DELFT UNIVERSITY OF TECHNOLOGY

REPORT 96-109

Parallelism in GMRES applied to the computation of incompressible flows

R.R.P. van Nooyen, C. Vuik,
P. Wesseling

ISSN 0922-5641

Reports of the Faculty of Technical Mathematics and Informatics no. 96-109

Delft 1996

Abstract

A parallel implementation of the preconditioned GMRES method is described. The method is applied to solve the discretized incompressible Navier-Stokes equations. A parallel implementation of the inner product is given, which appears to be scalable on a massively parallel computer. The most difficult part to parallelize is the ILU-preconditioner. We parallelize the preconditioner using ideas proposed by Bastian and Horton [2]. Their ideas are little used to parallelize preconditioned Krylov subspace methods. Contrary to other parallel methods the required number of iterations is independent of the number of processors used. A model is presented to predict the efficiency of the method. Experiments are done on the Cray T3D computing the solution of a two-dimensional flow. Predictions of computing time show good correspondence with measurements. It appears that the proposed method is scalable.

1 Introduction

To compute incompressible turbulent flows in complicated two- and three-dimensional domains we use a numerical method with the following properties. Boundary fitted coordinates and domain-decomposition are used to handle geometrically complicated domains. The finite volume method and a staggered grid are used for discretization in space. A combination of the Euler backward scheme and the pressure correction method is used to advance the solution in time. The computer program is referred to as the ISNaS (Information System for Navier-Stokes equations) incompressible flow solver.

Benchmark solutions for the discretization chosen can be found in [13]. Research into parallelism through multi-block techniques is described in [5]. Some results concerning the use

of multigrid solvers in this code can be found in [25, 11, 12]. Periodic and anti-periodic boundary conditions are discussed in [16]. Some aspects of the discretization are presented in [23, 17, 22, 24]. The treatment of turbulence is analysed in [27, 28, 26] and of compressibility in [3]. In the present paper we consider a two-dimensional single block laminar flow as a test case for implementation on a Cray T3D.

The ISNaS solver uses pressure correction, which means that an intermediate velocity field is determined. Thereafter a correction is calculated to obtain a velocity field with zero divergence. The intermediate velocity field is obtained by applying a linear solver to the Newton linearization of the discretized momentum equations. The pressure correction is obtained from the solution of the discretized pressure equation.

The equations are discretized on a staggered grid and with curvilinear coordinates. The curvilinear coordinates result in extra non-zero entries in the matrix. In 2D we initially find a 17 point stencil for the momentum vector component equations and a 9 point stencil for the pressure equation. Using divergence freedom, the number of points needed in the momentum stencil reduces to 13.

An initial analysis showed that matrix construction is embarrassingly parallel [18], so we concentrate on the linear solvers for the momentum and pressure equations. From many experiments it appears that GMRES combined with MILUD-preconditioning for the momentum equations and MILU-preconditioning, with the same sparsity pattern as the original matrix, for the pressure equations are robust and fast solvers [21]. Therefore we concentrate on parallelizable variants of these methods. A choice still remains: we can either parallelize the algorithms themselves or use some sort of multi-block algorithm. Results on multi-block techniques can be found in [5]. In this paper we present a parallelization of the algorithms themselves. In this case the convergence behaviour of the algorithm does not depend on the number of processors. Only the wall clock time will vary.

For later reference we include a short description of the GMRES(m) method as given in [15], with a left-preconditioner M_1 and a right-preconditioner M_2.

1. *Start*: Choose an initial estimate x_0, compute the initial preconditioned residual $r_0 = M_1(f - Ax_0)$ and determine $v_1 = r_0/\|r_0\|$.

2. *Iterate*: For $j = 1, 2, \ldots, m$ do:

$$\begin{aligned} h_{i,j} &= (M_1 A M_2 v_j, v_i), i = 1, 2, \ldots, j, \\ \hat{v}_{j+1} &= M_1 A M_2 v_j - \sum_{i=1}^{j} h_{i,j} v_i, \\ h_{j+1,j} &= \|\hat{v}_{j+1}\|, \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j} \end{aligned}$$

3. *Form the approximate solution*: $x_m = x_0 + M_2 \sum_{k=1}^{m} y_k v_k$ where the y_k minimize $\|M_1(r_0 - AM_2 \sum_{k=1}^{m} y_k v_k)\|$.

4. *Restart*: Compute $r_m = M_1(f - Ax_m)$, if the termination criterion is satisfied then stop, else compute $x_0 = x_m$, $v_1 = r_m/||r_m||$ and go to 2.

Note that in step 2 the new search direction is made perpendicular to all previous search directions with the classical Gram Schmidt orthogonalization method. For a full description of the GMRES method and its properties we refer to [15].

In Section 2 we consider the building blocks of the preconditioned GMRES method. A parallelization of the inner product is presented. The ILU-preconditioner part appears to be the most difficult to parallelize, details are given in Section 3. Section 4 contains various models to predict Megaflop rates, communication costs etc. Timing experiments on a test problem are reported in Section 5. Finally an analysis of the timing results is given in Section 6.

2 Parallelization of preconditioned GMRES

We consider a flow problem on a two-dimensional rectangular domain. We assume that the target machine behaves as a distributed memory machine (e.g. Cray T3D). The preconditioned GMRES method consists of the following building blocks: vector update, inner product, matrix vector product, preconditioner construction, and preconditioner times vector. In Subsection 2.1 we give the data distribution of our problem. We skip the description of a parallel vector update, because it is easily parallelizable without communication. The parallelization of the inner products is described in Subsection 2.2, together with a discussion of various Gram-Schmidt orthogonalization methods. Subsection 2.3 contains the parallel matrix vector product. Finally, in Subsection 2.4 we discuss the difficulties to obtain a parallel implementation of ILU-type preconditioners. The details of the parallel preconditioner are given in Section 3.

2.1 Data distribution

On distributed memory machines and workstation clusters it is important to keep information in local storage as much as possible. For this reason we assign storage space and update duties as follows. The domain is subdivided into a regular grid of rectangular sub-domains. The subdivision follows the cell edges of the space discretization grid. Each processor is responsible for all updates of variables associated to the grid cells in its sub-domain.

We use the following convention to assign the fluxes, which are given on cell edges for a

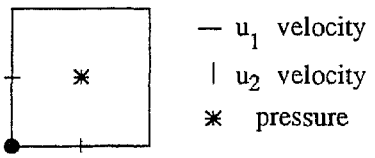


Figure 1: Assignment of unknowns to a cell using a staggered grid.

staggered discretization, to cells: the flux on the lower and the left-hand cell edge belongs to

the given cell (Figure 1). Furthermore, a sub-domain contains all fluxes on the lower and the left-hand boundary and all fluxes on the intersections of an outer boundary with a sub-domain boundary. Fluxes on an interior upper or right-hand sub-domain boundary are assigned to the

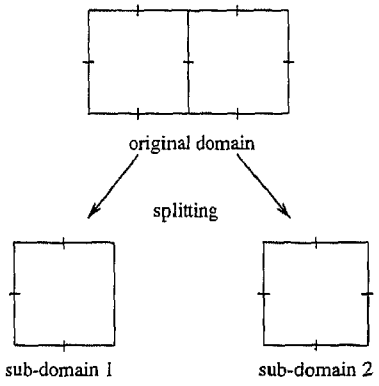


Figure 2: Decomposition of the domain in sub-domains

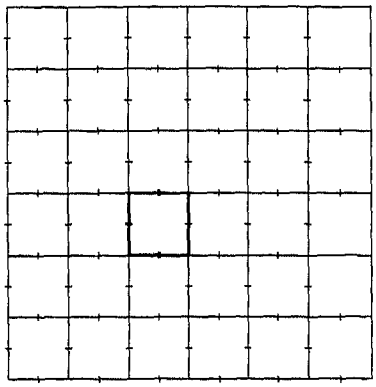


Figure 3: Sub-domain 1 with auxiliary cells

processor that is responsible for the sub-domain adjacent to that boundary (Figure 2). Two extra rows of cells are added on the lower and left boundary of a sub-domain and three extra rows of cells are added to the upper and right boundary of a sub-domain to provide storage space for variables used in matrix-vector products and preconditioner construction (Figure 3). Note that for small sub-domains the addition of the extra rows can result in a considerable increase in problem size. However, for large sub-domains the increase is negligible.

2.2 Inner products and Gram-Schmidt orthogonalization

In this section we show how to calculate an inner product on a Cray T3D efficiently. Furthermore some remarks are given on Gram-Schmidt orthogonalization methods.

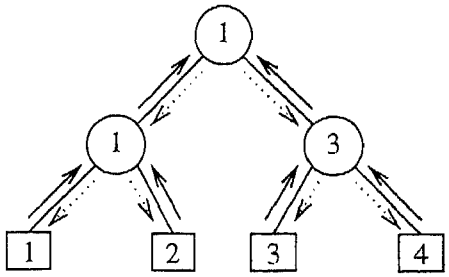
The work to compute an inner product is distributed as follows. First the inner product of the vector elements that reside on a processor is calculated for each processor. Thereafter these partial inner products should be summed up to form the full inner product. To obtain the full inner product global communication is required. We implement two different communication strategies.

Inner product I

We assume each processor to be a leaf of a binary tree (Figure 4). Each non-leaf node represents a summation of two partial inner products carried out by for instance the processor found by recursively ascending the left sub-tree. The node at the top of the binary tree obtains the full inner product at the last step of this process. Then the same tree is used in reverse order to distribute the full inner product from its top to all its leaves. When 2^n processors are used, there are $2n$ subsequent communication steps necessary.

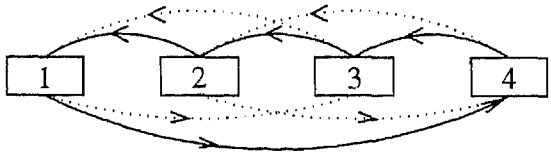
Inner product II

To explain the second communication strategy we consider a small 1D torus (Figure 5). In



→ communication partial inner products
⋯> communication full inner product

Figure 4: Communication pattern for inner product I



→ first communication step
⋯> second communication step

Figure 5: Communication pattern for inner product II

the first step each processor sends its partial inner product to its left neighbour. After summation every processor sends this result to its left neighbour with a distance of 2 links away. At the k^{th} step the distance between the processors is $2^{(k-1)}$. For 2^n processors it appears that after n subsequent communication steps every processor contains the full inner product. This implies that if sufficient bandwidth is available, the communication strategy of variant II is twice as fast as that of variant I.

Due to overhead the use of PVM communication subroutines for the inner product leads to unacceptable performance loss. Therefore we use Cray T3D specific shared arrays to implement the communication for the inner product. The shared memory array contains two memory positions for every processor. If n communication steps are done, the following implementation is used (note that only one synchronization point is necessary):

```
for all processors
   for i = 1, n

      if i = odd then pos = 1 else pos = 2 end if

      store the result in the shared memory array at [j,pos], where j is
      the number of the target processor

      synchronization

      processor k reads information from the shared memory array at [k,pos]

      calculations

   end for
end for all
```

It appears that the speed of the inner product II is comparable to the fastest Cray native inner product. We use inner product II because its behaviour is easier to analyse. Note that inner product II can also be used on machines without a native inner product.

Communication time can be seen as the sum of start-up time (latency) and send time. On many parallel computers the send time is an order of magnitude less than the latency. For this reason it is attractive to combine communications as much as possible. Using the Classical Gram-Schmidt (CGS) method in the GMRES algorithm (see Section 1), all inner products can be computed independently. So the communication steps can be clustered, which saves much start-up time. A drawback of CGS is that the resulting vectors may be not orthogonal due to rounding errors [4]. Therefore, the Modified Gram-Schmidt (MGS) method is preferred, which is stable with respect to rounding errors [4]. However, in MGS the inner products should be calculated sequentially. So clustering of the inner product communications is impossible. Since, for our T3D specific code, the difference between latency and send time is relatively small we use the Modified Gram-Schmidt method for stability reasons.

2.3 Matrix vector product

In principle only the vector elements normally updated by a given processor are kept up to date on that processor. If vector elements from adjacent processors are needed in the matrix vector product then calculations involving these elements are postponed until these elements have been obtained from another processor. This nearest neighbour communication on relatively long vectors is implemented by calls to PVM (or MPI) subroutines. The elements obtained from other processors are stored in the corresponding auxiliary cells (Figure 3). Calculations and communication steps are overlapped as much as possible.

2.4 Preconditioners

We use ILU-type preconditioners for the solution of the momentum and pressure equations. Suppose $Ax = b$ should be solved. A sparse lower triangular matrix L and a sparse upper triangular matrix U are constructed such that $L \cdot U \simeq A$. In the preconditioned GMRES algorithm it is necessary to calculate $x = U^{-1}L^{-1}b$. This is done by solving the triangular systems: $Ly = b$ and $Ux = y$. The construction of L and U and the solution of the triangular systems are not easy to parallelize. To illustrate this we consider the computation of y from $Ly = b$ (the other parts are comparable). A straight forward algorithm to calculate y is:

for $i = 1, ..., n$

$$y_i = \left(b_i - \sum_{j=1}^{i-1} L_{ij} y_j \right) / L_{ii}$$

end for

Note that y_i can only be computed if $y_1, ..., y_{i-1}$ are already known, so this leads to sequential code. For discretized partial differential equations it is possible to obtain parallel algorithms to construct L, U, and solve triangular systems. Our approach, as given in the next section, is based on the ideas presented in [2].

3 Parallel ILU-preconditioning

Preconditioners originating from an incomplete LU decomposition lead to very good performance of the preconditioned GMRES method [20],[21], and [19]. In this section we present the properties of the discretized pressure and momentum equations in 2D. These properties determine the form and contents of the triangular matrices L and U. Thereafter the details of the parallelization of the operations with the preconditioners are given.

3.1 Properties of the linear systems

Pressure equation

After discretizing the pressure equation, one obtains a linear system $Px = b$, with a non symmetric coefficient matrix P. The discretization stencil of the pressure equation consists of 9 points. Due to the structured grid approach the matrix P has only 9 non-zero diagonals [20]. The preconditioner used in this paper is the RILU(α)-preconditioner with $\alpha = 0.975$ [21]. Note that for the RILU-preconditioner the non-zero structure of $L + U$ is identical with the non-zero structure of P. Figure 6 displays the stencils of P, L, and U.

Momentum equations

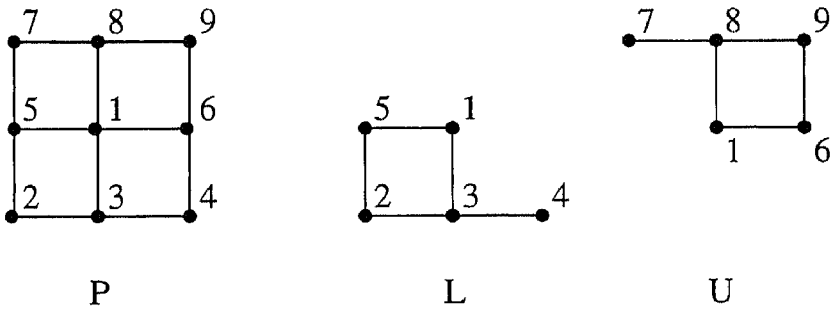


Figure 6: The stencils of P, L, and U.

The discretized momentum equations are denoted by

$$Mu = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{1}$$

The discretization stencil of M consists of 13 points. The diagonal blocks M_{11} and M_{22} contain 9 non-zero diagonals and their non-zero structure is the same as that of P. Both M_{21} and M_{12} contain 4 non-zero diagonals. As preconditioner we use RILUD_2(α) with $\alpha = 0.95$ [21]. Here the off-diagonal parts of M are the same as that of L and U, only the main diagonal elements of L, U, and M are different.

With respect to parallelization we consider the solution of $Lu = b$. First

$$L_{11}u_1 = b_1 \tag{2}$$

is solved. Thereafter u_2 is solved from

$$L_{22}u_2 = b_2 - L_{21}u_1. \tag{3}$$

To compute the right-hand side of (3) a matrix vector multiplication ($L_{21}u_1$) is needed. This can be parallelized in the same way as the original matrix vector product. Finally, the identical structure of M_{11}, M_{22}, and P leads to the same parallel algorithms to solve the lower triangular systems (2), (3) and that encountered at the solution of the pressure equation. For this reason, we restrict ourselves in the next paragraph to the parallelization of the pressure preconditioner.

3.2 Parallelization of the pressure preconditioner

Parallelization of the construction of L, U, and the solution of the triangular systems is comparable. Therefore we restrict ourselves to the parallel implementation of the solution of the lower triangular system $Lx = b$ for the pressure equation. The algorithm is first explained for a matrix originating from a 5 point stencil. Thereafter it is adapted for matrices based on a 9 point stencil.

The ideas for the parallelization come from [2]. The same approach is used in the vectorization of ILU-type preconditioners ([1] and [21]). First of all we decompose our rectangular computational domain in p strips parallel to the x_2-axis. We assume that the number of strips is equal to the number of processors. The number of grid points in x_i-direction is denoted by n_i. For ease of notation we assume that n_1 can be divided by p and set $n_x = n_1/p$ and $n_y = n_2$. The index i refers to the index in x_1-direction and j to the index in x_2-direction. The k^{th} strip is described by the following set $S_k = \{(i,j) | i \in [(k-1)\cdot n_x + 1, k \cdot n_x], j \in [1, n_y]\}$.

A 5 point stencil

The vector of unknowns is denoted by $x(i,j)$. For a 5 point stencil it appears that in the solution of $Lx = b$, unknown $x(i,j)$ only depends on $x(i-1,j)$ and $x(i,j-1)$. The parallel algorithm now runs as follows: first all elements $x(i,1)$ for $(i,1) \in S_1$ are calculated on processor 1. Thereafter communication takes place between processor 1 and 2. Now $x(i,2)$ for $(i,2) \in S_1$ and $x(i,1)$ for $(i,1) \in S_2$ can be calculated in parallel etc. After some start-up time all processors are busy (Figure 7).

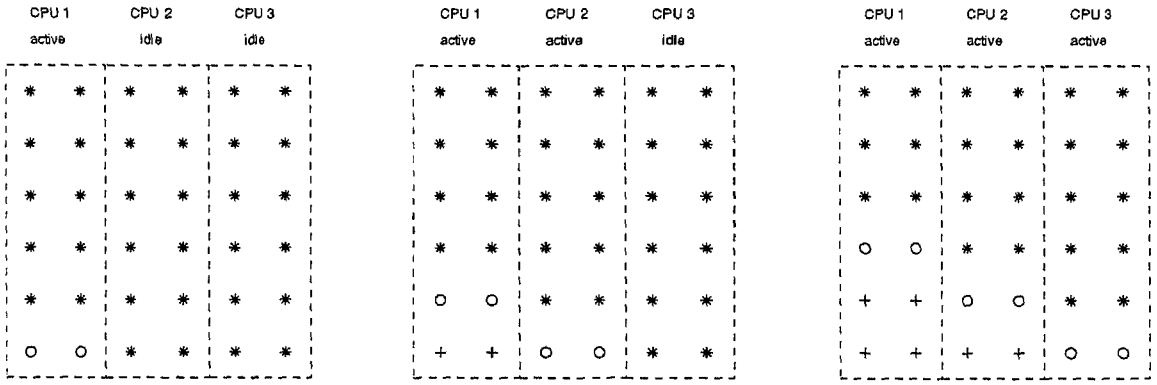


Figure 7: The first stages of the parallel solution of the lower triangular system $Lx = b$. The symbols denote the following: $*$ nodes to be calculated, o nodes are calculated, and $+$ nodes have been calculated.

A 9 point stencil
When a 9 point stencil is used the value of $x(i,j)$ depends on $x(i-1,j-1)$, $x(i,j-1)$, $x(i+1,j-1)$ and $x(i-1,j)$. Now the algorithm runs as follows: processor 1 calculates $x(i,1)$ for $(i,1) \in S_1$. The value of $x(n_x,1)$ is sent to processor 2. Thereafter processor 2 calculates $x(n_x+1,1)$ and sends it to processor 1. Then $x(i,2)$ for $(i,2) \in S_1$ and $x(i,1)$ for $(i,1) \in S_2$ are calculated in parallel etc. Note that some extra communication is necessary for a 9 point stencil. However, overlap of communication and computation is possible.

Note that the torus communication network of the Cray T3D is well adapted for this kind of communication. Only nearest neighbour communication takes place. Per message only one real is sent, so this approach is only useful on computers with a low latency with respect to send time. For this reason we use shared arrays (as in the inner product) instead of PVM subroutines. The auxiliary cells are again used to store information from neighbouring processors.

4 A performance model of the linear solver

We present a simple model to analyse the speedup of the components involved in the preconditioned GMRES algorithm on an abstract machine. This enables us to get an idea of the behaviour of various versions of the algorithm for different architectures. In this section we only consider the preconditioned GMRES(m) part of ISNaS.

4.1 An abstract machine

Many different aspects of parallel computer architecture play a role in determining the execution speed of a parallel algorithm. Some of them are: peak flop rate, design of the individual processors, cache size, latency, memory access time and structure, communications network, bandwidth etc. As a minimum we need to take into account the number of floating point operations that the processor can realistically be expected to perform per second, the communication latency, i.e. the time needed for the transmission of a zero length message, and the bandwidth, the maximum amount of data that can be transmitted per second.

We assume that the latency and bandwidth of our abstract machine do not depend on the distance between the sender and receiver. We define the following quantities: f is an estimate of the number of flops per second per processor,

$$R_l = \textit{latency in seconds} \times f,$$

and

$$R_b = \frac{f}{\textit{floating point numbers transported per second}}.$$

These quantities model efficiency loss due to latency and bandwidth restrictions. In words they mean the following: R_l is the number of flops, which can be done in the time necessary to send a zero length message and R_b is the number of flops, which can be done in the time that one floating point number is sent from a sending to a receiving processor.

4.2 An abstract algorithm

In an abstract algorithm there will be a number of floating point operations to be performed, split into a serial and a parallel fraction, and a number of messages of different lengths to be sent. To simplify things, assume the serial fraction of the algorithm is executed on all processors. When p processors are used, W(p) (=serial fraction+ parallel fraction/p) is the number of flops done on one processor. Let M(p) be the number of messages sent and received by one processor and the total length of those messages is denoted by L(p).

We give three speedup definitions, namely the theoretical speedup for a machine with instantaneous communication:

$$S_0(p) = \frac{W(1)}{W(p)}, \tag{4}$$

the true speedup found when running a program incorporating the algorithm:

$$S_t(p) = \frac{\textit{wall clock time for a run on one processor}}{\textit{wall clock time for a run on p processors}}, \tag{5}$$

and an estimate of the true speedup derived from the ratios introduced earlier:

$$S_e(p) = \frac{W(1)/f}{W(p)/f + M(p)R_l/f + L(p)R_b/f}. \tag{6}$$

Suppose that $W_{ov}(p)$ is the amount of work, which can be done independently of the communication. Then the estimated speedup with overlap of communication and calculation is

$$S_e(p) = \frac{W(1)/f}{W(p)/f + max(0, M(p)R_l/f + L(p)R_b/f - W_{ov}(p)/f)}, \tag{7}$$

which can also be written as

$$S_e(p) = S_0(p) \times \frac{1}{1 + max(0, R_l M(p)/W(p) + R_b L(p)/W(p) - W_{ov}(p)/W(p))}. \tag{8}$$

If we introduce the ratios $R_M(p) = M(p)/W(p), R_L(p) = L(p)/W(p)$ i.e. the start-up and transmission costs as fractions of the calculation costs, then we find the formula:

$$S_e(p) = S_0(p) \times \frac{1}{1 + max(0, R_l R_M(p) + R_b R_L(p) - W_{ov}(p)/W(p))}. \tag{9}$$

4.3 Speedup prediction

From now on we assume that the total number of grid points increases linearly with the number of processors. So we consider scaled speedup. Suppose n_1 and n_2 are given and the total number of grid points is equal to $n_1^{tot}(p) \times n_2^{tot}(p) = n_1\sqrt{p} \times n_2\sqrt{p}$, where we assume that p (the number of processors) is a square. Since the domain is split into p strips parallel to the x_2-axis, the number of grid points per processor: $n_x \times n_y = n_1/\sqrt{p} \times n_2\sqrt{p} = n_1 n_2$ is constant. This means we base our analysis on the Gustafsson model [8, 9]. Similar models are discussed in [14, 6].

4.3.1 Matrix-vector multiplication

This section contains a prediction of the optimal speedup obtainable for the matrix-vector product. Since the pressure matrix has 9 non-zero diagonals the number of flops per grid point is equal to 17 (9 multiplications and 8 additions). This leads to $W(p) = 17n_x n_y$. Per processor two communications are necessary, one to the left and one to the right neighbouring processor, so $M(p) = 2$ and

$$R_M(p) = \frac{2}{17n_x n_y} = \frac{2}{17n_1 n_2}. \tag{10}$$

The length of each communication is n_y which implies $L(p) = 2n_y$ and

$$R_L(p) = \frac{2n_y}{17n_x n_y} = \frac{2}{17n_x} = \frac{2\sqrt{p}}{17n_1}. \tag{11}$$

The calculation of the matrix vector product in the $n_x - 2$ interior nodes can be done independently of the communication. When overlap of communication and computation is used $W_{ov}(p)$ is given by

$$W_{ov}(p) = 17(n_x - 2)n_y, \tag{12}$$

otherwise $W_{ov}(p)$ is taken equal to zero. After substituting (10), (11), (12) into (9) and measuring the values of R_l and R_b one can predict the speedup of the matrix vector product.

4.3.2 Inner product

The Cray T3D computer is composed of pipelined RISC processors. It is well known that on such a processor the total time of a vector operation consists of a start-up time and the time to get one result multiplied by the length of the vector ([7] p. 56). The start-up time is denoted by t_{is} and f_{im} is the maximum flop rate once the routine runs. Using this notation the flop rate of the inner product is

$$f_i(n) = \frac{2n}{t_{is} + \frac{2n}{f_{im}}}. \tag{13}$$

We give this formula only for the inner product. However also for the other operations the flop rate depends on a start-up time and a maximum flop rate (see Section 6.3).

We also model the inner product speedup. The communication strategy of inner product II is used (see Section 2.2). For the inner product the ratios determining speedup are first of all a theoretical speedup for infinite communication speed. There are $2pn_1n_2$ flops done if one processor is used and $W(p) = 2n_1n_2 + \log_2 p$ flops on each processor when p processors are used, so

$$S_0(p) = \frac{2pn_1n_2}{2n_1n_2 + \log_2 p}.$$

To pass the global inner product to all processors $\log_2 p$ communication steps are necessary, where in each step p simultaneous messages are sent. Since only one floating point is sent in each message the values of $R_M(p)$ and $R_L(p)$ are the same

$$R_M(p) = R_L(p) = \frac{\log_2 p}{2n_1n_2 + \log_2 p}.$$

Since no overlap is possible $W_{ov}(p) = 0$.

4.3.3 The parallel preconditioner

In this section we analyse the speedup of the solution of the lower triangular system $Lx = b$, where a 9 point discretization stencil is used. The parallel algorithm to solve this system is given in Section 3.2. The amount of work is 9 flops (4 multiplications, 4 additions and 1 division) per grid point. For the theoretical speedup a delay is caused by the imbalance in the lower left and upper right corner. It takes some time before the p^{th} processor becomes active. This processor has to wait until $9(p-1)n_x \simeq 9pn_x$ operations are done, before it starts to perform the $9n_xn_y$ flops for its own sub-domain. Combination leads to $W(p) = 9(pn_x + n_xn_y)$. On one processor $9pn_xn_y$ flops are done, so

$$S_0(p) = \frac{9pn_xn_y}{9(pn_x + n_xn_y)} = \frac{pn_y}{p + n_y} = \frac{p}{\frac{p}{n_y} + 1} = \frac{p}{\frac{\sqrt{p}}{n_2} + 1}.$$

For $n_2 = 1$ the speedup is $S_0(p) = \frac{p}{\sqrt{p}+1} \simeq \sqrt{p}$ and for n_2 large $S_0(p) \simeq p$. For all values of n_2 the theoretical speedup $S_0(p)$ lies between these bounds.

To account for communication delays we note that 2 messages (one to the left and one to the right) per horizontal line are done on each processor. This leads to a total of $2n_y$ messages per processor. Again initial messages are necessary before the p^{th} processor becomes active. The number of these messages is $2(p-1) \simeq 2p$. Since all these messages contain only one floating point number, we have

$$R_M(p) = R_L(p) = \frac{2p + 2n_y}{9(pn_x + n_xn_y)} = \frac{2}{9n_x} = \frac{2\sqrt{p}}{9n_1}.$$

Note that only the communication to the left neighbouring processor can be overlapped by calculations.

5 Experiments with the parallel code

We consider the curved channel problem as described in [21] (see Figure 8). Boundary fitted coordinates are used to map this domain onto a rectangle. In mapping the physical domain to the computational domain, the x_1-coordinate is taken along the side containing the inner side of the bend. We start with zero velocities and pressures and a parabolic inflow at boundary 1 and take three time steps. The solution process was artificially interrupted after 19 iterations of full GMRES for the momentum equations and 26 iterations of full GMRES for the pressure equation to allow for direct comparison of timings on different sized grids.

The measurements reported in this section were done at the Cray T3D computer at the Edinburgh Parallel Computing Center. The tables are given at the end of this report. The wall clock times are measured for the matrix construction (Table 1 and 4), preconditioner construction (Table 2 and 5), and preconditioner construction combined with linear system solution (Table 3 and 6). These measurements are given for both the momentum equations

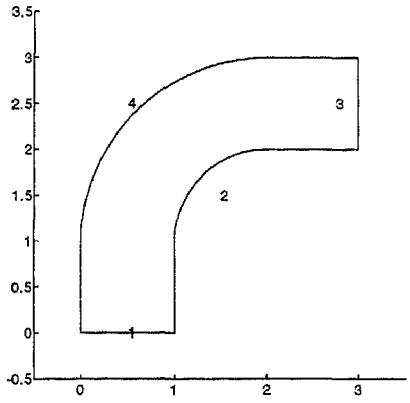


Figure 8: The physical domain of the curved channel problem.

and the pressure equation.

The results are only presented in a certain band. To explain this, consider the 32×8 grid. For the parallelization the computational domain is split into strips parallel to the x_2-axis. This means that if 8 processors are used the grid on one processor consists of 4×8 grid points. It is impossible to reduce this further, so no measurements are done for the 32×8 grid using a larger number of processors. On the other hand using 1 processor one cannot solve grid sizes larger than 256×64 due to memory limitations.

The matrix and preconditioner construction show a nice behaviour when the number of processors is increased. In general the momentum results are somewhat better than those for the pressure. A possible reason is that more calculations are done for the momentum equations so overhead costs are less important and more overlap is possible between communication and computation. With respect to the solution of the system we see that the wall clock time for 2 processors is the same as for 1 processor. We cannot explain this phenomenon, except by suggesting that the underlying architecture (2 processors share a communication node) slows down the computation for this particular case.

Estimated Megaflop rates for the preconditioner construction and solution of the linear system are given in Table 7 and 8. The Megaflop rates are estimated by an approximated flop count for the preconditioner construction and GMRES divided by the measured wall clock times. Table 9 contains the Megaflop rates for the inner product. There is also a table (Table 10) giving Megaflop rates for an inner product, where the global summation is omitted. This will be used later in Section 6.1 to estimate the communication costs associated with the global summation. The tables for the inner product (see the first three rows of Table 9 and 10) illustrate the somewhat irregular behaviour of the BLAS routine SDOT for very short vectors.

In theory the maximum Megaflop rate of 1 processor is 150 Mflop/s. The observed flop rates are much lower: 33.5 for the inner product (Table 9, 10) , and 12.5 for the solution of the

systems (Table 7, 8). For the inner product this leads to an expected rate of 4288 Mflop/s on 128 processors. For long vectors the observed rate (see Table 9 and 10) is very close to this value. The expected rate for preconditioning construction and system solution is 1600 Mflop/s using 128 processors. The observed rates (Table 7: 790, 8: 630) for the 2048×512 problem are approximately 50 % of the expected rate. This corresponds well with our observation that the wall clock time is the same using 1 or 2 processors.

To check the efficiency we determine the total time used per variable. This is calculated by dividing the wall clock time by the number of variables per processor. The number of variables per grid point is 2 for the momentum equations and 1 for the pressure equation. Again these results are given for the matrix construction (Table 11 and 14), preconditioner construction (Table 12 and 15), and preconditioner construction combined with linear system solution (Table 13 and 16).

In order to investigate scalability of our approach we visualise the efficiency for the momen-

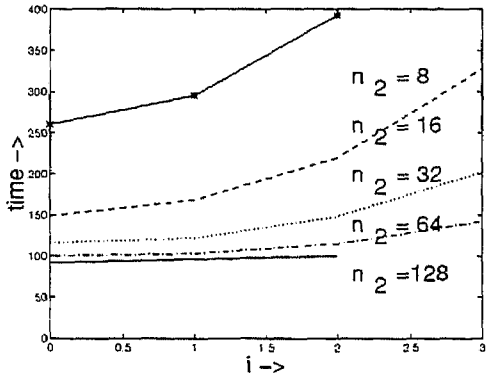


Figure 9: Total CPU time per variable in μseconds for the matrix construction

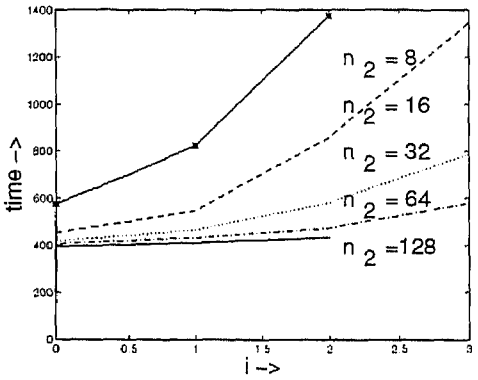


Figure 10: Total CPU time per variable in μseconds for the preconditioner construction and solution of the system

tum equations in Figure 9 (matrix construction) and 10 (system solution). In these figures the grid size is given by $4n_2 \cdot 2^i \times n_2 \cdot 2^i$ and the number of processors is equal to $2 \cdot 4^i$. A constant amount of CPU time per variable means that the considered algorithm is scalable. It appears that both algorithms are scalable when n_2 is large enough ($n_2 \geq 64$). For smaller values of n_2 the efficiency deteriorates. One reason for this is the overhead of computation due to the auxiliary grid cells. Another reason is a small vector length, which leads to low Megaflop rates on the RISC computers of the Cray T3D (compare Table 10). The efficiency loss for the solution algorithm is more severe than that for the matrix construction. This is probably caused by the fact that matrix construction is embarrassingly parallel (without communication), whereas the solution algorithm is parallelized using communication. It appears from Section 4.3 that the relative start-up $R_M(p)$ and transmission costs $R_L(p)$ increase for increasing p. Finally for small values of n_2 only a small amount of communication can be overlapped by computation.

6 Analysis of the timing experiments

In this section the measurements given in Section 5 are analysed using the models specified in Section 4. First some parameters are estimated: start-up time, transmission time and latency. Thereafter Megaflop rates are predicted for the various parts of GMRES and if possible compared with the rates observed. Finally we compute the total time per grid point for the solution of the pressure equation and compare them with the observed values.

6.1 Parameters of the Cray T3D

Before we give quantitative results it is helpful to consider the T3D machine on a qualitative level. Important characteristics are the high bandwidth of the inter-processor communication channels (300 Mbyte/sec, bi-directional along all torus directions), their low latency and the presence of latency hiding hardware. The high processor speed (150 MHz clock) and the small cache of 8 Kbyte imply severe penalties for code that is not specifically optimised for the T3D.

Estimates of communication costs are complicated by the fact that two processors share a node in the communication network. This means that the step from one to two processors may show atypical behaviour for communication intensive algorithms. Automatic rerouting, latency hiding and spare nodes make precise estimates of communication costs difficult. Of course rough estimates are still possible. When a small number of processors is required the T3D automatically restricts the communication pattern to a 1D or 2D torus, otherwise a 3D torus segment is used.

The start-up time, transmission time and latency are obtained from the measurements of the inner product. It appears from Table 10 that the inner product behaviour does not match model (13) for vectors with fewer than 64 elements. This is possibly caused by strip-mining effects ([7] p. 11). Applying the model to the remaining data and assuming a maximum flop rate of $f_{im} = 33.4$ Megaflops per second, we find a start-up time of approximately $t_{is} = 12\mu$seconds for Table 9 and $t_{is} = 7\mu$seconds for Table 10.

We estimate the communication latency t_l from Table 9 and 10. The inner product of two vectors with n elements costs $2n$ flops. The time $t(p)$ needed to perform the global summation on p processors consists of the following parts: a fixed overhead, $\log_2 p$ message start-ups, and $\log_2 p$ reals are sent. It appears that the transmission time t_b is equal to $\frac{16}{300}$ μseconds per real (300 MByte/sec per link per 2 processors). To eliminate the fixed overhead we consider $t(p)-t(2)$. The value $t(1)$ cannot be used because no communication takes place if 1 processor is used. When we fit the model

$$t(p) - t(2) = t_l \log_2 p + t_b \log_2 p - t_l - t_b \tag{14}$$

to the measurements we find a latency of approximately 4μseconds.

6.2 Predicted speed

The Megaflop rates are computed with the parameters specified in Section 6.1. Table 17 contains the predicted Megaflop rates of the inner product using the model discussed in Section 4.3.2. The predicted rates (Table 17) show a good correspondence with the observed rates (Table 9).

We also use the values of t_l and t_b from the inner product to predict the rate of the matrix vector product. Note that t_l and t_b are based on the Cray specific shared array code. The actual matrix-vector product code uses MPI, so it should be somewhat slower. On 1 processor the preconditioner construction and GMRES method runs at 10 Mflop/s (see Table 7 and 8). For this reason we present predicted speeds for the matrix vector product without communication overlap for $f_m = 10$ in Table 18 and $f_m = 5$ in Table 19. When overlap is used we see a perfect speedup. The Megaflop rate is then equal to $f_m p$.

Finally, a prediction of the performance is given for the solution of a lower triangular system, which is used in a preconditioner vector product. The results without communication overlap are given in Table 20 ($f_p = 10$) and Table 22 ($f_p = 5$), whereas Table 21 and 23 contain the predictions when overlap is taken into account. Only one half of the communication can be overlapped. When n_x (number of grid points in x_1-direction on one processor) is relatively small the Megaflop rate of the preconditioner vector product is much less than the matrix vector product. For $n_x > 64$ both Megaflop rates are comparable.

6.3 Prediction of the solution time for the pressure equation

In our experiments we take 26 iterations of the preconditioned GMRES method to solve the pressure equation. The amount of work of k iterations of full GMRES (applied to a problem with grid size $n_1\sqrt{p} \times n_2\sqrt{p}$) is: k matrix vector products with $17pn_1n_2$ flops, $3k$ back substitutions with $9pn_1n_2$ flops, $k^2/2$ inner products with $2pn_1n_2$ flops and $k^2/2$ vector updates with $2pn_1n_2$ flops. Let f_i be the inner product flop rate, f_m the matrix vector flop rate, f_p the back substitution flop rate, and f_u the vector update flop rate. The total time, measured in μ seconds, needed on 1 processor is:

$$t_{tot} = k\left(\frac{17}{f_m} + \frac{27}{f_p} + \frac{k}{f_i} + \frac{k}{f_u}\right) pn_1n_2 \ .$$

The time per grid point is then approximately

$$t = k\left(\frac{17}{f_m} + \frac{27}{f_p} + \frac{k}{f_i} + \frac{k}{f_u}\right) \ . \tag{15}$$

When k is very large it appears from (15) that the inner products and vector updates dominate the run time of the preconditioned GMRES method. In such a case it is important that the inner product is parallelized very well.

The Megaflop rates which are used in (15), are based on (13). From the experiments we

observe that the Megaflop rates on 2 processors are approximately 50% of the Megaflop rates on 1 processor. Therefore we use the following formulae:

$$f_m = \frac{17(n_x-2)}{14+\frac{17(n_x-2)}{5}} \quad , \quad f_p = \frac{9(n_x-2)}{14+\frac{9(n_x-2)}{5}} \quad , \quad f_i = \frac{2n_xn_y}{7+\frac{2n_xn_y}{16}} \quad , \quad f_u = \frac{2n_xn_y}{7+\frac{2n_xn_y}{10}} \; .$$

This together with the communication model (with overlap) is used to predict the total time per grid cell. The results are given in Table 24. There is a good correspondence between the predictions (Table 24) and the measurements (Table 16). So the described model can be used to predict the efficiency of the proposed parallel method also for larger grid sizes and/or a larger number of processors. Finally we note that $3k$ preconditioner vector products are used. This can be lowered to $2k$. Again the model can be used to predict the effect of this change. It appears that this is favourable for the scalability of the method.

In figures 11, 12, and 13 we present the percentage of the total time for the various parts of GMRES. Figure 11 contains the results for $p = 8$ and an increasing grid size. It appears that the preconditioner vector product is the most time consuming part, it takes 70 % of the time for a small grid size and 45 % for a large grid size. A comparable behaviour is seen for a fixed grid size and an increasing number of processors (Figure 12). In Figure 13 the results are shown for the Gustafsson model, the grid size increases linearly with the number of processors. There is only a small increase in the percentage used for the preconditioner vector product. This model suggests, as expected, that the preconditioner can be a bottle-neck especially if the number of grid cells in x_1-direction per processor is small.

The ratio between the computational cost and communication cost for the the preconditioner vector product is higher for the momentum equations than for the pressure equation. Therefore we expect a better efficiency for the momentum equations than for the pressure equation. This expectation is confirmed by the measurements in Table 13 and Table 16.

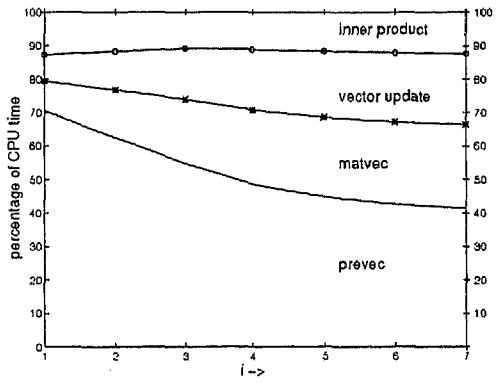


Figure 11: The percentage of CPU time used by the various parts (grid size $32 \cdot 2^{(i-1)} \times 8 \cdot 2^{(i-1)}$, 8 processors)

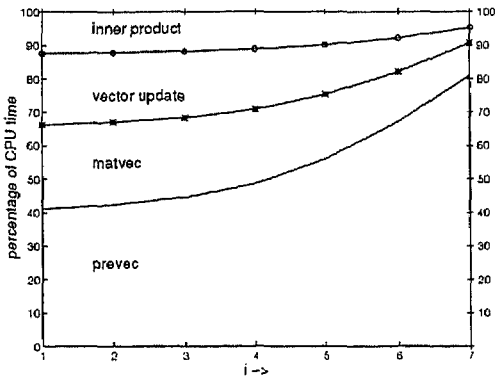


Figure 12: The percentage of CPU time used by the various parts (grid size 256×64, $2^{(i-1)}$ processors)

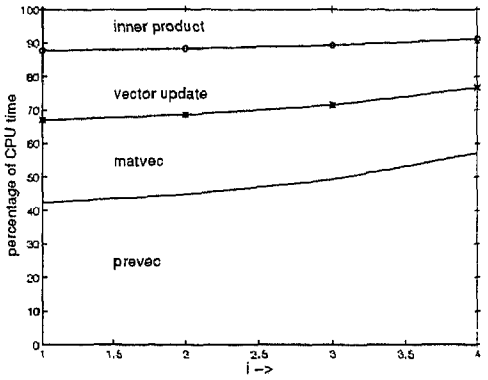


Figure 13: The percentage of CPU time used by the various parts (grid size $256 \cdot 2^{(i-1)} \times 64 \cdot 2^{(i-1)}$, $4^{(i-1)}$ processors)

7 Conclusions

A parallel implementation of the preconditioned GMRES methods is given, which is scalable when the grid size is large enough. The proposed model describes the required time per grid point adequately and can be used to analyse the method or to predict the efficiency on larger number of grid points or processors.

When the number of iterations increases the percentage of time spent in inner products and vector updates increases. Since these parts have a scalable parallel behaviour we see no parallelization problems, also when a large number of processors is used.

The ILU-preconditioner is parallelized using the ideas proposed in [2]. Advantage of this method is: the serial and parallel version of this method have the same behaviour with respect to convergence, size of the residual and effects of rounding errors. The only drawback is that the efficiency deteriorates when the domain is divided into thin slices. The reasons for this are: communication time is large with respect to computation time, many isolated floating point operations occur, an increase of overlap between the domains and a low flop rate for short vectors.

When the number of processors/domains becomes large it seems a good idea to combine coarse grain parallelism (domain decomposition [5]) and our parallel method (fine-grain parallelism). This is especially true on a Distributed Shared Memory (DSM) computer, which has a non-uniform latency [10]. A DSM computer consists of several identical building blocks. Every building block contains a number of processors connected to the same (shared) memory. The communication between the building blocks is done by message passing. The proposed combination is: the domains used in the domain decomposition method, are distributed over the building blocks (larger latency), whereas on each building block (small latency) our parallel method is used.

References

[1] C.C. Ashcraft and R.G. Grimes. On vectorizing incomplete factorization and SSOR preconditioners. *SIAM J. Sci. Stat. Comput.*, 9:122–151, 1988.

[2] P. Bastian and G. Horton. Parallization of robust multi-grid methods: ILU factorization and frequency decomposition method. *SIAM J. Stat. Comput.*, 12:1457–1470, 1991.

[3] H. Bijl and P. Wesseling. A numerical method for the computation of compressible flows with low mach number regions. In J.-A. Désid'eri, C. Hirsch, P. Le Tallec, M. Pandolfi, and J. Périaux, editors, *Computational Fluid Dynamics '96. Proc. Third ECCOMAS Computational Fluid Dynamics Conference*, pages 206 – 212, Chichester, 1996. Wiley.

[4] Å. Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT*, 7:1–21, 1967.

[5] E. Brakkee, A. Segal, and C.G.M. Kassels. A parallel domain decomposition algorithm for the incompressible Navier-Stokes equations. *Simulation Practice and Theory*, 3:185–205, 1995.

[6] A. L. Couch. Locating performance problems in massively parallel executions. *Proceedings of the IEEE*, 81:1116–1125, 1993.

[7] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Solving linear systems on vector and shared memory computers.* SIAM, Philadelphia, 1991.

[8] J. Gustafson. Reevaluating Amdahl's law. *Comm. ACM*, 31:532–533, 1988.

[9] J.L. Gustafson, G.R. Montry, and R.E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM J. Sci. Stat. Comput.*, 9:609–638, 1988.

[10] P.H. Michielse. High performance computing: trends and expectations. In P. Wesseling, editor, *Proceedings of the Summerschool on High Performance Computing in Fluid Dynamics, Delft University of Technology, The Netherlands, June 24-28, 1996*, ERCOFTAC series, pages 261–278, Dordrecht, 1996. Kluwer.

[11] C.W. Oosterlee and P. Wesseling. Multigrid schemes for time dependent incompressible Navier-Stokes equations. *Impact of Comp. in Science and Engng.*, 5:153–175, 1993.

[12] C.W. Oosterlee and P. Wesseling. Steady incompressible flow around objects in general coordinates with a multigrid solution method. *Num. Meth. Part. Diff. Eq.*, 10:295–308, 1994.

[13] C.W. Oosterlee, P. Wesseling, A. Segal, and E. Brakkee. Benchmark solutions for the incompressible Navier-Stokes equations in general co-ordinates on staggered grids. *Int. J. Num. Meth. Fluids*, 17:301–321, 1993.

[14] M.J. Quinn. *Parallel Computing; Theory and Practice (second edition).* McGraw-Hill, New York, 1994.

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

[16] Guus Segal, Kees Vuik, and Kees Kassels. On the implementation of symmetric and antisymmetric periodic boundary conditions for incompressible flow. *Int. J. Num. Meth. Fluids*, 18:1153–1165, 1994.

[17] P. van Beek, R.R.P. van Nooyen, and P. Wesseling. Accurate discretization on non-uniform curvilinear staggered grids. *J. Comp. Phys.*, 117:364–367, 1995.

[18] R.P.P. van Nooyen. Parallellism in the matrix construction routines and the linear system solver for the ISNaS program. Report 95-05, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1995.

[19] R.R.P. van Nooyen, C. Vuik, and P. Wesseling. Some parallelizable preconditioners for gmres. Report 96-50, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1996.

[20] C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. for Num. Meth. Fluids*, 16:507–523, 1993.

[21] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *Int. J. for Num. Meth. Fluids*, 22:195–210, 1996.

[22] P. Wesseling, C.G.M. Kassels, C.W. Oosterlee, A. Segal, C. Vuik, S. Zeng, and M. Zijlema. Computing incompressible flows in general domains. In F. Hebeker, R. Rannacher, and G. Wittum, editors, *Numerical methods for the Navier-Stokes equations Proceedings of the international workshop held at Heidelberg, October 25-28, 1993*, pages 298–314, Braunschweig, 1994. Vieweg.

[23] P. Wesseling, A. Segal, J.J.I.M. van Kan, C.W. Oosterlee, and C.G.M. Kassels. Finite volume discretization of the incompressible Navier-Stokes equations in general coordinates on staggered grids. *Comp. Fluid Dyn. J.*, 1:27–33, 1992.

[24] P. Wesseling, P. van Beek, and R.R.P. van Nooyen. Aspects of non-smoothness in flow computations. In A. Peters, G. Wittum, B. Herrling, U. Meissner, C.A. Brebbia, W.G. Gray, and G.F. Pinder, editors, *Computational Methods in Water Resources X*, pages 1263 – 1271, Dordrecht, 1994. Kluwer.

[25] S. Zeng and P. Wesseling. Multigrid solution of the incompressible Navier-Stokes equations in general coordinates. *SIAM J. Num. Anal.*, 31:1764–1784, 1994.

[26] M. Zijlema. On the construction of a third-order accurate monotone convection scheme with application to turbulent flow in general coordinates. *Int. J. Numer. Meth. in Fluids*, 22:619–641, 1996.

[27] M. Zijlema, A. Segal, and P. Wesseling. Finite volume computation of incompressible turbulent flows in general co-ordinates on staggered grids. *Int. J. Numer. Methods Fluids*, 20:621–640, 1995.

[28] M. Zijlema, A. Segal, and P. Wesseling. Invariant discretization of the $k-\varepsilon$ model in general co-ordinates for prediction of turbulent flow in complicated geometries. *Computers and Fluids*, 24:209–225, 1995.

#PE's	1	2	4	8	16	32	64	128	256
32×8	105.2	71.8	54.3	46.9					
64×16	278.2	158.9	103.5	78.4	63.7				
128×32	875.0	483.8	274.4	175.5	125.0	102.5			
256×64	3109.8	1645.4	861.9	507.1	320.4	226.3	178.5		
512×128		6028.1	3115.3	1708.7	959.9	610.6	430.3	337.8	
1024×256				6301.9	3317.7	1880.8	1174.0	832.0	659.9
2048×512						6550.2	3693.7	2337.5	1646.0

Table 1: Momentum matrix construction times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	13.3	12.3	7.4	5.9					
64×16	63.3	30.7	18.8	15.1	10.8				
128×32	191.0	103.4	63.0	36.8	25.8	22.5			
256×64	713.4	379.3	208.0	119.1	75.1	52.1	41.8		
512×128		1454.8	776.5	415.1	251.0	168.2	108.2	84.8	
1024×256				1568.0	887.6	487.3	320.1	217.7	172.0
2048×512						1781.2	1068.4	706.0	428.8

Table 2: Momentum preconditioner construction times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	136.6	159.2	122.9	113.5					
64×16	495.2	483.5	310.1	219.4	190.4				
128×32	1800.2	1746.7	969.4	573.1	398.6	359.2			
256×64	6913.1	6733.3	3612.3	1931.1	1138.7	887.2	692.7		
512×128		25980.5	13089.0	7121.5	3906.6	2392.0	1610.9	1388.0	
1024×256				27042.7	14198.3	7784.6	4755.1	3240.3	2764.1
2 048×512						28536.7	15824.5	9528.2	6627.9

Table 3: Momentum preconditioner construction and solution times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	63.1	46.6	39.6	36.3					
64×16	128.8	85.1	62.4	52.5	46.6				
128×32	347.1	203.6	132.8	95.7	78.2	68.9			
256×64	1103.0	602.5	345.8	224.6	160.9	129.8	113.0		
512×128		2055.1	1114.5	647.6	410.8	293.2	235.8	204.6	
1024×256				2139.8	1231.0	779.2	555.8	441.4	388.3
2048×512						2395.0	1520.0	1080.1	864.9

Table 4: Pressure matrix construction times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	9.4	6.7	5.4	4.9					
64×16	36.1	23.0	14.2	9.6	8.3				
128×32	144.2	82.5	44.8	27.3	19.0	16.8			
256×64	568.4	300.1	190.0	100.1	63.5	46.6	34.7		
512×128		1615.3	602.2	332.2	192.2	112.9	79.3	66.6	
1024×256				1233.7	670.9	406.7	231.4	173.8	133.3
2048×512						1335.0	745.5	502.1	335.9

Table 5: Pressure preconditioner construction times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	75.6	89.2	82.4	85.2					
64×16	225.6	228.6	167.0	133.2	134.3				
128×32	878.7	778.9	448.8	292.5	231.5	233.6			
256×64	3237.0	2911.2	1544.1	878.4	562.4	473.2	424.1		
512×128		11653.3	5695.5	3099.2	1718.1	1090.9	837.0	805.3	
1024×256				11630.8	6164.8	3478.7	2151.5	1671.3	1612.1
2048×512						12925.6	6867.1	4532.6	3294.0

Table 6: Pressure preconditioner construction and solution times in msec

#PE's	1	2	4	8	16	32	64	128	256
32×8	11.3	9.7	12.6	13.6					
64×16	11.6	11.9	18.5	26.1	30.1				
128×32	12.0	12.2	22.1	37.3	53.6	59.5			
256×64	12.6	12.9	24.0	44.8	75.9	86.7	124.8		
512×128		13.4	26.7	49.1	89.0	145.4	215.7	250.1	
1024×256				72.7	136.9	249.6	403.0	588.2	709.6
2048×512						271.6	478.4	786.7	1167.0

Table 7: Momentum preconditioner construction and solution performance in Megaflops per second

#PE's	1	2	4	8	16	32	64	128	256
32×8	10.7	9.1	9.8	9.5					
64×16	13.3	13.1	18.0	22.5	22.3				
128×32	13.2	14.8	25.7	39.5	49.8	49.4			
256×64	14.7	16.5	31.1	54.7	85.5	101.8	113.9		
512×128		15.5	31.6	58.1	104.8	165.0	215.2	223.7	
1024×256				61.6	116.2	206.0	333.0	428.8	445.1
2048×512						221.2	416.3	630.7	868.0

Table 8: Pressure preconditioner construction and solution performance in Megaflops per second

#PE's #elements per processor	1	2	4	8	16	32	64	128
32	10.1	7.2	11.7	20.0	34.8	61.2	108.9	194.9
64	8.2	8.9	15.5	27.8	50.3	91.5	167.2	307.2
128	13.2	15.5	27.6	50.1	91.5	167.3	307.8	568.3
256	19.0	25.2	45.8	84.5	155.8	289.6	538.8	1004.2
512	23.9	36.2	67.6	127.1	239.3	451.1	853.2	1612.3
1024	28.0	46.9	90.0	172.6	331.3	636.2	1221.6	2346.2
2048	30.5	55.3	107.4	209.6	409.1	797.2	1553.1	2997.7
4096	31.9	60.6	118.8	234.9	463.1	912.8	1796.8	3542.4
8192	32.6	63.5	125.9	250.1	495.7	974.1	1950.8	3875.3
16384	33.1	65.1	129.2	258.4	514.8	1025.6	2013.0	4057.3
32768	33.3	66.0	131.5	262.7	524.3	1046.5	2082.2	4152.9
65536	33.4	66.4	132.6	264.9	529.3	1057.7	2112.7	4212.0

Table 9: Inner product performance in Megaflops per second with communication

#PE's #elements per processor	1	2	4	8	16	32	64	128
32	25.0	49.9	99.9	199.8	399.6	799.2	1598.3	3196.5
64	11.9	24.2	48.5	97.0	193.8	388.0	775.3	1552.2
128	17.7	35.5	70.9	141.8	283.6	567.2	1132.9	2265.1
256	23.2	46.4	93.1	186.2	372.3	744.6	1490.2	2978.4
512	27.1	54.0	108.4	216.9	433.7	867.6	1733.5	3470.0
1024	30.0	60.0	120.0	239.3	479.9	959.8	1919.7	3839.7
2048	31.7	63.3	126.3	252.7	505.8	1011.6	2022.7	4046.7
4096	32.5	65.0	129.9	259.9	519.8	1039.6	2077.0	4157.7
8192	33.0	65.9	131.8	263.3	527.1	1053.1	2108.1	4212.8
16384	33.2	66.4	132.8	265.6	531.1	1062.2	2124.6	4247.9
32768	33.3	66.7	133.3	266.5	532.9	1062.4	2131.6	4262.4
65536	33.4	66.8	133.5	266.9	533.9	1067.6	2135.1	4270.2

Table 10: Inner product performance in Megaflops per second without communication

#PE's	1	2	4	8	16	32	64	128	256
32×8	190.5	260.3	393.8	679.4					
64×16	130.7	149.4	194.6	294.8	478.8				
128×32	104.8	115.9	131.4	168.1	239.5	392.6			
256×64	94.0	99.5	104.2	122.6	155.0	218.9	345.2		
512×128		91.5	94.6	103.8	116.6	148.3	209.1	328.3	
1024×256				95.9	101.0	114.5	143.0	202.6	321.4
2048×512						99.8	112.6	142.5	200.7

Table 11: Momentum equation matrix construction; total time per variable in μseconds

#PE's	1	2	4	8	16	32	64	128	256
32×8	24.15	44.62	53.69	85.73					
64×16	29.77	28.90	35.27	56.76	81.39				
128×32	22.87	24.76	30.15	35.23	49.48	86.23			
256×64	21.56	22.92	25.14	28.80	36.33	50.41	80.95		
512×128		22.09	23.58	25.21	30.50	40.87	52.59	82.42	
1024×256				23.87	27.02	29.67	38.98	53.03	83.78
2048×512						27.15	32.57	43.04	52.28

Table 12: Momentum preconditioner construction; total time per variable in μseconds

#PE's	1	2	4	8	16	32	64	128	256
32×8	247.5	576.8	890.6	1644.9					
64×16	232.7	454.4	582.9	824.8	1431.6				
128×32	215.5	418.3	464.3	548.9	763.6	1376.2			
256×64	208.9	407.0	436.7	466.9	550.6	858.0	1339.8		
512×128		394.5	397.5	432.5	474.6	581.1	782.8	1348.9	
1024×256				411.6	432.2	474.0	579.0	789.2	1346.4
2048×512						434.9	482.3	580.8	808.1

Table 13: Momentum preconditioner construction and solution; total time per variable in μseconds

#PE's	1	2	4	8	16	32	64	128	256
32×8	246.3	364.4	618.4	1133.8					
64×16	125.8	166.3	243.8	410.5	727.5				
128×32	84.7	99.4	129.7	186.8	305.5	538.1			
256×64	67.3	73.5	84.4	109.7	157.2	253.6	441.2		
512×128		62.7	68.0	79.1	100.3	143.2	230.2	399.6	
1024×256				65.3	75.1	95.1	135.7	215.5	379.2
2048×512						73.1	92.8	131.8	211.2

Table 14: Pressure equation matrix construction; total time per cell in μseconds

#PE's	1	2	4	8	16	32	64	128	256
32×8	36.59	52.07	83.93	152.27					
64×16	35.26	45.01	55.60	75.23	129.62				
128×32	35.20	40.27	43.78	53.29	74.35	131.10			
256×64	34.69	36.64	46.39	48.87	61.99	90.98	135.45		
512×128		49.29	36.76	40.55	46.93	55.14	77.46	130.06	
1024×256				37.65	40.95	49.65	56.50	84.85	130.20
2048×512						40.74	45.50	61.29	82.01

Table 15: Pressure preconditioner construction; total time per variable in μseconds

#PE's	1	2	4	8	16	32	64	128	256
32×8	295.3	696.9	1287.5	2662.5					
64×16	220.3	446.5	652.3	1040.6	2098.4				
128×32	214.5	380.3	438.3	571.3	904.3	1825.0			
256×64	197.6	355.4	377.0	428.9	549.2	924.2	1656.6		
512×128		355.6	347.6	378.3	419.5	532.7	817.4	1572.9	
1024×256				354.9	376.3	424.6	525.3	816.1	1574.3
2048×512						394.5	419.1	553.3	804.2

Table 16: Pressure preconditioner construction and solution; total time per cell in μseconds

#PE's	1	2	4	8	16	32	64	128
32	4.5	7.0	11.4	19.1	33.0	58.1	103.6	187.0
64	8.0	12.8	21.2	36.1	63.0	111.6	200.4	363.7
128	13.0	21.5	36.7	63.9	113.3	203.3	368.8	675.0
256	18.7	32.6	57.6	103.4	187.3	342.5	630.9	1169.3
512	24.0	43.8	80.6	149.2	277.6	519.3	975.2	1838.5
1024	27.9	52.9	100.5	191.5	365.5	699.2	1339.9	2572.3
2048	30.4	59.1	114.7	223.1	434.2	845.4	1647.4	3212.4
4096	31.8	62.7	123.5	243.2	479.1	944.1	1860.8	3668.5
8192	32.6	64.7	128.3	254.6	505.2	1002.6	1989.7	3948.7
16384	33.0	65.7	130.9	260.8	519.4	1034.6	2061.0	4105.5
32768	33.2	66.3	132.2	263.9	526.8	1051.4	2098.6	4188.6
65536	33.3	66.5	132.9	265.6	530.6	1060.1	2117.9	4231.5

Table 17: Estimated inner product Megaflop rates with communication

#PE's	1	2	4	8	16	32	64	128	256
32× 8	9.8	19.2	37.0	68.8					
64× 16	9.9	19.8	39.1	76.6	146.9				
128× 32	10.0	19.9	39.7	79.0	155.9	304.1			
256× 64	10.0	20.0	39.9	79.7	158.6	314.6	618.9		
512×128	10.0	20.0	40.0	79.9	159.5	318.0	632.1	1248.9	
1024×256	10.0	20.0	40.0	79.9	159.8	319.2	636.8	1267.1	2509.1
2048×512	10.0	20.0	40.0	80.0	159.9	319.6	638.6	1274.3	2537.2

Table 18: Estimated matrix vector product Megaflop rates ($f_m = 10$, no overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	4.9	9.8	19.2	37.0					
64× 16	5.0	9.9	19.8	39.1	76.6				
128× 32	5.0	10.0	19.9	39.7	79.0	155.9			
256× 64	5.0	10.0	20.0	39.9	79.7	158.6	314.6		
512×128	5.0	10.0	20.0	40.0	79.9	159.5	318.0	632.1	
1024×256	5.0	10.0	20.0	40.0	79.9	159.8	319.2	636.8	1267.1
2048×512	5.0	10.0	20.0	40.0	80.0	159.9	319.6	638.6	1274.3

Table 19: Estimated matrix vector product Megaflop rates ($f_m = 5$, no overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	7.8	11.4	13.8	13.2					
64× 16	8.8	14.7	21.6	26.3	25.6				
128× 32	9.3	17.0	28.6	42.2	51.6	50.4			
256× 64	9.7	18.4	33.5	56.4	83.3	102.1	100.1		
512×128	9.8	19.2	36.5	66.6	112.1	165.6	203.1	199.4	
1024×256	9.9	19.6	38.2	72.8	132.7	223.4	330.2	405.2	398.0
2048×512	10.0	19.8	39.1	76.3	145.3	264.9	446.0	659.3	809.5

Table 20: Estimated preconditioner vector product Megaflop rates ($f_m = 10$, no overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	8.8	13.9	18.6	15.5					
64× 16	9.3	16.5	26.3	35.6	30.0				
128× 32	9.7	18.1	32.1	51.2	69.7	59.1			
256× 64	9.8	19.0	35.7	63.2	101.1	137.9	117.2		
512×128	9.9	19.5	37.8	70.9	125.5	201.0	274.4	233.5	
1024×256	10.0	19.7	38.9	75.2	141.2	250.2	400.8	547.4	466.1
2048×512	10.0	19.9	39.4	77.6	150.2	281.9	499.6	800.3	1093.4

Table 21: Estimated preconditioner vector product Megaflop rates ($f_m = 10$, overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	4.4	6.9	9.3	10.1					
64× 16	4.7	8.3	13.2	17.9	19.5				
128× 32	4.8	9.1	16.0	25.7	35.0	38.5			
256× 64	4.9	9.5	17.9	31.7	50.7	69.3	76.4		
512×128	5.0	9.8	18.9	35.5	62.9	100.8	137.9	152.2	
1024×256	5.0	9.9	19.4	37.6	70.7	125.3	201.0	275.0	303.7
2048×512	5.0	9.9	19.7	38.8	75.1	141.1	250.2	401.3	549.3

Table 22: Estimated preconditioner vector product Megaflop rates ($f_m = 5$, no overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	4.7	7.8	11.4	13.1					
64× 16	4.8	8.8	14.8	21.7	25.4				
128× 32	4.9	9.4	17.1	28.8	42.5	50.0			
256× 64	5.0	9.7	18.5	33.7	56.8	84.1	99.2		
512×128	5.0	9.8	19.2	36.6	66.9	112.9	167.3	197.6	
1024×256	5.0	9.9	19.6	38.3	73.0	133.3	225.1	333.8	394.4
2048×512	5.0	10.0	19.8	39.1	76.4	145.7	266.2	449.5	666.8

Table 23: Estimated preconditioner vector product Megaflop rates ($f_m = 5$, overlap)

#PE's	1	2	4	8	16	32	64	128	256
32× 8	420.4	539.8	861.3	2033.3					
64× 16	378.5	424.9	534.7	832.1	1952.9				
128× 32	361.9	381.8	425.7	529.3	812.1	1902.2			
256× 64	354.6	363.8	383.1	425.4	525.3	799.3	1871.4		
512×128	351.3	355.7	364.7	383.6	424.9	522.5	791.5	1853.3	
1024×256	349.6	351.8	356.2	365.1	383.8	424.5	520.8	786.9	1842.8
2048×512	348.9	349.9	352.1	356.4	365.3	383.8	424.2	519.8	784.2

Table 24: Predicted total time per cell in μ seconds for the solution of the pressure equation

Parallelism in GMRES applied to the computation of incompressible flows*

R.R.P. van Nooyen, Faculty of Civil Engineering, Delft University of Technology, P.O. Box 5048, NL 2600 GA Delft, The Nerherlands

C. Vuik, P. Wesseling, Faculty of Technical Mathematics and Informatics, Delft University of Technology, P.O. Box 5031, NL 2600 GA Delft, The Netherlands

* This work was sponsored by the Stichting Nationale Computer Faciliteiten (National Computer Facilities Foundation, NCF) for the use of supercomputer facilities with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). The Training and Research in Advanced Computer Systems (TRACS) program of the Edinburgh Parallel Computing Centre contributed supercomputer time and financial support for a 3 month stay in Edinburgh for the first author.