1991

E.D. de Goede

3D Shallow water model on the CRAY Y-MP4/464

Department of Numerical Mathematics Report NM-R9106 March

CWI, nationaal instituut voor onderzoek op het gebied van wiskunde en informatica

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

3D Shallow Water Model on the CRAY Y-MP4/464

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

Simulation of three-dimensional shallow water flows requires the use of fast computers, along with numerical methods that fully exploit the potential of vector and parallel facilities of these computers. In this paper we discuss the implementation of such a numerical method on the CRAY Y-MP4/464, which has recently been installed at the Academic Computing Services Amsterdam (SARA).

1980 Mathematics Subject Classification: 65M10, 76D99.
Key Words & Phrases: 3D shallow water equations, numerical methods, CRAY Y-MP4/464.
Note: This work was supported by the Ministry of Transport and Public Works (RWS).
Note: This paper will be submitted for publication.

Introduction

In recent years, numerical modeling has become an important tool for computing shallow water models. For example, flows in rivers, estuaries and seas can be described by these numerical models. Whereas in the past it was necessary to use scale models, it is now possible to solve the three-dimensional shallow water equations using computers. 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 is a joint project of Rijkswaterstaat (Dutch Water Control and Public Works department) and CWI (Centre for Mathematics and Computer Science).

In [3] we described an unconditionally stable method for the three-dimensional shallow water equations. In that paper it was reported that this method can be implemented efficiently on an Alliant FX/4 (a shared memory mini-supercomputer with four vector processors). In this paper we will describe the implementation of this method on a CRAY Y-MP. In December 1990 a CRAY Y-MP4/464 has been installed at the Academic Computing Services Amsterdam (SARA). Like the Alliant FX/4, this computer is a shared memory system with four vector processors. However, the CRAY Y-MP has a much smaller clock cycle

time (6 ns versus 170 ns for the Alliant FX/4). The CRAY Y-MP4/464 replaces a Control Data Corporation Cyber 205 installed in 1983.

Both vectorization and parallelism will be examined on the CRAY Y-MP4/464. We will implement several types of parallelism, viz. macrotasking, microtasking and autotasking. A comparison will be made with results on the Alliant FX/4. Since the code was written in the ANSI Fortran 77 programming language, the same code was used for both computers.

Mathematical model

A mathematical model for the three-dimensional shallow water equations is described by

$$\frac{\partial u}{\partial t} = - u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \omega\frac{\partial u}{\partial \sigma} + fv - g\frac{\partial \zeta}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \frac{1}{\rho h^2}\frac{\partial}{\partial \sigma}\left(\mu\frac{\partial u}{\partial \sigma}\right) \tag{1}$$

$$\frac{\partial v}{\partial t} = - u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \omega\frac{\partial v}{\partial \sigma} - fu - g\frac{\partial \zeta}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + \frac{1}{\rho h^2}\frac{\partial}{\partial \sigma}\left(\mu\frac{\partial v}{\partial \sigma}\right) \tag{2}$$

$$\omega = \frac{1}{h}\left\{-(1-\sigma)\left(\frac{\partial}{\partial x}\left(h\int_0^1 u\,d\sigma\right) + \frac{\partial}{\partial y}\left(h\int_0^1 v\,d\sigma\right)\right) + \frac{\partial}{\partial x}\left(h\int_\sigma^1 u\,d\sigma\right) + \frac{\partial}{\partial y}\left(h\int_\sigma^1 v\,d\sigma\right)\right\} \tag{3}$$

$$\frac{\partial \zeta}{\partial t} = - \frac{\partial}{\partial x}\left(h\int_0^1 u\,d\sigma\right) - \frac{\partial}{\partial y}\left(h\int_0^1 v\,d\sigma\right) \tag{4}$$

where the following notation is used:

u,v,ω	velocity components in x- , y- and σ-direction
ζ	water elevation
x,y,σ	a left-handed set of co-ordinates
t	time
f	Coriolis term
g	acceleration due to gravity
h	depth of water
ν	eddy viscosity coefficient in the horizontal direction
μ	eddy viscosity coefficient in the σ-direction
ρ	density.

The equations (1)-(3) are the momentum equations and (4) denotes the continuity equation. In the vertical, the domain is bounded by the bottom topography and the time-dependent water elevation. To ensure that the three-dimensional domain is also constant in the vertical direction, system (1)-(4) has been transformed in the vertical into depth-following (sigma) co-ordinates.

In the experiments, we will examine a three-dimensional flow past a jetty. The geometry and some velocity patterns are given in Figure 1. The rectangular basin has a horizontal dimension of 1500 m by 300 m and a

constant depth of 25 m. At the left boundary, we prescribe an inflow condition of u=0.5 m/s and at the right boundary a uniform water level ζ=0 m is given. For a detailed description of the initial conditions and boundary conditions we refer to [3,6].

To obtain a discrete system representing (1)-(4), 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. On this grid the spatial derivatives are replaced by second-order finite differences, which results in a semi-discretized system of dimension $nx \cdot ny \cdot (3ns+1)$.

The time discretization of this semi-discrete system is performed by a numerical method developed in [3]. When applied to two-dimensional problems, this method is very similar to the method developed in [7]. Our time splitting method is unconditionally stable and consists of two stages. At both stages a system of equations has to be solved. The structure of these systems determines the efficiency of the numerical method. To solve these systems, a Jacobi-type iteration method is used at the first stage, whereas at the second stage a conjugate gradient method is used. The time splitting has been chosen in such a way that in the horizontal direction the computations are independent of each other. For example, the Jacobi-type iteration method requires the solution of $nx \cdot ny$ independent tridiagonal systems, all of dimension ns. In [3] it is reported that this method can be implemented efficiently on an Alliant FX/4.

Implementation

We now discuss the implementation of this time splitting method on the CRAY Y-MP4/464. A DO-loop in our code may be of the form

```
      DO 100  K=2,NS
        DO 100  J=1,NY
          DO 100  I=1,NX
100         A(I,J,K) = A(I,J,K) + A(I,J,K-1) * B(K)
```

In this example, the I- and J-loop do not show any dependencies, but the K-loop does. The loops with indices I and J are collapsed into a single DO-loop to obtain a longer vector length. This yields

```
      DO 100  K=2,NS
        J=1
        DO 100  I=1,NX·NY                                    (5)
100       A(I,J,K) = A(I,J,K) + A(I,J,K-1) * B(K)
```

On the CRAY Y-MP4/464, loop collapsing of simple operations is performed automatically by the compiler. However, for more complicated DO-loops this is not the case. Therefore, the code was collapsed by hand, instead of relying on the compiler.

Scalar and vector performance

In Table 1 we list the scalar and vector performance of our code on the CRAY Y-MP4/464 and on the Alliant FX/4. The speed-up due to scalar optimization and vectorization is satisfactory on both computers. Both compilers have no problems with vectorizing the DO-loops. It should be noted that Table 1 contains results for the complete code. The code not only consists of vectorizable instructions, but there are also non-vectorizable instructions like subroutine calls, which decrease the performance. For the vectorized code, the computation time on the CRAY Y-MP4/464 is seventy-five times smaller than for the Alliant FX/4. Notice that the ratio of the clock cycle times is only twenty-eight. On the Alliant FX/4 we obtain 15% of the peak performance, whereas on CRAY Y-MP4/464 we obtain 40%. Taking into account the performance of elementary operations on both computers (see [4,5]), we are satisfied with the performance of our code. In [4] it is also reported that the Alliant FX/4 performs considerably below its maximum level (≤40% of the theoretical maximum only). This is due to the insufficient bandwidth from memory to the functional units and/or from cache to the functional units.

Table 1. Mflop/s rates and speed-ups.

	ALLIANT FX/4		CRAY Y-MP4/64	
	Mflop/s	speed-up	Mflop/s	speed-up
no optimization	0.2		3.2	
		> 3		> 5
scalar optimization	0.7		15.0	
		> 3		> 10
vector optimization	2.0		148.9	

Parallelism

On the CRAY Y-MP4/464, parallel processing can be accomplished by three techniques: macrotasking, microtasking and autotasking. Macrotasking allows a program to be partitioned into several *tasks* at the subroutine level. The programmer inserts library routine calls into the code to initiate and synchronize tasks that can be executed in parallel. Macrotasking works best when the amount of work to be partitioned over

the processors is large. Microtasking, however, allows parallelism at the DO-loop level. The programmer identifies parallel regions in the code and inserts compiler directives accordingly. Then, at execution, the DO-loop iterations are spread over the processors, which is called strip-mining. Autotasking is a technique that shares the advantages of microtasking, while adding several new advantages. It identifies and exploits parallel regions in a program through the use of a preprocessor. This process may be completely automatic. However, the help of the programmer may be useful since not all types of parallelism can be detected by the compiler.

When applying macrotasking to an existing code, a significant amount of code restructuring may be necessary, which can introduce new errors. It requires a careful partitioning in equal-sized parallel tasks to obtain a good load balancing. On the other hand, it is relatively easy to implement microtasking, which yields an automatic load balancing. The success of the microtasking strategy depends on the synchronization overhead.

As mentioned earlier, the DO-loops in our code have been collapsed explicitly (see (5)). This maximizes the vector efficiency while it allows parallelism by the strip-mining technique. In our first experiment with parallelism on the CRAY Y-MP4/464 we implemented autotasking. We hoped that strip-mining of the innermost DO-loops was exploited automatically. However, for our code this is not the case. Thus, we did not obtain any speed-up by autotasking. It should be mentioned that on the Alliant FX/4 the collapsed DO-loops are automatically performed in a vector-parallel mode.

In the second experiment, we applied the microtasking technique. By inserting the CMIC$ DO GLOBAL LONG VECTOR directive the innermost DO-loops were strip-mined at execution. However, we obtained incorrect numerical results. Using the microtasking directives, it appeared that parallel processing always starts at the first executable statement of the subroutine and always ends at the last executable statement of the subroutine. Consequently, not only the microtasked DO-loop, but the complete subroutine was performed in parallel. To circumvent this unpleasant microtasking property, we implemented autotasking directives instead of microtasking directives. Thus, we used autotasking but not in an automatic way. With the CMIC$ DO ALL VECTOR AUTOSCOPE autotasking directive we obtained a strip-mining of the innermost DO-loops, while the numerical results remained correct.

Numerical results

In this section, we present the results for two different test problems. In the first experiment, the computations were performed on a grid with $nx=61$, $ny=13$ and $ns=10$. In this case, we have a vector length of about 800. In the second experiment, we used a grid with *121*, *49* and *10* grid points in the three spatial directions, respectively, yielding a vector length of about 6000. The main reason for the second experiment was to investigate the performance on the CRAY Y-MP4/464 for other vector lengths. In order to prevent many more iterations required by the iteration methods, we increased in the latter experiment the dimensions

of the geometry with a factor four. Consequently, in both experiments we obtained the same mesh sizes. In Table 2 we list the results for the experiments in which we introduced parallelism by strip-mining of the innermost DO-loops.

Table 2. Computation times for the CRAY Y-MP4 by strip-mining.

problem size	*61·13·10*	*121·49·10*
vector code	17.6 sec	137.4 sec
parallel code	11.6 sec	53.0 sec
speed-up	1.5	2.6

For the smallest test problem, we obtained a speed-up of 1.5 on four processors, which is disappointing. For the second and largest test problem, the speed-up of 2.6 is still small despite of its large vector length. Thus, especially for small vector lengths the synchronization overhead on the CRAY Y-MP4/464 appears to be considerable.

To obtain a higher efficiency on the CRAY Y-MP4/464, we implemented a domain decomposition technique. In the x-direction our domain was split into four subdomains. With the CMIC$ DO PARALLEL autotasking directive the computations for the four subdomains were performed in parallel. We applied autotasking instead of macrotasking, because it requires less programming effort while the computation times are comparable.

A decomposition in the x-direction has been chosen to minimize the restructuring of the code. A slightly more complicated decomposition would have been a decomposition of both the x- and y-direction into two parts, as used in [1]. Furthermore, we mention that for a spectral method a domain decomposition in the vertical direction has been applied in [2].

In our opinion, a domain decomposition in the horizontal direction is a good choice for the numerical method used in this paper, because of the many independent computations in the horizontal direction. In time-consuming 3D shallow water models, the value of $nx \cdot ny$ is very large, whereas the value of ns still may be rather small. Moreover, the water elevation and the vertical velocity ω are computed from vertical integrals (see (3)-(4)) and implicit relations, arising from an implicit treatment of the vertical diffusion term, have to be solved [3]. Therefore, it seems not a good idea to apply a domain decomposition in the vertical direction. This leads to a lot of communication between the processors.

In Table 3 we list for our two test problems the results for the domain decomposition approach. We now have vector lengths of 200 and 1500 respectively, which is one fourth of the vector lengths used in the former experiment. The overall speed-up is defined in the following way:

$$\frac{\text{time for the original vectorized code (see vector performance in Table 2)}}{\text{time for the parallel code with domain decomposition (see parallel performance in Table 3)}}$$

Table 3. Computation times for the CRAY Y-MP4 by domain decomposition.

problem size	*61·13·10*	*121·49·10*
vector code	20.7 sec	141.2 sec
parallel code	5.6 sec	38.1 sec
decomposition speed-up	3.7	3.7
overall speed-up	3.1	3.6

For both test problems, we now obtain a speed-up of 3.7 on four processors. Because of the synchronization overhead we do not obtain an optimum speed-up of four. The difference in overall and domain decomposition speed-up is due to the fact that vector lengths are now smaller by a factor of four. In the case of very large vector lengths, this hardly yields a decrease in performance. For the smallest test problem, the reduction is about 20%. The results in Table 3 clearly show that the domain decomposition approach gives rise to larger speed-ups than the strip-mining technique.

Conclusions

In this paper we investigated the performance of a numerical method for the three-dimensional shallow water equations on the CRAY Y-MP4/464. On one processor the highly vectorizable code yielded a performance of about 150 Mflop/s, which was no fewer than seventy-five times faster than on the Alliant FX/4. To exploit parallelism, a domain decomposition approach had to be used, since in general the strip-mining of the innermost DO loops did not give satisfactory speed-ups, even in the case of large vector lengths.

The experiments show that the CRAY Y-MP4/464 is very suitable for the time-consuming simulation of three-dimensional shallow water flows. With such supercomputers it is possible to simulate realistic models, e.g. including equations for salinity, temperature and turbulence, with a fine resolution. The vector

performance is very impressive. When the CRAY4/464 is not heavily loaded, an acceptable speed-up due to parallelism can be obtained.

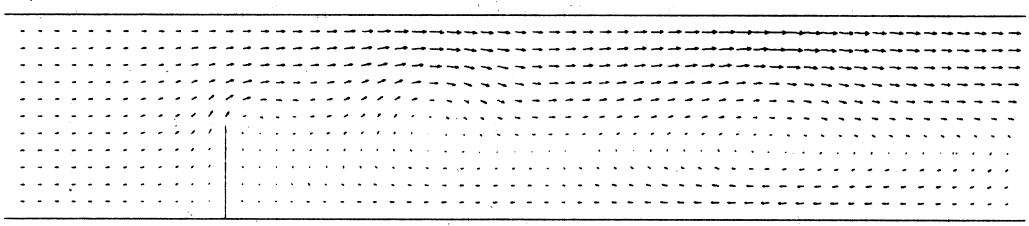

Figure 1. Flow past a jetty.

References

[1] P. Andrich, G. Madec and D. l'Hostis, Performance evaluation for an ocean general circulation model: vectorization and multitasking, *Proceedings of the International Conference on Supercomputing*, St. Malo, 295-303 (1988).

[2] A.M. Davies, R.B. Grzonka and M. O'Neill, Development and application of 3D shallow sea models on the CRAY computer, *Proceedings of the 5th International Symposium of Science and Engineering on Supercomputers*, London, 323-334 (1990).

[3] E.D. de Goede, A time splitting method for the three-dimensional shallow water equations, *Int. J. Numer. Methods Fluids*, to appear.

[4] J.M. van Kats and A.J. van der Steen, *Benchmarktests on an Alliant FX/4, a Convex C-1, an FPS 64/60 and an SCS-40*, report TR-24, ACCU, Utrecht, 1987.

[5] A.J. van der Steen and R.J. van der Pas, *A family portrait: Benchmarktests on a CRAY Y-MP and a CRAY-2S*, report TR-30, ACCU, Utrecht, 1989.

[6] G.S. Stelling, On the construction of computational methods for shallow water flow problems, *Ph.D. Thesis*, Delft University, 1983.

[7] P. Wilders, Th.L.van Stijn, G.S. Stelling and G.A. Fokkema, A fully implicit splitting method for accurate tidal computations, *Int. J. Numer. Methods Eng.*, **26**, 2707-2721 (1988).