# A Domain Decomposition Method for Almost Incompressible Flow

by

Lina Hemmingsson

Report No. 164/1995

UPPSALA UNIVERSITET
Institutionen för
teknisk databehandling

UPPSALA UNIVERSITY
Department of
Scientific Computing

# A Domain Decomposition Method for Almost Incompressible Flow [1]

Lina Hemmingsson

1995

[1] This work was supported by the Swedish National Board for Industrial and Technical Development, NUTEK, under contract No. 9303806

**Abstract**

A domain decomposition method for solving the Navier-Stokes equations for almost incompressible flow is examined. Due to a non-uniform decomposition of the domain, we have fast solvers in all subdomains. Hence each iteration on the Schur-complement system can be performed very efficiently. We have shown theoretically that the method requires much less memory positions and arithmetic operations than a direct method. Numerical experiments show that the iteration on the Schur-complement system converges very fast. We also show that the grid-quotient in the space-grid might be crucial for the performance of the method. Moreover, we show that for a given discretization of the problem, the rate of efficiency is larger than 100% for the problem studied here, due to the very nice parallelization properties of the algorithm.

# Contents

# 1 Introduction

We are interested in solving the compressible, isentropic Navier-Stokes equations in two space-dimensions for small Mach numbers $\epsilon$. This compressible model is a good approximation for incompressible flow, and has certain advantages. By introducing a transformed pressure $\phi = \frac{1}{\gamma\epsilon}(\rho^\gamma - 1)$ [4], where $\gamma$ is a constant characteristic for the fluid and $\rho$ is the density, the symmetrized equations for the unknowns $\bar{u} = \begin{pmatrix} \phi & u & v \end{pmatrix}^T$ read

$$\frac{\partial \bar{u}}{\partial t} + \frac{1}{\epsilon}\left(A_1' \frac{\partial \bar{u}}{\partial x_1} + A_2' \frac{\partial \bar{u}}{\partial x_2}\right) =$$

$$A_1(\bar{u})\frac{\partial \bar{u}}{\partial x_1} + A_2(\bar{u})\frac{\partial \bar{u}}{\partial x_2} + \frac{1}{Re}\left(B_1(\bar{u})\frac{\partial^2 \bar{u}}{\partial x_1^2} + B_2(\bar{u})\frac{\partial^2 \bar{u}}{\partial x_2^2} + B_{1,2}(\bar{u})\frac{\partial^2 \bar{u}}{\partial x_1 \partial x_2}\right). \tag{1}$$

Here

$$A_1' = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A_2' = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$

$$A_1(\bar{u}) = \begin{pmatrix} u & \gamma\phi & 0 \\ \gamma\phi & u & 0 \\ 0 & 0 & u \end{pmatrix}, \quad A_2(\bar{u}) = \begin{pmatrix} v & 0 & \gamma\phi \\ 0 & v & 0 \\ \gamma\phi & 0 & v \end{pmatrix},$$

$$B_1(\bar{u}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{\lambda+2\mu}{\rho} & 0 \\ 0 & 0 & \frac{\mu}{\rho} \end{pmatrix}, \quad B_2(\bar{u}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{\mu}{\rho} & 0 \\ 0 & 0 & \frac{\lambda+2\mu}{\rho} \end{pmatrix},$$

and

$$B_{1,2}(\bar{u}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{\lambda+\mu}{\rho} \\ 0 & \frac{\lambda+\mu}{\rho} & 0 \end{pmatrix},$$

where $\lambda$ and $\mu$ are viscosity coefficients and $Re$ is the Reynolds number. Note that $A_d'$, $d = 1, 2$ are constant, whereas $A_d$, $B_d$ depend on the solution $\bar{u}$.

In [5] it is shown that for almost incompressible flows (low Mach numbers), it is possible to compute an accurate solution even if the fast sound oscillations are not resolved by the time-marching method. For an explicit method in time, the CFL-criterion restricts the time-step as $\Delta t \leq \mathcal{O}(h\epsilon)$. The operator

$$\frac{1}{\epsilon}\left(A_1' \frac{\partial \bar{u}}{\partial x_1} + A_2' \frac{\partial \bar{u}}{\partial x_2}\right) \tag{2}$$

which contains the sound waves should hence be discretized implicitly in time. In [6], [12], (1) is discretized with the Leap-Frog/Backward Euler (LFBE) scheme.

The leap-frog scheme is employed for the nonlinear differential operator, while the backward Euler scheme is used for (2). For smooth solutions and almost incompressible flows, the LFBE scheme is accurate of order $\mathcal{O}(h^2 + \epsilon\Delta t)$. It is numerically stable for

$$\frac{\Delta t}{h_{min}}\left(|u| + |v|\right) < 1 - \frac{\Delta t}{h_{min}^2}\frac{4\left(\lambda + 3\mu\right)}{\rho Re}. \tag{3}$$

The semi-implicit discretization in time leads to a system of equations to solve for each time-step

$$A\bar{u} = \bar{b}. \tag{4}$$

In [12], (4) is solved using NAG-routines. Holmgren and Otto use an iterative method with a semi-circulant preconditioner to solve (4), [9]. They also use a "boundary correction method" [10]. These techniques require logically rectangular domains. A generalization to more complex domains is also presented in [10], but this method is not implemented.

In this paper we use a domain decomposition method presented in [8], to solve (4) on more general domains. Numerical results from an Intel Paragon are presented demonstrating the great flexibility with this method. We also present how the method can be generalized to three space-dimensions. The direct method presented in [12] lacks this possibilty due to its extreme demand of memory.

# 2 The numerical method

## 2.1 The geometry

We consider flow over a backward facing step, see Figure 1. For comparison we compute the same cases as in [12].

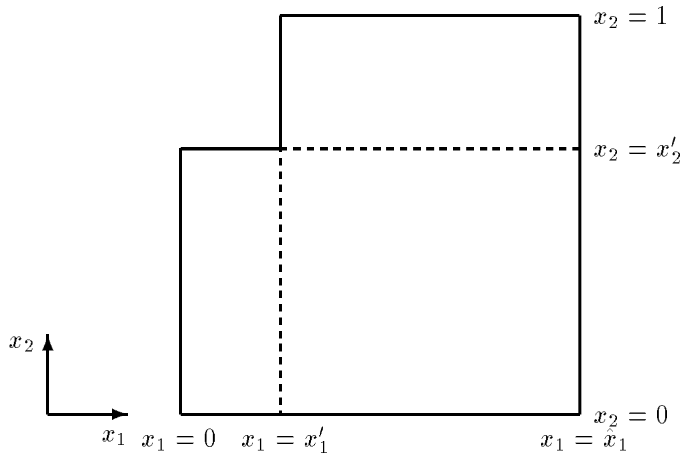


Figure 1: The domain.

The solid lines at $x_2 = 0$, $x_2 = x_2'$, and $x_2 = 1$ represent solid walls. The boundaries at $x_1 = 0$ and $x_1 = \hat{x}_1$ are inflow and outflow boundaries respectively.

In Section 3.1 we decompose this domain in vertical strips for the domain decomposition method.

## 2.2 The discretization

We consider the LFBE scheme, which for Equation (1) gives

$$\left(I + \frac{1}{\epsilon}Q_0\right)\bar{u}^{n+1} + Q_1(\bar{u}^n)\bar{u}^n = \left(I + \frac{1}{Re}Q_2(\bar{u}^{n-1})\right)\bar{u}^{n-1}, \tag{5}$$

where the operators $Q_0$, $Q_1(\bar{u})$ and $Q_2(\bar{u})$ are defined by

$$Q_0 = 2\Delta t\,(A_1' D_{0,x_1} + A_2' D_{0,x_2}),$$

$$Q_1(\bar{u}) = 2\Delta t\,(A_1(\bar{u}) D_{0,x_1} + A_2(\bar{u}) D_{0,x_2}),$$

and

$$Q_2(\bar{u}) = 2\Delta t\,(B_1(\bar{u}) D_{+,x_1} D_{-,x_1} + B_2(\bar{u}) D_{+,x_2} D_{-,x_2} + B_{1,2}(\bar{u}) D_{0,x_1} D_{0,x_2}).$$

We consider a grid defined by gridpoints $x_{1,k}$, $x_{2,j}$, $k = 1, \ldots, m_1$, $j = 1, \ldots, m_2$. At the corner we have $k = m_1'$, $j = m_2' + 1$. The space-step in the $x_d$-direction is denoted by $h_d$ and the grid function is defined by

$$\bar{u}_{k,j} = \begin{pmatrix} \phi_{k,j} \\ u_{k,j} \\ v_{k,j} \end{pmatrix}.$$

We will further discuss the decomposition of the domain in Section 3.1.

The discretization in time is $t_n = n\Delta t$. We start the calculation at rest. Since (5) requires initial data at two time-levels, we define $\bar{u} \equiv 0$ at $t = -\Delta t$ and $t = 0$.

## 2.3 Boundary conditions

At the inflow boundary $x_1 = 0$ we prescribe

$$\begin{aligned}
\phi(0, x_2, t) + \epsilon u(0, x_2, t) &= \epsilon g_1(x_2, t),\\
v(0, x_2, t) &= g_2(x_2, t),\\
\frac{\partial u}{\partial x_1}(0, x_2, t) &= g_3(x_2, t),
\end{aligned}$$

and at the outflow boundary $x_1 = \hat{x}_1$ we use

$$\begin{aligned}
u(\hat{x}_1, x_2, t) &= g_4(x_2, t),\\
\frac{\partial v}{\partial x_1}(\hat{x}_1, x_2, t) &= g_5(x_2, t).
\end{aligned}$$

To achieve second order accuracy, the boundaries are centered between the two utmost gridlines. For $j = 1, \ldots, m_2'$ this yields

$$\begin{aligned}
\frac{\phi_{0,j}^{n+1} + \phi_{1,j}^{n+1}}{2} + \epsilon \frac{u_{0,j}^{n+1} + u_{1,j}^{n+1}}{2} &= \epsilon g_1(x_{2,j}, t_{n+1}),\\
\frac{v_{0,j}^{n+1} + v_{1,j}^{n+1}}{2} &= g_2(x_{2,j}, t_{n+1}),\\
\frac{u_{1,j}^{n+1} - u_{0,j}^{n+1}}{h_1} &= g_3(x_{2,j}, t_{n+1}),
\end{aligned}$$

and for $j = 1, \ldots, m_2$

$$\begin{aligned}
\frac{u_{m_1,j}^{n+1} + u_{m_1+1,j}^{n+1}}{2} &= g_4(x_{2,j}, t_{n+1}),\\
\frac{v_{m_1+1,j}^{n+1} - v_{m_1,j}^{n+1}}{h_1} &= g_5(x_{2,j}, t_{n+1}).
\end{aligned}$$

At the solid walls and at the outflow boundary, second order extrapolation is used for $\phi$. At the corner, $\phi$ is defined by

$$\phi_{m_1',m_2'+1}^{n+1} = \phi_{m_1',m_2'}^{n+1} + \phi_{m_1'+1,m_2'+1}^{n+1} - \phi_{m_1'+1,m_2'}^{n+1}.$$

The velocities are set to 0 at all solid walls. These boundary conditions have been analyzed with respect to stability, well-posedness and accuracy in [6].

In this paper we consider

$$\begin{aligned}
g_1(x_2,t) &= \begin{cases} 0.6\cdot\sin^2\left(\frac{\pi}{2}t\right) & , \quad 0\le t\le 1,\\ 0.6 & , \quad t>1,\end{cases}\\
g_2(x_2,t) &= 0.0,\\
g_3(x_2,t) &= 0.0,\\
g_4(x_2,t) &= \begin{cases} 0.6\cdot\sin^2\left(\frac{\pi}{2}t\right) & , \quad 0\le t\le 1,\\ 0.6 & , \quad t>1,\end{cases}\\
g_5(x_2,t) &= 0.0,
\end{aligned}$$

which is also used in [6].

## 2.4 Grid stretching

In order to resolve boundary layers near the solid walls we use a stretched grid in the $x_2$-direction. The stretching is defined by

$$x_2(\xi_2) = \begin{cases} \frac{1-\xi_2'}{2}\tanh\left(a\left(\xi_2-\frac{1+\xi_2'}{2}\right)\right)/\tanh\left(a\frac{1-\xi_2'}{2}\right)+\frac{1+\xi_2'}{2} & \xi_2'<\xi_2\le 1,\\ \frac{\xi_2'}{2}\tanh\left(b\left(\xi_2-\frac{\xi_2'}{2}\right)\right)/\tanh\left(b\frac{\xi_2'}{2}\right)+\frac{\xi_2'}{2} & 0\le\xi_2\le\xi_2'.\end{cases} \tag{6}$$

To achieve continous derivative of $x_2$ with respect to $\xi_2$ at $\xi_2' = x_2'$ we set $b = a(1-\xi_2')/\xi_2'$. We do not exploit a stretched grid in the $x_1$-direction, since there will not be any sharp boundary layers in this direction. In a more general case some stretching should be taken into account even in this direction. This will be further discussed in Section 6.

# 3 The domain decomposition method

## 3.1 The decomposition of the domain

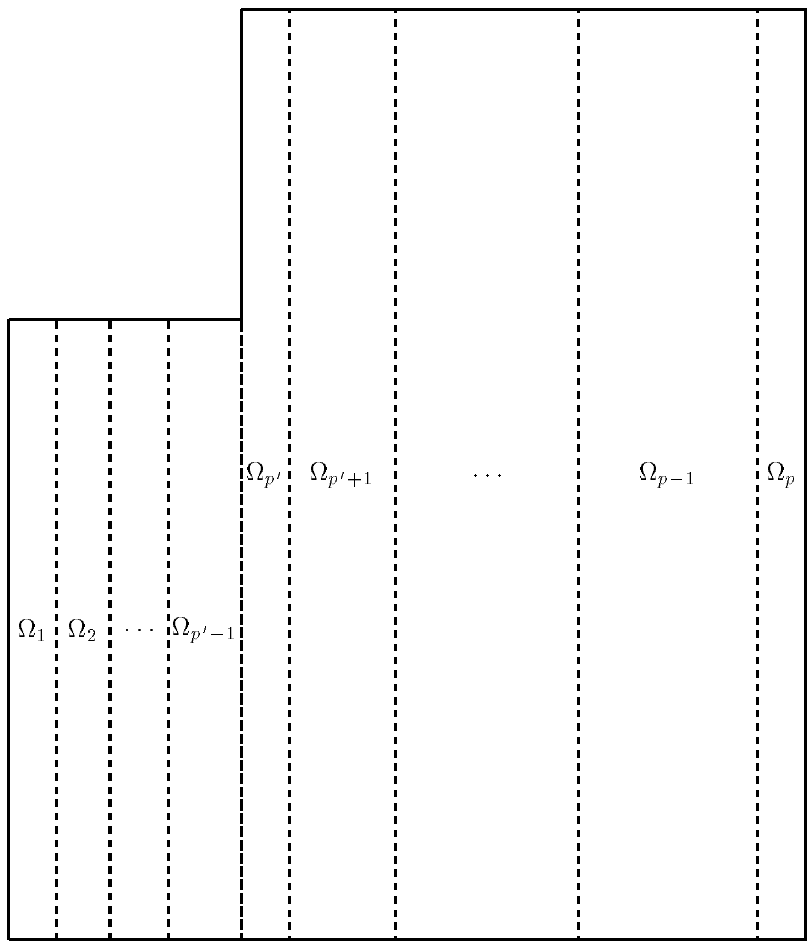


Figure 2: The domain.

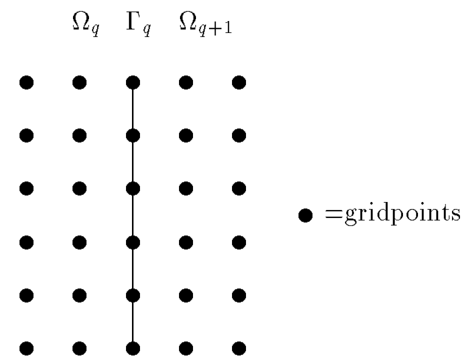


Figure 3: A blow-up of $\Gamma_q$.

The domain is decomposed in $p$ vertical strips $\Omega_q$, $q = 1, \ldots, p$, see Figure 2. We denote the number of vertical inner gridlines in $\Omega_q$ by $m_1^q - 1$. Each such gridline contains $m_2^q$ inner gridpoints. The boundary between $\Omega_q$ and $\Omega_{q+1}$ is denoted by $\Gamma_q$, and consists of one gridline. Note that there is no overlap between $\Omega_q$, $\Gamma_q$, and $\Omega_{q+1}$. In $\Omega_1$, $\Omega_{p'}$, and $\Omega_p$ we only have one inner gridline. This is due to the fact that we then have fast solvers for all subdomain systems which we will show in Section 3.4.

### 3.2 The Schur-complement method

In this paper we consider the so called Schur-complement method. The original system of equations (4) is reduced to a smaller system for the unknowns on $\cup_{q=1}^{p-1}\Gamma_q$. Once this Schur-complement system is solved, the local subdomain systems can be solved independently of each other. The inherent parallelism can be exploited using a computer with a parallel architecture.

Denote the unknowns on $\cup_{q=1}^{p-1}\Gamma_q$ by $\bar{u}_0$ and those in $\cup_{q=1}^{p}\Omega_q$ by $\bar{u}_1$. We then have

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \begin{pmatrix} \bar{u}_0 \\ \bar{u}_1 \end{pmatrix} = \begin{pmatrix} \bar{b}_0 \\ \bar{b}_1 \end{pmatrix}. \tag{7}$$

The algorithm for solving (7) is

1. Solve $A_{11}\bar{w}_1 = \bar{b}_1$
2. Solve $C\bar{u}_0 = \bar{b}_0 - A_{01}\bar{w}_1$
3. Solve $A_{11}\bar{v}_1 = A_{10}\bar{u}_0$
4. $\bar{u}_1 = \bar{w}_1 - \bar{v}_1$.

Here $C = A_{00} - A_{01}A_{11}^{-1}A_{10}$ is the Schur-complement. In Section 3.3 we show how to solve the Schur-complement system, and in Section 3.4 we discuss the solution of $A_{11}\bar{x}_1 = \bar{y}_1$.

### 3.3 Schur-complement system solver

We solve the Schur-complement system

$$C\bar{u}_0 = \bar{b}_0 - A_{01}\bar{w}_1 \equiv \bar{f}_0 \tag{8}$$

using a CG-like iterative method. In this paper we consider GCR($\ell$), Generalized Conjugate Residuals with restarting length $\ell$, [2]. The reason for not using a more modern method such as GMRES($\ell$) [11], is that we had well-tested code for GCR($\ell$). However, GCR($\ell$) and GMRES($\ell+1$) are mathematically equivalent.

GCR is a minimal residual method [3], which in iteration $k$ fulfills

$$||\bar{r}_0^k||_2 = \min_{p_k \in \mathcal{P}, p_k(0)=1} ||p_k(C)\bar{r}_0^0||_2.$$

Here $\mathcal{P}$ is the set of all polynomials of degree $k$, $\bar{r}_0^k \equiv C\bar{u}_0^k - \bar{f}_0$, and $\bar{u}_0^k$ is the approximation of $\bar{u}_0$ obtained in iteration $k$. If $C$ is diagonalizable we obtain

$$\frac{||\bar{r}^k||_2}{||\bar{r}^0||_2} \leq \mathrm{cond}_2(W_C) \min_{(p_k \in \mathcal{P}, p_k(0)=1)} \max_{1 \leq j \leq \eta} |p_k(\lambda_j)| \equiv \mathrm{cond}_2(W_C) \cdot \varepsilon_k,$$

where $W_C$ is the eigenvector-matrix and $\lambda_j$ are the eigenvalues of $C$. We also define the asymptotic convergence factor $\varrho$ as

$$\varrho \equiv \lim_{k \to \infty} \varepsilon_k^{1/k}. \tag{9}$$

In [8] a scalar first-order PDE is considered

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2} = g(x_1, x_2, t).$$

This equation is discretized exploiting the trapezoidal rule in time, and centered differences in space in the interior of the domain. In one space-dimension a periodic boundary condition is prescribed, and in the other one-sided differences are employed at the outflow boundary. It is theoretically proved that the Schur-complement $C$ obtained from this discretization is well-conditioned, and that the convergence-rate depends on the grid-quotient

$$\varphi = \frac{h_1}{h_2}. \tag{10}$$

We restate the theorem here without proof.

**Theorem 3.1** *Define*

$$\kappa_d = \frac{\Delta t}{h_d} \quad , \quad d = 1, 2.$$

*For $\kappa_1$ large and fixed, the asymptotic convergence factor $\varrho$ defined in (9), fulfills*

$$\varrho \leq \sqrt{\frac{4\varphi^4}{\kappa_1^2(1-\varphi^2)} + 2 - \varphi^2 - 2\sqrt{1-\varphi^2}}.$$

□

In Figure 4 we display the upper limit for $\varrho$ as a function of $\varphi$, for $\kappa_1 = 10$.

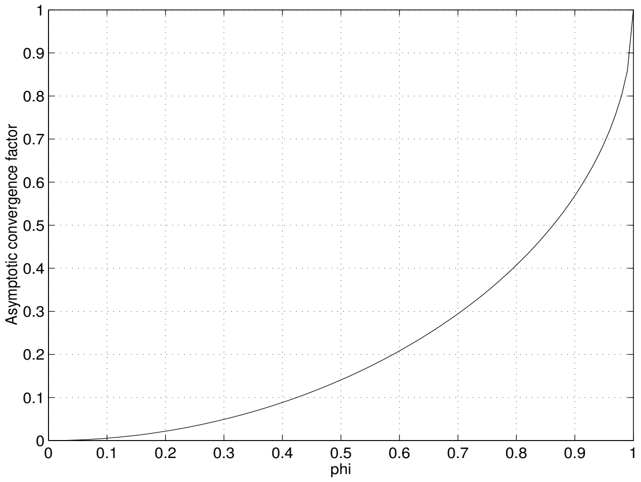


Figure 4: Upper limit for $\varrho$.

The correspondance to $\kappa_d$ for the LFBE scheme to the Navier-Stokes equations is

$$\mathcal{K}_1 = \begin{pmatrix} 0 & \kappa_1' & 0 \\ \kappa_1' & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\mathcal{K}_2 = \begin{pmatrix} 0 & 0 & \kappa_2' \\ 0 & 0 & 0 \\ \kappa_2' & 0 & 0 \end{pmatrix},$$

$$\kappa_d' = \tfrac{\Delta t}{\epsilon h_d} \quad , \quad d = 1, 2. \tag{11}$$

In Section 5 we will examine how Theorem 3.1 applies when we solve (8) with GCR for the Navier-Stokes equations.

Since the iterative method includes matrix-vector-multiplications $Cx$ we have to solve each subdomain system once each iteration. In Section 3.4 we discuss how to accomplish these solves.

## 3.4 Subdomain systems solvers

In Section 3.1 we prescribed that $\Omega_1$, $\Omega_{p'}$, and $\Omega_p$ should consist of only one gridline. This nonuniform decomposition is introduced since the numerical boundary conditions at the inflow and outflow boundary and at $x_1 = x_1'$ disturb the structure of the coefficient-matrices. By imposing this nonuniform decomposition we isolate these disturbances to $\Omega_1$, $\Omega_{p'}$, and $\Omega_p$. This methodology is further discussed in [8] and is shown numerically to be very efficient for scalar PDEs.

The coefficient-matrix $A_{11}$ is given by

$$A_{11} = \mathrm{diag}(A_{11}^1, \ldots, A_{11}^p).$$

In [8] it is shown that

$$\begin{array}{c} A_{11}^q = (I_{m_2^q} \otimes S_{m_1^q-1} \otimes I_3) T^q (I_{m_2^q} \otimes S_{m_1^q-1} \otimes I_3)^H \\ q = 2, \ldots, p'-1, p'+1, \ldots, p-1 \end{array} , \tag{12}$$

where

$$T^q = \begin{pmatrix} \Delta_1^q & \Theta_1^q & & & \\ \Psi_2^q & \Delta_2^q & \Theta_2^q & & \\ & \ddots & \ddots & \ddots & \\ & & \Psi_{m_2^q-1}^q & \Delta_{m_2^q-1}^q & \Theta_{m_2^q-1}^q \\ & & & \Psi_{m_2^q}^q & \Delta_{m_2^q}^q \end{pmatrix},$$

and

$$\begin{aligned} \Delta_j^q &= \mathrm{diag}\left(\delta_{1,j}^q, \ldots, \delta_{m_1^q-1,j}^q\right), \\ \Psi_j^q &= \mathrm{diag}\left(\sigma_{1,j}^q, \ldots, \sigma_{m_1^q-1,j}^q\right), \\ \Theta_j^q &= \mathrm{diag}\left(\theta_{1,j}^q, \ldots, \theta_{m_1^q-1,j}^q\right). \end{aligned}$$

Here $\delta_{k,j}^q$, $\sigma_{k,j}^q$, and $\theta_{k,j}^q$ are dense blocks of size $3 \times 3$. Hence the solution of

$$T^q \bar{x}_1^q = \bar{y}_1^q \tag{13}$$

decouples into the solution of $m_1^q - 1$ independent banded systems of order $3m_2^q$ with bandwidth 11. Exploiting symmetry relations it can be shown that only $m_1^q/2$ of these systems have to be solved.

The matrix $S_{m_1^q-1}$ is the modified sine matrix of order $m_1^q - 1$ defined in [7]. This matrix has entries

$$S_{m_1^q-1}(j,k) = \sqrt{\frac{2}{m_1^q}}\, i^{j+k+1} \sin\left(\frac{jk\pi}{m_1^q}\right).$$

From (12) we get that

$$A_{11}^q \bar{w}_1^q = \bar{b}_1^q \quad , \quad q = 2, \ldots, p'-1, p'+1, \ldots, p-1$$

can be solved as

1. $\bar{z}_1^q = (I_{m_2^q} \otimes S_{m_1^q - 1} \otimes I_3)^H \bar{b}_1^q$
2. Solve $T^q \bar{x}_1^q = \bar{z}_1^q$
3. $\bar{w}_1^q = (I_{m_2^q} \otimes S_{m_1^q - 1} \otimes I_3) \bar{x}_1^q$

Step 1. and 3. consists of $3m_2^q$ modified sine transforms and inverse modified sine transforms respectively. In [7] it is shown that each such transform requires $\mathcal{O}(2.5 m_1^q \log_2 m_1^q)$ arithmetic operations. The system in step 2. is solved using banded Gaussian elimination.

The subdomain systems in $\Omega_1$, $\Omega_{p'}$, and $\Omega_p$ are of the same form as the systems in step 2. above, and hence solved using the same algorithm.

# 4 A comparison with a direct method

In [12], (4) is discretized on a single domain, and the system of equations is solved using NAG-routines. These routines implement a direct solution method, which try to exploit the sparsity in the coefficient-matrix. Here, we will compare the method presented in this paper and standard banded Gaussian elimination with respect to memory requirements and arithmetic operations. We consider the decomposition defined in Section 3.1 and

$$m_1^q = \frac{m_1 - 5}{p - 3} \quad , \quad q = 2, \ldots, p'-1, p'+1, \ldots, p-1 \ . \tag{14}$$

In this paper we also assume that each processor in the parallel computer solves for the unknowns in one subdomain and one inner boundary. Only processor number $p$ has no inner boundary.

## 4.1 Storage requirements

We start by studying the Domain Decomposition method (DD) presented in this paper. Denote by $M_q$, the number of memory positions processor $q$ requires. Then

$$M_q = \frac{159}{2} m_1^q m_2^q + 27 m_1^q + (233 + 6\ell) m_2^q.$$

As we run the same program in each processor, we need to allocate $\hat{M}$ memory positions in each processor where

$$\hat{M} = \max_q M_q,$$

and

$$\hat{M} = \frac{159}{2}\frac{m_1 - 5}{p-3}m_2 + 27\frac{m_1-5}{p-3} + (233 + 6\ell)m_2.$$

Thus we need

$$\hat{M}_{tot} = p\hat{M} = p\left(\frac{159}{2}\frac{m_1 - 5}{p-3}m_2 + 27\frac{m_1-5}{p-3} + (233 + 6\ell)m_2\right)$$

memory positions in total. If we consider large problems this gives that we need approximately 159/2 memory positions per gridpoint.

For banded Gaussian elimination (bandGE) we need $27m_1$ memory positions per gridpoint. In Figure 5 we compare the number of memory positions required for bandGE and DD.

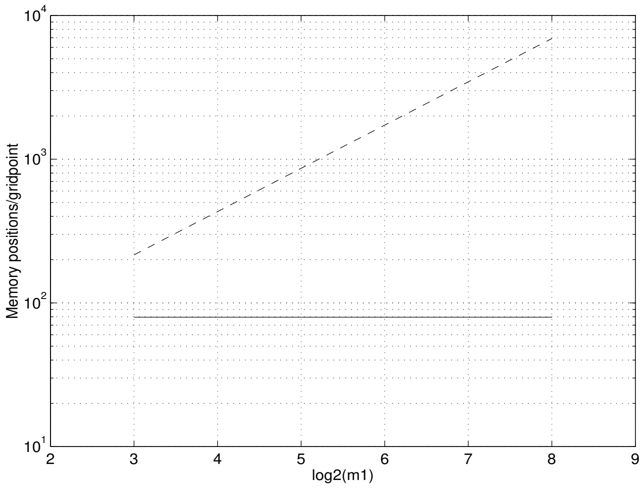


Figure 5: Memory positions required per gridpoint. Solid line represents DD and dashed line bandGE.

## 4.2 Arithmetic operations

### 4.2.1 Factorization

We start by considering the arithmetic operations that are performed before the time-stepping. These operations are basically the factorization of the systems defined in Section 3.4, that are solved with banded Gaussian elimination. Since the PDE has constant coefficients, these systems can be factorized once prior to the time-stepping. In Table 1 we present the requirement of arithmetic operations for the different subdomains/processors.

| $q$ | #a.o. |
|---|---|
| $1$ | $1465m_2'$ |
| $2,\ldots,p'-1$ | $\frac{1393(m_1-5)m_2'}{2(p-3)}+4\frac{m_1-5}{p-3}+74m_2'$ |
| $p'$ | $1459m_2-2m_2'$ |
| $p'+1,\ldots,p-1$ | $\frac{1393(m_1-5)m_2}{2(p-3)}+4\frac{m_1-5}{p-3}+74m_2$ |
| $p$ | $1459m_2$ |

Table 1: Number of arithmetic operations needed in the factorization phase in each processor.

By introducing

$$\vartheta=\frac{m_2'}{m_2} \tag{15}$$

we get that the total work in the factorization phase for DD, denoted by $A_{DD,fac}$ becomes

$$\begin{aligned} A_{DD,fac} &= \frac{1393(m_1-5)m_2}{2(p-3)}\left(p-(1-\vartheta)p'-(1+2\vartheta)\right)+4(m_1-5)+ \\ &+ (1533+1537\vartheta)m_2. \end{aligned}$$

For $\vartheta$ close to 1 we get

$$A_{DD,fac}\approx\frac{1393}{2}m_1m_2+4m_1-\frac{821}{2}m_2.$$

Hence the work per gridpoint in the factorization phase becomes approximately 1393/2 for the DD method. For the direct method bandGE the corresponding quantity is $108m_1^2$.

In Figure 6 we display the work in the factorization phase for the different methods.

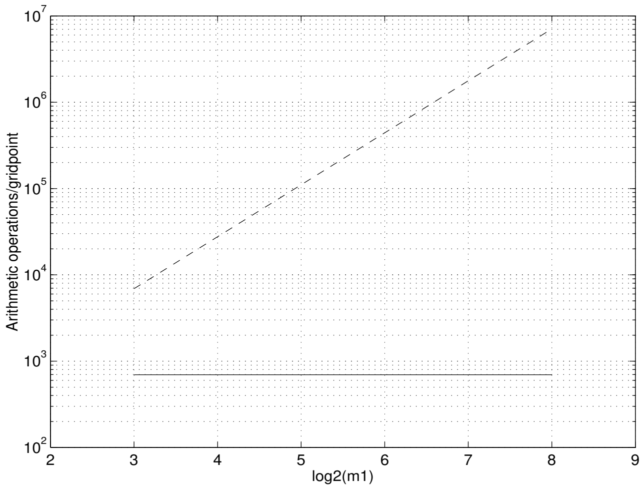


Figure 6: Arithmetic operations in the factorization phase required per gridpoint. Solid line represents DD and dashed line bandGE.

#### 4.2.2 Time-stepping

Here we consider the number of arithmetic operations required in each time-step for bandGE and DD. We do not take into account the number of arithmetic operations it takes to set up the right-hand side as this is an equal amount of work for the two methods. We start by presenting the number of operations required in each processor with the decomposition defined in Section 3.1 and (14). Here $it$ is the number of iterations required for convergence.

| $q$ | #a.o. |
|---|---|
| $1$ | $1623m_2' + it\left(437m_2' + 9m_2'\frac{\ell^2}{\ell+1}\right)$ |
| $2,\ldots,p'-1$ | $\frac{(m_1-5)m_2'}{p-3}\left(987 + 266\cdot it + \log_2\left(\frac{m_1-5}{2(p-3)}\right)(60+15it)\right) + {} + m_2'\left(-131 + it\left(2 + 9\frac{\ell^2}{\ell+1}\right)\right)$ |
| $p'$ | $1626m_2 + 5m_2' + it\left(438m_2 + 5m_2' + 9m_2\frac{\ell^2}{\ell+1}\right)$ |
| $p'+1,\ldots,p-1$ | $\frac{(m_1-5)m_2}{p-3}\left(987 + 266\cdot it + \log_2\left(\frac{m_1-5}{2(p-3)}\right)(60+15it)\right) + {} + m_2\left(-131 + it\left(2 + 9\frac{\ell^2}{\ell+1}\right)\right)$ |
| $p$ | $1582m_2 + 393\cdot it\cdot m_2$ |

Table 2: Number of arithmetic operations needed in each processor.

With the definition of $\vartheta$ in (15) we get that the total number of arithmetic operations in each time-step for DD, denoted by $A_{DD,sol}$ becomes

$$
\begin{aligned}
A_{DD,sol} &= \tfrac{(m_1-5)m_2}{p-3}\left(987+266\cdot it+\log_2\left(\tfrac{m_1-5}{2(p-3)}\right)(60+15\cdot it)\right)\times \\
&\times \quad (p-(1-\vartheta)p'-(1+2\vartheta))+ \\
&+ \quad m_2\left(\left(-131+it\left(2+9\tfrac{\ell^2}{\ell+1}\right)\right)(p-(1-\vartheta)p'-(1+2\vartheta))+\right. \\
&+ \quad \left. 3208+831\cdot it+1628\vartheta+442\cdot it\cdot\vartheta+9\cdot it\tfrac{\ell^2}{\ell+1}(1+\vartheta)\right).
\end{aligned}
$$

If we assume that $\vartheta$ is close to 1 we get

$$
\begin{aligned}
A_{DD,sol} &\approx (m_1-5)m_2\left(987+266\cdot it+\log_2\left(\tfrac{m_1-5}{2(p-3)}\right)(60+15\cdot it)\right)+ \\
&+ \quad m_2\left(\left(-131+it\left(2+9\tfrac{\ell^2}{\ell+1}\right)\right)(p-3)+\right. \\
&+ \quad \left. 4836+1273\cdot it+18\cdot it\tfrac{\ell^2}{\ell+1}\right).
\end{aligned}
$$

The total arithmetic complexity per gridpoint hence becomes approximately

$$
987+266\cdot it+\log_2\left(\frac{m_1-5}{2(p-3)}\right)(60+15\cdot it)\,.
$$

Now consider $p\rightarrow\frac{m_1}{2}$ and $m_1\rightarrow\infty$. We then get that the total arithmetic complexity is $(987+266\cdot it)m_1m_2$. So if the number of iterations stays constant for increasing $m_1$, we have a solution method with arithmetic complexity $\mathcal{O}(const.m_1m_2)$. In Section 5 we will investigate how the number of iterations varies with problem-size and number of processors.

For bandGE the arithmetic complexity per gridpoint is given by

$$
54m_1.
$$

In Figure 7 we display the number of arithmetic operations per gridpoint for DD and bandGE. We display $A_{DD,sol}$ per gridpoint for $it=1$, i.e. the number of arithmetic operations per iteration and gridpoint. If more than one iteration is needed for convergence the graph should be scaled with $it$.

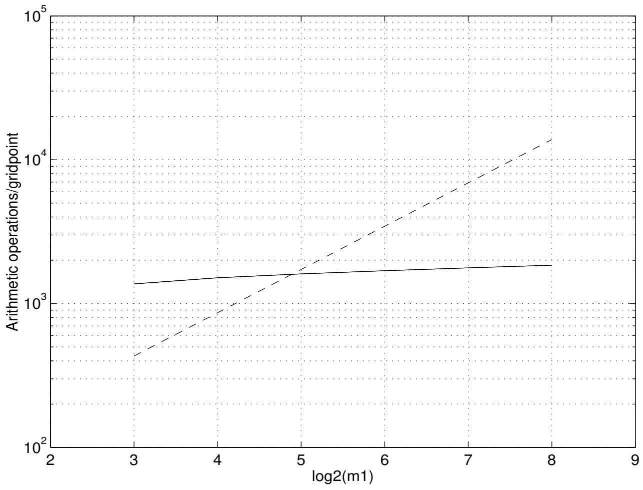


Figure 7: Arithmetic operations required per gridpoint. Solid line represents DD for $p = 5$ and dashed line bandGE.

### 4.3 Conclusions

From Section 4.1 and 4.2 we draw the following conclusions.

- The DD method is superior to bandGE regarding memory requirements, due to the fill ins in the empty bands for the direct methods. When it comes to problems in three space-dimensions this will be an enormous problem for the latter method. This will be further discussed in Section 6.
- In the factorization phase the DD method beats the direct method for all problem sizes. The superiority increases rapidly with increasing problem size.
- When it comes to time-stepping, the DD method is superior to the direct method for large problems, if this method does not require too many iterations in each time-step. In Section 5 we will study the convergence properties of the DD method.

## 5 Numerical results

In this section we present numerical results from an Intel Paragon, located at Para//ab in Bergen. In all cases we consider the iteration on (8) to have converged in iteration $k$ when

$$\frac{||C(\bar{u}_0^{n+1})^k - \bar{f}_0||_2}{||\bar{f}_0||_2} < 10^{-6}. \tag{16}$$

As initial guess we have used linear extrapolation in time to get

$$(\bar{u}_0^{n+1})^0 = 2\bar{u}_0^n - \bar{u}_0^{n-1}.$$

The restarting length is 6.

### 5.1 An example

Here we show the solution for $Re = 200$ and $\epsilon = 0.01$. The geometric parameters are $x_1' = 0.2512$, $\hat{x}_1 = 0.9975$, and $x_2' = 0.6667$. The spatial discretization is defined by $m_1' = 34$, $m_1 = 133$, $m_2' = 43$, $m_2 = 65$. The stretching parameter, defined in (6), is $a = 10$. This discretization yields $h_{min} = 0.00387$ and, utilizing (3), we set $\Delta t = 0.00015$. Hence $\max \kappa_d' \approx 3.9$. We have used 19 processors, $p' = 6$ and the decomposition defined in Section 3.1 and (14).

In Figure 8 we show the solution at $t = 3.0$. At that time steady state is achieved. In Figure 9 the arrows are normalized to equal length in order to show the stream lines. The average number of iterations at each time-step was 2.27. The total cpu-time was 43.8 minutes and the arithmetic speed for this example was 112 Mflops.

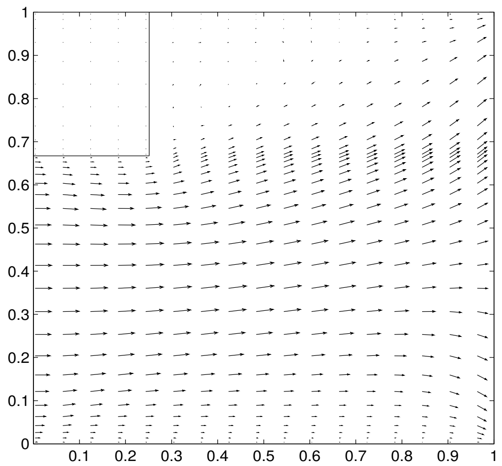


Figure 8: The flow pattern for $\epsilon = 0.01$, and $Re = 200$ at $t = 3.0$.

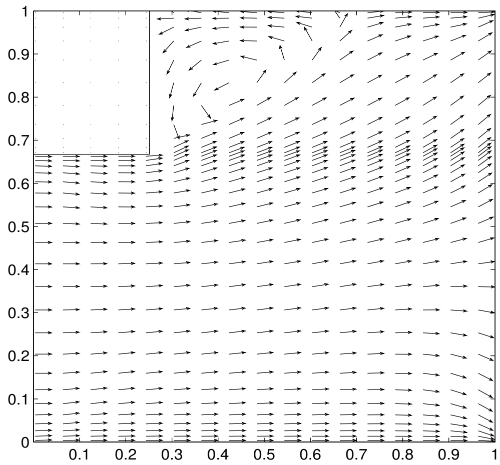


Figure 9: Stream lines for $\epsilon = 0.01$, and $Re = 200$ at $t = 3.0$.

## 5.2 Numerical experiments

We start by studying how the convergence depends on the physical parameters $\epsilon$ and $Re$. We consider the discretization defined in Section 5.1. In Table 3 we display the average number of iterations $\bar{it}$, required over 20000 timesteps, and the total cpu-time, $T_p$.

| $\epsilon$ | $Re$ | $\bar{it}$ | $T_p$[minutes] |
|---|---|---|---|
| 0.01 | 200 | 2.27 | 43.8 |
| 0.01 | 250 | 2.33 | 44.3 |
| 0.02 | 200 | 1.65 | 38.4 |
| 0.02 | 250 | 1.68 | 38.6 |

Table 3: Average number of iterations and total cpu-time for different physical parameters.

In Table 3 we see that at least in this regime, the iteration converges faster for lower Reynolds numbers and higher Mach numbers. In the following we will use $Re = 200$, and $\epsilon = 0.01$.

Next we examine how the iteration on (8) scales with the number of processors. We consider the discretization defined in Section 5.1 but consider 7, 11 and 19 processors, i.e., we solve exactly the same problem in all cases. In Table 4 we show the average number of iterations and the total cpu-time.

| $p$ | $it$ | $T_p$[minutes] |
|---|---|---|
| 7 | 2.27 | 200.9 |
| 11 | 2.25 | 92.9 |
| 19 | 2.27 | 43.8 |

Table 4: Average number of iterations and total cpu-time required for different number of processors.

In Table 4 we see that the number of iterations does *not* grow with increasing number of processors, i.e., increasing number of unknowns in the Schur-complement $C$. This is a desirable property. In Figure 10 we display the rate of efficiency $E_p$ and $E_{p,eff}$, defined by

$$E_p = \frac{p_{\min}}{p} \cdot \frac{T_{p_{\min}}}{T_p},$$

and

$$E_{p,eff} = \frac{p_{\min} - 3}{p - 3} \cdot \frac{T_{p_{\min}}}{T_p}.$$

Here $p_{\min}$ is the minimum number of processors used in the experiment. Hence $p_{\min} = 7$. The definition of $E_{p,eff}$ is motivated by the fact that the processors 1, $p'$, and $p$ do not really take part in the parallelization. They are solving the same problem, independent of the number of processors.

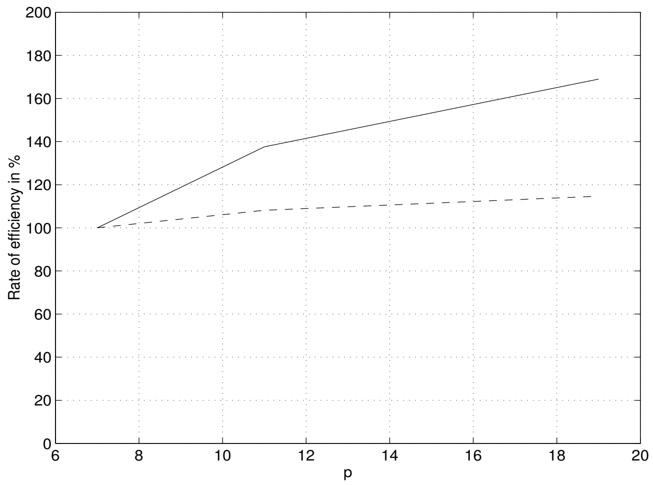


Figure 10: Rate of efficiency $E_p$ solid line and effective rate of efficiency $E_{p,eff}$ dashed line.

From Figure 10 we draw the conclusion that the method clearly has favorable parallelization properties. Both $E_p$ and $E_{p,eff}$ are greater than 1. The reason is the fact that this domain decomposition method is beneficial even sequentially, i.e., the number of iterations is constant, and the cost for the subdomain solves decreases as the number of subdomains grow.

Now we will examine another form of scalability. We will let the problem-size increase with the number of processors. The physical parameters are the same as in the previous problem. We will examine the three different problem-settings defined in Table 5.

| Problem | $p$ | $p'$ | $m_1$ | $m_1'$ | $h_1$ | $m_2'$ | $m_2$ | $h_2$ |
|---|---|---|---|---|---|---|---|---|
| A | 7 | 3 | 10 | 37 | 0.03 | 21 | 32 | 0.0303 |
| B | 11 | 4 | 18 | 69 | 0.015 | 43 | 65 | 0.0152 |
| C | 19 | 6 | 34 | 133 | 0.0075 | 87 | 131 | 0.00758 |

Table 5: Different problem-settings.

We do not use a stretched grid. The resason is that we want to keep the quotient $\kappa_d'$, $d = 1, 2$ constant within each grid. When we increase the number of unknowns, we decrease the time-step used. In Table 6 we display the results for $\kappa_1' = 4.0$ and $\kappa_2' = 3.96$. In all cases we solve the problem to $t = 3.0$.

| Problem | $\Delta t$ | $it$ | $T_p$[minutes] |
|---|---|---|---|
| A | 0.0012 | 23.36 | 14.8 |
| B | 0.0006 | 4.04 | 14.6 |
| C | 0.0003 | 2.96 | 49.7 |

Table 6: Average number of iterations and total cpu-time for the different problem settings.

In Figure 11 we show $E_p$ and $E_{p,eff}$ again for this scaled problem. Now we have to scale these properties such that we in some sense compare equally sized problems. We define

$$E_{p,s} = E_p \cdot \frac{m_1 m_2 n}{(m_1 m_2 n)_{\min}}$$

and

$$E_{p,eff,s} = E_{p,eff} \cdot \frac{m_1 m_2 n}{(m_1 m_2 n)_{\min}}.$$

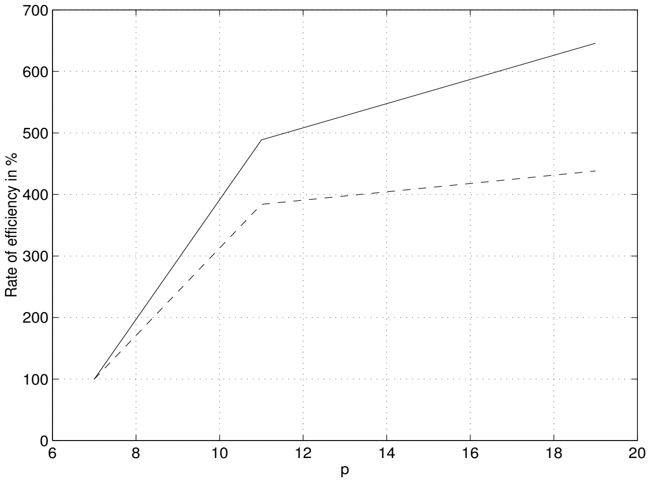


Figure 11: Rate of efficiency $E_{p,s}$ solid line and effective rate of efficiency $E_{p,eff,s}$ dashed line.

In Table 6 we see that the number of iterations decreases rapidly with increasing problem-size. This is an extremely good property of an iterative method. Moreover, in Figure 11 we see that the rate of efficiency is extremely high.

Inspired by the theorem in Section 3.3, we now study how the convergence depends on the grid-quotient $\varphi$ defined in (10). We consider the parameters $p = 19$, $p' = 6$, $m_1' = 34$, $m_1 = 133$, and let $\varphi$ vary between 0.85 and 1.05. The time-step used is $\Delta t = 0.0003$. In Figure 12 we display the average number of iterations $\bar{it}$ over 10000 time-steps as a function of $\varphi$. Figure 13 shows the total cpu-time obtained as a function of $\varphi$.

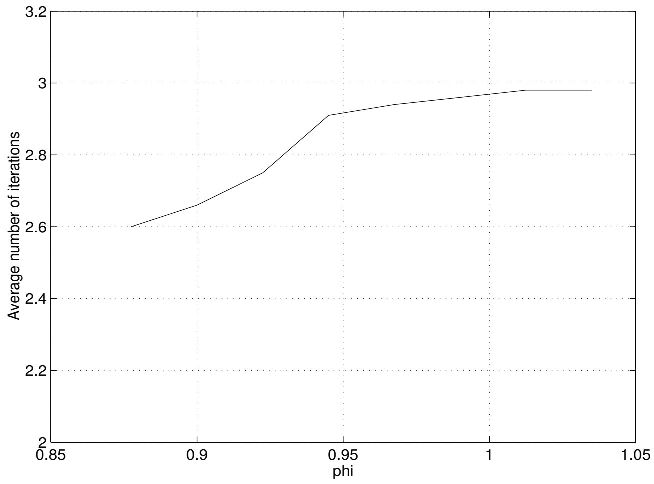


Figure 12: Average number of iterations $\bar{it}$ as a function of the grid-quotient $\varphi$.

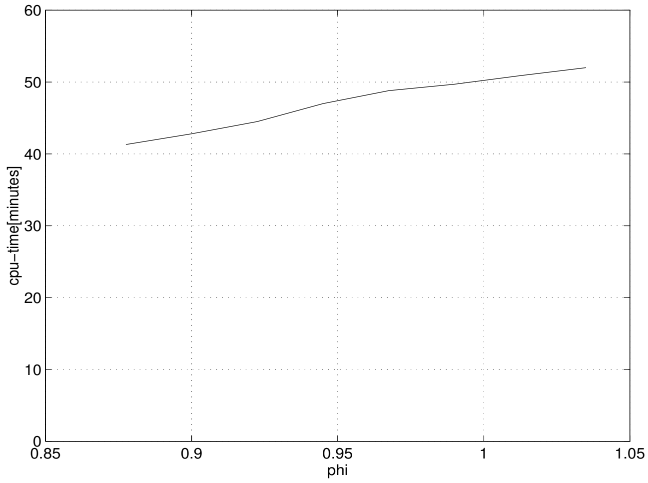


Figure 13: Total cpu-time as a function of the grid-quotient $\varphi$.

From Figure 12 it is clear that the result in Theorem 3.1 extends also to this system of PDEs, in the sence that the number of iterations decreases with decreasing $\varphi$. From Table 6 we conslude that this does not depend on the fact that we are solving a smaller problem for lower $\varphi$. In Figure 13 we see that the gain in number of iterations yields a significant gain in cpu-time. Of course this is reinforced by the fact that we perform fewer arithmetic operations for a lower $\varphi$. Anyhow, it is clear that for large problems, we could gain cpu-time by simply letting $\varphi$ decrease slightly from 1.

Another interesting property of the algorithm is the dependence on $\kappa'_d$, defined in (11). We study the three different parameter-settings defined in Table 5, but let $\Delta t$ vary so that $\kappa'_1 = 1.0$, 2.0, 3.0 and 4.0. In Figure 14 we show the total cpu-time for 100 time-steps. The total cpu-time required to solve the problem to $t = 3.0$ is shown in Figure 15.

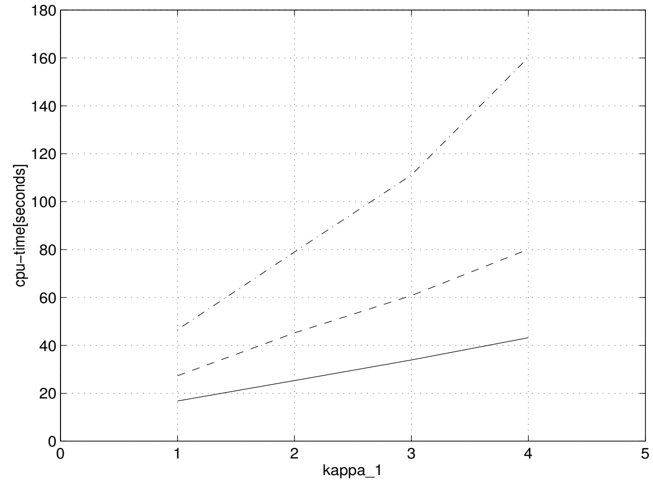


Figure 14: Total cpu-time for 100 time-steps as a function of $\kappa'_1$. Solid line represents Problem A, dashed line Problem B, and dashed-dotted line Problem C.

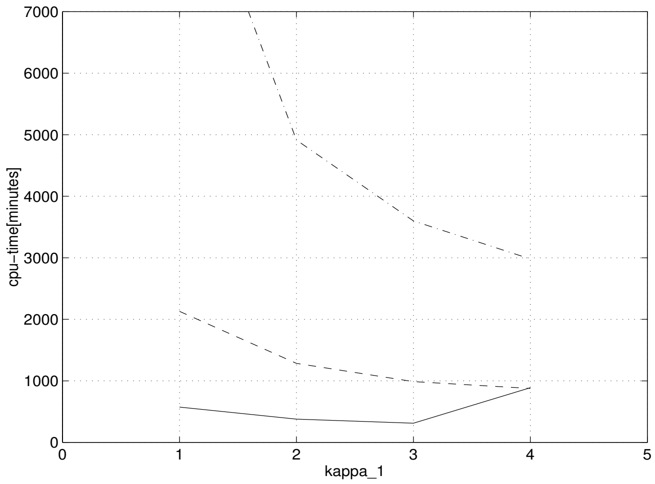


Figure 15: Total cpu-time to solve the problem to $t = 3.0$ as a function of $\kappa'_1$. Solid line represents Problem A, dashed line Problem B, and dashed-dotted line Problem C.

In Figure 14 we see that the time per time-step increases with increasing $\kappa_1'$. However, in Figure 15 we see that this clearly is compensated for by the fact that we take fewer time-steps, with one exception for the smallest problem-size. Hence we should use the largest time-step possible that fulfills the stability-criterion (3).

# 6 Possible generalizations

In this paper, we consider a PDE where the part that is discretized implicitly in time has constant coefficients. For a more general PDE the implicit operator can have variable coefficients. For the method presented here, the fast direct subdomain solvers require constant coefficients in one space-direction. However, in [8] an iterative solver for these subdomain systems is presented for a scalar PDE. The generalization of this more flexible method to systems of PDEs is straight-forward.

We consider an equidistant grid in the $x_1$-direction. Again, the reason for this is that we want to use the fast direct solvers in the subdomains. If we have a problem where we want to have a non-equidistant grid in this direction too, the method must be modified. An iterative method for the subdomain solves as mentioned above may be utilized, or equidistant grids may be used within each subdomain. However, note that then the transformation from the computational grid to the physical grid has discontinous derivatives at $\Gamma_q$, and it is not clear that the scheme is stable.

For all numerical experiments, we have used $m_1^q = 2^s$, $q = 2, \ldots, p' - 1, p' + 1, \ldots, p$. The reason for this is that then the Fourier transforms in the fast modified sine transforms are of size $2^{s-1}$. However, by employing the fractional Fourier transform [1], we can have any number of gridpoints in these subdomains.

If we go to three space-dimensions, the direct method presented in [12] becomes impossible to implement. The bandwidth of the system (4) is of order $\mathcal{O}(m_1 m_2)$, and the memory requirement becomes unrealistic. However, by generalizing the method presented here, we decompose the three-dimensional domain in rectangular blocks, and solve first for the unknowns on each edge that couples 4 subdomains. Then we solve for the unknowns on the faces that couple 2 different subdomains, and finally we solve for the unknowns within each block. The memory requirement for a method of this type will grow only linearly with the number of unknowns.

# 7 Conclusions

We have used a domain decomposition method defined in [8] for solving the Navier-Stokes equations for almost incompressible flow. The PDE is discretized using the leap-frog/backward Euler scheme in time. In space we use centered differences. This discretization has been analyzed in [6] with respect to well-posedness, stability and accuracy.

The original domain is divided into $p$ subdomains. The decomposition is chosen such that all subdomain systems have a fast direct solver. The original system is reduced to a smaller system on the interior boundaries; the Schur-complement system. This system is solved using an iterative method, where the fast direct solvers for the subdomain systems are utilized. For a scalar PDE, the iteration on the Schur-complement system is analyzed theoretically in [8].

We have compared the method presented in this paper with a standard direct method. Numerical experiments have been performed to study how the method depends on different parameters. The conclusions are:

- The domain decomposition method presented in this paper is superior to the direct method in memory requirements and in arithmetic operations. This allows us to solve much larger problems and problems in three space-dimensions.

- For a given discretization of the PDE, as many processors as possible should be used. Moreover, for the problem-sizes that we have studied, the number of iterations on the Schur-complement system is constant independent of the number of processors. The rate of efficiency is larger than 100%, since the domain decomposition method is beneficial even sequentially.

- In [8] a theoretical result for the convergence for the Schur-complement system is presented for a scalar PDE. By letting the grid-quotient $\varphi$ defined in (10) decrease slightly from 1, we decrease both the number of iterations and the cpu-time. In this paper we have studied the extension of this theorem to the systems of equations arising from the Navier-Stokes equations. The numerical experiments confirm that the result holds, and we should use $\varphi \approx 0.9$.

- The numerical experiments show that for a given quotient between the time-step and the space-step, the number of iterations decreases with increasing problem-size. This is an extremely good property of an iterative method. Moreover, we have shown that for larger problems it is beneficial to use as large time-step as possible with respect to (3). For the Euler equations, $Re = \infty$, and we expect that even larger time-steps can be used.

# 8 Acknowledgments

I would like to thank my adviser Bertil Gustafsson for helpful support. Also, I owe many thanks to Para//ab for letting me use their Intel Paragon. Finally, I thank Dr Sverker Holmgren for his careful reading of this paper.

# References

[1] D. H. Bailey and P. N. Swarztrauber, *The fractional Fourier transform and applications*, SIAM Rev., 33 (1991), pp. 389-404.

[2] H. C. Elman, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph. D. thesis and Report RR-229, Departments of Computer Science, Yale University, New Haven, CT, 1982.

[3] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solution of linear systems*, Acta Numerica (1992), pp. 57-100.

[4] J. Guerra and B. Gustafsson, *A numerical method for incompressible and compressible flow problems with smooth solutions*, J. Comput. Phys., 63 (1986), pp. 377-397.

[5] J. Guerra and B. Gustafsson, *A semi-implicit method for hyperbolic problems with different time-scales*, SIAM J. Numer. Anal., 23 (1986), pp. 734-749.

[6] B. Gustafsson and H. Stoor, *Navier-Stokes equations for almost incompressible flow*, SIAM J. Numer. Anal., 28(6) (1991).

[7] L. Hemmingsson, *A fast modified sine transform for solving block-tridiagonal systems with Toeplitz blocks.*, Numer. Algor., 7 (1994), pp. 375-389.

[8] L. Hemmingsson, *A domain decomposition method for first-order PDEs*, To appear in SIAM J. Matrix Anal. Appl.

[9] S. Holmgren and K. Otto, *Iterative solution methods and preconditioners for block-tridiagonal systems of equations*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 863-886.

[10] S. Holmgren and K. Otto, *Semicirculant solvers and boundary corrections for first-order PDE*, To appear in SIAM J. Sci. Comput.

[11] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856-869.

[12] H. Stoor, *A semi-implicit difference method for the Navier-Stokes equations for almost incompressible flow*, Report No. 126, Department of Scientific Computing, Uppsala University, Uppsala, Sweden, 1990.