Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

E.D. de Goede

A computational model for three-dimensional shallow water flows on the ALLIANT FX/4

Department of Numerical Mathematics Report NM-R8910 May

A Computational Model for Three-Dimensional Shallow Water Flows on the ALLIANT FX/4

Erik D. de Goede
Centre for Mathematics and Computer Science
P.O.Box 4079, 1009AB Amsterdam, The Netherlands

Numerical methods for the simulation of three-dimensional water flows require a very great computational effort. Therefore, numerical methods should exploit the full potential of vector and parallel computers. In this contribution we investigate the efficiency of a numerical method on the ALLIANT FX/4.

1980 Mathematics Subject Classification: 65M10, 76D99.
Key Words & Phrases: three-dimensional shallow water equations, numerical methods, ALLIANT FX/4.
Note: This work was supported by the Ministry of Public Works (RWS).
Note: This report will be submitted for publication in the Journal Supercomputer.

Introduction

Flows in rivers, estuaries and seas can be described by mathematical models based on the shallow water equations (SWEs). In hydraulic engineering, these models are becoming more and more important. A well known application is made for the storm surge barrier in the mouth of the Oosterschelde (Eastern Scheldt), which is in the south-western part of The Netherlands. An accurate prediction of the water level is required to ensure that the barrier will be closed in time in case of extremely high water. Mathematical models are applied not only to estimate the water levels, but also the flow patterns. In many cases the flow exhibits a three-dimensional structure and contains variable densities in all directions. This is important in cases where a detailed dispersion of a pollutant is desired. However, the application of three-dimensional models requires a great computational effort, especially when a high resolution is needed. Therefore, it is necessary to construct numerical methods that are not only robust and accurate, but also efficient on vector and parallel computers. To obtain a computationally efficient method the VECPARCOMP project has been started. This project is a cooperation between the Data Processing Division of Rijkswaterstaat (Water Control and Public Works department) and the CWI (Centre for Mathematics and Computer Science).
This contribution deals with the efficient implementation on vector and parallel computers of a numerical method for the simplified three-dimensional shallow water equations. The second section contains the mathematical model and a brief description of the numerical method. This method results in the solution of a large number of tridiagonal systems. In the third section we describe two possible methods for the solution of these systems. The performance of these two methods has been tested on the ALLIANT FX/4 at CWI. For the most efficient method we also list the computation times on a number of computers (ranging from supercomputers to personal computers). The results clearly illustrate the necessity of supercomputers for such time-consuming computations.

Mathematical model

In this section we present the simplified three-dimensional shallow water equations [1,2]

$$\frac{\partial u}{\partial t} = fv - g\frac{\partial \zeta}{\partial x} + \frac{1}{\rho h^2}\frac{\partial}{\partial \sigma}\left(A^{\sigma}\frac{\partial u}{\partial \sigma}\right) \qquad (1)$$

$$\frac{\partial v}{\partial t} = -fu - g\frac{\partial \zeta}{\partial y} + \frac{1}{\rho h^2}\frac{\partial}{\partial \sigma}\left(A^\sigma \frac{\partial v}{\partial \sigma}\right) \qquad (2)$$

$$\frac{\partial \zeta}{\partial t} = -\frac{\partial}{\partial x}\left(h\int_0^1 u\,d\sigma\right) - \frac{\partial}{\partial y}\left(h\int_0^1 v\,d\sigma\right), \qquad (3)$$

where the following notation is used:

A^σ	vertical diffusion coefficient
f	Coriolis term
g	acceleration due to gravity
h	undisturbed depth of water
t	time
u,v	velocity components in x,y-direction
x,y,σ	a left-handed set of coordinates
ρ	density
ζ	elevation above undisturbed depth.

The equations (1) and (2) are the momentum equations and (3) denotes the continuity equation. In the vertical the domain is bounded by the bottom topography and the time-dependent water elevation. Here, system (1)-(3) has been transformed in the vertical into depth-following coordinates to ensure the domain to be constant in time. For the description of the domain, the initial conditions and boundary conditions we refer to [1,2].
To obtain a discrete system representing (1)-(3), the equations are discretized in space and time. Firstly, we apply a finite difference space discretization on a spatial grid that is staggered in both the horizontal and the vertical direction. The computational domain is covered by an $nx \cdot ny \cdot ns$ rectangular grid. E.g., $U_{i,j,k}$ denotes the approximation of the velocity u at the horizontal grid point (i,j) and layer k. Figure 1 shows the horizontal grid spacing.

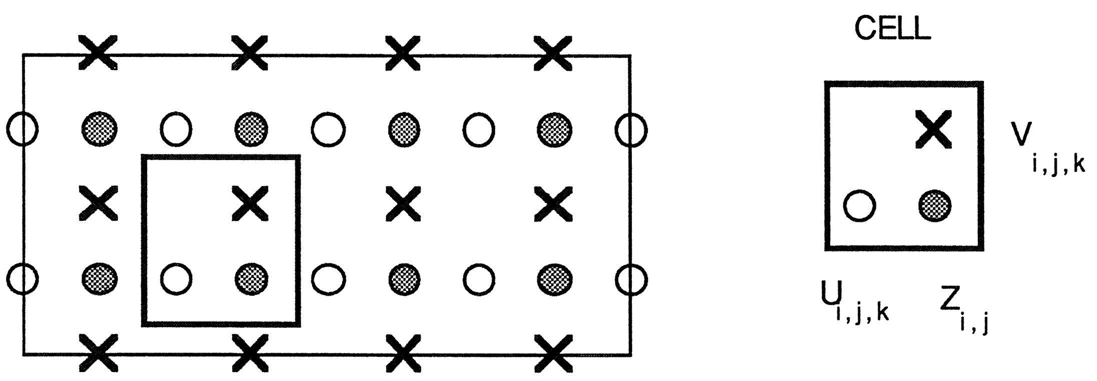

Fig. 1. Staggered grid in (x,y)-plane.

The main advantage of a staggered grid is that it reduces the computation time and the storage requirements with a factor 8 (in our case), while the accuracy is hardly influenced.
The time discretization is performed by a method developed in [2]. In the appendix this method is given in detail. Here, we describe this method in the following schematical form:

$$\begin{aligned} T\,U^{n+1} &= F_u(U^n, V^n, Z^n) \\ T\,V^{n+1} &= F_v(U^{n+1}, V^n, Z^n) \\ Z^{n+1} &= F_z(U^{n+1}, V^{n+1}, Z^n), \end{aligned} \qquad (4)$$

where $F_{u/v/z}$ contains for the corresponding component the computation of all terms except for the vertical diffusion term, which is included in matrix T. Matrix T is a block tridiagonal matrix with equal blocks. **U**, **V** and **Z** are grid functions approximating the velocities u and v and the water elevation ζ, respectively. The upper indices denote the time level. As shown in (4) the components are computed sequentially (firstly the **U**-, then the **V**- and finally the **Z**-component). This is advantageous for both the stability of the method and the storage requirements. The **Z**-component can be computed explicitly from the updated velocity components and from the 'old' **Z**-component. The implicit treatment of the vertical diffusion term results in a tridiagonal system for each (i,j)-point in the horizontal. Thus, nx·ny tridiagonal systems of dimension ns have to be solved for the velocity components **U** and **V**.
Method (4) possesses second-order accuracy in space and first-order accuracy in time. In [2] it is shown that the time step restriction for this method is

$$\tau < \frac{1}{\sqrt{gh}} \frac{1}{\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}}} \sqrt{1 + \frac{\tau}{2(\Delta\sigma)^2} \frac{A^\sigma}{\rho h^2}}, \tag{5}$$

where τ denotes the time step and Δx and Δy denote the horizontal mesh sizes and $\Delta\sigma$ the vertical mesh size.

Solution of the tridiagonal systems

For the solution of the tridiagonal systems the Gaussian Elimination (double sweep) method can be applied. Since this is a recursive method, it seems to be unattractive on vector and parallel computers. However, in our case a **large number** of tridiagonal systems have to be solved. Therefore, the systems can be solved in a vector-parallel mode on the ALLIANT FX/4. This means that each processor is executing vector instructions to compute a certain operation of the Gaussian Elimination method for all tridiagonal systems. This method, which we denote by method A, can be described schematically by

```
Method A (Gaussian Elimination method (GE))
for k=1,...,ns do   (in scalar mode)
  for j=1,...,ny
  for i=1,...,nx  do   (in vector-parallel mode)                 (A)
    perform some step of the GE method at layer k and point (i,j).
```

The loops with indices i and j are formed into a single DO-loop to obtain a more efficient code (loop collapsing). These iterations are executed in vector-parallel mode. For relatively small values of nx and ny we may expect a considerable overhead for this innermost DO-loop, which is due to vectorization and especially to synchronization. Therefore, we also examine methods that contain parallelism at a higher level. For method A the loop with index k is recursive and is therefore not suited for parallel execution. Several methods have been developed to reduce this recursion problem to smaller recursion problems. Here, we use a variant of Wang's method that has been developed for bidiagonal systems in [3]. A similar approach can be followed for tridiagonal systems. We briefly describe this method. Let us assume that ns can be factorized as ns = pq. Then, we write the tridiagonal system as a p by p block matrix, the elements of which are q by q matrices. Now, the off-diagonal elements on the p diagonal blocks are eliminated in parallel. This method, which we denote by method B, can be considered as a method in which the Gaussian Elimination method is applied in parallel for all p diagonal blocks. Moreover, for all tridiagonal systems this elimination is performed in vector mode. Schematically, this method reads

<u>Method B (variant of Wang's method (WANG))</u>

```
for k1=1,...,p  do   (in scalar-parallel mode)
 for k2=1,...,q  do
   for j=1,...,ny
   for i=1,...,nx  do  (in vector mode)                                   (B)
      perform some step of the WANG method at layer k2+(k1-1)q and point (i,j).
```

Here, the loop with index k2 is recursive. Method B requires more operations than method A (about 2.5 times as many).

Numerical illustration

In this section we give the results for a wind driven test problem [1,2]. We integrated over a period of 24 hours with a time step $\tau = 1200$ sec. The computations have been performed on a grid with $nx = 10$, $ny = 18$ and $ns = 24$. In this case, 180 tridiagonal systems of dimension 24 have to be solved. In Table 1 we list the computation times for the solution of the tridiagonal systems on the ALLIANT FX/4 for method A and B for various optimizations (- = no optimization, G = Global, V = Vector and P = Parallel). This mini-supercomputer consists of four vector processors. For method B we partitioned the 24 vertical layers into four blocks (i.e., $p = 4$ and $q = 6$). The blocks were solved in parallel. Moreover, for each block the computations were performed in vector mode.

Table 1 : Computation times on the ALLIANT FX/4 (in seconds).

		OPTIMALIZATION			
	# PROC.	(-)	(G)	(GV)	(GVP)
METHOD A	1	37.2	9.6	2.47	2.59
	2				1.36
	3				1.00
	4		9.6	2.47	0.87
METHOD B	1	87.7	17.3	4.91	4.95
	2				2.64
	3				2.47
	4		17.3	4.01	1.60

In scalar mode method A is about a factor 2.4 faster, which is in accordance with the number of operations. Also in vector mode, method A is better. Both methods have the same vectorization properties. Although method B requires about 2.5 times as many operations, the number of divisions is equal for both methods. Since divisions are more expensive than additions and multiplications we obtain a gain factor of 1.8 for method A. It is remarkable that for the (GV)-optimization we obtain different computation times on one and four processors for method B. It appears that some operations are done in parallel already. This is one of the few unexpected (unwanted) results that we encountered on the ALLIANT FX/4.

In parallel mode, we expected the smallest computation time for method B. Although method B requires more operations, we expected the synchronization overhead for method A to be relatively larger. However, it turns out that even for our relatively small test problem method A is faster. Thus, it can be concluded that the ALLIANT FX/4 allows parallel processing with a very low overhead. On four processors we obtain a speed-up of about three for method A.

Method A was developed for execution on the CDC CYBER 205 [2]. The entire code was structured in such a way that large vector lengths were obtained (of course depending on the size of the test

problem). This is very important on the CDC CYBER 205, because of its relatively large start-up time for DO-loops. It took very limited effort to optimize this code on the ALLIANT FX/4. Moreover, it turns out that it is not worthwhile to develop a method that contains parallelism at a higher level (viz. method B), because a large number of tridiagonal systems had to be solved. Method A is also very efficient on irregular regions, because of the constant number of grid layers in the vertical direction due to the sigma-coordinates (see (1)-(3)). It should be noted that method B is superior when e.g. one large tridiagonal system has to be solved [3].
For method A we have also measured the **total** computation times (thus, inclusive computation of the function $F_{u/v/z}$ in (4), etc.) on various computers, ranging from supercomputers to personal computers. This could be easily done, since the code was written in the ANSI FORTRAN 77 programming language. In Table 2 we list the results (* = 32 bits precision, otherwise 64 bits precision).

Table 2 : Computation times (in seconds).

	OPTIMALIZATION			
COMPUTER	(-)	(G)	(GV)	(GVP)
MACINTOSH PLUS *	5076			
VAX-11/780 *	422	148		
ALLIANT FX/4 *	104.6	25.7	7.0	2.34
ALLIANT FX/4				3.05
CDC CYBER 990	44.3	8.8		
CDC CYBER 205	9.9	4.8	0.54	
CRAY X-MP/28			0.196	
NEC SX/2 *			0.073	

For this test problem the CDC CYBER 205 is about 5 times faster than the ALLIANT FX/4, and the NEC SX/2 supercomputer is about 70.000 times faster than the MACINTOSH PLUS !

Conclusions

In this contribution we investigated the performance of a numerical method for the solution of the simplified shallow water equations on the ALLIANT FX/4. The results showed that even for a relatively small test problem, a considerable reduction in computation time can be obtained when the numerical method fully exploits the potential of vector and parallel facilities of this mini-supercomputer.

References

[1] Davies, A.M., Application of the DuFort-Frankel and Saul'ev methods with time splitting to the formulation of a three dimensional hydrodynamic sea model, *Intern. J. Numer. Methods Fluids*, **5** (1985), pp. 405-425.
[2] Goede, E.D. de, *Finite difference methods for the three-dimensional hydrodynamic equations*, Report NM-R8813, CWI, Amsterdam, 1988.
[3] Vorst, H.A. van der & K. Dekker, The vectorization of linear recurrence relations, *SIAM J. Sci. and Stat. Comput.*, **2** (1989), pp. 27-35.

Appendix: a detailed description of the time integrator

Here, we give a detailed description of the method that has been briefly described in (4).

$$U^{n+1}_{i,j,k} - \frac{1}{\rho h^2}\frac{\tau}{\Delta\sigma_k}\left\{\frac{A_{k+1}(U^{n+1}_{i,j,k+1} - U^{n+1}_{i,j,k})}{0.5(\Delta\sigma_{k+1}+\Delta\sigma_k)} - \frac{A_k(U^{n+1}_{i,j,k} - U^{n+1}_{i,j,k-1})}{0.5(\Delta\sigma_k+\Delta\sigma_{k-1})}\right\}$$

$$= U^n_{i,j,k} + \tau f\,\tilde{V}^n_{i,j,k} - \tau g\left(\frac{Z^n_{i,j} - Z^n_{i-1,j}}{\Delta x}\right)$$

$$V^{n+1}_{i,j,k} - \frac{1}{\rho h^2}\frac{\tau}{\Delta\sigma_k}\left\{\frac{A_{k+1}(V^{n+1}_{i,j,k+1} - V^{n+1}_{i,j,k})}{0.5(\Delta\sigma_{k+1}+\Delta\sigma_k)} - \frac{A_k(V^{n+1}_{i,j,k} - V^{n+1}_{i,j,k-1})}{0.5(\Delta\sigma_k+\Delta\sigma_{k-1})}\right\}$$

$$= V^n_{i,j,k} - \tau f\,\tilde{U}^{n+1}_{i,j,k} - \tau g\left(\frac{Z^n_{i,j+1} - Z^n_{i,j}}{\Delta y}\right)$$

$$Z^{n+1}_{i,j} = Z^n_{i,j} - \frac{\tau}{\Delta x}\left\{h\sum_{k=1}^{ns}\Delta\sigma_k U^{n+1}_{i+1,j,k} - h\sum_{k=1}^{ns}\Delta\sigma_k U^{n+1}_{i,j,k}\right\}$$

$$- \frac{\tau}{\Delta y}\left\{h\sum_{k=1}^{ns}\Delta\sigma_k V^{n+1}_{i,j,k} - h\sum_{k=1}^{ns}\Delta\sigma_k V^{n+1}_{i,j-1,k}\right\},$$

where $\tilde{V}_{i,j,k} = 0.25\cdot\left(V_{i,j,k} + V_{i,j-1,k} + V_{i-1,j,k} + V_{i-1,j-1,k}\right)$,

$\tilde{U}_{i,j,k} = 0.25\cdot\left(U_{i,j,k} + U_{i+1,j,k} + U_{i,j+1,k} + U_{i+1,j+1,k}\right)$.

This averaging is a consequence of the space staggering. The velocities are denoted by $U^n_{i,j,k}$ and $V^n_{i,j,k}$, where i,j refers to a horizontal grid point, k to a vertical layer and n to a time level. The surface elevation points are denoted by $Z_{i,j}$ and are specified at the sea surface. In the vertical we have varying mesh sizes $\Delta\sigma_k$, where k refers to the kth grid layer. The vertical diffusion coefficient is assumed to vary only through the vertical. Hence, A_k refers to the coefficient at layer k.