Solving large linear systems in an implicit thermohaline ocean model

de Niet, Arie Christiaan

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2007

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Chapter 7

A tailored solver for bifurcation analysis of ocean-climate models

7.1 Introduction

In the previous chapters we paid much attention to the saddle point problem. We discussed a direct solver (Chapter 4), preconditioning via incomplete LU factorization (Chapter 5) and preconditioning with block-preconditioners (Chapter 6). In this chapter we will return to the main issue, namely the solution of large linear systems from the model as presented in Chapter 2.

In Section 2.4.2 we showed that the Jacobian matrix of the discrete ocean equations ($\Phi$) has the following structure

$$
\begin{bmatrix}
A_{uv} & E_{uv} & G_{uv} & 0 \\
0 & 0 & G_w & B_{TS} \\
D_{uv} & D_w & 0 & 0 \\
B_{uv} & B_w & 0 & A_{TS}
\end{bmatrix}
\begin{bmatrix}
u_{uv} \\
u_w \\
u_p \\
_uT_S
\end{bmatrix}
= 
\begin{bmatrix}
b_{uv} \\
b_w \\
b_p \\
b_{TS}
\end{bmatrix}
. 
$$

(7.1)

In previous versions of THCM (the thermohaline continuation model) this system was solved with MRILU, a multilevel renumbering incomplete LU factorization method described in [10]. MRILU was not applied to the matrix with the block ordering as in (7.1) but to a matrix where the variables were interleaved, that is, the variables (u,v,w,p,T,S) that belong to the same cell were put after each other in the ordering. The resulting system was manipulated by MRILU with blocksize 6, i.e. the 6 variables were treated as one single variable. The major disadvantages of MRILU are the memory requirements and the construction time. Both are related to the large amount of fill (the nonzeros) in the final factorization.

We cannot apply the methods of the previous chapters blindly to (7.1), simply because the matrix is not of saddle point type. However based on the conclusions of Chapter 5 we can say that it is unlikely that the incorporation of fill-reducing orderings in MRILU or another incomplete LU factorization method will result in a good preconditioner with low fill. It was not able to do so for the Stokes problem, let alone for the ocean system. Chapter 6 showed us that approaches that exploit the structure of the saddle point problem, can result in very efficient
preconditioners. This motivated us to design a preconditioner for the matrix in (7.1) based on its structure.

We carefully studied the submatrices in (7.1) and, based on their properties, we constructed a preconditioner that subsequently solves different variables. After the development of the preconditioner we noted that there was a striking similarity with a technique well known and widely applied in explicit time integration methods for ocean flows: \textit{barotropic-baroclinic time splitting}, see e.g. [42]. This technique exploit the fact that fast external gravity waves are approximately independent of depth. Hence the time-dependent equations are solved in the following way:

1. apply depth-averaging to momentum and continuity equations (2.4a-b,d);
2. solve the depth averaged horizontal velocity field using the barotropic equations (two-dimensional);
3. construct full pressure field (p) and horizontal velocity field (u,v) using the baroclinic equations;
4. solve vertical velocity field (w) via the continuity equation;
5. solve tracer equations (T,S).

Both depth-averaging and the typical ordering of the solution of variables plays an important role in our preconditioner. The techniques used in explicit time stepping appear to be helpful in the design of a preconditioner for the Jacobian matrix that occurs in the implicit time stepping methods. Note that the Jacobian that occurs in continuation of steady states is nothing but an extreme case ($\Delta t = \infty$) of the Jacobian in implicit time stepping (see Section 2.6).

In this chapter we describe the tailored solver and motivate the choices we made in the design. To be able to exploit the structure of the equations we will rewrite the system in Section 7.2. The rewritten system will serve as a starting point for the construction of an (incomplete) block LU factorization. In Section 7.4 we will discuss the practical implementation of the preconditioner and the solution of the subsystems.

The performance of the preconditioner is not discussed in this chapter. We leave that for Chapter 8, where we solve several large scale ocean flows.

### 7.2 Rewriting the equations to ease the solution process

Before we describe the preconditioner for (7.1) we first rewrite the system in four subsequent steps, we (i) transform the pressure, (ii) transform the continuity equation, (iii) rename some submatrices and finally (iv) we rearrange columns and rows. The aim of all these operations is to get a system that is a better starting point for the development of a preconditioner. We remark that for the construction of the preconditioner we ignore the block $E_{uv}$ in (7.1), because the entries in this block are very small compared to entries in the rest of the matrix. There is a physical motivation for this as well. The terms $w u_z$ and $w v_z$ are small relative to the
Coriolis force and the pressure gradient in the momentum equations for \( u \) (Equation (2.4a)) and \( v \) (Equation (2.4b)) respectively.

The difficulties in solving Equation (7.1) are partially caused by the zero blocks on the diagonal. We could get rid of these blocks if it was allowed to interchange the (block) rows 2 and 3, which is not the case, because the matrices \( G_w \) and \( D_w \) are not square \( (d_w < d_p) \). However, after a smart transformation of the pressure and the continuity equation, we can split the matrices in two subblocks: a square invertible matrix and a zero matrix. The square and invertible submatrices then allow to interchange columns and rows. In the next paragraph we will pay attention to the transformation of the pressure.

**Step 1: transform the pressure**

In the continuous formulation the hydrostatic pressure equation, see Equation (2.4c), contains the term \( p_z \). Note that we can decompose the pressure in \( p = p + \tilde{p} \), where \( p(\phi, \theta) = \int_0^{-h} p(\phi, \theta, z) dz \) is a depth-averaged pressure, that vanishes in the hydrostatic pressure equation, because it is in the kernel of the operator \( \frac{\partial}{\partial z} \). From this it follows that \( \int_0^{-h} \tilde{p}(\phi, \theta, z) dz = 0 \). The consequence of this decomposition is that the problem \( f_z = g \) in the space of all functions with \( \int_0^{-h} f dz = 0 \), is now uniquely solvable.

We can do something similar on the discrete equations. As we pointed out in Section 2.4 the matrix \( G_w \) has dimensions \( d_w \times d_p \) and because \( d_w < d_p \) it is not square. The matrix has full rank, so the kernel has dimension \( d_p - d_w \). Because of the structured grid and the fact that \( G_w \) is a discrete gradient operator, it is quite easy to construct an orthonormal basis for this kernel. Given such a basis we can build the operator \( M_1 \), who’s rows are precisely these orthonormal basis vectors. Consequently \( G_w M_1^T = 0 \) and \( M_1 M_1^T = I_p \). The operator \( M_1 \) acts on the pressure space and is up to a factor a depth-averaging of the pressure field.

We now define the transformation matrix

\[
T_1 = \begin{bmatrix} I_w & M_1^T \\ 0 & I_1 \end{bmatrix},
\]

which is a square matrix, because it is the matrix \( M_1^T \) with dimension \( d_p \times (d_p - d_w) \) put together with a \( d_w \times d_w \) identity matrix and a \( (d_p - d_w) \times d_w \) zero block. If we choose an appropriate ordering for the variables \( u_p \) it is guaranteed that the matrix is invertible. In practice it suffices to order the pressure nodes such that all nodes at the surface are numbered last. Each surface node corresponds to one singular vector of \( G_w \).

With this \( T_1 \) we can rewrite the pressure field:

\[
u_p = T_1 \tilde{p} = \begin{bmatrix} I_w & M_1^T \\ 0 & I_1 \end{bmatrix} \begin{bmatrix} \tilde{p} \\ u_p \end{bmatrix}.
\]

(7.2)

If we use this equation in system (7.1) we get two transformed gradient matrices, namely

\[
G_{uv} T_1 = \begin{bmatrix} \tilde{G}_{uv} & G_{uv} M_1^T \end{bmatrix} \quad \text{and} \quad G_w T_1 = \begin{bmatrix} \tilde{G}_w & G_w M_1^T \end{bmatrix} = \begin{bmatrix} \tilde{G}_w & 0 \end{bmatrix},
\]

(7.3)
where the last equality holds because of the definition of $M_1$. The matrices $\tilde{G}_{uv}$ and $\tilde{G}_{w}$ contain the first $d_w$ columns of $G_{uv}$ and $G_{w}$ respectively. The matrix $\tilde{G}_w$ has $d_w$ rows as well, so it is square. Because $G_{w}$ and $\tilde{G}_w$ have the same rank the latter matrix necessarily is invertible.

**Step 2: transform the continuity equation**

Since $w_z = f$ with $w(0) = w(-h) = 0$ is overdetermined, there is no solution unless $\int_{-h}^0 f \, dz = 0$. The latter is obtained from the depth integration of the continuity equation, see Equation (2.4d), where $w$ vanishes because of the boundary conditions. This gives an equation for the horizontal velocities. With the horizontal velocities satisfying this equation, one can solve $w$ uniquely and leave out even one of the boundary conditions.

We can perform this depth integration as well on the discrete equations. In analogy to step 1 we use the operator $M_2$, who’s rows form an orthonormal basis for the kernel of $D_w^T$. So each row of $M_2$ is a left-singular vector of $D_w$. Consequently $M_2D_w = 0$ and $M_2M_2^T = I_p$. With $M_2$ we define the following transformation matrix

$$T_2 = \begin{bmatrix} I_w & 0 \\ M_2 & \end{bmatrix}.$$

The matrix is square and invertible.

Note that if $D_w^T = G_w$, we would have $M_1 = M_2$ and $T_1 = T_2^T$. However equality of the discrete divergence and gradient matrices only holds for equidistant grids. In case of a stretched grid the matrices $D_w^T$ and $G_w$ need some diagonal scaling to make them equal. The last property is important for stability.

We apply $T_2$ to the discrete equation of conservation mass, that is the third block-row in (7.1). We get two transformed divergence matrices

$$T_2D_{uv} = \begin{bmatrix} \tilde{D}_{uv} \\ M_2D_{uv} \end{bmatrix} \quad \text{and} \quad T_2D_w = \begin{bmatrix} \tilde{D}_w \\ M_2D_w \end{bmatrix} = \begin{bmatrix} \tilde{D}_w \\ 0 \end{bmatrix}.$$

(7.4)

Where the matrices $\tilde{D}_{uv}$ and $\tilde{D}_w$ contain the first $d_w$ rows of $D_{uv}$ and $D_w$ respectively. The last matrix has $d_w$ columns as well so $\tilde{D}_w$ will be square and invertible. We get the zero block in the last equality because of the definition of $D_w$. Furthermore in the right-hand-side we get

$$T_2b_p = \begin{bmatrix} b_{\bar{\rho}} \\ b_{\bar{\rho}r} \end{bmatrix},$$

where $b_{\bar{\rho}} = [I_w \ 0]b_p$ and $b_{\bar{\rho}r} = M_2b_p$. 

Step 3: isolate saddle point problem and rename subblocks

If we put the transformations together we get the following system

\[
\begin{bmatrix}
A_{uv} & 0 & \tilde{G}_{uv} & G_{uv}M_1^T & 0 \\
0 & 0 & \tilde{G}_w & 0 & B_{TS} \\
\tilde{D}_{uv} & \tilde{D}_w & 0 & 0 & 0 \\
M_2D_{uv} & 0 & 0 & 0 & 0 \\
B_{uv} & B_w & 0 & 0 & A_{TS}
\end{bmatrix}
\begin{bmatrix}
u_{uv} \\
\tilde{u}_v \\
\tilde{u}_w \\
u_p \\
u_T \\
\end{bmatrix}
= 
\begin{bmatrix}
b_{uv} \\
b_w \\
b_{T}\tilde{\rho} \\
b_{\tilde{\tau}} \\
b_{T}\tilde{\tau} \\
b_{TS}
\end{bmatrix}. \tag{7.5}
\]

To simplify the matrix we define

\[K_{uv\tilde{\tau}} = \begin{bmatrix}
A_{uv} & G_{uv}M_1^T \\
M_2D_{uv} & 0
\end{bmatrix}, \tag{7.6}
\]

which is a saddle point matrix involving the full velocity field and the depth-averaged pressure field. \(K_{uv\tilde{\tau}}\) is a submatrix of (7.5) (involving block-row and -column 1 and 4) and we want to isolate this system. To do so we have to define

\[
\hat{B}_{uv} = [ \begin{bmatrix}
B_{uv} & 0
\end{bmatrix} ], \hat{D}_{uv} = [ \begin{bmatrix}
\tilde{D}_{uv} & 0
\end{bmatrix} ], \quad \hat{G}_{uv} = [ \begin{bmatrix}
\tilde{G}_{uv} \\
0
\end{bmatrix}], u_{uv\tilde{\tau}} = [ \begin{bmatrix}
u_{uv} \\
u_{\tilde{\tau}}
\end{bmatrix} ] \text{ and } b_{uv\tilde{\tau}} = [ \begin{bmatrix}
b_{uv} \\
b_{\tilde{\tau}}
\end{bmatrix} ]. \tag{7.7}
\]

Furthermore we introduce \(A_p = \tilde{G}_w\) and \(A_w = \tilde{D}_w\), to reflect that the matrices are square and invertible. After some rearrangement we get the system

\[
\begin{bmatrix}
K_{uv\tilde{\tau}} & 0 & \hat{G}_{uv} & 0 \\
0 & 0 & A_p & B_{TS} \\
\hat{D}_{uv} & A_w & 0 & 0 \\
\hat{B}_{uv} & B_w & 0 & A_{TS}
\end{bmatrix}
\begin{bmatrix}
u_{uv\tilde{\tau}} \\
u_{w} \\
\tilde{u}_p \\
u_T
\end{bmatrix}
= 
\begin{bmatrix}
b_{uv\tilde{\tau}} \\
b_w \\
b_{T}\tilde{\rho} \\
b_{TS}
\end{bmatrix}. \tag{7.9}
\]

This system looks the same as system (7.1). The only difference seems the names of the blocks and some hats and tildes. So what is the gain of the transformations and redefinition of all the subblocks? Well first of all the systems (7.1) and (7.9) may look similar, but they are different. The most important difference is that the blocks \(A_w\) and \(A_p\) in (7.9) are square and invertible, whereas \(G_w\) and \(D_w\) in (7.1) have a different number of columns and rows. Because these two matrices now are of the same size and invertible we can swap rows or columns 2 and 3, such that the blocks move to the diagonal. Doing so we reduce the number of zeros on the diagonal dramatically. The second important difference is that by the transformation, rearrangement and redefinition of subblocks, we moved part of the discrete gradient and the divergence matrices to the heading block \(K_{uv\tilde{\tau}}\) that became a saddle point matrix instead of the simple discrete convection-diffusion equation with Coriolis \(A_{uv}\) in (7.1). As we saw in Chapter 6 in general saddle point problems are more difficult to solve, but fortunately in this case we have good preconditioners available.
Step 4: rearrange columns and rows

Note that in system (7.9) there is a cyclic dependency of the different types of unknowns. If
the pressure field $\mathbf{u}_p$ is given, we can use the first (block) row to solve the horizontal velocity
and the depth-averaged pressure $\mathbf{u}_{av\overline{p}}$. Given these two we can compute the vertical velocity
$\mathbf{u}_w$ with the third row. Then we can use the last row to solve $\mathbf{u}_{TS}$, which on its turn can be used
to compute a new $\mathbf{u}_p$ with the second row. In a scheme this dependency is

$$
\mathbf{u}_p \Rightarrow \mathbf{u}_{av\overline{p}} \Rightarrow \mathbf{u}_w \Rightarrow \mathbf{u}_{TS} \Rightarrow \mathbf{u}_p.
$$

This dependency can be made more explicit if we asymmetrically rearrange the columns (vari-
ables) and rows (equations) in the matrix. Numbering the rows and columns from 1 to 4, we
use for the rows the permutation $q_r = (2, 1, 3, 4)$ and for the columns $q_c = (3, 1, 2, 4)$. This
rearrangement gives a system with an almost lower block triangular matrix

$$
\begin{bmatrix}
A_p & 0 & 0 & B_{TS} \\
\hat{G}_{av} & K_{av\overline{p}} & 0 & 0 \\
0 & \hat{D}_{av} & A_w & 0 \\
0 & \hat{B}_{av} & B_w & A_{TS}
\end{bmatrix}
\begin{bmatrix}
\mathbf{u}_p \\
\mathbf{u}_{av\overline{p}} \\
\mathbf{u}_w \\
\mathbf{u}_{TS}
\end{bmatrix}
= \begin{bmatrix}
\mathbf{b}_w \\
\mathbf{b}_{av\overline{p}} \\
\mathbf{b}_w \\
\mathbf{b}_{TS}
\end{bmatrix}.
$$

We will call this transformed and permuted Jacobian matrix $\hat{\Phi}$.

If we use wind forcing only, the blocks $\hat{B}_{av}$ and $B_w$ are empty. Then we can solve $\mathbf{u}_{TS}$ first,
followed by $\mathbf{u}_p$, $\mathbf{u}_{av\overline{p}}$ and $\mathbf{u}_w$.

Remark 7.2.1. There are two other permutations of (7.9) that give an almost lower block
triangular matrix. If we use $q_r = (1, 3, 4, 2)$ and $q_c = (1, 2, 4, 3)$, the matrix $A_p$ becomes the
last diagonal block. Then $\hat{G}_{av}$ is the only upper diagonal block. The last possibility is $q_r =
(3, 4, 2, 1)$ and $q_c = (2, 4, 3, 1)$, which gives $K_{av\overline{p}}$ as last diagonal block and two nonzero upper
diagonal blocks, namely $\hat{D}_{av}$ and $\hat{B}_{av}$. The permutation that we chose corresponds best to the
ordering of variables in barotropic-baroclinic splitting, that we discussed in the introduction.
We will further motivate this choice in Remark 7.3.1.

Note that so far we did not essentially change the system; equations (7.1) and (7.10) have
the same solution, although they look quite different. The most important advantage of (7.10)
is that the amount of zeros on the diagonal has reduced dramatically. (7.10) has only one zero
block (hidden in the heading block) where (7.1) has two. Furthermore, the zero block in the
rewritten system is rather small because it is part of the depth-averaged continuity equation.
The rewritten system appears to be more convenient as a starting point for the development of
a preconditioner than (7.1).
7.3 Block (incomplete) LU factorization

From \( \hat{\Phi} \) in Equation (7.10) we can construct the following block LU factorization

\[
L_{\hat{\Phi}} U_{\hat{\Phi}} = \begin{bmatrix}
A_p & 0 & 0 & 0 \\
\hat{G}_{uv} & K_{uvp} & 0 & 0 \\
0 & \hat{D}_{uv} & A_w & 0 \\
0 & \hat{B}_{uv} & \hat{B}_w & S_{TS}
\end{bmatrix}
\begin{bmatrix}
I_p & 0 & 0 & A_p^{-1}B_{TS} \\
0 & I_{uv+p} & 0 & -K_{uvp}G_{uvp}A_p^{-1}B_{TS} \\
0 & 0 & I_w & A_w^{-1}\hat{D}_{uv}K_{uvp}^{-1}G_{uvp}A_p^{-1}B_{TS} \\
0 & 0 & 0 & I_{TS}
\end{bmatrix},
\]  

(7.11)

where the Schur complement is given by

\[
S_{TS} = A_{TS} + (\hat{B}_{uv} - \hat{B}_wA_w^{-1}\hat{D}_{uv})K_{uvp}^{-1}\hat{G}_{uvp}A_p^{-1}B_{TS}.
\]  

(7.12)

This factorization is exact. However in practice it is impossible to construct the Schur complement explicitly, because the matrix will be dense. Now there are two possibilities:

(i) Totally forget the Schur complement and simply use \( S_{TS} = A_{TS} \). In that case it is of no use to apply the block \( U_{\hat{\Phi}} \). This is the same as ignoring the block \( B_{TS} \) in (7.10), which leads to a block Gauss-Seidel preconditioner, which we will refer to as BLOCK-GS.

(ii) We solve Equation (7.12) via an iterative procedure. The application of the Schur complement to a vector can be decomposed in a number of matrix-vector multiplications and system solves, which is relatively cheap. The problem that remains is the search of a good preconditioner for \( S_{TS} \). This results in a kind of block incomplete LU factorization, which we will call the BLOCK-ILU preconditioner.

Remark 7.3.1. If we choose a different permutation of (7.9), as we mentioned in Remark 7.2.1, we get a different block LU factorization with a different Schur complement. Numerical experiments showed that in approach (i) the timing results for all permutation are close to each other, although the permutation that we chose seems slightly better. Within approach (ii) the differences in timing results are much bigger. In that case iterative solution of the Schur complement \( S_{TS} \) gives the best results.

7.4 The implementation of the preconditioner

The application of the factorization (7.11) as preconditioner requires the solution of two equations, one with \( L_{\hat{\Phi}} \) and one with \( U_{\hat{\Phi}} \). The solution of both equations can be decomposed in matrix-vector multiplications and the solution of systems with the diagonal blocks \( A_p, A_w, K_{uvp} \) and \( A_{TS} \) or the Schur complement \( S_{TS} \). We will discuss the solution of these systems in this section.
7.4.1 Solving the system for the pressure

The matrix $A_p = \tilde{G}_w$ is a square and invertible submatrix of $G_w$ as implicitly defined by Equation (7.3). $G_w$ is the discretization of the term $p_z$ in Equation (2.4c). Because of the B-grid (see Figure 2.3) and the fact that the hydrostatic pressure equation is discretized at the $w$-points, the matrix $G_w$ is a bi-diagonal matrix. In fact it is even a gradient-type matrix as in Definition 4.1.2 in Chapter 4. With row scaling we can get the entries of $A_p$, an invertible gradient-type matrix, in \{-1,0,1\}, which allows us to apply Lemma 4.4.3. We can conclude there exist row and column permutations such that $A_p$ is upper diagonal. Consequently an equation with $A_p$ can be solved exactly at low cost. Because we use structured grids it is easy to find the ordering such that $A_p$ is upper diagonal.

7.4.2 Solving the system for the vertical velocity

For the matrix $A_w = \hat{D}_w$ we have a similar reasoning. It is a square and invertible submatrix of $D_w$ as implicitly defined by Equation (7.4). The continuity equation (2.4d) is discretized at the position of $p$ on a B-grid (see Figure 2.3), which causes $\hat{D}_w^T = A_w^T$ to be a gradient-type matrix. Via row scaling and Lemma 4.4.3 we conclude that $A_p$ is a permutation of a lower diagonal matrix. So equations with $A_p$ are easy to solve exactly.

7.4.3 Solving the saddle point problem

Unfortunately not all subsystems are as easy to solve as the previous two. The saddle point problem $K_{uv\overline{p}}$ as defined in Equation (7.6) is much more difficult. To the reader this will not come as a surprise because all three previous chapters deal with the solution of saddle point problems. We will use the knowledge developed in those chapters here.

First of all let us recall the system that we want to solve:

$$
K_{uv\overline{p}} u_{uv\overline{p}} = 
\begin{bmatrix}
A_{uv} & G_{uv} M_1^T \\
M_2 D_{uv} & 0
\end{bmatrix}
\begin{bmatrix}
u_{uv} \\
u_{\overline{p}}
\end{bmatrix}
= 
\begin{bmatrix}
b_{uv} \\
b_{\overline{p}}
\end{bmatrix}.
$$

(7.13)

This saddle point system is a little weird, because it involves the full three dimensional field of horizontal velocities and the depth averaged pressure field, which is essentially two dimensional.

We will describe two ways to solve the equation

(a) Reduce the number of unknowns by depth-averaging the velocity field as well. Solve that system and use the equation $A_{uv} u_{uv} = b_{uv} - G_{uv} M_1^T u_{\overline{p}}$ to compute the full velocity field from the depth-averaged pressure.

(b) Solve the system at once.

Option (a) requires an operator for the depth averaging of the velocity field. Suppose the matrix $M_3$ is that operator. It can be constructed such that the rows of the matrix are orthonormal: $M_3 M_3^T = I_{\overline{p}}$. If we apply this operator to the velocity field and to the momentum...
7.4 The implementation of the preconditioner

We get the following saddle point problem

\[
\begin{bmatrix}
A_{uv} & G_{uv} \\
D_{uv} & 0
\end{bmatrix}
\begin{bmatrix}
u_{uv} \\
u_{pf}
\end{bmatrix} =
\begin{bmatrix}
b_{uv} \\
b_{pf}
\end{bmatrix},
\]  

(7.14)

where

\[
A_{uv} = M_3 A_{uv} M_3^T,
\]

\[
G_{uv} = M_3 G_{uv} M_1^T,
\]

\[
D_{uv} = M_2 D_{uv} M_3^T.
\]

It is important to note that Equations (7.14) and (7.13) are essentially different: the solution for \( u_{pf} \) is not the same, but hopefully the approximation is good enough. By depth integration, we at least use information that is spread all over the velocity field. One could choose a different operator than \( M_3 \), for example one that picks a single horizontal velocity field instead of averaging. Then the approximation becomes worse and the overall performance of the preconditioner deteriorates.

Note that the system (7.14) corresponds to the two-dimensional barotropic equations, that we mentioned in the introduction of this chapter.

The depth-averaged saddle point problem can be solved with a Krylov subspace method (see Section 3.2) using one of the preconditioners as described in Chapter 6. In Section 6.4.3, especially Table 6.8, we showed that for the depth-averaged equations the artificial compressibility preconditioner is a good candidate. The application of the artificial compressibility preconditioner requires the solution of the grad-div added matrix

\[
A^{GD}_{uv} = A_{uv} + \omega G_{uv} D_{uv},
\]

(7.15)

where the parameter \( \omega \) needs to be chosen quite large, due to bad scaling of the starting equations, which influences the size of the entries in the subblocks \( A_{uv}, G_{uv} \) and \( D_{uv}, \) and their depth-integrated counterparts. The modified simpler approach, that we described in Section 6.2.1 and 6.4.3, is a good alternative. As the method applies as well to the larger saddle point problem (7.13), we will discuss it in more detail at the end of this paragraph.

Note that the size of the depth averaged saddle point problem (7.14) is \( L \) (the number of cells in the depth) times smaller than the original saddle point problem. It is much cheaper to solve than the big saddle point system (7.13). The time to solve it is only a fraction of the time needed to solve the much bigger systems with \( A_{uv} \) and \( S_{TS} \). Therefore, the choice of the preconditioner is not a crucial factor.

Given \( u_{pf} \) we can compute \( u_{uv} \) from the equation

\[
A_{uv} u_{uv} = b_{uv} - G_{uv} M_1^T u_{pf}.
\]

This is again done with a Krylov subspace method involving a preconditioner for the matrix \( A_{uv} \).

The choice of \( M_3 \) is an important factor if we want to apply this method successfully. If we have no bottom topography and an equidistant grid the choice for the depth-averaging operator is quite clear. The method we described here gives a good preconditioner for the saddle point...
problem (7.13). However as soon as we introduce bottom topography or stretched grids, it is more difficult to construct the right $M_3$. If we choose the wrong one, the results deteriorate quickly. Apparently the solutions $u_{\mathcal{P}}$ of Equations (7.14) and (7.13) differ too much in that case. It might be possible to find a matrix $M_3$ such that the saddle point problem (7.14) is a better approximation to the barotropic equations and the solutions $u_{\mathcal{P}}$ of both saddle point problems are close to each other. A further study of the barotropic-baroclinic splitting, like in [42], could help to find the appropriate depth averaging operator for the horizontal momentum equations. So far we did not succeed to construct this matrix.

For the global ocean flows the bottom topography and a stretched grid are important, so we have to use option (b): we solve the saddle point system at once. Of the preconditioners that we described in the previous section, there is only one that is useful as preconditioner for the saddle point system (7.13): the MODIFIED SIMPLER approach. Unfortunately the artificial compressibility preconditioner, that performed quite well on the depth averaged saddle point problem, is useless here. It requires the construction of an (incomplete) LU-factorization for the grad-div stabilized system $A_{uv} + \omega G_{uv} M_1^T M_2 D_{uv}$, which is a matrix with much more entries than $A_{uv}$ itself. This is mainly caused by the fact that the matrix $M_1^T M_2$ has many nonzeros.

Fortunately the MODIFIED SIMPLER preconditioner - we introduced the SIMPLER method in Section 6.2.1 - for $K_{uv}$ is relatively cheap to construct and it appears to be a good preconditioner. For a more extended description of SIMPLE(R) including eigenvalue analysis see [85] and [50]. The modification of SIMPLER that we use is the following: instead of the diagonal of $A_{uv}$ we use a $2 \times 2$ block-diagonal $D_{A_{uv}}$. Because the unknowns $u$ and $v$ are clustered, this matrix $D_{A_{uv}}$ includes the Coriolis force, which dominates the momentum equations. This modification is crucial, since the standard SIMPLER preconditioner overlooks the important Coriolis force and therefore it fails to converge for Equation (7.13).

The application of MODIFIED SIMPLER can be reduced to the application of MODIFIED SIMPLE and SIMPLEL (see Section 6.3). Both require the solution of systems with $A_{uv}$ and
\[
C_{SI} = M_2 D_{uv} D_{A_{uv}}^{-1} G_{uv} M_1^T,
\]
which is an approximation to the Schur complement. We will discuss the solution of these systems at the end of this section.

### 7.4.4 Solving the system for temperature and salt

Here we have to make a distinction between the two possibilities that we sketched in Section 7.3: (i) the BLOCK-GS approach with $S_{TS} = A_{TS}$ or (ii) the BLOCK-ILU approach with the Schur complement $S_{TS}$ as in Equation (7.12).

The second approach is the most complex. It is expensive and in many cases even impossible to construct the exact Schur complement. To avoid the explicit construction we can use a Krylov subspace method (see Section 3.2) to solve the Schur complement. A Krylov method only requires that the matrix $S_{TS}$ can be applied to a vector. This is relatively easy and cheap, because (7.12) shows that we can decompose it in a number of smaller matrix-vector products.
and a few system solves (i.e. with $A_p$, $A_w$ and $K_{uv\eta}$, precisely the ones we just treated). The
next question we have to answer is: how do we obtain a good preconditioner for $S_{TS}$? It appears
that in our case an incomplete LU factorization of $A_{TS}$ suffices. One can get a better precondi-
tioner by constructing a factorization for a better approximation to the Schur-complement. For
example one can take into account more of the terms of (7.12). In general the construction of
such a better approximation will be quite expensive and the number of nonzeros will increase
rapidly, which makes the construction of an incomplete factorization much more expensive.
We tried several approximations, but none of them did beat the relatively cheap incomplete LU
factorization of $A_{TS}$.

7.4.5 Nested iterations?

There are three systems left to solve: the large submatrices $A_{TS}$ and $A_{uv}$, and one of the two
smaller systems $C_{S\ell}$ (as in Equation (7.16)) or $A_{uv}^{GD}$ (as in Equation (7.15)). Unlike the systems
$A_p$ and $A_w$ these four systems cannot be solved exactly at low cost. We use MRILU (see [10]
and Section 3.2.2) to build a preconditioner for the matrices. For the matrices $A_{TS}$ and $A_{uv}$
we use clustering of the variables $T, S$ and $u, v$ respectively. MRILU is able to build a good
factorization for these matrices.

We have to decide whether we use these preconditioners in an iterative method to solve
the systems accurately or apply the preconditioner only once every time a subsystem needs to
be solved. In the last case we in fact replace the matrices $S_{TS}$ and $K_{uv\eta}$ in (7.11) with their
preconditioners. We then need a relatively good MRILU factorization (i.e. with small drop
tolerance). The number of outer iterations will increase anyway, but we avoid nested iteration
schemes. If we apply nested iterations, we choose the flexible Krylov method (FGMRES, [62])
for the outer iteration.

7.5 Advantages of the tailored solver

At the end of this chapter we want to point at a few advantages of the BLOCK-GS/ILU solver
over the previous solver: MRILU directly applied to system (7.1), which needed a huge amount
of construction time, scaled badly and required much memory.

We expect the BLOCK-GS/ILU solver to scale almost linearly with the problem size. Most
of the operations that have to be performed for the construction and application of the precon-
ditioner, scale linearly with the problem size. For example, extraction of submatrices, com-
putation of sparse matrix transposes, matrix-vector products and the solution of systems with
lower or upper-triangular matrices can all be computed in $O(d)$ time. The only trouble could
come from the computation and application of the MRILU factorizations of the matrices $A_{uv}$,
$A_{TS}$ and $A_{uv}^{GD}$ or $C_{S\ell}$. Fortunately, these matrices are of convection-diffusion type and MRILU
shows almost grid-independent convergence for that kind of problems [10].
We also expect that less memory is needed for the BLOCK-GS/ILU solver. MRILU applied directly to (2.34) needs much memory because a small drop tolerance is required. The BLOCK-GS/ILU preconditioner requires the storage of the subblocks and the MRILU preconditioners for the much smaller matrices $A_{uv}$, $A_{TS}$ and $A^{GD}_{uv}$ or $C_{SI}$. Because MRILU is a suitable preconditioner for these matrices, the fill generated by the MRILU factorization will be rather small. Immediate consequence is that both construction and storage of the MRILU factors is much cheaper. Overall, we expect a serious decrease of the memory requirements when BLOCK-GS/ILU is used.

### 7.5.1 Data-assimilation

The solver that we developed in this chapter is primarily designed for use in the context of continuation of steady states, as we described in Section 2.3. However in the introduction in Chapter 1 we mentioned that we want to use the ocean model THCM in combination with the data-assimilation algorithm developed in [76]. In Section 2.6 we explained what this means for the solver: instead of systems with the steady state Jacobian $\Phi$, we have to solve systems with the time dependent Jacobian $\Phi_t$, as defined in Equation (2.43), as well as systems with its transpose $\Phi^T_t$. Does the BLOCK-GS/ILU solver apply to these two systems?

Let's start with $\Phi_t$. This matrix is given by $\Phi_t = 1/\Delta t M + \Phi$, with $M$ the diagonal matrix given in (2.42). The matrix has the same structure as (7.1), but the diagonals of $A_{TS}$ and $A_{uv}$ will be stronger, because we add $1/\Delta t$ to the diagonal entries. The solution of both subsystems plays an important role in the BLOCK-GS/ILU solver. As diagonal dominant matrices are easier to solve, we can expect the solution of time dependent systems to be much easier than the solution of steady-state Jacobian matrices, especially for small time steps $\Delta t$.

The solution of transposed system is no problem either. Except for the block $E_{uv}$, the structure of the matrix in (7.1) is symmetric. It is precisely this submatrix that we ignored in the design of the BLOCK-GS/ILU preconditioner (see the first paragraph of Section 7.2). In the derivation of the preconditioner all properties of $\Phi$ that we used are properties of $\Phi^T$ as well. Hence the BLOCK-GS/ILU suits data-assimilation.

### 7.5.2 Parallelization

The last issue about the tailored solver that we want to address is parallelization. If the tailored solver indeed gives the desired speed up, we will increase the resolution and push the solver to the new limit. Which means that on a single processor PC we easily meet the memory limits again. Therefore we would like to be able to run the ocean model on a parallel super computer. Hence the question whether the BLOCK-GS/ILU solver can be parallelized is important.

Many operations in the construction and the application of the solver allow for parallelization. Extraction of submatrices, matrix-vector products, do not give any trouble here.
7.6 Conclusions

The solution of the triangular matrices $A_w$ and $A_p$ is no problem either, because with appropriate reordering these matrices are block-diagonal matrices with as many blocks as there are horizontal grid points. The blocks itself are smaller triangular matrices, they have dimension $L$, the number of grid points in the depth. As long as all nodes with the same $\theta$ and $\phi$ coordinate are stored on the same processor, no communication between processors is required to solve these systems.

Without doubt the bottle neck in the parallelization of the solver is the solution of the convection-diffusion problem for the tracers $A_{TS}$ and the saddle point problem $K_{uv}$. In Section 7.4.3 we exposed that the last problem can be reduced to the solution of $A_{uv}$ and $A_{GD}^{uv}$ or $C_{SI}$.

The parallelization of the solver has been carried out within a Trilinos [40] framework. Trilinos contains many parallel matrix solvers, like incomplete LU factorizations, multilevel methods and it can be coupled to the parallel direct matrix solver MUMPS [4]. As the parallel code of THCM is still under development, showing results would be premature. Nevertheless we can say that we succeeded to parallelize the construction and the application of the preconditioner and that scaling of both with problem size and number of processors is reasonable. However the tuning of the submatrix solvers takes some time before we can present results.

7.6 Conclusions

In this chapter we presented a new solver for the Jacobian matrices in THCM. The most important difference with the old solver is the fact that we exploit the structure of the matrix, which hides a cyclic dependency of the variables. To reveal this dependency, in Section 7.2 we transformed the pressure field and the continuity equations, swapped columns and rows and renamed certain subblocks. These operations resulted in the almost upper block-triangular system (7.9). If we simply drop the upper block (a coupling between temperature, salinity and pressure) we get a block-Gauss-Seidel solver (BLOCK-GS). If we leave it untouched, we can construct an exact block LU factorization. As it is unpractical to construct the full Schur complement involved in the exact factorization, it has to be solved iteratively, which gives the BLOCK-ILU solver. The application of either one of the two solvers requires the solution of a number of subsystems, which are all easier to solve than the system as a whole, because they are well known and understood.

There are a few open questions: which of the two (GS/ILU) gives the best preconditioner, can we profit from nested iterations and how is the performance compared to the old solver? We leave the answer to these questions for the next chapter where will use the solver in several large scale computations with THCM.

The advantages of the new solver are the expected decrease in memory requirements and construction time and better scalability. Hence the BLOCK-GS/ILU solver has the potential to give the desired speed up in large scale ocean circulation problem. In the next chapter we will see if it is able to do so in practice.