j+\frac{1}{2}}+ V_{i+1,j-\frac{1}{2}}+ V_{i,j-\frac{1}{2}})\\ &\quad -(H_{i+1,j}- H_{i,j})/\Delta x\end{aligned} \tag{4.16}$$

$$\begin{aligned}\frac{v^{(n+1)}_{i,j+\frac{1}{2}} - v^{(n-1)}_{i,j+\frac{1}{2}}}{2\Delta t} &= -\tfrac{1}{2}(Z_{i+\frac{1}{2},j+\frac{1}{2}}+ Z_{i-\frac{1}{2},j+\frac{1}{2}})x\\ &\quad 1/4(U_{i-\frac{1}{2},j+1}+ U_{i-\frac{1}{2},j}+ U_{i+\frac{1}{2},j+1}+ U_{i+\frac{1}{2},j})\\ &\quad -(H_{i,j+1} - H_{i,j})/\Delta y\end{aligned} \tag{4.17}$$

$$\begin{aligned}\frac{P^{(n+1)}_{i,j} - P^{(n-1)}_{i,j}}{2\Delta t} &= -(U_{i+\frac{1}{2},j}- U_{i-\frac{1}{2},j})/\Delta x\\ &\quad -(V_{i,j+\frac{1}{2}}- V_{i,j-\frac{1}{2}})/\Delta y\end{aligned} \tag{4.18}$$

The leapfrog scheme is used to derive (4.16), (4.17), and (4.18). The superscripts on the left side of these equations designate the time level. All quantities on the right side are evaluated at $t_n = n\Delta t$ where Δt must satisfy the stability criterion $H^{\frac{1}{2}}\frac{\Delta t}{\Delta x} < 1$.

The initial velocity fields u(x,y,0) and v(x,y,0) will be selected so that the balance conditions

$$\text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{4.19}$$

and

$$\frac{\partial}{\partial t} \text{div } \mathbf{v} = 0 \tag{4.20}$$

are satisfied. To this end we define a stream function

$$\Psi = A \sin \frac{2\pi x}{b-a} \sin \frac{2\pi y}{d-c} \tag{4.21}$$

and set

$$u(x,y,0) = -\frac{\partial \Psi}{\partial y} \text{ and } v(x,y,0) = \frac{\partial \Psi}{\partial x} \tag{4.22}$$

It is clear that these initial values will satisfy (4.19).

From (4.9) and (4.10)

$$\frac{\partial}{\partial t} \text{div } \mathbf{v} = \frac{\partial}{\partial x}(ZV) - \frac{\partial}{\partial y}(ZV) - \nabla^2 H \tag{4.23}$$

Substituting (4.4) through (4.8) into (4.23), we obtain

$$\frac{\partial}{\partial t} \text{div } \mathbf{v} = 2\left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\frac{\partial v}{\partial x}\right) - \nabla^2 P \tag{4.24}$$

In order to satisfy the balance equation (4.20), we set (4.24) equal to zero and solve for P. Substituting (4.20), (4.21), and (4.22) into (4.24), we obtain

$$\nabla^2 P = -\frac{16\pi^4 A^2}{(b-a)^2(d-c)^2}\left(\cos \frac{4\pi x}{b-a} + \cos \frac{4\pi y}{d-c}\right) \tag{4.25}$$

Integrating we obtain

$$P = \frac{A^2}{4}\left[\left(\frac{2\pi}{d-c}\right)^2 \cos\frac{4\pi x}{b-a} + \left(\frac{2\pi}{b-a}\right)^2 \cos\frac{4\pi y}{d-c}\right] + P_0 \qquad (4.26)$$

where P_0 is any constant. Equation (4.26) is used to define the initial values of P at t=0. In practice, the finite-difference approximation of Poisson's equation is solved in order to initialise P.

The discrete forms of (4.21) and (4.22) are

$$\Psi_{i+\frac{1}{2},j+\frac{1}{2}} = A \sin\frac{2\pi x_{i+\frac{1}{2}}}{b-a} \sin\frac{2\pi y_{j+\frac{1}{2}}}{d-c} \qquad (4.27)$$

$$u_{i+\frac{1}{2},j} = -\frac{(\Psi_{i+\frac{1}{2},j+\frac{1}{2}} - \Psi_{i+\frac{1}{2},j-\frac{1}{2}})}{\Delta y} \qquad (4.28)$$

$$v_{i,j+\frac{1}{2}} = \frac{(\Psi_{i+\frac{1}{2},j+\frac{1}{2}} - \Psi_{i-\frac{1}{2},j+\frac{1}{2}})}{\Delta x} \qquad (4.29)$$

The leapfrog time differencing implied by (4.16), (4.17), and (4.18) requires values of the dependent variables u, v, and P at two levels, namely, $t_0 = 0$ and $t_1 = \Delta t$. The values at t_0 are provided by the initial data in (4.26) through (4.29). The values at t_1 are computed using one step of Euler's method

$$f^{(1)} = f^{(0)} + \frac{\partial f^{(0)}}{\partial t}\Delta t \qquad (4.30)$$

where $\partial f^{(0)}/\partial t$ for f = u, v, and P, is computed from (4.16), (4.17), and (4.18) evaluated at t_0. Finally, to handle the weak instability in the leapfrog scheme, we use the following time filter for f = u, v, and P.

$$F^{(n)} = f^{(n)} + \alpha(f^{(n+1)} - 2f^{(n)} + F^{(n-1)}) \qquad (4.31)$$

where α is the filter parameter.

The filtered values $F^{(n)}$ enter the computation in (4.16), (4.17), and (4.18) at time level t_{n-1}.

5 IMPLEMENTATION AND PERFORMANCE OF THE MODEL

In this section we examine the implementation and performance of the shallow water model on several supercomputers. The performance of a supercomputer is difficult to evaluate because it depends on many factors. In practice, maximum computational rates are rarely achieved, and a gap, which appears to be widening, exists between expected and possible performance. The extent of this gap depends both on the problem and on the user's ability to adapt the computation to a particular architecture. More responsibility has been placed on the user for the performance of the computer, which may be one to two orders of magnitude below maximum unless the computation is formulated in a way that takes advantage of the architectural features.

The computations in the model are all parallelisable or distributable, which resulted in relatively high computational rates on all of the computers. The remainder of this section is divided into four parts in which the results for each of the computers are presented. In addition to performance figures, each part includes a discussion about programming the model, which includes any modifications that were necessary in order to take advantage of the particular architecture.

5.1 The CRAY-1

The program for solving the shallow-water equations is listed in Appendix A. It was first written for the CRAY-1 and subsequently adapted to the other computers. The first version of the program contained about 120 lines of code and took about two hours to write using a screen editor. This version then took about four hours to debug. The final (second) version in Appendix A contains 242 lines of code and required an additional eight hours for both code development and debugging.

The program contains three main double loops that account for most of the computing time. The quantities $U_{i+\frac{1}{2}j}$, $V_{ij+\frac{1}{2}}$, $Z_{i+\frac{1}{2}j+\frac{1}{2}}$, and H_{ij} that are defined in (4.12) through (4.15) are computed in loops 100. Their FORTRAN names are CU(I+1,J), CV(I,J+1), Z(I+1,J+1), and H(I,J), respectively. The half subscripts $i+\frac{1}{2}$ or $j+\frac{1}{2}$ are converted to the FORTRAN subscripts I+1 or J+1. In general, for fractional indices, the FORTRAN subscripts are generated by adding one-half.

The quantities $U^{(n-1)}_{i+\frac{1}{2},j}$, $V^{(n+1)}_{i,j+\frac{1}{2}}$, and $P^{(n+1)}_{i,j}$ that are defined in (4.16, (4.17), and (4.18) are computed in loops 200. Their FORTRAN names are UNEW(I+1,J), VNEW(I,J+1), and PNEW(I,J), respectively. Time filtering and updating occur in loops 300. The variables at the three different time levels t_{n-1}, t_n, and t_{n+1} are stored in three separate two dimensional arrays. For example $U^{(n-1)}_{i+\frac{1}{2},j}$, $U^{(n)}_{i+\frac{1}{2},j}$, and $U^{(n+1)}_{i+\frac{1}{2},j}$ are stored in the arrays UOLD(I+1,J), U(I+1,J), and UNEW(I+1,J), respectively.

Problem and program parameters are initialized beginning with Δt = DT = 90 seconds. The variable TDT is set equal to Δt on the first pass where Euler's method is used to compute the values of u, v, and P at $t = t_1$ as discussed near (4.30). The variable TDT is then set equal to $2\Delta t$ on all subsequent cycles. Most of the parameters that are defined at the beginning of the program are self-explanatory, with the possible exception of ITMAX, which is the maximum number of time cycles, and MPRINT, which is the number of cycles between prints. The variable EL is set equal to both b - a and d - c and hence the domain is square.

The initial stream function $\psi_{i+\frac{1}{2},j+\frac{1}{2}}$ and the balanced $P_{i,j}$, as defined in (4.27) and (4.26), are computed in loops 50. Their FORTRAN equivalents are PSI(I+1,J+1) and P(I,J). The initial velocities, as defined in (4.28) and (4.29), are computed in loops 60. Periodic boundary conditions are applied throughout the code in loops 70, 75, 86, 110, 115, 210, 215, 320, and 325. Output is produced beginning with the cycle number and model time at the statement WRITE(6,350) NCYCLE,PTIME. Computational times in seconds are computed for loops 100, 200, and 300 and stored in the variables T100, T200, and T300, respectively. The megaflops for each of these loops are computed based on 24, 26, and 15 floating operations per loop and stored in the variables MFS100, MFS200, and MFS300, respectively. The computer times and megaflops are listed in Table 1. These results were obtained using the CRAY FORTRAN complier CFT 1.13.

TABLE 1

CRAY-1 performance on the shallow water model for a 64 x 64 grid

loop	time (ms)	Mflops
100	1.405	70.0
200	1.670	63.8
300	1.282	47.9

5.2 The CYBER-205

The Control Data Corporation CYBER 205 [1], unlike the CRAY-1, has no intermediate storage through which the vector arithmetic units receive operands and return results. Instead, the vector operands and results are streamed directly to and from main memory. Furthermore, on the CYBER 205 the elements of a vector must occupy consecutive locations in memory. In order to guarantee conflict-free memory access to any vectors during an operation a sophisticated microcoded algorithm is executed before each vector instruction. Consequently, there are rather large start-up times for vector operations. To reduce this cost one must arrange the calculations in such a way as to create the longest possible vectors.

For example, operations on modest-sized (on the order of 100 x 100) arrays may be done column- or row-wise on the CRAY-1 at near peak efficiency. On the CYBER 205 however, column operations alone (using vectors of length 100) yield a throughput on the order of 50% of the maximum, whereas if the operations can be performed over the entire array (working on vectors of length 10,000), the throughput will rise to about 99% of the maximum. All numbers related to speed or efficiency are based on the timing formula for vector addition and multiplication which are the fastest-executing. Other vector operations give slower speeds.

In order to increase vector lengths, bit (logical) vectors have been defined in the CYBER 205 extension to FORTRAN. Bit vectors may be used to control the storage of elements of result vectors and are absolutely required for efficiency when one is working over a subarray that is large compared to the

array which contains it. Continuing the example above, suppose that one wants to perform a calculation on a 99 x 99 sub-array of a 100 x 100 array. It is no longer possible to perform one vector calculation over the entire subarray because the elements of the vector are no longer contiguously located in memory. Without bit vectors the calculation would have to be performed over vectors of length 99 (at 50% efficiency). Defining a bit vector that causes every 100th result to not be stored will result in the ability to perform operations on vectors of length 99 x 100 = 9,900 (at 99% efficiency).

A two-pipe CYBER 205 has an asymptotic megaflop rating of 100 for memory-to-memory vector operations on full 64-bit words. However, it is possible to direct the output of one vector arithmetic unit into an input port of another vector arithmetic unit. With this linking process it is possible to double the asymptotic megaflop rating to 200.

The FORTRAN program SHALOW, described in the previous section and listed in Appendix A, was run on a two-pipe CYBER 205 located at Colorado State University in Fort Collins, Colorado. There were two versions run on the machine: (i) the original FORTRAN program listed in Appendix A, and (ii) a modified version in which the principal do loops were recoded using the special vector extensions to FORTRAN developed by CDC [2]. The modified version is listed in Appendix B.

For the first version, no changes were made to the original code. The vectorization and optimization options specified to the FORTRAN 2.1.5 compiler were O=UOV, which directed the compiler to fully optimize all scalar code and vectorize as much of it as it could automatically. Examination of the main

time-stepping loop (statements 99 through 400) revealed that the compiler vectorized the inner portion of the double loops 100, 200, 300, and 400 and the single loops 115, 215, and 325. The performance for the three main loops is given in Table 2.

For the modified version of the code all of the loops mentioned in the previous paragraph were recoded in special explicit vector notation. The double loops 100 and 200 could not be replaced by single vector operations because they only worked over the submatrices I = 1,2,...,M, J = 1,2...,N of the M+1 X N+1 matrices. A bit vector was defined that allowed those double loops to be replaced by single vector operations over the contiguous subarrays I = 1,2,...,M+1, J = 1,2,...,N. The resulting vector lengths were 4159 (=N*(M+1)-1) so we expect the vector portion of the code to be running at about 97% of its peak rate. The double loops 300 and 400 were replaced by vector operations over the entire M+1 X N+1 arrays. The conversion effort required about eight hours to study, code, and debug. The results of the customisation for the CYBER 205 are given in Table 3.

Comparison of the results in Tables 2 and 3 vividly demonstrate the benefit derived by operating on long vectors on the CYBER 205. An unfortunate side effect is the nonportability of the resulting FORTRAN program.

TABLE 2

CYBER 205 performance on the standard FORTRAN version of the shallow-water model for a 64 x 64 grid.

loop	time (ms)	Mflops
100	2.66	37.0
200	2.53	42.1
300	1.60	38.4

TABLE 3

CYBER 205 performance on the customized FORTRAN version of the shallow-water model for a 64 x 64 grid.

loop	time (ms)	Mflops
100	1.08	92.6
200	0.861	126
300	0.526	119

5.3 The Denelcor HEP-1

Shallowp, a parallel version of the shallow-water equations allowing for an arbitrary number of parallel processes, written from the original CRAY-1 version, is listed in Appendix C. In the parallel version each of the primary loops 100, 200, and 300 were put into subroutines, as were the periodic continuation loops. Each of these routines is called in turn from a subroutine called PARSUB. Each loop routine has two inputs: iproc = the process number and nproc = total number of processes. In each case the outer loop is split across the processes by the technique called prescheduling. The loop has the form:

```
      DO 100 I = iproc, N, nproc
         .
         .
         .
  100 CONTINUE
```

For example, if there were two processes the first would perform all odd rows and the second would perform all even rows. The value of nproc is input at run time. In PARSUB a call to barrier is made between each of the various routines for the purpose of synchronising the processes. As each process enters the barrier it is terminated, until the last process enters, at which time all processes are recreated and processing resumes. Using this technique a larger problem could be run on a four-to eight-PEM system with 128-256 parallel processes without modifying the program.

These changes required about 40 minutes of design and editing time with a full screen editor. The program ran on the first trial, but no correct answers were available at the time for comparisons. The program has since been debugged.

The program was run on System-0 at Denelcor's home offices in Aurora, Colorado, under the company's UNIX™-based operating system HEP/UPX. System-0 is a minimum 1-PEM configuration. Since the Fortran 77 optimizer was unavailable, we felt it only fair for the purposes of comparison that some simple optimisation should be performed by hand. David Snelling, analyst for Denelcor, adapted the compiler output in such a way as to simulate the loop invariant removal function of the optimiser. No other optimisation was done. The Mflop rates for loops 100, 200, and 300 are 3.9, 3.5, and 3.9, respectively.

System-0 at Denelcor was configured with an Eulerian switch, in which each message visits all ports on the network rather than taking the shortest route. The memory latency characteristics created by this routing simulate those of an eight to ten-PEM system, and hence 20 processes were required to achieve full machine utilisation instead of the 13 processes normally required on a single PEM.

5.4 The CRAY X-MP/2

The CRAY X-MP/2 has two vector processors, each of which is similar to the CRAY-1, except for the following major differences:

(i) The clock period has been reduced from 12.5 to 9.5 nanoseconds.

(ii) The ports to memory have been increased to four per processor, which permits two vector loads, a store, and an I/O transfer to occur simultaneously.

(iii) The solid-state disk (SSD), which is available, can be written at a rate of about one word per CPU cycle.

The two CPUs operate in a multitask mode. The user generates tasks which are then scheduled on the processors by the operating system. The user generates a task either as a main program or by executing a CALL TSKSTART statement. In effect, this calls a subroutine which can be executed at the same time as the main program. However, unless the computer is dedicated to a particular user it is likely that the two processors will be executing the programs of two different users.

At certain times within the user's program it may be necessary to wait for a set of tasks to be completed before starting new ones. This can either be achieved by a CALL TSKWAIT statement or by synchronisation using the event statements, CALL EVPOST, CALL EVWAIT, and CALL EVCLEAR. The subroutine EVPOST posts an event, EVWAIT waits for an event to occur in the other processor, and EVCLEAR clears the event, which can then be reused. A detailed description of the use of these subprograms is contained in the multitasking guide [4] available from Cray Research. An overview of multitasking is given in [7].

Two multiprocessor programs for the sample model were written and run on a dedicated CRAY X-MP/2. The first program is listed in Appendix D and uses TSKSTART and TSKWAIT subroutines in order to synchronise the processors. Six tasks are defined for each time step, two for each of the three main loops, namely, 100, 200, and 300. These loops were extracted from the main program and made into three subroutines. Each of the three subroutines was called with different parameters which define the region of computation for each processor.

For example, loop 100 has been replaced by the statements

```
CALL TSKSTART(I100A,L100,M,1,N/2)
CALL TSKSTART(I100B,L100,M,N/2+1,N)
CALL TSKWAIT(I100A)
CALL TSKWAIT(I100B)
```

The first argument, I100A, is called the task array and is used by the system when a task is defined. The second argument, L100, is the subroutine that performs the task, and the remaining arguments are used by the subroutine L100. The first TSKSTART statement initiates the computations in loop 100 that correspond to the values J=1,N/2. The second TSKSTART statement initiates the computations that correspond to the remaining values J=N/2+1,N. The tasks are synchronised prior to the periodic continuation by the CALL TSKWAIT statements. This procedure is repeated for loops 200 and 300. The performance of this program is presented in Table 5. For the purpose of comparison, the performance of the single-processor code in Appendix A on a single CRAY X-MP/2 processor is listed in Table 4.

TABLE 4

Performance on single-processor CRAY X-MP of shallow-water model on a 64 x 64 grid

loop	time (ms)	mflops
100	0.963	102
200	1.10	97
300	0.653	94

TABLE 5

CRAY X-MP/2 performance on "tasked" version of shallow water model in Appendix D on a 64 x 64 grid

loop	time (ms)	Mflops	speedup
100	0.559	176	1.73
200	0.616	173	1.78
300	0.407	151	1.61

The second multiprocessor code is listed in Appendix E. It is synchronised using events. The program was developed by duplicating loops 100, 200, and 300 and including them in a subroutine called TASK, which is called only once. The loops are synchronised with those in the main program by using EVPOST, EVWAIT, and EVCLEAR subroutines.

The performance of the "events" program is given in Table 6. As suggested by Larson [7] and others, its performance is superior to the "tasks" program in Table 5. The reason is simply that the TSKSTART subprogram is time-consuming compared to the events program. Of course the results in Tables 5 and 6 would be closer if the grid size were increased.

TABLE 6

CRAY X-MP/2 performance on "events" version of the shallow-water model in Appendix E on a 64 x 64 grid

loop	time (ms)	mflops	speedup
100	0.504	195	1.91
200	0.563	189	1.95
300	0.351	175	1.86

The program in Appendix D that uses tasks was both easier to write and easier to debug. The total time required was about eight hours using a line editor. The program in Appendix E took about the same time, but that was only because the author (Swarztrauber) had developed more knowledge about both the editor and multitasking.

6 THE DISTRIBUTED SOLUTION OF POISSON'S EQUATION

In this section we describe the solution of Poisson's equation in a distributed processing environment. Although Poisson's equation does not appear in the sample model, it is nevertheless solved in a number of atmospheric applications and is therefore deserving of some discussion. As we will see, the distributed solution of Poisson's equation is not quite as straightforward as the solution of the sample model. On the other hand, the effort that is required to solve Poisson's equation is likely more representative of the effort that is required to distribute the computations in a more realistic and complex model. We will discuss the solution of Poisson's equation on a computer with p processors, where p is a relatively small number. We will also assume that the processors share memory and that each processor is pipelined. These two assumptions significantly influence the method of solution. A different approach can be taken if memory is not shared [10].

We wish to solve Poisson's equation:

$$\Delta u = g \tag{6.1}$$

on a rectangle $a \leq x \leq b$ and $c \leq y \leq d$ and subject to an appropriate boundary condition. We assume that $u = 0$ is specified on the boundary; however, the results are applicable to any of the standard boundary conditions [11,12]. More precisely, we will obtain an approximate solution of (6.1) in the form of a tabulation. Given the integers M and N, a grid (x_i,y_j) is defined by $x_i=a+i\Delta x$ and $y_i=c+j\Delta y$, where $\Delta x=(b-a)/(M+1)$ and $\Delta y=(d-c)/(N+1)$. We seek a tabulation $u_{i,j}$ that approximates the exact solution $u(x_i,y_j)$ at each of the grid points. To this end we require $u_{i,j}$ to satisfy the following finite-difference approximation of (6.1).

$$\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{\Delta y^2} = g_{i,j} \tag{6.2}$$

where $g_{i,j} = g(x_i,y_j)$. A convenient notation is obtained by introducing the vectors

$$u_j^T = (u_{1,j}, u_{2,j}, \ldots, u_{M,j}) \text{ and } g_j^T = \Delta y^2 (g_{1,j}, g_{2,j}, \ldots, g_{M,j}).$$

Then (6.2) can be written:

$$u_{j-1}+Au_j+u_{j+1} = g_j \quad j=1,\ldots,N \tag{6.3}$$

where

$$A = \begin{bmatrix} -2\alpha & 1 & & & & \\ 1 & -2\alpha & 1 & & & \\ & 1 & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & 1 \\ & & & & 1 & -2\alpha \end{bmatrix} \tag{6.4}$$

and $\alpha = \left(\frac{\Delta y}{\Delta x}^2\right) + 1$. With no loss of generality we can assume that $\Delta x = \Delta y$.

There are a number of methods for solving the system (6.3), which may be equally attractive, particularly in a time-dependent setting. In this paper we investigate the Fourier method, which is known to take only 5 to 10% of the total computer time in current atmospheric models.

Let Q be the orthogonal matrix whose columns are the eigenvectors of A; then

$$q_{i,j} = \sqrt{\frac{2}{M+1}} \sin ij\frac{\pi}{M+1} \tag{6.5}$$

and $Q^T A Q = \text{diag}\,(\lambda_1 \ldots, \lambda_M)$, where

$$\lambda_k = 2\left[\sin^2 k\frac{\pi}{2(M+1)} - \left(\frac{\Delta y}{\Delta x}\right)^2\right] \qquad k=1,\ldots,M \tag{6.6}$$

If we define $\hat{u}_j = Q^T u_j$ and compute $\hat{g}_j = Q^T g_j$, then, substituting into (6.2), we obtain

$$\hat{u}_{j-1} + Q^T A Q \hat{u}_j + \hat{u}_{j+1} = g_j \tag{6.7}$$

Let $\hat{u}_{k,j}$ and $\hat{g}_{k,j}$ denote the kth component of $\hat{u}_j$ and $\hat{g}_j$, respectively; then for each $k=1,\ldots M+1$ we obtain a tridiagonal system of order N.

$$\hat{u}_{k,j-1} + \lambda_k \hat{u}_{k,j} + \hat{u}_{k,j+1} = \hat{g}_{k,j} \quad j=1,\ldots N \tag{6.8}$$

The Fourier method can now be summarized.

(i) Compute $\hat{g}_j = Q^T g_j$ using the FFT.

(ii) Solve the tridiagonal systems (6.8) for $\hat{u}_{k,j}$ and hence $\hat{u}_j$.

(iii) Compute the solution $u_j = Q\hat{u}_j$ again using the FFT.

Note that the Fourier transforms are computed in the x (or i) direction, whereas the tridiagonal systems are solved in the y (or j) direction, as depicted in Figure 6.

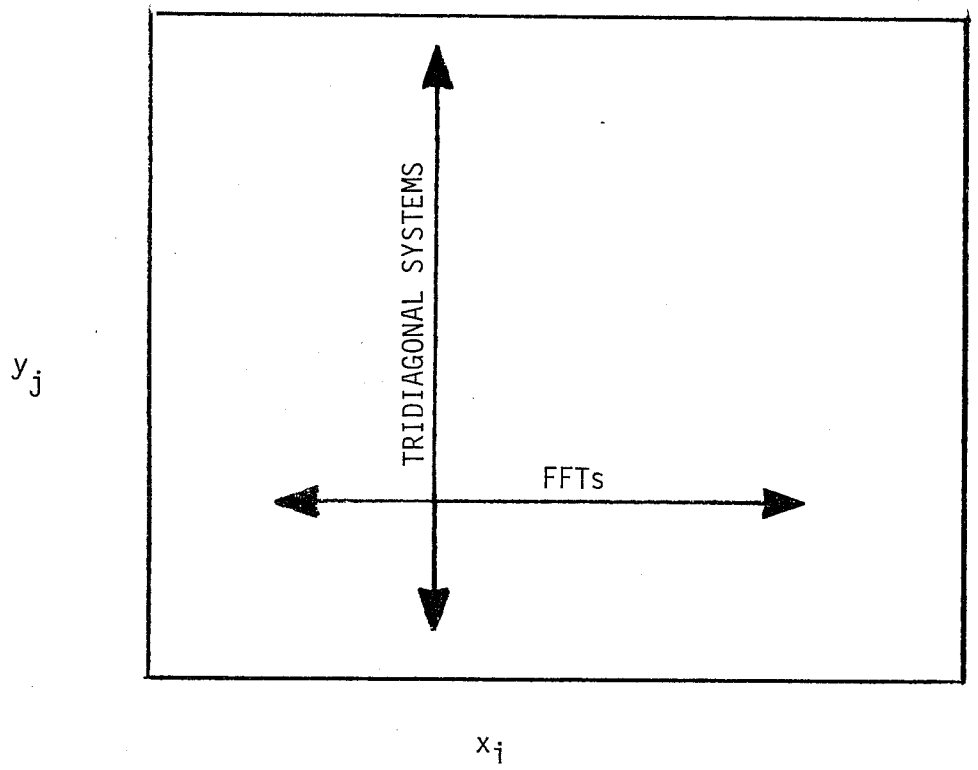


Figure 6

We proceed now to discuss the implementation of the Fourier method on a vector multiprocessor. We assume that the memory is shared ad that the computer has p processors. Consider first the FFTs of N sequences of length M, which is required in steps (i) and (iii) of the Fourier method. Four methods are presented below. The fourth method will be discussed in detail in the next section.

(i) Method I for FFTs

The simplest approach is to assign N/p FFTs to each processor and to perform the FFTs individually. The minimum vector length is $\sqrt{M}/2$, which can be relatively small.

(ii) Method II for FFTs

Again N/p FFTs are assigned to each processor. However, each operation in the FFT is applied to all N/p sequences. All vectors have length N/p, which can also be relatively short. This is an example of how distribution and vectorisation can be at odds. As the number of processors increases, the vectors become shorter.

(iii) Method III for FFTs

Apply each operation in a distributed version of the FFT to all N sequences. All vectors have length N. This approach requires a distributed version of the FFT which is discussed in the next section. This approach is relatively efficient, since the FFT distributes quite well; i.e., the number of operations in the distributed FFT is the same as in the non-distributed FFT.

(iv) Method IV for FFTs

All N sequence of length M are transformed as a single sequence of length MN using the "truncated" FFT. The minimum vector length is $\sqrt{MN}/p$ and the maximum vector length is MN/p. This approach is discussed in detail in the next section.

Next we consider the efficient solution of the tridiagonal systems. The asymptotic operation count for solving the tridiagonal systems is O(MN), which is less than O(MNlogM) for the FFTs. However, it is more difficult to vectorise the solution of the tridiagonal systems, and in practice it takes a significant part of the total computing time.

(i) Method I for Solving the Tridiagonal Systems

The simplest approach is to assign M/p tridiagonal systems to each processor. The systems can then be solved individually using the tridiagonal version of Gauss elimination. The difficulty with this approach is that the Gauss algorithm does not vectorise.

(ii) Method II for Solving the Tridiagonal Systems

In each processor, each operation in the Gauss algorithm is applied to all M/P systems. All vector lengths are M/P; however, a significant amount of additional storage is required for the LU decompositions.

(iii) Method III for Solving the Tridiagonal Systems

Each operation in a distributed version of the Gauss algorithm is applied to all tridiagonal systems. All vector lengths are M. This method also requires a substantial amount of storage. Another problem is that the Gauss algorithm does not distribute well. The number of operations in the distributed algorithm is greater than in the non-distributed algorithm [9].

(iv) Method IV for Solving the Tridiagonal Systems

Each operation in a distributed version of the cyclic reduction algorithm is applied to all tridiagonal systems, with the result that the vector lengths are all equal to M. The cyclic reduction algorithm distributes well and requires only a modest amount of additional storage.

7 COMPUTING THE FFT ON A VECTOR MULTIPROCESSOR

In the previous section an overview of the Fourier method for solving Poisson's equation was presented. At the end of the section, vector method IV was presented in which N sequences of length M are transformed as a single sequence of length MN using a distributed version of the truncated FFT. In this section we discuss this method in greater detail. A survey of vector methods for the FFT is given in [13].

This section is divided into the following parts:

7.1 A review of the Cooley-Tukey FFT

7.2 The loop inversion method for maintaining long vectors

7.3 The "truncated" FFT for multiple transforms

7.4 Eliminating the sort phase in the FFT

7.5 The distributed FFT on the Cray X-MP/2

7.1 A review of the Cooley-Tukey FFT

The presentation in this section involves modifications of the computation and storage allocations that are internal to the FFT. Therefore, in order to develop notation and to describe the results, it is necessary to review the FFT algorithm for computing the discrete Fourier transform. Given the complex sequence X_m, then we wish to compute the discrete Fourier transform:

$$X_k = \frac{1}{M} \sum_{m=0}^{M-1} x_m \, e^{-ikm \frac{2\pi}{M}} \qquad (7.1)$$

This computation requires M^2 complex multiplications and M(M-1) complex additions. We begin our discussion of the FFT with a description of the splitting algorithm which is fundamental to the FFT and which computes the X_k with about half the number of operations required by (7.1).

If M is even then the sum on the right side of (7.1) can be split into two sums in which the subscripts on X_m are even and odd, respectively. It is customary to ignore the scale factor of M^{-1}, in which case:

$$X_k = \sum_{m=0}^{M/2-1} x_{2m} e^{-ik2m\frac{2\pi}{M}} + \sum_{m=0}^{M/2-1} x_{2m+1} e^{-ik(2m+1)\frac{2\pi}{M}} \qquad (7.2)$$

If we define the new transforms

$$Y_k = \sum_{m=0}^{M/2-1} x_{2m} e^{-ik2m\frac{2\pi}{M/2}} \tag{7.3}$$

$$Z_k = \sum_{m=0}^{M/2-1} x_{2m+1} e^{-ik2m\frac{2\pi}{M/2}} \tag{7.4}$$

then (7.2) takes the form:

$$X_k = Y_k + e^{-ik\frac{2\pi}{M}} Z_k \tag{7.5}$$

From (7.3) and (7.4) it can be determined that Y_k and Z_k are periodic with period M/2. That is, $Y_{k+M/2} = Y_k$ and $Z_{k+M/2} = Z_k$. Also, since

$$e^{-i(k+M/2)\frac{2\pi}{M}} = -e^{-ik\frac{2\pi}{M}} \tag{7.6}$$

$$X_{k+N/2} = Y_k + e^{-ik\frac{2\pi}{M}} Z_k \tag{7.7}$$

the splitting algorithm can now be given:

(i) For k=1,...M/2 compute Y_k and Z_k from (7.3) and (7.4).

(ii) Compute X_k from Y_k and Z_k using (7.5) and (7.7)

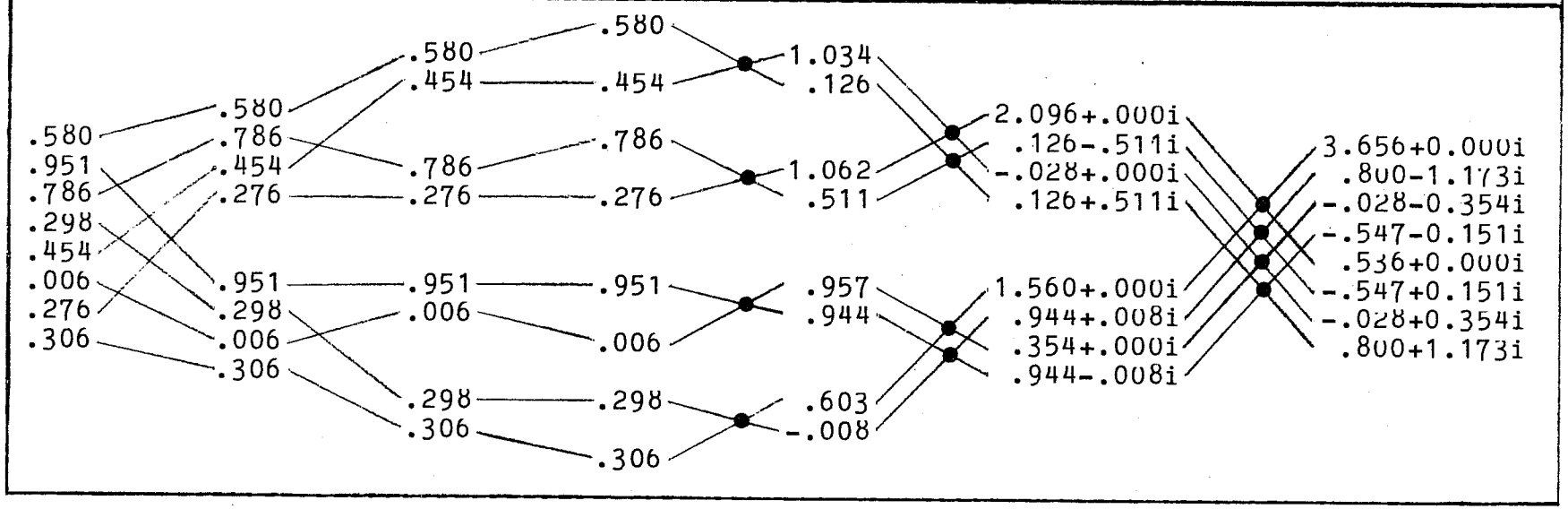
The computation of Y_k and Z_k requires $M^2/2$ multiplications and $M(M-1)/2$ additions. The computation of X_k from (7.5) and (7.7) is proportional to M which can be ignored for sizeable M. Therefore, the splitting algorithm requires about half the time taken by (7.1).

The FFT is now a simple extension of the splitting algorithm. Since (7.3) and (7.4) have the same form as (7.1) but with M replaced by M/2, the splitting algorithm can also be used to compute Y_k and Z_k. If $N=2^{\alpha}$ for some integer α then the splitting can be continued α times, which results in the FFT. A sample FFT is shown in Table 7 for the case $M=2^3=8$. The original sequence is in column 1 and subsequent splittings are in columns 2, 3 and 4. From (7.1) we note that the transform of a sequence of length 1 is just the element itself and therefore column 4 also contains the Fourier transforms of the sequences of length 1.

Columns 5, 6 and 7 are computed using the splitting formulas (7.5) and (7.7) with M = 2, 4 and 8, respectively. Columns 1 through 4 are called the sort phase of the FFT, and columns 5, 6, and 7 are called the combine phase. In Section 7.4 it is shown that sorting can be eliminated. This is important since sorting is expensive on a vector processor.

Table 7

The Cooley-Tukey FFT for N=8



7.2 The loop inversion method for maintaining long vectors

One of the difficulties associated with computing the FFT on a vector processor is that the vectors get smaller as the computation proceeds. This is evident in Table 7 where the lengths of the sequences are halved from one column to the next until column 4, after which the lengths of the transforms double from column to column. The short vectors in the centre columns result in the inefficient use of a vector processor. This is also evident in the expository FORTRAN programs, given in Figure 7, for the combine phase of the Cooley-Tukey FFT. The sort phase is not presented since, as we show later, it can be eliminated.

```
      SUBROUTINE CTCMBN (IS,M,C)
C
C     COMBINE PHASE FOR THE COOLEY-TUKEY FFT
C
      COMPLEX C(1)
      N = 2**M
      DO 100 L=1,M
      LS = 2**(L-1)
      NS = N/(LS+LS)
      CALL CTCMB1 (IS,LS,NS,C)
100   CONTINUE
      RETURN
      END
      SUBROUTINE CTCMB1 (IS,LS,NS,C)
      COMPLEX OMEGA,OMEGK,WYK,C(LS,2,NS)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      DO 200 J=1,NS
      OMEGK=1.
      DO 200 I=1,LS
      WYK=OMEGK*C(I,2,J)
      C(I,1,J)=C(I,1,J)+WYK
      C(I,2,J)=C(I,2,J)-WYK
      OMEGK=OMEGA*OMEGK
200   CONTINUE
      RETURN
      END
```

Figure 7

In the program, the length of the original sequence is N and M is $\log_2 N$. IS=-1 produces a forward transform and IS=1 produces a backward transform. Loop 100 with index L computes the columns in the combine phase of the FFT. In Table 7, columns 5, 6, and 7 are computed with L = 1, 2, and 3, respectively. The variable 2LS is the length of the sequences in the Lth column, and NS contains the number of sequences. For example, when computing column 6 in Table 7, LS=2 and NS=2. The splitting formulas (7.5) and (7.7) are evaluated in loop 200.

For L=1 the inner loop has length LS=1 which significantly degrades the performance of a vector processor. This problem could be eliminated by using the Pease FFT, in which all vectors have length N/2. However, the Pease algorithm requires $N\log_2 N$ additional storage locations, which makes it unattractive for large N. A more attractive approach is to lengthen the vectors by simply inverting the order of the loop. That is, we replace loop 200 in Figure 7 by the FORTRAN code in Figure 8.

```
      IF (LS.LT.NS)  GO TO 300
      DO 200 J=1,NS
      OMEGK=1.
      DO 200 I=1,LS
      WYK=OMEGK*C(I,2,J)
      C(I,1,J)=C(I,1,J)+WYK
      C(I,2,J)=C(I,2,J)-WYK
      OMEGK=OMEGA*OMEGK
200   CONTINUE
      RETURN
300   OMEGK=1.
      DO 500 I=1,LS
      DO 400 J=1,NS
      WYK=OMEGK*C(I,2,J)
      C(I,1,J)=C(I,1,J)+WYK
      C(I,2,J)=C(I,2,J)-WYK
400   CONTINUE
      OMEGK=OMEGA*OMEGK
500   CONTINUE
```

Figure 8

Loop 200 is duplicated but with the order of the loops inverted and renumbered as loops 500 and 400. The IF statement ensures that longest vectors correspond to the inner loop. This straightforward modification can reduce computing time by 50% depending on N. The vector lengths still vary as the computation proceeds but this minimum length is now N/2. In practice the crossover point may not be LS = NS but rather a function of the architecture. For example, in loop 400 the data are not accessed with a stride of 1, which may create memory conflicts or require a scatter-gather operation.

7.3 The "truncated" FFT for multiple transforms

In this part we will continue the search for long vectors which is particularly important for a distributed FFT. Consider now the problem in which N sequences of length M are to be transformed. In Table 7 the two sequences in column 6 are Y_k and Z_k, defined in (7.3) and (7.4) which are the Fourier transforms on the two sequences X_{2m} and X_{2m+1} in column 2. More generally, the sequences in columns 5, 6, and 7 are the Fourier transforms of the sequences in columns 3, 2, and 1, respectively. If we choose to ignore columns 1 and 7 and assume that the two sequences in column 2 are given, then these transforms Y_k and Z_k are computed in column 6. Therefore columns 2 through 6 correspond to a multiple transform in which M=4 and N=2. Similarly, columns 3, 4, and 5 correspond to a multiple transform problem in which M=2 and N=4.

This result generalises to arbitrary M and N, and the resulting algorithm is called the truncated FFT. It is attractive on a vector processor since the minimum vector length is now $\sqrt{MN/2}$. A FORTRAN program for the truncated FFT is given in the next part. However, it is based on an algorithm for the FFT that does not require sorting.

7.4 Eliminating the sort phase in the FFT

Fortunately, sorting can be eliminated, which results in a significant improvement on a vector processor. There are several variants of the Cooley-Tukey FFT which differ only in the way that the intermediate computations are stored. These variants, together with the Cooley-Tukey FFT are discussed in detail in [13]. Of particular interest are the Stockham FFTs because they do not have a sort phase. The intermediate computations in the combine phase are stored in such a manner as to ensure that the final transform is ordered.

An expository FORTRAN program for the Stockham algorithm is given in Fig. 9. It differs from the Cooley-Tukey FFT in Fig. 7, only in the order of the subscripts in loop 200 and in its use of a work array. Unlike the Cooley-Tukey FFT, the Stockham FFT cannot be computed in place.

```
      SUBROUTINE STOCK(IS,M,C,WORK)
C
C     THE STOCKHAM AUTO-SORT FFT
C
      COMPLEX C(1), WORK(1)
      N = 2**M
      DO 100 L=1,M
      LS = 2**(L-1)
      NS = N/(LS+LS)
      CALL STOCK1(IS,LS,NS,C,WORK)
      DO 100 I=1,N
      C(I)=WORK(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE STOCK1(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE=FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA=CMPLX(COS(ANGLE),SIN(ANGLE))
      DO 200 J=1,NS
      OMEGK=1.
      DO 200 I=1,LS
      WYK=OMEGK*C(J,2,I)
      CH(J,I,1)=C(J,1,I)+WYK
      CH(J,I,2)=C(J,1,I)-WYK
      OMEGK=OMEGA*OMEGK
200   CONTINUE
      RETURN
      END
```

Figure 9

The Stockham FFT is particularly suited for use with the truncated FFT. If the N sequences are stored in an array C(I,J), where I=1,...,M and J=1,...,N and C is dimensioned as C(M,N), then the truncated FFT computes the transforms in the J direction. That is, M transforms of length N are computed. This is particularly useful when the transforms are being performed as part of a solution to Poisson's equation. Since the FFTs are performed in the J direction, the solution of the tridiagonal systems can then be performed in the I direction where the vector elements are stored consecutively. An expository FORTRAN program for the truncated FFT is given in Figure 10.

```
      SUBROUTINE MSTOCK(IS,LI,LJ,C,WORK)
C
C     THE TRUNCATED STOCKHAM AUTOSORT FFT FOR MULTIPLE TRANSFORMS
C
C     DEFINE M=2**LI AND N=2**LJ THEN M TRANSFORMS OF LENGTH N
C     ARE COMPUTED. TTHE TRANSFORMS ARE COMPUTED IN THE DIRECTION
C     OF THE SECOND INDEX OF THE TWO DIMENSIONAL ARRAY C.
C     THE FIRST DIMENSION OF C MUST BE EQUAL TO M. THAT IS,
C     THE SEQUENCES MUST BE STORED CONSECUTIVELY.
C
C     IS = -1 FORWARD TRANSFORM
C     IS =  1 BACKWARD TRANSFORM
C
      COMPLEX C(1),WORK(1)
      N = 2**(LI+LJ)
      DO 100 L=1,LJ
      LS = 2**(L-1)
      NS = N/(LS+LS)
      CALL STOCKH(IS,LS,NS,C,WORK)
      DO 100 I=1,N
      C(I) = WORK(I)
  100 CONTINUE
      RETURN
      END
      SUBROUTINE STOCKH(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE = FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA = CMPLX(COS(ANGLE),SIN(ANGLE))
      DO 200 J=1,NS
      OMEGK = 1.
      DO 200 I=1,LS
      WYK = OMEGK*C(J,2,I)
      CH(J,I,1) = C(J,1,I)+WYK
      CH(J,I,2) = C(J,1,I)-WYK
      OMEGK = OMEGA*OMEGK
  200 CONTINUE
      RETURN
      END
```

Figure 10

7.5 The distributed FFT on the Cray X-MP/2

With the results presented above, the distribution of the FFT on a vector multiprocessor is straightforward. From Table 7 it would seem reasonable to partition the computations between the upper and lower halves, that is, to assign the computations in the upper and lower halves of Table 7 to processors 0 and 1, respectively. The processors would be synchronised at only one point, namely, between columns 6 and 7.

However, longer vectors are possible if the computations are distributed in the FORTRAN program for the truncated Stockham FFT given in Fig. 9. Although the processors must be synchronised between each column (or value of L) the long vectors should result in a net increase in speed. A FORTRAN program for the distributed computation is given in Appendix F. This program differs from the single processor program in the following respects.

(i) The computations in loops 200 on the rectangle J=1, NS; I=1,LS are distributed between the processors. The half that corresponds to J=1,NS/2 is computed in subroutine STOCKH, and the other half for J=NS/2+1, NS, is computed in subroutine STOCKG. One processor executes subroutine MSTOCK, which calls STOCKH. The other processor executes subroutine TASK, which calls STOCKG.

(ii) The processors are synchronised using events following the computations in loops 200 in STOCKH and STOCKG, respectively.

(iii) To avoid the need for additional synchronisation points, the transfer from the WORK array to the C array has been eliminated by duplicating the CALL STOCKH and CALL STOCKG statements, with the order of the arguments C and WORK reversed.

It is important to note that the program is strictly expository. The loops 200 in STOCKH and STOCKG will not vectorise because of the statement OMEGK=OMEGA*OMEGK. Also, the loops 200 should be inverted as discussed in Section 7.2. The computations for the inverted loops would then be distributed on the index I. That is, the computations for I=1,LS/2 and I=LS/2+1, LS would be assigned to processors 0 and 1, respectively.

In addition, the CALL TSKSTART statement should be moved to the program that calls subroutine MSTOCK or to an initialising program. In practice the trigonometric computations for OMEGK are computed only once in a separate subroutine that is called the initialising program. The reason for this is that the trigonometric computations take a substantial amount of time, and they can be used repeatedly for subsequent transforms. The situation is similar for the CALL TSKSTART statement, which can require up to 4 milliseconds.

The task is started in the initialising program, and its computations are synchronised using the statements CALL EVPOST, CALL EVWAIT, and CALL EVCLEAR, whose combined time is about 4 microseconds. The use of these statements is illustrated in Appendix F.

The distribution of the FFT for a larger number of processors is somewhat more complicated because the number of processors is greater than the range NS on the index J. For example, when L=M then NS=2. If four processors are available, then the inner loop must also be distributed, which will result in shorter vectors. However, LS achieves its maximum value N/2 when L=M, so the length of the inner loop on I is N/4 which is still a relatively long vector and hence any reduction in performance should be minimal.

8 ACKNOWLEDGEMENT

The authors gratefully acknowledge the valuable assistance afforded them by colleagues and staff at Cray Research Inc., Control Data Corporation, Denelcor Inc. and ETA Corporation. In particular, the contributions made by D. Dent (ECMWF) and J. Larson (Cray Research Inc.) deserve mention.

REFERENCES

(1) CDC CYBER 200 Model 205 Computer System Hardware Reference Manual, Rev. A, Pub. No. 60256020, Control Data Corp., Minneapolis, Minnesota, 1981.

(2) CDC CYBER 200 FORTRAN Version 2 Reference Manual, Rev. B, Pub. No. 60485000, Control Data Corp., Minneapolis, Minnesota, 1981.

(3) T. Cheung and J.E. Smith, An analysis of the CRAY X-MP memory system, Proceedings of the 1984 International Conference on Parallel Processing, Bellaire, Michigan, IEEE Computer Society, Los Angeles, California 1984.

(4) Cray Research Multitasking User Guide, CRAY Computer Systems Technical Note No. SN-0222, Cray Research, Inc., Mendota Heights, Minnesota, 1984.

(5) R. Hockney and C. Jesshope, Parallel Computers, Adam Hilger Ltd., Bristol, England, 1981.

(6) K. Hwang and F.A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984.

(7) J. Larson, "Multitasking on the CRAY X-MP/2 Multiprocessor", Computer, 17 (1984), pp. 62-69.

(8) R. Sadourney, The dynamics of finite difference models of the shallow water equations, J. Atmos. Sciences, 32 (1975), pp. 680-689

(9) A.H. Sameh and D. Kuck, On stable parallel linear system solvers, JACM, 25 (1978), pp. 80-91.

(10) A.H. Sameh, A fast Poisson solver for multiprocessors, Elliptic Problem Solvers II, Academic Press Inc., New York, 1984.

(11) Paul N. Swarztrauber, The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle, SIAM Rev., 19 (1977), pp.490-501.

(12) ----------, Fast Poisson Solvers, MAA Studies in Numerical Analysis, Vol.24, Mathematical Association of America, 1984.

(13) ----------, FFT algorithms for vector computers, Parallel Computing, 1(1984), pp.45-63.

APPENDIX A

Single processor CRAY-1 version of shallow water model

```
      PROGRAM SHALOW
C
C     BENCHMARK WEATHER PREDICTION PROGRAM FOR COMPARING THE
C     PERFORMANCE OF CURRENT SUPERCOMPUTERS. THE MODEL IS
C     BASED OF THE PAPER - THE DYNAMICS OF FINITE-DIFFERENCE
C     MODELS OF THE SHALLOW-WATER EQUATIONS, BY ROBERT SADOURNY
C     J. ATM. SCIENCES, VOL 32, NO 4, APRIL 1975.
C
C     CODE BY PAL N. SWARZTRAUBER, NATIONAL CENTER FOR
C     ATMOSPHERIC RESEARCH, BOULDER, CO, OCTOBER 1984.
C
      DIMENSION U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
      REAL MFS100,MFS200,MFS300
C
C     NOTE BELOW THAT TWO DELTA T (TDT) IS SET TO DT ON THE FIRST
C     CYCLE AFTER WHICH IT IS RESET TO DT+DT
C
      DT = 90.
      TDT = DT
C
      DX = 1.E5
      DY = 1.E5
      A = 1.E6
      ALPHA = .001
      ITMAX = 1200
      MPRINT = 120
      M = 64
      N = 64
      MP1 = M+1
      NP1 = N+1
      EL = FLOAT(N)*DX
      PI = 4.*ATAN(1.)
      TPI = PI+PI
      DI = TPI/FLOAT(M)
      DJ = TPI/FLOAT(N)
      PCF = PI*PI*A*A/(EL*EL)
C
C     INITIAL VALUES OF THE STREAM FUNCTION AND P
C
      DO 50 J=1,NP1
      DO 50 I=1,MP1
      PSI(I,J) = A*SIN((FLOAT(I)-.5)*DI)*SIN((FLOAT(J)-.5)*DJ)
      P(I,J) = PCF*(COS(2.*FLOAT(I-1)*DI)
     1             +COS(2.*FLOAT(J-1)*DJ))+50000.
   50 CONTINUE
C
C     INITIALIZE VELOCITIES
C
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J) = -(PSI(I+1,J+1)-PSI(I+1,J))/DY
```

```
      V(I,J+1) = (PSI(I+1,J+1)-PSI(I,J+1))/DX
   60 CONTINUE
C
C     PERIODIC CONTINUATION
      DO 70 J=1,N
      U(1,J) = U(M+1,J)
      V(M+1,J+1) = V(1,J+1)
   70 CONTINUE
      DO 75 I=1,M
      U(I+1,N+1) = U(I+1,1)
      V(I,1) = V(I,N+1)
   75 CONTINUE
      U(1,N+1) = U(M+1,1)
      V(M+1,1) = V(1,N+1)
      DO 86 J=1,NP1
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
   86 CONTINUE
C
C     PRINT INITIAL VALUES
C
      WRITE(6,390) N,M,DX,DY,DT,ALPHA
  390 FORMAT(*1NUMBER OF POINTS IN THE X DIRECTION* I8/
     1       * NUMBER OF POINTS IN THE Y DIRECTION* I8/
     2       * GRID SPACING IN THE X DIRECTION    * F8.0/
     3       * GRID SPACING IN THE Y DIRECTION    * F8.0/
     4       * TIME STEP                          * F8.0/
     5       * TIME FILTER PARAMETER              * F8.3)
      MNMIN = MIN0(M,N)
      WRITE(6,391) (POLD(I,I),I=1,MNMIN)
  391 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,392) (UOLD(I,I),I=1,MNMIN)
  392 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF U * //(8E15.5))
      WRITE(6,393) (VOLD(I,I),I=1,MNMIN)
  393 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF V * //(8E15.6))
      TSTART = 9.5E-9*RTC(DUM)
      T300 = 1.
      TCYCH = 0.
      TIME = 0.
      NCYCLE = 0
   90 NCYCLE = NCYCLE + 1
      TCYC = 9.5E-9*RTC(DUM)
C
C     COMPUTE CAPITAL U, CAPATIL V, Z AND H
C
      FSDX = 4./DX
      FSDY = 4./DY
      T100 = 9.5E-9*RTC(DUM)
      DO 100 J=1,N
      DO 100 I=1,M
      CU(I+1,J) = .5*(P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = .5*(P(I,J+1)+P(I,J))*V(I,J+1)
```

```
      Z(I+1,J+1) = (FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1)
     1          -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
      H(I,J) = P(I,J)+.25*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
     1               +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
  100 CONTINUE
      T100 = 9.5E-9*RTC(DUM)-T100
C
C     PERIODIC CONTINUATION
C
      DO 110 J=1,N
      CU(1,J) = CU(M+1,J)
      CV(M+1,J+1) = CV(1,J+1)
      Z(1,J+1) = Z(M+1,J+1)
      H(M+1,J) = H(1,J)
  110 CONTINUE
      DO 115 I=1,M
      CU(I+1,N+1) = CU(I+1,1)
      CV(I,1) = CV(I,N+1)
      Z(I+1,1) = Z(I+1,N+1)
      H(I,N+1) = H(I,1)
  115 CONTINUE
      CU(1,N+1) = CU(M+1,1)
      CV(M+1,1) = CV(1,N+1)
      Z(1,1) = Z(M+1,N+1)
      H(M+1,N+1) = H(1,1)
C
C     COMPUTE NEW VALUES U,V AND P
C
      TDTS8 = TDT/8.
      TDTSDX = TDT/DX
      TDTSDY = TDT/DY
      T200 = 9.5E-9*RTC(DUM)
      DO 200 J=1,N
      DO 200 I=1,M
      UNEW(I+1,J) = UOLD(I+1,J)+
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2       +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
      VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1       *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2       -TDTSDY*(H(I,J+1)-H(I,J))
      PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
     1       -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
      T200 = 9.5E-9*RTC(DUM)-T200
C
C     PERIODIC CONTINUATION
C
      DO 210 J=1,N
      UNEW(1,J) = UNEW(M+1,J)
      VNEW(M+1,J+1) = VNEW(1,J+1)
      PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
      DO 215 I=1,M
      UNEW(I+1,N+1) = UNEW(I+1,1)
```

```
      VNEW(I,1) = VNEW(I,N+1)
      PNEW(I,N+1) = PNEW(I,1)
  215 CONTINUE
      UNEW(1,N+1) = UNEW(M+1,1)
      VNEW(M+1,1) = VNEW(1,N+1)
      PNEW(M+1,N+1) = PNEW(1,1)
      IF(NCYCLE .GT. ITMAX) CALL EXIT
      TIME = TIME + DT
      TCYC = 9.5E-9*RTC(DUM) - TCYC
      IF(MOD(NCYCLE,MPRINT) .NE. 0) GO TO 370
      PTIME = TIME/3600.
      WRITE(6,350) NCYCLE,PTIME
  350 FORMAT(//* CYCLE NUMBER*I5* MODEL TIME IN  HOURS* F6.2)
      WRITE(6,355) (PNEW(I,I),I=1,MNMIN)
  355 FORMAT(/*DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,360) (UNEW(I,I),I=1,MNMIN)
  360 FORMAT(/* DIAGONAL ELEMENTS OF U * //(8E15.6))
      WRITE(6,365) (VNEW(I,I),I=1,MNMIN)
  365 FORMAT(/* DIAGONAL ELEMENTS OF V * //(8E15.6))
      MFS100 = 24.*M*N/T100/1.E6
      MFS200 = 26.*M*N/T200/1.E6
      MFS300 = 15.*M*N/T300/1.E6
      CTIME = 9.5E-9*RTC(DUM)-TSTART
      TCYC = CTIME/FLOAT(NCYCLE)
      WRITE(6,375) NCYCLE,CTIME,TCYCH,T100,MFS100,T200,MFS200,T300,
     1             MFS300
  375 FORMAT(* CYCLE NUMBER*I5* TOTAL COMPTER TIME* E15.6
     1       * TIME PER CYCLE* E15.6 /
     2       * TIME AND MEGAFLOPS FOR LOOP 100 * E15.6,F6.1/
     3       * TIME AND MEGAFLOPS FOR LOOP 200 * E15.6,F6.1/
     4       * TIME AND MEGAFLOPS FOR LOOP 300 * E15.6,F6.1/ )
  370 IF(NCYCLE .LE. 1) GO TO 310
      CTCYC = 9.5E-9*RTC(DUM)
      T300 = 9.5E-9*RTC(DUM)
      DO 300 J=1,N
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      T300 = 9.5E-9*RTC(DUM)-T300
C
C     PERIODIC CONTINUATION
C
      DO 320 J=1,N
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)
      V(M+1,J) = V(1,J)
      P(M+1,J) = P(1,J)
```

```
320 CONTINUE
    DO 325 I=1,M
    UOLD(I,N+1) = UOLD(I,1)
    VOLD(I,N+1) = VOLD(I,1)
    POLD(I,N+1) = POLD(I,1)
    U(I,N+1) = U(I,1)
    V(I,N+1) = V(I,1)
    P(I,N+1) = P(I,1)
325 CONTINUE
    UOLD(M+1,N+1) = UOLD(1,1)
    VOLD(M+1,N+1) = VOLD(1,1)
    POLD(M+1,N+1) = POLD(1,1)
    U(M+1,N+1) = U(1,1)
    V(M+1,N+1) = V(1,1)
    P(M+1,N+1) = P(1,1)
    TCYC = TCYC + 9.5E-9*RTC(DUM) - CTCYC
    TCYCH = TCYC
    GO TO 90
310 TDT = TDT+TDT
    DO 400 J=1,NP1
    DO 400 I=1,MP1
    UOLD(I,J) = U(I,J)
    VOLD(I,J) = V(I,J)
    POLD(I,J) = P(I,J)
    U(I,J) = UNEW(I,J)
    V(I,J) = VNEW(I,J)
    P(I,J) = PNEW(I,J)
400 CONTINUE
    GO TO 90
    END
```

APPENDIX B

Cyber 205 version of shallow water model

```
      PROGRAM SHALOW(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C
C     BENCHMARK WEATHER PREDICTION PROGRAM FOR COMPARING  THE
C     PERFORMANCE OF CURRENT SUPERCOMPUTERS. THE MODEL IS
C     BASED ON THE PAPER - THE DYNAMICS OF FINITE-DIFFERENCE
C     MODELS OF THE SHALLOW-WATER EQUATIONS, BY ROBERT SADOURNY
C     J. ATM. SCIENCES, VOL 32, NO 4, APRIL 1975.
C
C     CODE BY PAUL N. SWARZTRAUBER, NATIONAL CENTER FOR
C     ATMOSPHERIC RESEARCH, BOULDER, CO, OCTOBER 1984.
C
C     CONVERSION TO CYBER 205 VECTOR FORTRAN BY ROLAND SWEET,
C     NATIONAL BUREAU OF STANDARDS, BOULDER, CO. OCTOBER 1984.
C
      DIMENSION U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
      REAL MFS100,MFS200,MFS300
C
C     CYBER 205 DECLARATION OF BIT VECTOR FOR CONTROLLED STORE OPERATION
C
C     BIT INTER((M+1)*N-1)
C
      BIT INTER(4159)
C
C     NOTE BELOW THAT TWO DELTA T (TDT) IS SET TO DT ON THE FIRST
C     CYCLE AFTER WHICH IT IS RESET TO DT+DT.
C
      DT = 90.
      TDT = DT
C
      DX = 1.E5
      DY = 1.E5
      A = 1.E6
      ALPHA = .001
      ITMAX = 1200
      MPRINT = 120
      M = 64
      N = 64
      MP1 = M+1
      NP1 = N+1
C
C     DEFINE CYBER 205 BIT VECTOR TO STORE ONLY IN THE SUB-ARRAY
C     I=1,2,...,M, J=1,2,...,N OF THE ARRAY DEFINED FOR
C     I=1,2,...,M,M+1, J=1,2,...,N,N+1.
C
      LV = MP1*N-1
      INTER(1;LV) = Q8VMK0(M,MP1;INTER(1;LV))
      L = MP1*NP1
C
```

```
      EL = FLOAT(N)*DX
      PI = 4.ATAN(1.)
      TPI = PI+PI
      DI = TPI/FLOAT(M)
      DJ = TPI/FLOAT(N)
      PCF = PI*PI*A*A/(EL*EL)
C
C     INITIAL VALUES OF THE STREAM FUNCTION AND P
C
      DO 50 J=1,NP1
      DO 50 I=1,MP1
      PSI(I,J) = A*SIN((FLOAT(I)-.5)*DI)*SIN((FLOAT(J)-.5)*DJ)
      P(I,J) = PCF*(COS(2.*FLOAT(I-1)*DI)
     1               +COS(2.*FLOAT(J-1)*DJ))+50000.
   50 CONTINUE
C
C     INITIALIZE VELOCITIES
C
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J) = -(PSI(I+1,J+1)-PSI(I+1,J))/DY
      V(I,J+1) =  (PSI(I+1,J+1)-PSI(I,J+1))/DX
   60 CONTINUE
C
C     PERIODIC CONTINUATION
C
      DO 70 J=1,N
      U(1,J) = U(M+1,J)
      V(M+1,J+1) = V(1,J+1)
   70 CONTINUE
      DO 75 I=1,M
      U(I+1,N+1) = U(I+1,1)
      V(I,1) = V(I,N+1)
   75 CONTINUE
      U(1,N+1) = U(M+1,1)
      V(M+1,1) = V(1,N+1)
      DO 86 J=1,NP1
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
   86 CONTINUE
C
C     PRINT INITIAL VALUES
C
      WRITE(6,390) N,M,DX,DY,DT,ALPHA
  390 FORMAT(*1NUMBER OF POINTS IN THE X DIRECTION* I8/
     1       * NUMBER OF POINTS IN THE Y DIRECTION* I8/
     2       * GRID SPACING IN THE X DIRECTION    * F8.0/
     3       * GRID SPACING IN THE Y DIRECTION    * F8.0/
     4       * TIME STEP                          * F8.0/
     5       * TIME FILTER PARAMTER               * F8.3)
      MNMIN = MIN0(M,N)
      WRITE(6,391) (POLD(I,I),I=1,MNMIN)
```

```
  391 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF P * //(4E25.13))
      WRITE(6,392) (UOLD(I,I),I=1,MNMIN)
  392 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF U * //(4E25.13))
      WRITE(6.393) (VOLD(I,I),I=1,MNMIN)
  393 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF V * //(4E25.13))
      TSTART = SECOND(DUM)
      T300 = 1.
      TIME = 0.
      NCYCLE = 0
   90 NCYCLE = NCYCLE + 1
C
C     COMPUTE CAPITAL U, CAPITAL V, Z AND H
C
      FSDX = 4./DX
      FSDY = 4./DY
      T100 = SECOND(DUM)
C
C     CYBER 205 VERSION OF DO 100 LOOP
C
      WHERE (INTER(1;LV))
         CU(2,1;LV) = .5*(P(2,1;LV)+P(1,1;LV))*U(2,1;LV)
         CV(1,2;LV) = .5*(P(1,2;
         Z(2,2;LV)  = (FSDX*(V(2,2;LV)-V(1,2;LV))-
     *                 FSDY*(U(2,2;LV)-U(2,1;LV)))/
     *                (P(1,1;LV)+P(1,2;LV)+P(2,2;LV)+P(2,1;LV))
         H(1,1;LV)  = P(1,1;LV)+.25*(U(2,1;LV)*U(2,1;LV)
     *                +U(1,1;LV)*U(1,1;LV)+V(1,2;LV)*V(1,2;LV)
                     +V(1,1;LV)*V(1,1;LV))
      END WHERE
C
      T100 = SECOND(DUM)-T100
C
C     PERIODIC CONTINUATION
C
      DO 110 J=1,N
      CU(1,J) = CU(M+1,J)
      CV(M+1,J+1) = CV(1,J+1)
      Z(1,J+1) = Z(M+1,J+1)
      H(M+1,J) = H(1,J)
  110 CONTINUE
C
C     CYBER 205 VERSION OF DO 115 LOOP
C
      CU(2,NP1;M) = CU(2,1;M)
      CV(1,1;M) = CV(1,NP1;M)
      Z(2,1;M) = Z(2,NP1;M)
      H(1,NP1;) = H(1,1;M)
C
      CU(1,N+1) = CU(M+1,1)
      CV(M+1,1) = CV(1,N+1)
      Z(1,1) = Z(M+1,N+1)
      H(M+1,N+1) = H(1,1)
C
```

```
C     COMPUTE NEW VALUES U,V AND P
C
      TDTS8 = TDT/8.
      TDTSDX = TDT/DX
      TDTSDY = TDT/DY
      T200 = SECOND(DUM)
C
C     CYBER 205 VERSION OF DO 200 LOOP
C
      WHERE (INTER(1;LV))
         UNEW(2,1;LV) = UOLD(2,1;LV)+TDTS8*(Z(2,2;LV)+Z(2,1;LV))*
     *                  (CV(2,2;LV)+CV(1,2;LV)+CV(1,1;LV)+CV(2,1;LV))
     *                  -TDTSDX*(H(2,1;LV)-H(1,1;LV))
         VNEW(1,2;LV) = VOLD(1,2;LV)-TDTS8*(Z(2,2;LV)+Z(1,2;LV))*
     *                  (CU(2,2;LV)+CU(2,1;LV)+CU(1,1;LV)+CU(1,2;LV))
                        -TDTSDY*(H(1,2;LV)-H(1,1;LV))
         PNEW(1,1;LV) = POLD(1,1;LV)-TDTSDX*(CU(2,1;LV)-CU(1,1;LV))
     *                  -TDTSDY*(CV(1,2;LV)-CV(1,1;LV))
      END WHERE
C
      T200 = SECOND(DUM)-T200
C
C
C     PERIODIC CONTINUATION
C
      DO 210 J=1,N
      UNEW(1,J) = UNEW(M+1,J)
      VNEW(M+1,J+1) = VNEW(1,J+1)
      PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
C
C     CYBER 205 VERSION OF DO 215 LOOP
C
      UNEW(2,NP1;M) = UNEW(2,1;M)
      VNEW(1,1;M) = VNEW(1,NP1;M)
      PNEW(1,NP1;M) = PNEW(1,1;M)
C
      UNEW(1,N+1) = UNEW(M+1,1)
      VNEW(M+1,1) = VNEW(1,N+1)
      PNEW(M+1,N+1) = PNEW(1,1)
      IF(NCYCLE .GT. ITMAX) STOP
      TIME = TIME + DT
      IF(MOD(NCYCLE,MPRINT) .NE. 0) GO TO 370
      PTIME = TIME/3600.
      WRITE(6,350) NCYCLE,PTIME
  350 FORMAT(//* CYCLE NUMBER*I5* MODEL TIME IN HOURS* F6.2)
      WRITE(6,355) (PNEW(I,I),I=1,MNMIN)
  355 FORMAT(/* DIAGONAL ELEMENTS OF P * //(4E25.13))
      WRITE(6,360) (UNEW(I,I),I=1,MNMIN)
  360 FORMAT(/* DIAGONAL ELEMENTS OF U * //(4E25.13))
      WRITE(6,365) (VNEW(I,I),I=1,MNMIN)
```

```
  365 FORMAT(/* DIAGONAL ELEMENTS OF V * //(4E25.13))
      MFS100 = 24.*((M+1)*N-1)/T100/1.E6
      MFS200 = 26.*((M+1)*N-1)/T200/1.E6
      MFS300 = 15.*((M+1)*N-1)/T300/1.E6
      CTIME = SECOND(DUM)-TSTART
      TCYC = CTIME/FLOAT(NCYCLE)
      WRITE(6,375) NCYCLE,CTIME,TCYC,T100,MFS100,T200,MFS200,T300,MFS300
  375 FORMAT(* CYCLE NUMBER*I5* TOTAL COMPUTER TIME* E15.6
     1       * TIME PER CYCLE* E15.6 /
     2       * TIME AND MEGAFLOPS FOR LOOP 100 * E15.6,F6.1/
     3       * TIME AND MEGAFLOPS FOR LOOP 200 * E15.6,F6.1/
     4       * TIME AND MEGAFLOPS FOR LOOP 300 * E15.6,F6.1/ )
C
C     TIME SMOOTHING AND UPDATE FOR NEXT CYCLE
C
  370 IF(NCYCLE .LE. 1) GO TO 310
      T300 = SECOND(DUM)
C
C     CYBER 205 VERSION OF DO 300 LOOP
C
      UOLD(1,1;L) = U(1,1;L)+ALPHA*(UNEW(1,1;L)-2.*U(1,1;L)+UOLD(1,1;L))
      VOLD(1,1;L) = V(1,1;L)+ALPHA*(VNEW(1,1;L)-2.*V(1,1;L)+VOLD(1,1;L))
      POLD(1,1;L) = P(1,1;L)+ALPHA*(PNEW(1,1;L)-2.*P(1,1;L)+POLD(1,1;L))
      U(1,1;L) = UNEW(1,1;L)
      V(1,1;L) = VNEW(1,1;L)
      P(1,1;L) = PNEW(1,1;L)
C
      T300 = SECOND(DUM)-T300
C
C     PERIODIC CONTINUATION
      DO 320 J=1,N
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)
      V(M+1,J) = V(1,J)
      P(M+1,J) = P(1,J)
  320 CONTINUE
C
C     CYBER 205 VERSION OF DO 325 LOOP
C
      UOLD(1,NP1;M) = UOLD(1,1;M)
      VOLD(1,NP1;M) = VOLD(1,1;M)
      POLD(1,NP1;M) = POLD(1,1;M)
      U(1,NP1;M) = U(1,1;M)
      V(1,NP1;M) = V(1,1;M)
      P(1,NP1;M) = P(1,1;M)
C
      UOLD(M+1,N+1) = UOLD(1,1)
      VOLD(M+1,N+1) = VOLD(1,1)
      POLD(M+1,N+1) = POLD(1,1)
      U(M+1,N+1) = U(1,1)
      V(M+1,N+1) = V(1,1)
      P(M+1,N+1) = P(1,1)
      GO TO 90
```

```
  310 TDT = TDT+TDT
C
C     CYBER 205 VERSION OF DO 400 LOOP
C
      UOLD(1,1;L)L = U(1,1;L)
      VOLD(1,1;L)L = V(1,1;L)
      POLD(1,1;L)L = P(1,1;L)
      U(1,1;L) = UNEW(1,1;L)
      V(1,1;L) = VNEW(1,1;L)
      P(1,1;L) = PNEW(1,1;L)
C
      GO TO 90
      END
```

APPENDIX C

HEP.1 version of shallow water model

```
      PROGRAM SHALLOWP
C
C     BENCHMARK WEATHER PREDICTION PROGRAM FOR COMPARING THE
C     PERFORMANCE OF CURRENT SUPERCOMPUTERS. THE MODEL IS
C     BASED ON THE PAPER - THE DYNAMICS OF FINITE-DIFFERENCE
C     MODELS OF THE SHALLOW-WATER EQUATIONS, BY ROBERT SADDURNY
C     J. ATM. SCIENCES VOL. 32,NO. 4 APRIL 1975.
C
C     CODE BY DAVID SNELLING, DENELCOR, DENVER, CO OCTOBER 1984.
C
C     PROGRAM PERFORMS NDIM**2 * (29+25) + (NDIM+1)**2 * 15 FLOATING
C     POINT OPERATIONS PER TIME STEP.
C
C     ON THE CRAY-1 AT NCAR FOR NDIM=64
C             RUN TIME PER TIME STEP = 0.00525 SECONDS.
C             MFLOPS                 = 54.2
C             MACHINE CYCLE TIME (MS) * MFLOPS = 0.677 (EFFICIENT)
C
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)

C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C
      COMMON /SYNC/ KOUT,NPROC,DONE,IBARR(52),FINISH
C
      EXTERNAL PARSUB
C
C
```

```
C             INITIALIZE THE RUNTIME PARAMETERS.
C
      A       = 1.0 E 6
      TDT     = 90.0
      DX      = 1.0 E 5
      DY      = 1.0 E 5
      ALPHA   = 0.001
      ITMAX   = 1200
      M       = NDIM
      N       = NDIM
      MP1     = M + 1
      NP1     = N + 1
      PI      = 4.0 * ATAN(1.0)
      TPI     = 2.0 * PI
      DI      = TPI/FLOAT(M)
      DJ      = TPI/FLOAT(N)
      EL      = FLOAT (N) * DX
      PCF     = PI*PI*A*A/(EL*EL)
C
      WRITE (6,77)
   77 FORMAT (' ENTER NPROC AND MPRINT (32  120).')
      READ (5,1001) NPROC,MPRINT
 1001 FORMAT (I2,I5)
      WRITE (6,78) NPROC
   78 FORMAT(///,'NPROC = ',I5,///)
C
C             INITIALIZE DATA FOR PROBLEM.
C
      CALL INIT
C
      CALL PRINTIT (6)
C
      CALL INTIME
      CALL CLOCK (ISTART)
C
      CALL SETE (DONE)
      CALL SETE (KOUNT)
      IBARR (1) = NPROC
      IBARR (2) = 0
      FINISH = 0
C
      DO 10 I=1,NPROC-1
      CALL AWRITE (KOUNT,I)
      CALL CREATE (PARSUB)
   10 CONTINUE
      CALL AWRITE (KOUNT,NPROC)
      CALL PARSUB
C
      CALL WAITF (DONE)
C
      CALL OUTIME (6)
      STOP
      END
      SUBROUTINE PARSUB
```

```
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),

     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1  ,ISTART,ITMAX,MPRINT,EL,PCF
C
      COMMON /SYNC/ KOUNT,NPROC,DONE,IBARR(52),FINISH
C
C
      IPROC = IAREAD(KOUNT)
      NCYCLE = 0
C
C            TOP OF LOOP.
C
   90 CONTINUE

      NCYCLE = NCYCLE + 1
C
C             COMPUTE CAPITAL U, CAPITAL V, Z, AND H.
C
      CALL CAPUV (IPROC,NPROC)
C
      CALL BARRIER (IBARR)
C
C             PERIODIC CONTINUATION.
C
      CALL CONCAPS (IPROC,NPROC)
C
      CALL BARRIER (IBARR)
C
C             COMPUTE NEW U, V, AND P.
C
      CALL NEWUVP (IPROC,NPROC)
C
      CALL BARRIER (IBARR)
C
C             PERIODIC CONTINUATION.
C
      CALL CONNEW(IPROC,NPROC)
C
      IF (IPROC .EQ. 1) THEN
C
C            TEST FOR EXIT AND PRINT INTERMEDIATE RESULTS.
```

```
C
      IF (NCYCLE .GT. ITMAX) FINISH = 1
      CALL CLOCK (ITIME)
      TIME = 1.0E-7 * FLOAT(ITIME-ISTART)
      TYC = TIME/FLOAT(NCYCLE)
      IF (MOD(NCYCLE,MPRINT) .EQ. 0) WRITE (6,1000) NCYCLE,TIME,TCYC
 1000 FORMAT (' NCYCLE = ',I5,' TOTAL TIME =',E12.4,
     1   ' TIME PER CYCLE =',E12.4)
      IF (FINISH .EQ. 1) CALL AWRITE (DONE,1.)
      ENDIF
C
C             TIME SMOOTHING AND UPDATE FOR NEXT CYCLE.
C
      CALL BARRIER (IBARR)
C
      IF (FINISH .EQ. 1) GOTO 999
C
      IF (NCYCLE .GT. 1) THEN
              CALL SMOOTH (IPROC,NPROC)
              CALL BARRIER (IBARR)
              CALL CONSMO (IPROC,NPROC)
      ELSE
              IF (IPROC .EQ. 1) TDT = 2.0 * TDT
              DO 400 J=IPROC,NP1,NPROC
              DO 400 I=1,MP1
              UOLD(I,J) = U(I,J)
              VOLD(I,J) = V(I,J)
              POLD(I,J) = P(I,J)
              U(I,J) = UNEW(I,J)
              V(I,J) = VNEW(I,J)
              P(I,J) = PNEW(I,J)
  400         CONTINUE
      ENDIF
C
      IF (MOD(NCYCLE,MPRINT) .EG. 0) THEN
              CALL BARRIER (IBARR)
              IF (IPROC .EQ. 1) CALL PRINTIT(6)
      ENDIF
      CALL BARRIER (IBARR)
C
      GOTO 90
  999 CONTINUE
      RETURN
      END
C
      SUBROUTINE INIT
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),

     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
```

```
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C
C             INITIALIZE VALUES OF THE STREAM FUNCTION.
C
      DO 50 J=1,NP1
      DO 50 I=1,MP1
      PSI(I,J) = A*SIN((FLOAT(I)-.5)*DI) * SIN((FLOAT(J)-.5)*DJ)
      P(I,J)=PCF*(COS(2.0*FLOAT(I-1)*DI)+COS(2.0*FLOAT(J-1)*DJ))
     1                + 50000.0
   50 CONTINUE
C
C             INITIALIZE VELOCITIES.
C
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J) = -(PSI(I+1,J+1)-PSI(I+1,J))/DY
      V(I,J+1) = (PSI(I+1,J+1)-PSI(I,J+1))/DX
   60 CONTINUE
C
C             PERIODIC CONTINUATION.
C
      DO 70 J=1,N
      U(1,J) = U(M+1,J)
      V(M+1,J+1) = V(1,J+1)
   70 CONTINUE
C
      DO 80 I=1,M
      U(I+1,N+1) = U(I+1,1)
      V(I,1) = V(I,N+1)
   80 CONTINUE
C
      U(1,N+1) = U(M+1,1)
      V(M+1,1) = V(1,N+1)
C
C             INITIALIZE OLD VALUES.
C
      DO 86 J=1,NP1
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
   86 CONTINUE
C
      RETURN
      END
      SUBROUTINE CAPUV (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
```

```
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 100 J=IPROC,N,NPROC
      DO 100 I=1,M
      CU(I+1,J) = 0.5 * (P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = 0.5 * (P(I,J+1)+P(I,J))*V(I,J+1)
      Z(I+1,J+1) = 4.0 * ((V(I+1,J+1)-V(I,J+1))/DX-(U(I+1,J+1)
     1    -U(I+1,J)).DY) / (P(I,J)+P(I,J+1)+P(I+1,J)+P(I+1,J+1))
      H(I,J) = P(I,J)+0.25*(U(I+1,J)**2+U(I,J)**2+V(I,J+1)**2+V(I,J)**2)
  100 CONTINUE
C
      RETURN
      END

      SUBROUTINE CONCAPS (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 110 J=IPROC,N,NPROC
      CU(1,J) = CU(M+1,J)
      CV(M+1,J+1) = CV(1,J+1)
      Z(1,J+1) = Z(M+1,J+1)
      H(M+1,J) = H(1,J)
  110 CONTINUE
      DO 115 I=IPROC,M,NPROC
      CU(I+1,N+1) = CU(I+1,1)
      CV(I,1) = CV(I,N+1)
      Z(I+1,1) = Z(I+1,N+1)
      H(I,N+1) = H(I,1)
  115 CONTINUE

C
      IF (IPROC .EQ.1) THEN
              CU(1,N+1) = CU(M+1,1)
              CV(M+1,1) = CV(1,N+1)
              Z(1,1) = Z(M+1,N+1)
              H(M+1,N+1) = H(1,1)
      ENDIF
```

```
C
      RETURN
      END
      SUBROUTINE NEWUVP (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)

C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C
      DO 200 J=IPROC,N,NPROC
      DO 200 I=1,M
      UNEW(I+1,J) = UOLD(I+1,J)+TDT*
     1(0.125*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2+CV(I+1,J))-(H(I+1,J)-H(I,J))/DX)
      VNEW(I,J+1) = VOLD(I,J+1)-TDT*
     1(0.125*(Z(I+1,J+1,)+Z(I,J+1))*(CU(I+1,J+1)+CU(I,J+1(+CU(I,J)

     2+CU(I+1,J))-((H(I,J+1)-H(I,J))/DY)
      PNEW(I,J) = POLD(I,J)-TDT*
     1((CU(I+1,J)-CU(I,J))/DX + (CV(I,J+1)-CV(I,J))/DY)
  200 CONTINUE
C
      RETURN
      END
      SUBROUTINE CONNEW (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 210 J=IPROC,N,NPROC
      UNEW(1,J) = UNEW(M+1,J)
      VNEW(M+1,J+1) = VNEW(1,J+1)
      PNEW(M+1,J) = PNEW(1,J)
```

```
  210 CONTINUE
      DO 215 I=IPROC,M,NPROC
      UNEW(I+1,N+1) = UNEW(I+1,1)
      VNEW(I,1) = VNEW(I,N+1)
      PNEW(I,N+1) = PNEW(I,1)
  215 CONTINUE
C
      IF (IPROC .EQ. 1) THEN
              UNEW(1,N+1) = UNEW(M+1,1)
              VNEW(M+1,1) = VNEW(1,N+1)
              PNEW(M+1,N+1) = PNEW(1,1)
      ENDIF
C

      RETURN
      END
      SUBROUTINE SMOOTH (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 300 J=IPROC,N,NPROC
      DO 300 I=1,M
      UOLD(I,J) = U(I,J) + ALPHA*(UNEW(I,J)-2.0*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J) + ALPHA*(VNEW(I,J)-2.0*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J) + ALPHA*(PNEW(I,J)-2.0*V(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
C
      RETURN
      END

      SUBROUTINE CONSMO (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
```

```
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1  ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 100 J=IPROC,N,NPROC
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)

      SUBROUTINE CONSMO (IPROC,NPROC)
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1  ,ISTART,ITMAX,MPRINT,EL,PCF
C

      DO 100 J=IPROC,N,NPROC
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)
      V(M+1,J) = V(1,J)
      P(M+1,J) = P(1,J)
  100 CONTINUE
      DO 200 I=IPROC,M,NPROC
      UOLD(I,N+1) = UOLD(I,1)
      VOLD(I,N+1) = VOLD(I,1)
      POLD(I,N+1) = POLD(I,1)
      U(I,N+1) = U(I,1)
      V(I,N+1) = V(I,1)
      P(I,N+1) = P(I,1)
  200 CONTINUE IF (IPROC .EQ. 1) THEN
              UOLD(M+1,N+1) = UOLD(1,1)
              VOLD(M+1,N+1) = VOLD(1,1)
              POLD(M+1,N+1) = POLD(1,1)
              U(M+1,N+1) = U(1,1)
              V(M+1,N+1) = V(1,1)
              P(M+1,N+1) = P(1,1)
      ENDIF
      RETURN
      END
C
      PARAMETER (NDIM = 64)
      PARAMETER (NDP1 = NDIM + 1)
```

```
C
      COMMON /DATA/ U(NDP1,NDP1),V(NDP1,NDP1),P(NDP1,NDP1),
     1UNEW(NDP1,NDP1),VNEW(NDP1,NDP1),PNEW(NDP1,NDP1),
     2UOLD(NDP1,NDP1),VOLD(NDP1,NDP1),POLD(NDP1,NDP1),
     3CU(NDP1,NDP1),CV(NDP1,NDP1),Z(NDP1,NDP1),H(NDP1,NDP1),
     4PSI(NDP1,NDP1)
C
      COMMON /PARA/ A,TDT,DX,DY,ALPHA,M,N,MP1,NP1,PI,TPI,DI,DJ
     1   ,ISTART,ITMAX,MPRINT,EL,PCF
C
      SUBROUTINE PRINTIT (IOUT)

      WRITE (IOUT,1000) N,M,DX,DY,TDT,ALPHA
 1000 FORMAT ( 'N = ',I3,' M = ',I3,' DX = ',F8.0,' DY = ',F8.0
     1       ,' DT = ',F8.0,' ALPHA = ',F8.3)
      WRITE (IOUT,1001) (P(I,I),I=1,N)
      WRITE (IOUT,1001) (U(I,I),I=1,N)
      WRITE (IOUT,1001) (V(I,I),I=1,N)
 1001 FORMAT (///(8E15.6))
      RETURN
      END
```

APPENDIX D

The "tasked" multiprocessor program for the CRAY X-MP 2

```
      PROGRAM SHALOW
C
C     BENCHMARK WEATHER PREDICTION PROGRAM FOR COMPARING THE
C     PERFORMANCE OF CURRENT SUPERCOMPUTERS. THE MODEL IS
C     BASED OF THE PAPER - THE DYNAMICS OF FINITE-DIFFERENCE
C     MODELS OF THE SHALLOW-WATER EQUATIONS, BY ROBERT SADOURNY
C     J. ATM. SCIENCES, VOL 32, NO 4, APRIL 1975.
C
C     CODE BY PAUL N. SWARZTRAUBER, NATIONAL CENTER FOR
C     ATMOSPHERIC RESEARCH, BOULDER, CO, NOVEMBER 1984.
C
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA
      DIMENSION I100A(3),I100B(3),I200A(3),I200B(3),I300A(3),I300B(3)
      EXTERNAL L100,L200,L300
      REAL MFS100,MFS200,MFS300
C
C     NOTE BELOW THAT TWO DELTA T (TDT) IS SET TO DT ON THE FIRST
C     CYCLE AFTER WHICH IT IS RESET TO DT+DT
C
      I100A(1) = 3
      I100A(3) = 'I100A'
      I100B(1) = 3
      I100B(3) = 'I100B'
      I200A(1) = 3
      I200A(3) = 'I200A'
      I200B(1) = 3
      I200B(3) = 'I200B'
      I300A(1) = 3
      I300A(3) = 'I300A'
      I300B(1) = 3
      I300B(3) = 'I300B'
      DT = 90.
      TDT = DT
C
      DX = 1.E5
      DY = 1.E5
      A = 1.E6
      ALPHA = .001
      ITMAX = 1200
      MPRINT = 120
      M = 64
      N = 64
      MP1 = M+1
      NP1 = N+1
      EL = FLOAT(N)*DX
      PI = 4.*ATAN(1.)
      TPI = PI+PI
      DI = TPI/FLOAT(M)
      DJ = TPI/FLOAT(N)
      PCF = PI*PI*A*A/(EL*EL)
```

```
C
C     INITIAL VALUES OF THE STREAM FUNCTION AND P
C
      DO 50 J=1,NP1
      DO 50 I=1,MP1
      PSI(I,J) = A*SIN((FLOAT(I)-.5)*DI)*SIN((FLOAT(J)-.5)*DJ)
      P(I,J) = PCF*(COS(2.*FLOAT(I-1)*DI)
     1               +COS(2.*FLOAT(J-1)*DJ))+50000.
   50 CONTINUE
C
C     INITIALIZE VELOCITIES
C
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J) = -(PSI(I+1,J+1)-PSI(I+1;J))/DY
      V(I,J+1) = (PSI(I+1,J+1)-PSI(I,J+1))/DX
   60 CONTINUE
C
C     PERIODIC CONTINUATION
C
      DO 70 J=1,N
      U(1,J) = U(M+1,J)
      V(M+1,J+1) = V(1,J+1)
   70 CONTINUE
      DO 75 I=1,M
      U(I+1,N+1) = U(I+1,1)
      V(I,1) = V(I,N+1)
   75 CONTINUE
      U(1,N+1) = U(M+1,1)
      V(M+1,1) = V(1,N+1)
      DO 86 J=1,NP1
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
   86 CONTINUE
C
C     PRINT INITIAL VALUES
C
      WRITE(6,390) N,M,DX,DY,DT,ALPHA
  390 FORMAT(*1NUMBER OF POINTS IN THE X DIRECTION* I8/
     1       * NUMBER OF POINTS IN THE Y DIRECTION* I8/
     2       * GRID SPACING IN THE X DIRECTION    * F8.0/
     3       * GRID SPACING IN THE Y DIRECTION    * F8.0/
     4       * TIME STEP                          * F8.0/
     5       * TIME FILTER PARAMETER              * F8.3)
      MNMIN = MIN0(M,N)
      WRITE(6,391) (POLD(I,I),I=1,MNMIN)
  391 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,392) (UOLD(I,I),I=1,MNMIN)
  392 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF U * //(8E15.5))
      WRITE(6,393) (VOLD(I,I),I=1,MNMIN)
  393 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF V * //(8E15.6))
      TSTART = 9.5E-9*RTC(DUM)
```

```
      T300 = 1.
      TCYCH = 0.
      TIME = 0.
      NCYCLE = 0
   90 NCYCLE = NCYCLE + 1
      TCYC = 9.5E-9*RTC(DUM)
C
C     COMPUTE CAPITAL U, CAPITAL V, Z AND H
C
      T100 = 9.5E-9*RTC(DUM)
      CALL TSKSTART(I100A,L100,M,1,N/2)
      CALL TSKSTART(I100B,L100,M,N/2+1,N)
      CALL TSKWAIT(I100A)
      CALL TSKWAIT(I100B)
      T100 = 9.5E-9*RTC(DUM)-T100
C
C     PERIODIC CONTINUATION
C
      DO 110 J=1,N
      CU(1,J) = CU(M+1,J)
      CV(M+1,J+1) = CV(1,J+1)
      Z(1,J+1) = Z(M+1,J+1)
      H(M+1,J) = H(1,J)
  110 CONTINUE
      DO 115 I=1,M
      CU(I+1,N+1) = CU(I+1,1)
      CV(I,1) = CV(I,N+1)
      Z(I+1,1) = Z(I+1,N+1)
      H(I,N+1) = H(I,1)
  115 CONTINUE
      CU(1,N+1) = CU(M+1,1)
      CV(M+1,1) = CV(1,N+1)
      Z(1,1) = Z(M+1,N+1)
      H(M+1,N+1) = H(1,1)
C
C     COMPUTE NEW VALUES U,V AND P
C
      T200 = 9.5E-9*RTC(DUM)
      CALL TSKSTART(I200A,L200,M,1,N/2)
      CALL TSKSTART(I200B,L200,M,N/2+1,N)
      CALL TSKWAIT(I200A)
      CALL TSKWAIT(I200B)
C
      T200 = 9.5E-9*RTC(DUM)-T200
C
C     PERIODIC CONTINUATION
C
      DO 210 J=1,N
      UNEW(1,J) = UNEW(M+1,J)
      VNEW(M+1,J+1) = VNEW(1,J+1)
      PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
      DO 215 I=1,M
      UNEW(I+1,N+1) = UNEW(I+1,1)
```

```
      VNEW(I,1) = VNEW(I,N+1)
      PNEW(I,N+1) = PNEW(I,1)
  215 CONTINUE
      UNEW(1,N+1) = UNEW(M+1,1)
      VNEW(M+1,1) = VNEW(1,N+1)
      PNEW(M+1,N+1) = PNEW(1,1)
      IF(NCYCLE .GT. ITMAX) CALL EXIT
      TIME = TIME + DT
      TCYC = 9.5E-9*RTC(DUM) - TCYC
      IF(MOD(NCYCLE,MPRINT) .NE. 0) GO TO 370
      PTIME = TIME/3600.
      WRITE(6,350) NCYCLE,PTIME
  350 FORMAT(//* CYCLE NUMBER*I5* MODEL TIME IN  HOURS* F6.2)
      WRITE(6,355) (PNEW(I,I),I=1,MNMIN)
  355 FORMAT(/* DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,360) (UNEW(I,I),I=1,MNMIN)
  360 FORMAT(/* DIAGONAL ELEMENTS OF U * //(8E15.6))
      WRITE(6,365) (VNEW(I,I),I=1,MNMIN)
  365 FORMAT(/* DIAGONAL ELEMENTS OF V * //(8E15.6))
      MFS100 = 24.*M*N/T100/1.E6
      MFS200 = 26.*M*N/T200/1.E6
      MFS300 = 15.*M*N/T300/1.E6
      CTIME = 9.5E-9*RTC(DUM)-TSTART
      TCYC = CTIME/FLOAT(NCYCLE)
      WRITE(6,375) NCYCLE,CTIME,TCYCH,T100,MFS100,T200,MFS200,T300,
     1             MFS300
  375 FORMAT(* CYCLE NUMBER*I5* TOTAL COMPUTER TIME* E15.6
     1       * TIME PER CYCLE* E15.6 /
     2       * TIME AND MEGAFLOPS FOR LOOP 100 * E15.6,F6.1/
     3       * TIME AND MEGAFLOPS FOR LOOP 200 * E15.6,F6.1/
     4       * TIME AND MEGAFLOPS FOR LOOP 300 * E15.6,F6.1/ )
  370 IF(NCYCLE .LE. 1) GO TO 310
      CTCYC = 9.5E-9*RTC(DUM)
      T300 = 9.5E-9*RTC(DUM)
      CALL TSKSTART(I300A,L300,M,1,N/2)
      CALL TSKSTART(I300B,L300,M,N/2+1,N)
      CALL TSKWAIT(I300A)
      CALL TSKWAIT(I300B)
      T300 = 9.5E-9*RTC(DUM)-T300
C
C     PERIODIC CONTINUATION
C
      DO 320 J=1,N
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)
      V(M+1,J) = V(1,J)
      P(M+1,J) = P(1,J)
  320 CONTINUE
      DO 325 I=1,M
      UOLD(I,N+1) = UOLD(I,1)
      VOLD(I,N+1) = VOLD(I,1)
      POLD(I,N+1) = POLD(I,1)
```

```
      U(I,N+1) = U(I,1)
      V(I,N+1) = V(I,1)
      P(I,N+1) = P(I,1)
  325 CONTINUE
      UOLD(M+1,N+1) = UOLD(1,1)
      VOLD(M+1,N+1) = VOLD(1,1)
      POLD(M+1,N+1) = POLD(1,1)
      U(M+1,N+1) = U(1,1)
      V(M+1,N+1) = V(1,1)
      P(M+1,N+1) = P(1,1)
      TCYC = TCYC + 9.5E-9*RTC(DUM) - CTCYC
      TCYCH = TCYC
      GO TO 90
  310 TDT = TDT+TDT
      DO 400 J=1,NP1
      DO 400 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  400 CONTINUE
      GO TO 90
      END
      SUBROUTINE L100(M,NS,NF)
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA
      FSDX = 4./DX
      FSDY = 4./DY
      DO 100 J=NS,NF
      DO 100 I=1,M
      CU(I+1,J) = .5*(P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = .5*(P(I,J+1)+P(I,J))*V(I,J+1)
      Z(I+1,J+1) = (FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1)
     1          -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
      H(I,J) = P(I,J)+.25*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
     1               +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
  100 CONTINUE
      RETURN
      END
      SUBROUTINE L200(M,NS,NF)
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA
      TDTS8 = TDT/8.
      TDTSDX = TDT/DX
      TDTSDY = TDT/DY
      DO 200 J=NS,NF
      DO 200 I=1,M
      UNEW(I+1,J) = UOLD(I+1,J)+
```

```
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2       +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
      VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1       *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2       -TDTSDY*(H(I,J+1)-H(I,J))
      PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
     1       -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
      RETURN
      END
      SUBROUTINE L300(M,NS,NF)
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA
      DO 300 J=NS,NF
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      RETURN
      END
```

APPENDIX E

The "events" multiprocesor program for the CRAY X-MP 2

```
      PROGRAM SHALOW
C
C     BENCHMARK WEATHER PREDICTION PROGRAM FOR COMPARING THE
C     PERFORMANCE OF CURRENT SUPERCOMPUTERS. THE MODEL IS
C     BASED OF THE PAPER - THE DYNAMICS OF FINITE-DIFFERENCE
C     MODELS OF THE SHALLOW-WATER EQUATIONS, BY ROBERT SADOURNY
C     J. ATM. SCIENCES, VOL 32, NO 4, APRIL 1975.
C
C     CODE BY PAUL N. SWARZTRAUBER, NATIONAL CENTER FOR
C     ATMOSPHERIC RESEARCH, BOULDER, CO, NOVEMBER 1984
C
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA,ITMAX
     4         ,IV1,IV2,IV3,IV4,IV5,IV6,IV7,IV8,IV9,IV10,IV11,IV12
      DIMENSION ITASK(3)
      EXTERNAL TASK
      REAL MFS100,MFS200,MFS300
      ITASK(1) = 3
      ITASK(3) = 'TASK'
      CALL EVASGN(IV1)
      CALL EVASGN(IV2)
      CALL EVASGN(IV3)
      CALL EVASGN(IV4)
      CALL EVASGN(IV5)
      CALL EVASGN(IV6)
      CALL EVASGN(IV7)
      CALL EVASGN(IV8)
      CALL EVASGN(IV9)
      CALL EVASGN(IV10)
      CALL EVASGN(IV11)
      CALL EVASGN(IV12)
C
C     NOTE BELOW THAT TWO DELTA T (TDT) IS SET TO DT ON THE FIRST
C     CYCLE AFTER WHICH IT IS RESET TO DT+DT
C
      DT = 90.
      TDT = DT
C
      DX = 1.E5
      DY = 1.E5
      A = 1.E6
      ALPHA = .001
      ITMAX = 1200
      MPRINT = 120
      M = 64
      N = 64
      MP1 = M+1
      NP1 = N+1
      NS2 = N/2
      NS2P1 = NS2+1
      EL = FLOAT(N)*DX
```

```
      PI = 4.*ATAN(1.)
      TPI = PI+PI
      DI = TPI/FLOAT(M)
      DJ = TPI/FLOAT(N)
      PCF = PI*PI*A*A/(EL*EL)
C
C     INITIAL VALUES OF THE STREAM FUNCTION AND P
C
      DO 50 J=1,NP1
      DO 50 I=1,MP1
      PSI(I,J) = A*SIN((FLOAT(I)-.5)*DI)*SIN((FLOAT(J)-.5)*DJ)
      P(I,J) = PCF*(COS(2.*FLOAT(I-1)*DI)
     1                +COS(2.*FLOAT(J-1)*DJ))+50000.
   50 CONTINUE
C
C     INITIALIZE VELOCITIES
C
      DO 60 J=1,N
      DO 60 I=1,M
      U(I+1,J) = -(PSI(I+1,J+1)-PSI(I+1,J))/DY
      V(I,J+1) = (PSI(I+1,J+1)-PSI(I,J+1))/DX
   60 CONTINUE
C
C     PERIODIC CONTINUATION
      DO 70 J=1,N
      U(1,J) = U(M+1,J)
      V(M+1,J+1) = V(1,J+1)
   70 CONTINUE
      DO 75 I=1,M
      U(I+1,N+1) = U(I+1,1)
      V(I,1) = V(I,N+1)
   75 CONTINUE
      U(1,N+1) = U(M+1,1)
      V(M+1,1) = V(1,N+1)
      DO 86 J=1,NP1
      DO 86 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
   86 CONTINUE
C
C     PRINT INITIAL VALUES
C
      WRITE(6,390) N,M,DX,DY,DT,ALPHA
  390 FORMAT(*1NUMBER OF POINTS IN THE X DIRECTION* I8/
     1       * NUMBER OF POINTS IN THE Y DIRECTION* I8/
     2       * GRID SPACING IN THE X DIRECTION    * F8.0/
     3       * GRID SPACING IN THE Y DIRECTION    * F8.0/
     4       * TIME STEP                          * F8.0/
     5       * TIME FILTER PARAMETER              * F8.3)
      MNMIN = MIN0(M,N)
      WRITE(6,391) (POLD(I,I),I=1,MNMIN)
  391 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,392) (UOLD(I,I),I=1,MNMIN)
```

```
  392 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF U * //(8E15.5))
      WRITE(6,393) (VOLD(I,I),I=1,MNMIN)
  393 FORMAT(/* INITIAL DIAGONAL ELEMENTS OF V * //(8E15.6))
      TSTART = 9.5E-9*RTC(DUM)
      T300 = 1.
      TCYCH = 0.
      TIME = 0.
      NCYCLE = 0
      CALL TSKSTART(ITASK,TASK,M,NS2P1,N)
   90 NCYCLE = NCYCLE + 1
      TCYC = 9.5E-9*RTC(DUM)
C
C     COMPUTE CAPITAL U, CAPITAL V, Z AND H
C
      T100 = 9.5E-9*RTC(DUM)
      CALL EVPOST(IV1)
      CALL EVWAIT(IV7)
      CALL EVCLEAR(IV7)
      FSDX = 4./DX
      FSDY = 4./DY
      DO 100 J=1,NS2
      DO 100 I=1,M
      CU(I+1,J) = .5*(P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = .5*(P(I,J+1)+P(I,J))*V(I,J+1)
      Z(I+1,J+1) = (FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1)
     1          -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
      H(I,J) = P(I,J)+.25*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
     1               +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
  100 CONTINUE
      CALL EVPOST(IV2)
      CALL EVWAIT(IV8)
      CALL EVCLEAR(IV8)
      T100 = 9.5E-9*RTC(DUM)-T100
C
C     PERIODIC CONTINUATION
C
      DO 110 J=1,N
      CU(1,J) = CU(M+1,J)
      CV(M+1,J+1) = CV(1,J+1)
      Z(1,J+1) = Z(M+1,J+1)
      H(M+1,J) = H(1,J)
  110 CONTINUE
      DO 115 I=1,M
      CU(I+1,N+1) = CU(I+1,1)
      CV(I,1) = CV(I,N+1)
      Z(I+1,1) = Z(I+1,N+1)
      H(I,N+1) = H(I,1)
  115 CONTINUE
      CU(1,N+1) = CU(M+1,1)
      CV(M+1,1) = CV(1,N+1)
      Z(1,1) = Z(M+1,N+1)
      H(M+1,N+1) = H(1,1)
C
C     COMPUTE NEW VALUES U,V AND P
```

```
C
      T200 = 9.5E-9*RTC(DUM)
      CALL EVPOST(IV3)
      CALL EVWAIT(IV9)
      CALL EVCLEAR(IV9)
      TDTS8 = TDT/8.
      TDTSDX = TDT/DX
      TDTSDY = TDT/DY
      DO 200 J=1,NS2
      DO 200 I=1,M
      UNEW(I+1,J) = UOLD(I+1,J)+
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2       +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
      VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1       *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2       -TDTSDY*(H(I,J+1)-H(I,J))
      PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
     1       -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
      CALL EVPOST(IV4)
      CALL EVWAIT(IV10)
      CALL EVCLEAR(IV10)
      T200 = 9.5E-9*RTC(DUM)-T200
C
C     PERIODIC CONTINUATION
C
      DO 210 J=1,N
      UNEW(1,J) = UNEW(M+1,J)
      VNEW(M+1,J+1) = VNEW(1,J+1)
      PNEW(M+1,J) = PNEW(1,J)
  210 CONTINUE
      DO 215 I=1,M
      UNEW(I+1,N+1) = UNEW(I+1,1)
      VNEW(I,1) = VNEW(I,N+1)
      PNEW(I,N+1) = PNEW(I,1)
  215 CONTINUE
      UNEW(1,N+1) = UNEW(M+1,1)
      VNEW(M+1,1) = VNEW(1,N+1)
      PNEW(M+1,N+1) = PNEW(1,1)
      IF(NCYCLE .GT. ITMAX) CALL EXIT
      TIME = TIME + DT
      TCYC = 9.5E-9*RTC(DUM) - TCYC
      IF(MOD(NCYCLE,MPRINT) .NE. 0) GO TO 370
      PTIME = TIME/3600.
      WRITE(6,350) NCYCLE,PTIME
  350 FORMAT(//* CYCLE NUMBER*I5* MODEL TIME IN  HOURS* F6.2)
      WRITE(6,355) (PNEW(I,I),I=1,MNMIN)
  355 FORMAT(/* DIAGONAL ELEMENTS OF P * //(8E15.6))
      WRITE(6,360) (UNEW(I,I),I=1,MNMIN)
  360 FORMAT(/* DIAGONAL ELEMENTS OF U * //(8E15.6))
      WRITE(6,365) (VNEW(I,I),I=1,MNMIN)
  365 FORMAT(/* DIAGONAL ELEMENTS OF V * //(8E15.6))
      MFS100 = 24.*M*N/T100/1.E6
      MFS200 = 26.*M*N/T200/1.E6
```

```
      MFS300 = 15.*M*N/T300/1.E6
      CTIME = 9.5E-9*RTC(DUM)-TSTART
      WRITE(6,375) NCYCLE,CTIME,TCYCH,T100,MFS100,T200,MFS200,T300,
     1             MFS300
  375 FORMAT(* CYCLE NUMBER*I5* TOTAL COMPUTER TIME* E15.6
     1       * TIME PER CYCLE* E15.6 /
     2       * TIME AND MEGAFLOPS FOR LOOP 100 * E15.6,F6.1/
     3       * TIME AND MEGAFLOPS FOR LOOP 200 * E15.6,F6.1/
     4       * TIME AND MEGAFLOPS FOR LOOP 300 * E15.6,F6.1/ )
  370 IF(NCYCLE .LE. 1) GO TO 310
      CTCYC = 9.5E-9*RTC(DUM)
      T300 = 9.5E-9*RTC(DUM)
      CALL EVPOST(IV5)
      CALL EVWAIT(IV11)
      CALL EVCLEAR(IV11)
      DO 300 J=1,NS2
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      CALL EVPOST(IV6)
      CALL EVWAIT(IV12)
      CALL EVCLEAR(IV12)
      T300 = 9.5E-9*RTC(DUM)-T300
C
C     PERIODIC CONTINUATION
C
      DO 320 J=1,N
      UOLD(M+1,J) = UOLD(1,J)
      VOLD(M+1,J) = VOLD(1,J)
      POLD(M+1,J) = POLD(1,J)
      U(M+1,J) = U(1,J)
      V(M+1,J) = V(1,J)
      P(M+1,J) = P(1,J)
  320 CONTINUE
      DO 325 I=1,M
      UOLD(I,N+1) = UOLD(I,1)
      VOLD(I,N+1) = VOLD(I,1)
      POLD(I,N+1) = POLD(I,1)
      U(I,N+1) = U(I,1)
      V(I,N+1) = V(I,1)
      P(I,N+1) = P(I,1)
  325 CONTINUE
      UOLD(M+1,N+1) = UOLD(1,1)
      VOLD(M+1,N+1) = VOLD(1,1)
      POLD(M+1,N+1) = POLD(1,1)
      U(M+1,N+1) = U(1,1)
      V(M+1,N+1) = V(1,1)
      P(M+1,N+1) = P(1,1)
      TCYC = TCYC + 9.5E-9*RTC(DUM)-CTCYC
```

```
      TCYCH = TCYC
      GO TO 90
  310 TDT = TDT+TDT
      DO 400 J=1,NP1
      DO 400 I=1,MP1
      UOLD(I,J) = U(I,J)
      VOLD(I,J) = V(I,J)
      POLD(I,J) = P(I,J)
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  400 CONTINUE
      GO TO 90
      END
      SUBROUTINE TASK(M,NS,NF)
      COMMON U(65,65),V(65,65),P(65,65),UNEW(65,65),VNEW(65,65),
     1          PNEW(65,65),UOLD(65,65),VOLD(65,65),POLD(65,65),
     2          CU(65,65),CV(65,65),Z(65,65),H(65,65),PSI(65,65)
     3         ,DT,TDT,DX,DY,ALPHA,ITMAX
     4         ,IV1,IV2,IV3,IV4,IV5,IV6,IV7,IV8,IV9,IV10,IV11,IV12
      NCYCLE = 0
   90 NCYCLE = NCYCLE+1
      CALL EVPOST(IV7)
      CALL EVWAIT(IV1)
      CALL EVCLEAR(IV1)
      FSDX =4./DX
      FSDY= 4./DY
      DO 100 J=NS,NF
      DO 100 I=1,M
      CU(I+1,J) = .5*(P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = .5*(P(I,J+1)+P(I,J))*V(I,J+1)
      Z(I+1,J+1) = (FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1)
     1          -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
      H(I,J) = P(I,J)+.25*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
     1               +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
  100 CONTINUE
      CALL EVPOST(IV8)
      CALL EVWAIT(IV2)
      CALL EVCLEAR(IV2)
      CALL EVPOST(IV9)
      CALL EVWAIT(IV3)
      CALL EVCLEAR(IV3)
      TDTS8 = TDT/8.
      TDTSDX = TDT/DX
      TDTSDY = TDT/DY
      DO 200 J=NS,NF
      DO 200 I=1,M
      UNEW(I+1,J) = UOLD(I+1,J)+
     1    TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
     2       +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
      VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(I,J+1))
     1       *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
     2       -TDTSDY*(H(I,J+1)-H(I,J))
      PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
```

```
     1       -TDTSDY*(CV(I,J+1)-CV(I,J))
  200 CONTINUE
      CALL EVPOST(IV10)
      CALL EVWAIT(IV4)
      CALL EVCLEAR(IV4)
      IF(NCYCLE.GT.ITMAX) GO TO 55
      IF(NCYCLE .LE. 1) GO TO 90
      CALL EVPOST(IV11)
      CALL EVWAIT(IV5)
      CALL EVCLEAR(IV5)
      DO 300 J=NS,NF
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      CALL EVPOST(IV12)
      CALL EVWAIT(IV6)
      CALL EVCLEAR(IV6)
      GO TO 90
   55 RETURN
      END
```

APPENDIX F

The following program is an implementation of the "truncated" Stockham FFT for multiple sequences on the CRAY X-MP 2

```
      SUBROUTINE MSTOCK(IS,LI,LJ,C,WORK)
C
C     THE STOCKHAM AUTOSORT FFT FOR MULTIPLE TRANSFORMS
C     MULTIPROCESSOR VERSION FOR THE CRAY-XMP-2
C
C     DEFINE M=2**LI AND N=2**LJ THEN M TRANSFORMS OF LENGTH N
C     ARE COMPUTED. THE TRANSFORMS ARE COMPUTED IN THE DIRECTION
C     OF THE SECOND INDEX OF THE TWO DIMENSIONAL ARRAY C.
C     THE FIRST DIMENSION OF THE ARRAY C MUST BE EQUAL TO M.
C     THAT IS, THE SEQUENCES MUST BE STORED CONSECUTIVELY.
C
C     IS = -1  FORWARD TRANSFORM
C     IS =  1  BACKWARD TRANSFORM
C
      COMPLEX C(1),WORK(1)
      DIMENSION IVT(2),ITASK(2)
      EXTERNAL TASK
      ITASK(1) = 2
      CALL EVASGN(IVT(1))
      CALL EVASGN(IVT(2))
      CALL TSKSTART(ITASK,TASK,IS,LI,LJ,C,WORK,IVT)
      N = 2**(LJ+LI)
      L01 = 0
      DO 100 L=1,LJ
      LS=2**(L-1)
      NS = N/(LS+LS)
      L01 = 1-L01
      IF(L01 .EQ. 0) GO TO 102
      CALL STOCKH(IS,LS,NS,C,WORK)
      GO TO 103
  102 CALL STOCKH(IS,LS,NS,WORK,C)
  103 CALL EVPOST(IVT(1))
      CALL EVWAIT(IVT(2))
      CALL EVCLEAR(IVT(2))
  100 CONTINUE
      IF(L01 .EQ. 0) RETURN
      DO 101 I=1,N
      C(I) = WORK(I)
  101 CONTINUE
      RETURN
      END
      SUBROUTINE STOCKH(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE = FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA = CMPLX(COS(ANGLE),SIN(ANGLE))
      NS2 = NS/2
      DO 200 J=1,NS2
      OMEGK = 1.
      DO 200 I=1,LS
      WYK = OMEGK*C(J,2,I)
      CH(J,I,1) = C(J,1,I)+WYK
```

```
      CH(J,I,2) = C(J,1,I)-WYK
      OMEGK = OMEGA*OMEGK
  200 CONTINUE
      RETURN
      END
      SUBROUTINE TASK(IS,LI,LJ,C,WORK,IVT)
      COMPLEX C(1),WORK(1)
      DIMENSION IVT(1)
      N = 2**(LJ+LI)
      L01 = 0
      DO 100 L=1,LJ
      LS=2**(L-1)
      NS = N/(LS+LS)
      L01 = 1-L01
      IF(L01 .EQ. 0) GO TO 102
      CALL STOCKG(IS,LS,NS,C,WORK)
      GO TO 103
  102 CALL STOCKG(IS,LS,NS,WORK,C)
  103 CALL EVPOST(IVT(2))
      CALL EVWAIT(IVT(1))
      CALL EVCLEAR(IVT(1))
  100 CONTINUE
      RETURN
      END
      SUBROUTINE STOCKG(IS,LS,NS,C,CH)
      COMPLEX OMEGA,OMEGK,WYK,C(NS,2,LS),CH(NS,LS,2)
      ANGLE = FLOAT(IS)*4.*ATAN(1.)/FLOAT(LS)
      OMEGA = CMPLX(COS(ANGLE),SIN(ANGLE))
      NS2P1 = NS/2+1
      DO 200 J=NS2P1,NS
      OMEGK = 1.
      DO 200 I=1,LS
      WYK = OMEGK*C(J,2,I)
      CH(J,I,1) = C(J,1,I)+WYK
      CH(J,I,2) = C(J,1,I)-WYK
      OMEGK = OMEGA*OMEGK
  200 CONTINUE
      RETURN
      END
```