Performance of a Three Dimensional Hydrodynamic Model on a Range of Parallel Computers

Dr Mike Ashworth[1] and Dr Alan M. Davies[2]

[1]NERC Computer Services, Bidston Observatory, Birkenhead L43 7RA, UK
[2]Proudman Oceanographic Laboratory, Bidston Observatory, Birkenhead L43 7RA, UK

Abstract

A three dimensional shallow sea model is briefly described in which the three dimensional flow field in a shallow sea region is represented using a finite difference grid in the horizontal and a spectral expansion in the vertical. A horizontal domain decomposition is employed, in which each processor works on a patch of sea and communicates boundary values with neighbouring processors as required. Performance characteristics of this code are presented using computers with a range of parallel architectures, including shared memory vector-parallel, distributed memory message passing, and data parallel. Whereas good parallel efficiency is readily obtained, the performance on most highly parallel computers is limited by the performance of the individual processors.

1: Introduction

With the significant rise in computing power over the last ten years, there has been increased activity in the development of three dimensional sea models, aimed at simulating current patterns in shallow sea regions, The ability to accurately predict these currents is particularly important in a wide range of pollution problems.

It is becoming increasingly clear that parallel processing is the best way of satisfying the future demands of these and other applications. However, to date parallel systems have not proved very attractive for large scale environmental modelling projects in the UK. This paper describes one of a number of activities which aims to develop parallel algorithms and evaluate codes on a range of parallel architectures.

2: Description of the POLMP model

The Proudman Oceanographic Laboratory Multiprocessing Program (POLMP) is a three-dimensional hydrodynamic shallow sea model which has been formulated to run efficiently on a range of modern parallel computers. The code was developed using a set of portable programming conventions based upon standard Fortran 77. The hydrodynamic partial differential equations are solved using a mixed explicit/implicit forward time integration scheme. The explicit component corresponds to a horizontal finite difference scheme (figure 1) and the implicit to an expansion in terms of functions in the vertical yielding a continuous current profile from sea surface to sea bed [2,3].

The model computes the wind induced flow in a closed rectangular basin. The facility exists for the inclusion of a number of arbitrary land areas, though none of the results presented here contain land. Recently more physical

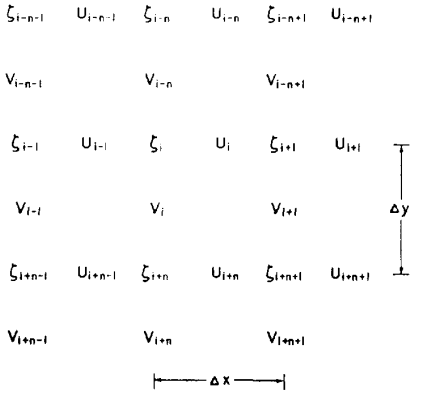

Figure 1. The staggered grid finite difference scheme. ζ is the free surface elevation, u and v the x- and y-components of velocity respectively,

1066-6192/92 $3.00 © 1992 IEEE

processes have been incorporated within this type of model giving rise to a larger computational task at each grid point, a trend favouring the solution on more highly parallel computers.

We will briefly describe the solution of the linear (for clarity) hydrodynamic equations. The working equations in sigma co-ordinates $\sigma=z/h$ are given by

$$\frac{\partial \zeta}{\partial t} + \frac{\partial}{\partial x}\left(h\int_0^1 u\,d\sigma\right) + \frac{\partial}{\partial y}\left(h\int_0^1 v\,d\sigma\right) = 0$$

$$\frac{\partial u}{\partial t} - \gamma v = -g\frac{\partial \zeta}{\partial x} + \frac{1}{h^2}\frac{\partial}{\partial \sigma}\left(\mu\frac{\partial u}{\partial \sigma}\right)$$

$$\frac{\partial v}{\partial t} + \gamma u = -g\frac{\partial \zeta}{\partial y} + \frac{1}{h^2}\frac{\partial}{\partial \sigma}\left(\mu\frac{\partial v}{\partial \sigma}\right)$$

In these equations, t denotes time, x, y and z are Cartesian co-ordinates and u, v are the x- and y-components of velocity respectively. The acceleration due to gravity, g, and the geostrophic coefficient γ are taken as constant, with μ the vertical eddy viscosity, ζ the free surface elevation and h the mean water depth.

The functions used in the vertical are arbitrary, although the computational advantages of using eigenfunctions (modes) of the eddy viscosity profile have been demonstrated [2,5]. Recently Davies [3] has shown that by using a mixed basis set in which the modal expansion is enhanced by an additional function (an "enhanced" spectral approach) an improved rate of convergence over the "classical" approach can be obtained with associated saving in computer time and memory.

3: Requirements for computation and communication

Each timestep in the forward time integration of the model, involves successive updates to the three fields, the ζ field, the u field and the v field. New field values computed in each update are used in the subsequent calculations. The new ζ_i values depend on ζ_i, u_i, u_{i-1}, v_i, and v_{i-n}, as can be seen in figure 1. Similarly, the new u_i values are obtained from u_i, ζ_i, ζ_{i-1}, v_i, v_{i+1}, v_{i-n}, v_{i-n+1}, and the new v_i values from v_i, ζ_i, ζ_{i+n}, u_i, u_{i-1}, u_{i+n}, u_{i+n-1}, where n is the number of grid points in the x-direction.. The calculations require 7, 11 and 11 floating point operations respectively, making a total of 29 flops per grid point per time step, of which 18 are add or subtract operations and 11 are multiplies. Thus, if a processor relies on overlapping adds with multiplies to obtain peak performance, as most do, we can only expect to reach 81% of peak, simply due to the imbalance in the operation count. The actual flop count used to calculate Megaflop rates in POLMP is slightly less than this because it takes into account boundary effects. Each inner loop vectorizes, using unit stride vectors over the full range of the domain, ensuring that vector lengths are long and high efficiency is therefore obtained on vector processors.

We can imagine that the data are mapped onto a processor array such that each grid point resides on a different processor, but with the fields aligned so that ζ_i, u_i and v_i are on the same processor. In this case processor$_i$ requires data points v_{i-n} from the north, ζ_{i+n} and u_{i+n} from the south, u_{i-1} from the west, ζ_{i+1} and v_{i+i} from the east, v_{i-n+1} from the north-east and u_{i+n-1} from the south-west. This is a total of eight words from six directions per grid point per time step. Clearly this communication rate is a maximum, and larger data arrays or smaller processor arrays which result in processors working on a patch of sea rather than on a single point, will reduce inter-processor communications.

The serial implementation of the model described above contains the following computational kernel, represented in pseudo-code.

```
do for all timesteps
  do j for all modes
    do i for all gridpoints
      update ζ(i) from u(i,j), v(i,j),
           u(i-1,j),  v(i-n,j)
    enddo
  enddo
  do j for all modes
    do i for all gridpoints
      update u(i,j) from ζ(i), ζ(i+1), v(i,j),
           v(i+1,j), v(i-n,j), v(i-n+1,j)
    enddo
  enddo
  do j for all modes
    do i for all gridpoints
      update v(i,j) from ζ(i), ζ(i+n), u(i,j),
           u(i-1,j), u(i+n,j), u(i+n-1,j)
    enddo
  enddo
enddo
```

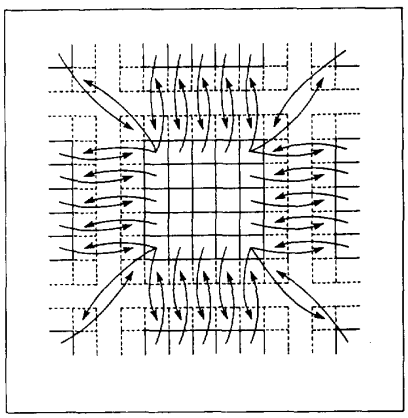

Figure 2. Horizontal decomposition of the domain into sub-domains (solid) with guard bands (dashed). Arrows show the data transfers from the central sub-domain and to its guard band.

4: Partitioning the Problem

The most natural partitioning scheme for finite difference problems is to partition the horizontal domain between the processors, for example [6,8]. For the five-point finite difference operator used here, this results in the decomposition shown in figure 2. Each processor works on a sub-domain and maintains in a guard band around its own data a copy of the neighbouring values from adjacent sub-domains. After each update within each timestep, certain data values are exchanged to keep the data in the guard band up-to-date. By the time the timestep is complete, exchanges of data in six directions will have taken place. Clearly, as the size of the sub-domain is increased the communications overhead will become relatively less important.

5: Shared memory parallel implementation

The horizontal partitioning method has been implemented in the following way for shared memory parallel machines. In computational terms, this is a coarse grain (macrotasking) approach, as opposed to the use of a fine grain approach using microtasking directives [5] to achieve parallelism over the vertical modes. The master process executes a control loop over the number of sub-domains, nsub, which initiates nsub-1 processes, each pointing to its own sub-domain, with the final sub-domain being left for the master. The structure for the kernel within the time stepping loop of each process is as follows.

update ζ
send ζ values to N and W
receive ζ values from E and S
update u
send u values to N, E and NE
receive u values from S, W and SW
update v
send v values to S, W and SW
receive v values from N, E and NE

In order for such an approach to be implemented, the explicit time integration method used here is required [2,3]. The application of a semi-implicit method using a sweep approach [11] would be more involved and in some cases impossible to implement efficiently.

On a shared memory multiprocessor, the sending of data is achieved by one process writing to the data array containing the other's sub-domain. During this phase of data exchange, each data location in the guard band of each sub-domain is written to by one, and only one, process. Therefore, the processes can write in parallel. The only safeguard which is required is that there is a synchronization point before and after the data exchange phase to ensure that it does not overlap with a computation phase. This is achieved by setting up a barrier at which each process waits until all processes have reached the barrier. Barriers are available on most shared memory multiprocessors.

The data arrays for each sub-domain are allocated in such a way that all data for a particular sub-domain are located sequentially in memory. This allows the inner loop for each update to be vectorized over the entire sub-domain with unit stride memory accesses, maintaining maximum efficiency on vector processors. Masking was used for land and boundary points, as this has been shown to be most generally effective [5], but alternative code using the strip-mining method was also included for use on scalar processors and for models where a significant fraction of the domain is land.

6: Distributed memory parallel implementation

The program structure required for distributed memory machines is very similar to that shown in the previous section for shared memory machines. A copy of the program runs on each processor, so that the control loop now only invokes a single process. The sending and receiving of data is now implemented by calls to a proprietary message passing library.

If the message passing interface allows explicit access to asynchronous communications, then computation may be overlapped with communications in the following way. For example for the ζ update:

update ζ at the sub-domain boundaries
send ζ values to N and W
initiate receives for ζ values from E and S
update ζ in the sub-domain interior
check receives have completed

The new boundary values are computed first, the new data sent and receives posted. Computation of the interior points then follows and by the time the receives are issued, communications should be complete. Clearly, the greater the size of the sub-domain, the greater the ratio of work between the interior and the boundary, and the greater the possibility for overlap in this way.

7: Data parallel implementation

When programming data parallel or SIMD processors, the data arrays must be mapped onto the array of processors. This mapping is communicated to the compiler by the use of compiler directives. For the POLMP code the horizontal dimensions were mapped across the processor array, with each gridpoint's spectral data being held within the processor memory. The compiler thus performs horizontal partitioning implicitly. If the number of gridpoints equals the number of processors, then clearly each processor is allocated one gridpoint. For larger grids each processor works on a patch of sea just as if the domain had been explicitly partitioned. The computational kernel was rewritten using Fortran 90 array syntax, which forms the interface between the programmer and the processor array on these computers.

8: Transarchitectural portability

In order to combine the ability to transport the code between machines of different architectures with the ease of maintenance that goes with having a single copy of the code, the preceding coding structures were combined in a single hybrid implementation. Details of the structure of the composite coding are given by Ashworth and Davies [1].

Standard Fortran 77 was used wherever possible. In certain areas, such as process spawning and message passing, machine dependent constructs must be used. Machine dependent sections of code are enabled and disabled using the ANSI standard C preprocessor to form the source file for a particular machine from the master copy. This preprocessor is available on most Unix systems, thus further enhancing the portability of the code.

9: Performance on the Cray vector multiprocessors

In this and subsequent sections, performance results are presented for a number of parallel machines. The problem size is governed by four parameters: *nx* and *ny*, the number of grid points in the horizontal finite difference scheme; *m*, the number of vertical modes in the spectral expansion; and *nts*, the number of time steps. The speed in Mflops is calculated from the number of floating point operations which are known to be required in the kernel of the program and is a simple function of the problem size.

A subset of the domain was printed out at the end of each run in order to check the correctness of the results. In production modelling runs all field arrays would need to be written out after every 100-1000 timesteps. Efficient implementation of this i/o requirement is an important issue which demands careful consideration and which, on most parallel machines, involves machine dependent coding.

For the Cray Y-MP and Cray Y-MP C90 shared memory vector multiprocessors, performance was measured using a grid of size 224 x 416 x 16 run for 3200 timesteps. The eight processor Cray Y-MP/8 gave 1495 Mflops which is 56% of the peak performance of 2667 Mflops. On a sixteen processor Cray Y-MP/16 C90, which with a clock speed of 4 ns has a peak performance of 16000 Mflops, the code achieved 7308 Mflops or 46% of peak.

10: Performance on the Intel iPSC/860

The Intel iPSC/860 is a distributed memory MIMD message-passing machine, using i860 processors rated at 80 peak Mflops each in single, 32-bit, precision. Horizontal partitioning was implemented using the native message passing library available on the machine. One instance of the code, the one running on node zero, was deemed to be the master process and handled the input of steering data, the division of the problem into sub-domains, the sending of steering data to the other, slave, processes and the collection of results at the end of the calculation. This last function, that of writing results, should in a full production model be distributed across the processing nodes using the concurrent file system which is available on the iPSC machines.

Despite the kernels of the code being highly vectorizable, the single node performance was initially poor, only achieving a rate of 2.7 Mflops using the Portland Group compiler (if77). By selecting the -Knoieee compiler switch, which runs the i860 in native floating point mode rather than according to the IEEE standard, an improvement to 14.7 Mflops was realised. The change of floating point format is not significant to the results produced from this code. This performance is still well short of the 80 Mflops peak. The shortfall is believed to be mainly due to the inadequacy of the off-chip memory bandwidth. Techniques for attaining higher performance from the i860 by programming in assembler have been described [9], but we do not consider this to be an attractive route for a modern environmental modelling project.

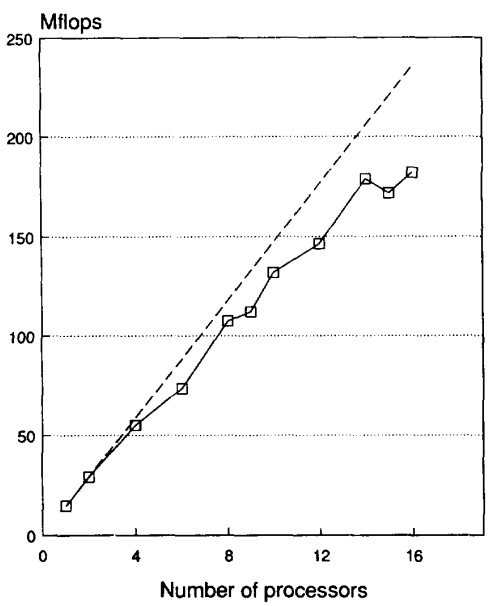

Figure 3. Scaled performance in Mflops of the POLMP model on the Intel iPSC/860 as a function of the number of processors used. The dashed line shows the ideal performance obtained by scaling the single node performance. Each processor works on a sea area of 128 x 64 with 16 vertical modes and the model was run for 200 timesteps.

Using 16 nodes of the Intel iPSC/860 the code ran at 161 Mflops for a problem size of 256 x 256 x 16 run for 200 timesteps. A larger problem of 512 x 512 x 16 achieved 215 Mflops. With a peak performance of 1280 Mflops these figures correspond to 13% and 17% of peak respectively.

In addition to running large problems on 16 nodes, the parallel performance of the code was tested using a scaled performance test. The difficulty with using a single problem size for a wide range of numbers of processors is well-known. A large problem will not fit within the memory of a single node, and a small problem will not demonstrate the full performance available from a large number of nodes. It is therefore sensible to fix the grid size *on each node* so that, as the number of processors is increased, a larger and larger problem size is being solved. This corresponds to the likely usage of the machine, as scientists will always want to run the largest problem that will fit onto a given size of machine.

Figure 3 shows the performance using a sub-domain size of 128 x 64 on each processor node with 16 vertical modes run for 200 timesteps. The program ran for about 60 seconds independent of the number of nodes, and covered domains up to 512 x 256 in size. The results show a good speed-up with increasing number of nodes, but with the efficiency falling off to about 80% at 16 nodes.

11: Performance on the MasPar MP-1104

The MasPar MP-1104 is an SIMD, data parallel computer with 4096 RISC-like floating point processors. By experimentation it was found that best performance was obtained when communications was effected using array sections rather than the Fortran 90 explicit functions CSHIFT or EOSHIFT. Code with fixed, statically allocated arrays performed better than using automatic

array allocation, but has the disadvantage that a program must be compiled for a particular problem size.

A well-known feature of data parallel computers is the wide variation in performance depending upon how the data arrays map onto the processor array. Figure 4 shows the performance of the POLMP kernel in Mflops plotted against the size of the side of a square domain. The graph is a sawtooth with performance rising steeply to a maximum at 64 x 64, when the data arrays fit perfectly onto the processor array (i.e. there is one grid point per processor). For larger data sizes, the speed drops off and rises to a peak at 128 x 128 and then to another near 192 x 192. However, the height of the peaks appears to be diminishing, with no problem size performing better than the 64 x 64. Sawtooth behaviour is also seen on vector processors as performance varies with vector length, but in that case the overall performance asymptotically approaches the peak speed.

The full POLMP code was built into two versions, one with a fixed, statically allocated array size of 64 x 64 x 16 and the other 128 x 128 x 16. As on the Intel, output of results was disabled. Passing the arrays back to the front-end was prohibitively expensive, and it was felt that a proper implementation of the application would use parallel input/output direct from the array processor to disk.

The data arrays are of size (nx+2) x (ny+2), due to the addition of a boundary strip in each direction, so the best performance is attained by setting the problem size to 62 x 62 and 126 x 126. The maximum performance achieved for the smaller problem was 155 Mflops, when running with 16 modes for 20000 timesteps. The larger problem only reaches 122 Mflops under the same conditions. With an advertised peak performance of 300 Mflops these results represent 52% and 41% of peak respectively.

12: Performance on the Meiko i860 Computing Surface

The Meiko Computing Surface is a distributed memory message passing MIMD computer. The machine used for these measurements was equipped with i860 compute nodes. Implementation of POLMP was effected by using the Intel version of the code together with a compatibility library which forms an interface between the Intel message passing library calls and Meiko's CS Tools environment. For a problem of 256 x 256 x 16 run for 200 timesteps on 16 nodes of the Computing Surface the code ran at 226 Mflops. The larger problem of 512 x 512 x 16 achieved 279 Mflops. With a peak rating of 1280 Mflops these speeds correspond to 18% and 22% of peak performance respectively. These results indicate that the overhead in introducing the extra layer of software were small.

13: Performance on the CM-200 Connection Machine

The CM-200 Connection Machine is an SIMD, data parallel, computer. We used a machine with 16384 single-bit processors, which are supported by 512 32-bit floating point accelerator units. Originally CSHIFT functions were used to generate the shifted arrays needed for the finite difference stencil, but these calls were replaced with PSHIFT calls from the CMSSL Library, which allow communication to take place simultaneously in all four directions, thus making full use of the communications network. Jordan [7] describes how higher performance may be obtained for two-dimensional finite difference schemes by using the CM-Stencil Library. Output of results from the model was, again, omitted from the code as this would have required considerable additional effort in special coding.

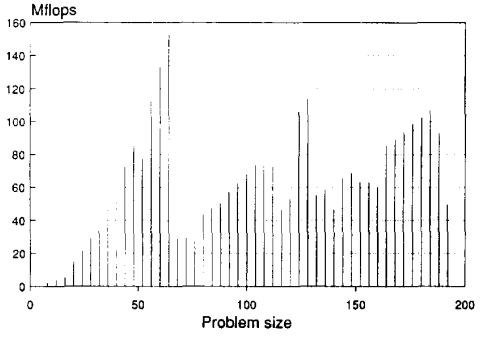

Figure 4. Performance in Mflops of the kernel of the hydrodynamic model on the MasPar MP-1104 as a function of the problem size. The problem size parameter is the size of a square domain with 16 vertical modes run for 200 timesteps.

As with the MasPar, there was a wide variation in performance with the size of the data arrays. Figure 5 shows the performance of the POLMP kernel on 8192 processors as a function of the problem size. *Nice* values such as 128 x 128, 192 x 192 and 256 x 256 achieve high performance, but intermediate values are slower by up to a factor of 40. Whereas it is understandable that arrays of size 257 x 257 should perform badly, as any division by a power-of-two number of processors will leave a small remainder and create a load imbalance, it was, on the face of it, surprising that 255 x 255 should also be poor. It is clear from other work on data parallel machines, for example [10], that *nice* problem sizes have been chosen for maximum performance and there is no reason for most applications why this restriction should be a disadvantage.

As for the MasPar, the data arrays are of size (nx+2) x (ny+2), so the best performance is attained by setting the problem size to 126 x 126, 254 x 254 and 510 x 510. The problems were run on 16384 processors with 16 modes and for 20000 timesteps. Performance for the three problem sizes was 597, 864 and 1249 Mflops corresponding to 7%, 11% and 16% respectively of the peak performance of 8000 Mflops.

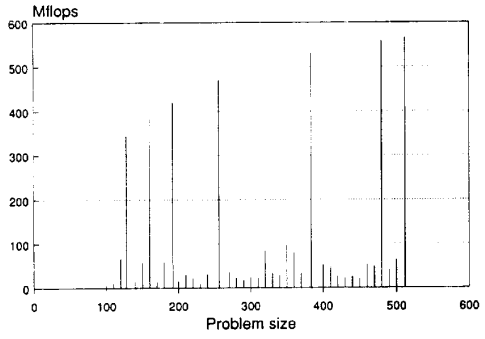

Figure 5. Performance in Mflops of the kernel of the hydrodynamic model on 8192 processors of the CM-200 Connection Machine as a function of the problem size. The problem size parameter is the size of a square domain with 16 vertical modes run for 200 timesteps.

14: Conclusions

In most cases there was no difficulty in obtaining a good parallel efficiency provided that a sufficiently large problem was used. When this is done and communications is overlapped with computation, the communications speed itself does not appear to be a limiting factor for most applications on most machines.

With a few notable exceptions, the fraction of the peak performance obtained with this code is in the range 7-22% . The MasPar gave a good fraction of peak and the Cray supercomputers demonstrated why vector multiprocessors have been the mainstay of supercomputing for the last two decades. The performance in the other cases was disappointing bearing in mind the high parallel efficiency and results almost entirely from poor performance of a single processor or processing element. There are several reasons for this. In some cases it is clear that the combination of compiler and RISC architecture does not produce as efficient code as a vector processor. It is possible that there may yet be improvements in this area or that parallel computer manufacturers may start to use vector processing nodes to good effect. For some machines the performance of computationally intensive parts of the code is limited by the bandwidth between the processor and its memory, the on-chip cache being too small to alleviate the problem.

15: Acknowledgements

I should like to express my thanks to staff at the SERC Daresbury Laboratory, Edinburgh Parallel Computing Centre and the Atlas Centre of the SERC Rutherford Appleton Laboratory and to Mike O'Neill of Cray (UK) Ltd, Vic Knight of MasPar (UK) Ltd, and Duncan Roweth of Meiko Ltd for help and assistance with the benchmarks.

16: References

1. Ashworth, M. and Davies, A.M., Restructuring three-dimensional hydrodynamic models for computers with low and high degrees of parallelism, in Parallel Computing '91, eds D.J.Evans, G.R.Joubert and H.Liddell (North Holland, 1992), 553-560.

2. Davies, A.M., Formulation of a linear three-dimensional hydrodynamic sea model using a Galerkin-eigenfunction method, International Journal for Numerical Methods in Fluids, 3, (1983) 33-60.

3. Davies, A.M., Solution of the 3D linear hydrodynamic equations using an enhanced eigenfunction approach, International Journal for Numerical Methods in Fluids, **13**, (1991) 235-250.

4. Davies, A.M., Grzonka R.G. and Stephens, C.V., A microtasked numerical model of seas on the RAL Cray X-MP, NERC Computing, No. 44, (1989) 32-36.

5. Davies, A.M., Grzonka, R.G. and Stephens, C.V., The implementation of hydrodynamic numerical sea models on the Cray X-MP, in Advances in Parallel Computing, Vol. 2, ed. D.J. Evans (JAI Press, 1992)

6. Harding, T., An oceanographic model on a transputer array, Internal report of the Department of Statistics and Computational Mathematics, University of Liverpool, UK (1992).

7. Jordan, K., Use of finite difference patterns to achieve high performance on the CM-2, Thinking Machines Corporation technical report TMC-194 (1992).

8. McBryan, O., A comparison of the Intel iPSC/860 and the Suprenum-1 parallel computers, Supercomputer, **41**, (1991) 6-17.

9. Purvis, W., Programming the i860, Parallelogram, part 1 Nov 1990, part 2 Jan 1991, part 3 Sep 1991.

10. Smith, R.D., Dukowicz, J.K. and Malone, R.C., Parallel ocean circulation modelling, Physica D in press (1992).

11. Wolf, J., A comparison of a semi-implicit with an explicit scheme in a three-dimensional hydrodynamic model, Continental Shelf Research, **2**, (1983) 215-229.