http://dx.doi.org/10.1090/conm/218/03029

Contemporary Mathematics
Volume **218**, 1998
B 0-8218-0988-1-03029-6

# A Comparison of Scalability of Different Parallel Iterative Methods for Shallow Water Equations

Arnt H. Veenstra, Hai Xiang Lin, and Edwin A.H. Vollebregt

## 1. Introduction

The scalability of the iterative methods Jacobi, SOR, CG and AMS-CG will be analyzed for the solution of systems of equations with symmetric positive definite matrices, arising from the shallow water equations. For the parallel solution of these systems domain decomposition is applied and each subdomain is computed by a processor. The AMS-CG (Approximate Multi-Subdomain CG) method is a relatively new CG-type method specially designed for parallel computation. We will show that it compares favorably in our experiments to the other methods with respect to speedup, communication overhead, convergence rate and scalability.

In order to predict the water movement and the dispersion of dissolved substances for the management of coastal seas and estuaries, computer simulations are performed which comprises the numerical solution of a.o. the shallow water equations. The shallow water equations are a set of coupled nonlinear partial differential equations which describe a 3-dimensional flow in shallow water [**3**]. The SWE are derived from the incompressible Navier Stokes equations by assuming a hydrostatic pressure distribution. In this way the pressure is eliminated and replaced by the unknown water level $\zeta$. Practical simulation requires an accurate solution using a fine grid and a small time step. Thus, computation time becomes very important. In this paper we consider parallel solution in order to reduce computation time.

The partial differential equations are discretized using a staggered grid, see e.g. [**7**], and a two stage time splitting method [**3**]. This leads to a nonlinear pentadiagonal system of equations to be solved in each time step. These nonlinear systems are solved by linearization [**3**] and by using iterative methods for the resulting systems, which are symmetric positive definite. For the parallel solution of these systems domain decomposition is applied and each subdomain is computed by a separate processor. We analyze and evaluate the performance and scalability of different iterative methods for solving these systems: Jacobi, CG, SOR and AMS-CG [**6**].

The AMS-CG (Approximate Multi-Subdomain-CG) method is a relatively new CG-type method. The method is specially designed for (massively) parallel computation. Unlike CG, it only requires communication between neighboring subdomains and the unknowns in each subdomain are approximated with an independent

1991 *Mathematics Subject Classification.* Primary 65F10; Secondary 65Y05, 76B15.

search direction. It is known that the overhead of global communications, such as inner product computations in CG, can strongly affect the parallel performance and scalability on massively parallel processors. AMS-CG eliminates the global communication and therefore should have a better scalability.

We applied the following preconditioners for CG and AMS-CG: ILU [**5**], MILU [**4**] and FSAI [**2**]. Each of them is applied with overlapping subdomains and non-overlapping subdomains and with varying internal boundary conditions. We compare the above mentioned iterative methods combined with domain decomposition from the experimental results on a Cray T3D. The speedup, communication overhead, convergence rate, and scalability of the different methods are discussed.

## 2. Problem formulation

The application problem in our study is a 3-dimensional hydrodynamic sea model: A simplified form of the shallow water equations. These equations describe the motion and the elevation of the water:

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

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

$$\frac{\partial \zeta}{\partial t} = -\frac{\partial}{\partial x}\left(h_0\int^1 ud\sigma\right) - \frac{\partial}{\partial y}\left(h_0\int^1 vd\sigma\right) \tag{3}$$

The equations have been transformed in the vertical direction into depth-following coordinates ($\sigma$) by the so-called sigma transformation: $\sigma = \frac{\zeta - z}{d+\zeta}$. The boundary conditions at the sea surface ($\sigma$=0) and at the bottom ($\sigma$=1) are:

$$\left(\mu\frac{\partial u}{\partial \sigma}\right)_{\sigma=0} = -\frac{h}{\rho}W_f\, cos\,(\phi)\,, \left(v\frac{\partial v}{\partial \sigma}\right)_{\sigma=0} = -\frac{h}{\rho}W_f\, sin\,(\phi) \tag{4}$$

$$\left(\mu\frac{\partial u}{\partial \sigma}\right)_{\sigma=1} = \frac{gu_d}{C^2}, \left(v\frac{\partial v}{\partial \sigma}\right)_{\sigma=1} = \frac{gv_d}{C^2} \tag{5}$$

$W$=wind force, $\phi$=direction of wind force and $u_d$ and $v_d$ velocities near the bottom. In the horizontal direction a staggered grid is used , see e.g. [**7**] for more information about the use and advantages of staggered grids. In the vertical direction the grid is divided into a few equidistant layers. The variable $\zeta$ is discretized as $Z$, which is the water level compared to a horizontal plane of reference. The spatial derivatives are discretized using second-order central differences. The time integration is performed by a two-stage time splitting method [**3**]. The time step is split in two stages: $t = n$ to $t = n + 1/2$ and $t = n + 1/2$ to $t = n + 1$. At the first stage the equations describing the vertical diffusion are treated implicitly. Each vertical line of grid points corresponds to a separate tridiagonal system of equations.

At the second stage the equations describing the propagation of the surface waves are treated implicitly. This pentadiagonal system of equations describes the water level related to the water level in 4 neighboring grid points:

$$(1+c_1+c_2+c_3+c_4)Z_{i,j}^{n+1} - c_1 Z_{i+1,j}^{n+1} - c_2 Z_{i-1,j}^{n+1} - c_3 Z_{i,j+1}^{n+1} - c_4 Z_{i,j-1}^{n+1} = Z_{i,j}^{n+1/2}$$

$Z_{i,j}^{n+1}$ is the water level in grid point $(i,j)$ at time $t{=}n{+}1$. The coefficients $c_i$ are functions depending on the water level $Z$. This equation can be put in the form:

$$A\left(Z^{n+1}\right)Z^{n+1} = Z^{n+1/2}$$

The equations are first linearized before applying an iterative solution method by:

$$A\left(X^{m}\right)X^{m+1} = Z^{n+1/2}$$

Here $X^0{=}Z^{n+1/2}$ and $A$ is symmetric positive definite. These systems are linear in $X^{m+1}$. The iterative methods are used to compute $X^{m+1}$. Thus there are two iteration processes: The outer iteration process updates $A(X^m)$ and continues until $\left\|X^{m+1} - X^m\right\| < \varepsilon$ and the inner iteration process solves each $X^{m+1}$. Both processes are executed until convergence. In our experiments we compare the results of the iterative methods for the entire two iteration processes in the second stage.

## 3. Approximate Multi-Subdomain-CG

Starting with a basic form of CG without any optimizations, we can derive the Multi Subdomain CG (MS-CG) method by splitting the global search direction into local search directions, one for each subdomain. The local search direction $p_d$ for subdomain $d$ is a vector with zero entries for variables corresponding to positions outside subdomain $d$. Each iteration a $p_d$ for each subdomain must be determined. The $p_d$'s are put in the columns of a matrix $P$. Instead of the scalars $\alpha$ and $\beta$ in the CG method, $\alpha$ and $\beta$ are vectors with a separate entry for each subdomain. We require that $p_d^k = z_d^{k-1} + \beta^k P^{k-1}$ is conjugate to all previous $p_d^j$ with $j{<}k$. This is satisfied by computing $\beta$ from $C^{k-1}\beta^k = -(Q^{k-1})^T z^{k-1}$. In order to minimize $\left\|x - x^k\right\|_A = \left(x - x^k, b - Ax^k\right)$, $\alpha$ is computed by $C^k\alpha^k{=}(P^k)^T r^{k-1}$, where $C^k{=}(Q^k)^T P^k$. It can be proven that the errors $\left\|x - x^k\right\|_A$ are non-increasing [**6**]. The preconditioned MS-CG method can be derived from PCG.

```
Algorithm: Multi-Subdomain CG
$x^0$ = initial guess, $k = 0$, $r^0 = b - Ax^0$
while 'not converged'
    solve $Mz_d^k = r_d^k$
    $k = k + 1$
    $if\ (k = 1)$
        for each subdomain $d$: $p_d^1 = z_d^0$
    else
        $C^{k-1}\beta^k = -(Q^{k-1})^T z^{k-1}$
        for each subdomain $d$: $p_d^k = z_d^{k-1} + \beta^k P^{k-1}$
    end
    $Q^k = AP^k$
    $C^k = (Q^k)^T P^k$
    $C^k\alpha^k = (P^k)^T r^{k-1}$
    $x^k = x^{k-1} + P^k\alpha^k$
    $r^k = r^{k-1} - Q^k\alpha^k$
end
```

$$x = x^k$$

The matrix $M$ has to be a symmetric positive definite preconditioning matrix. $p_d^i$=0 outside subdomain $i$. The matrix $C$ is symmetric positive definite when the matrix $A$ is. The CG method has the disadvantage that for each iteration 2 dot products are computed. Instead of these dot products the MS-CG method requires 2 matrix equations with the matrix $C$ to be solved. The matrix $C$ is symmetrical positive definite in our problem. The idea of the Approximate MS-CG (AMS-CG) method is to approximate the 2 equations with the matrix $C$ by a few Jacobi iterations. The matrix $C$ is relatively small: $l \times l$, where $l$ = number of subdomains. The Jacobi method only requires local communication. The convergence of the Jacobi iterations is not tested, because we don't want to introduce global communication. Usually a small fixed number of Jacobi iterations (2 to 4) is sufficient. The AMS-CG method requires no global communication, except to check convergence. This convergence check can be done once after performing a number of iterations.

## 4. Preconditioning

In this section we describe several preconditioners for the CG-method and the AMS-CG method. ILU preconditioning [**5**] is well known. It is derived from the LU-decomposition. The LU-decomposition generates full matrices. The ILU(0) algorithm allows no fill in, ie: An entry in $L$ and $U$ is zero whenever the corresponding entry in the matrix $A$ is zero.

Gustafsson [**4**] proposed the MILU (Modified ILU) method. The components which are disregarded in ILU are added to the main diagonal. Therefore the row sum of $LU$ is equal to the row sum of $A$. In our experiments the MILU preconditioner performed much better than the ILU preconditioner. Therefore most experiments with preconditioning were done with MILU.

The ILU and MILU preconditioners have to be uncoupled in order to compute different subdomains in parallel. There are a few possibilities: We can simply disregard the coupling, we can impose boundary conditions on the internal boundaries or we can use a Schur complement method. We did not use a Schur complement method, however. This would have resulted in more communication, while we tried to precondition without communication or with very little communication. Consequently we found an increasing number of iterations for an increasing number of subdomains. With a Schur complement method the number of iterations will almost certainly be roughly the same for different numbers of subdomains.

The FSAI (Factorised Sparse Approximate Inverse) preconditioner [**2**] uses the following approximate inverse of $A$: $G^T D^{-1} G$, where $D$=diag($G$) and $g_{ij} = 0$ when $a_{ij} = 0$, $(GA)_{ij} = \delta_{ij}$ when $a_{ij} \neq 0$. This preconditioner can be applied to different numbers of subdomains without any problem. This only requires some local communication between neighboring processors. These preconditioners are constructed with non-overlapping or overlapping subdomains.

## 5. Numerical results

The test problem used is that of de Goede [**3**, p. 63]. We simulate a rectangular basin of 400 km $\times$ 800 km. The bottom is inclined in $x$-direction with a depth of 20 m at one end and a depth of 340 m at the other end. The grid used is 100 $\times$ 100 points with 5 vertical layers. Simulations are carried out with time steps equal to:

TABLE 1. Amount of computation per iteration

| Method | Computations |
|---|---|
| Jacobi | $6n$ multiplications |
| SOR | $6n$ multiplications |
| CG | $10n$ multiplications |
| CG+MILU | $14n$ multiplications + $2n$ divisions |
| AMS-CG | $12n$ multiplications |
| AMS-CG+MILU | $16n$ multiplications + $2n$ divisions |

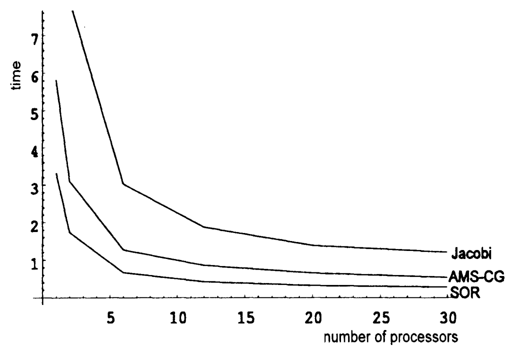

FIGURE 1. Average execution time per time step plotted against the number of processors

600 s, 1200 s and 2400s. We simulate a period of 4 hours and use a wind force of 1.0 N/m$^2$ in $x$-direction. For the AMS-CG method we take a number of 3 Jacobi iterations.

**Convergence.** A good indication of the convergence properties of an iterative method is the average number of iterations required per time step. The convergence properties of AMS-CG are just as good as the convergence properties of CG in our experiments. This holds with and without preconditioning. As expected the convergence properties of Jacobi are bad, especially for larger time steps. The convergence of the SOR method is better than the convergence of CG and AMS-CG when the right relaxation parameter is chosen. The convergence of the MILU-preconditioned CG and AMS-CG is best of all. To give an indication: With a time step of 1200 s and $6 \times 5$ subdomains the number of iterations for Jacobi is 1008, for CG/AMS-CG: 202, for SOR: 125, for CG+MILU/AMS-CG+MILU: 55.

Convergence of MILU-preconditioned CG or AMS-CG becomes worse as domain numbers increase. The largest difference is between 1 global domain $(1 \times 1)$ and 2 subdomains $(2 \times 1)$ . We tried to improve this situation, but the number of iterations remained largely the same with overlapping or non-overlapping subdomains and with different internal boundary conditions. The convergence of FSAI-preconditioned AMS-CG is worse than the convergence of MILU-preconditioned AMS-CG. Therefore, in case of preconditioning, we will only present results with MILU using non-overlapping subdomains.

**Execution time.** For $n$ grid points, the amount of computation per iteration for the different methods is shown in Table 1. CG and AMS-CG require more

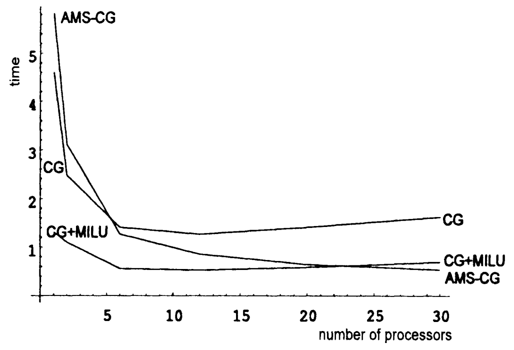

FIGURE 2. Average execution time per time step plotted against the number of processors

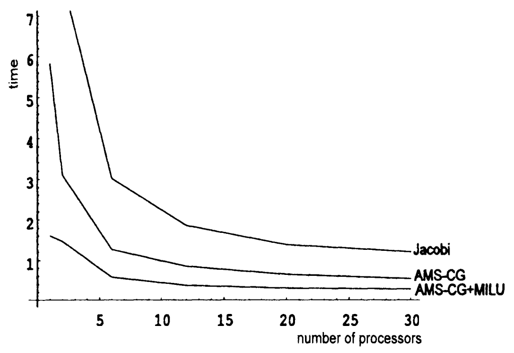

FIGURE 3. Average execution time per time step plotted against the number of processors

computations per iteration, with or without preconditioning. However, the number of iterations is less for CG and AMS-CG. This results in a better overall performance of CG and AMS-CG for a complete time step. Figure 1, 2 and 3 show the average execution time per time step plotted against the number of processors. Figure 1 shows that the average execution time of AMS-CG lies just between the execution time of Jacobi and SOR. Figure 2 shows that AMS-CG is worse than CG and CG+MILU on a small number of processors. If the number of processors increases AMS-CG becomes better than CG and CG+MILU. Figure 3 shows that the average execution time of AMS-CG+MILU is better then the execution time of AMS-CG. The execution time of AMS-CG+MILU is also better than the execution time of SOR.

**Optimization.** The convergence check implies that for every iteration a global dot product must be computed. Therefore it is advantageous not to check the convergence every iteration. In our experiments we choose to perform the convergence check only every 10 iterations. This reduces global communication considerably. For the standard CG method it is not relevant to do this, because one of the 2 dot products required per iteration is also used for the convergence check.

**Speedup and scalability.** Figure 4 shows the relative speedup compared to the execution time of the respective method on 2 processors with $2 \times 1$ subdomains. The figure is clear: The global communication of CG and CG+MILU results in a

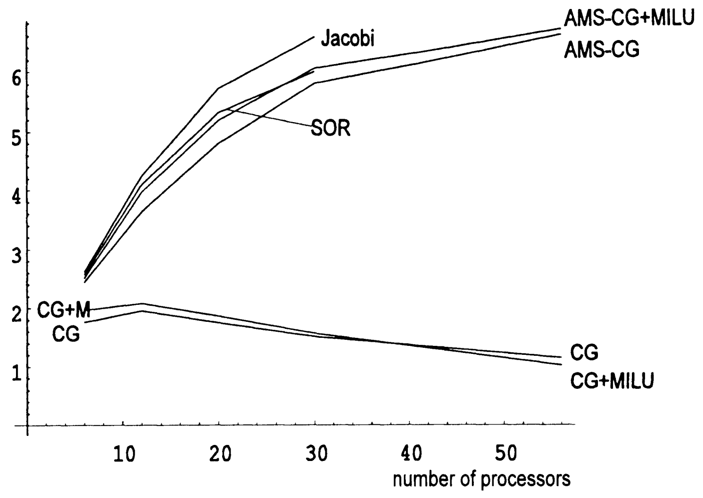

FIGURE 4. Relative speedup compared to the execution time on 2 processors (2 × 1 subdomains) of the respective method

TABLE 2. The ratio between the execution time on 30 processors (6 × 5 subdomains) and on 1 processor (1 × 1 subdomain)

| method | time step 600 s | 1200 s | 2400 s |
|---|---|---|---|
| Jacobi | 20.0 | - | - |
| SOR | 6.3 | 11.4 | 11.3 |
| CG | 16.3 | 16.6 | 16.9 |
| CG+MILU | 8.5 | 11.0 | 15.7 |
| AMS-CG | 10.0 | 9.9 | 8.4 |
| AMS-CG+MILU | 4.4 | 7.3 | 15.8 |

very bad speedup. A slowdown has been observed for larger number of processors. AMS-CG and AMS-CG+MILU have roughly the same speedup as Jacobi and SOR, which are known to be well parallelizable. For the evaluation of the scalability we kept the size of each subdomain constant. Each subdomain consisted of 50 × 50 grid points. The parallel execution times are compared with the execution time for a problem equal to 1 subdomain. Table 2 shows the execution time for 6 × 5 subdomains using 30 processors divided by the execution time for 1 × 1 subdomain using 1 processor. Note that the problem size grows proportionally with the number of subdomains. Thus, with the same number of processors, the execution time of 6 × 5 subdomains will be at least 30 times that of 1 × 1 subdomain. With perfect scalability, the ratio between the execution time for 6 × 5 subdomains using 30 processors and the execution time for 1 × 1 subdomain using 1 processor should be equal to 1. But in practice this ratio will be generally larger than 1, due to the increased number of iterations and the communication overhead.

The execution time of the Jacobi method with increasing number of subdomains grows fast. This is caused by the slow convergence of this method when the problem size increases. The SOR method does quite well in this case but its performance soon worsens as the time step increases (e.g. with an integration step of 1200 s the execution time ratio becomes 11.4). The CG and CG+MILU methods are not good scalable because of the fast increase of the overhead of global communication. The AMS-CG+MILU has the best scalability for small integration time steps, but it becomes worse for large integration time steps. The reason for this is that the preconditioning is then less effective. The performance of the AMS-CG method is in between and its convergence is insensitive to the integration time step.

## 6. Conclusions

We compared the iterative methods Jacobi, SOR, CG and AMS-CG on the solution of a system of equations which describes wave propagation. The matrix associated with the system of equations is sparse (pentadiagonal). This results in few computations and relatively more communication. The (preconditioned) AMS-CG method is well suited for a parallel solution of this application problem in combination with domain decomposition. In this case the method has convergence properties comparable to (preconditioned) CG. The speedup of AMS-CG is much better than the speedup of CG. This is due to the fact that AMS-CG requires only local communication. MILU preconditioning results in better convergence.

It is remarkable that the number of iterations of the preconditioned AMS-CG makes a big jump from 1 subdomain to 2 subdomains, while this is not observed for AMS-CG. The number of iterations increases only slightly from 2 subdomains to 3 subdomains and more. Future research should analyze the preconditioner and possibilities to improve it. An interesting topic is to implement a Schur complement preconditioner and compare the convergence and scalability. The Schur complement preconditioner generally has a faster convergence [**1**], but it requires more communication.

Also interesting is to apply the AMS-CG method to other type of problems. It is still an open question if the AMS-CG method is a suitable parallel method for the solution of systems of equations arising from other applications.

## References

1. T.F. Chan and D. Goovaerts, *A note on the efficiency of domain decomposed incomplete factorizations*, SIAM J. Sci. Stat. Comput. **11** (1990), no. 4, 794–802.
2. M. R. Field, *An efficient parallel preconditioner for the conjugate gradient algorithm*, presented at the IMACS 97 conference, Jackson Hole, Wyoming, USA, july 1997.
3. E. de Goede, *Numerical methods for the three-dimensional shallow water equations on supercomputers*, Ph.D. thesis, University of Amsterdam, The Netherlands, 1992.
4. I. Gustafsson, Preconditioning Methods. Theory and Applications ( David J. Evans, ed.), ch. Modified Incomplete Choleski Methods, pp. 265–293, Gordon and Breach, Science Publishers, New York, London, Paris, 1983, pp. 265–293.
5. J.A. Meijerink and H.A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp. **31** (1977), 148–162.
6. E.A.H. Vollebregt, *Multi-subdomain conjugate gradients: A global inner product free cg-type method*, Tech. report, Department of Applied Math.&Informatics, TU Delft, 1997.
7. C.B. Vreugdenhil, *Numerical methods for shallow-water flow*, Water Science and Technology Library, vol. 13, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1994.

Faculty of Information Technology and Systems, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands
*E-mail address*: h.x.lin@math.tudelft.nl