
ELSEVIER

Parallel Computing 24 (1998) 1927–1946

PARALLEL COMPUTING

Practical aspects

# Parallelism in ILU-preconditioned GMRES

C. Vuik [a,*], R.R.P. van Nooyen [b], P. Wesseling [a]

[a] *Technische Wiskunde & Inform. Technische Universiteit Delft, P.O. Box 5031, 2600 GA Delft, The Netherlands*

[b] *Faculty of Civil Engineering, Delft University of Technology, P.O. Box 5048, NL 2600 GA Delft, The Netherlands*

Received 15 January 1997; received in revised form 10 March 1998

**Abstract**

A parallel implementation of the preconditioned GMRES method is described. The method is used 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 (P. Bastian, G. Horton, SIAM. J. Stat. Comput. 12 (1991) 1457–1470). Contrary to some 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 incompressible flow. Predictions of computing time show good correspondence with measurements. 

*Keywords:* Parallel ILU preconditioner; Distributed memory computer; Parallel performance model; Computational fluid dynamics

## 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. Some aspects

* Corresponding author. E-mail: c.vuik@math.tudelft.nl

0167-8191/98/$ – see front matter 
PII: S0167-8191(98)00084-2

of the discretization are presented in [29,30]. The treatment of turbulence with our discretization is analysed in [44]. Research into (coarse grain) parallelism for our code through domain decomposition techniques is described in [8]. In this paper another (fine grain) parallelization is used for a problem without domain decomposition. It seems a good idea to combine both techniques when a non-uniform memory access (NUMA) computer is used ([20]).

Our solver uses the pressure correction method, which means that an intermediate velocity field is determined using an estimate for the pressure. Thereafter a correction is calculated to obtain a velocity field with zero divergence. The intermediate velocity field is obtained by applying an iterative solver to the Newton linearization of the discretized momentum equations. The pressure correction is obtained from a discrete pressure equation.

A preliminary analysis showed that matrix construction is embarrassingly parallel [35], 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 equation are robust and fast solvers [37–39,43,7]. Therefore we concentrate on parallelizable variants of these methods. The convergence behaviour of the parallel algorithm is the same as that of the serial algorithm.

For later reference we include a short description of the GMRES(m) method as given in [26], 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:

$$h_{i,j} = (M_1 A M_2 v_j, v_i), \quad i = 1, 2, \ldots, j,$$

$$\hat{v}_{j+1} = M_1 A M_2 v_j - \sum_{i=1}^{j} h_{i,j} v_i,$$

$$h_{j+1,j} = \|\hat{v}_{j+1}\|,$$

$$v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$$

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 (CGS) orthogonalization method.

Preconditioned Krylov subspace methods are very popular to solve large algebraic systems of linear equations. Many practical preconditioners are based on Incomplete LU decompositions. Such a preconditioner is first described in [22]. Later

on various alternatives have been formulated, such as MILU [18], RILU [2], and ILUM [25]. Recently, a number of preconditioners have been proposed for the discretized Poisson equation, where the rate of convergence does not depend on the grid size. Examples are: NGILU [31] and DRIC [23]. A comparison of these and related preconditioners is given in [7].

Parallel implementations of Krylov subspace methods are given in [28,27,11,9], and [20]. The large number of inner products used in the GMRES method can be a drawback on distributed memory computers, because an inner product needs global communication. In general the preconditioner is the most difficult part to parallelize. In many references a slightly adapted ILU decomposition is considered, which has better parallel properties. To illustrate such an approach we consider a domain decomposed into a number of blocks. Each block is distributed to a different processor. To parallelize the ILU preconditioner the couplings between the blocks are deleted. Such an approach is considered in: [27,28,24,40,10,4,9], and [20]. For most of these preconditioners the rate of convergence deteriorates when the number of blocks (processors) increases. When overlapping blocks are used the convergence behaviour depends only slightly on the number of blocks. However, overlapping costs extra work. Other parallel preconditioners are based on a renumbering of the unknowns such as: the nested twisted approach [33] and red black orderings [14,12,15], and [9]. Recently, Sparse Approximate Inverses [16] and multigrid [41] have also been used as parallel preconditioners.

In our ILU preconditioner the matrices $L$ and $U$ satisfy the following rules:

- $diag\ (L) = I$;
- the nonzero structure of the matrix $L + U$ is identical to the nonzero structure of $A$;
- if $A_{ij} \neq 0$ then $(LU)_{ij} = A_{ij}$.

For more details see Section 3 and [39]. Our choice to parallelize this preconditioner is based on the following considerations:

- From [37] and [43] it appears that for our problems the ILU based preconditioners are competitive with SPAI and multigrid,
- The nested twisted approach is only useful for a small number of processors,
- In general the red black orderings and the block ILU preconditioners without overlap lead to a worse convergence behaviour,
- It is hard to adapt our code to solve the Navier–Stokes equations such that overlapping block ILU preconditioners can be used.

To understand the parallel properties of the various methods, models are used to analyse the measured CPU or wall-clock times. Various models are given in [19,20], and [13]. In [11,3,20], and [13] these models are used to understand the properties of parallel Krylov methods.

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 for a test problem on a Cray T3D are reported and analysed in Section 5.

## 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 Section 2.1 we give the data distribution of our problem. We skip the description of a parallel vector update and the matrix vector product, because they are easily parallelizable. The parallelization of the inner products is described in Section 2.2, together with a discussion of various Gram–Schmidt orthogonalization methods. Finally, in Section 2.3 we discuss the difficulties associated with 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 it is important to keep information in local storage as much as possible. For this reason we assign storage space and update tasks as follows. The domain is subdivided into a regular grid of rectangular blocks. The subdivision follows the cell edges of the space discretization grid. Each processor is responsible for all updates of variables associated with the grid cells in its block.

We use the following convention to assign the fluxes, which are given on cell edges for a staggered discretization, to cells: the flux on the lower and the left-hand cell edge belongs to the given cell (Fig. 1). Furthermore, a block contains all fluxes on the lower and the left-hand boundary and all fluxes on the intersections of an outer boundary with a block boundary. Fluxes on an interior upper or right-hand block boundary are assigned to the processor that is responsible for the block adjacent to that boundary (Fig. 2). Two extra rows of cells are added on the lower and left boundary of a block and three extra rows of cells are added to the upper and right boundary of a block to provide storage space for variables used in matrix-vector products and preconditioner construction (Fig. 3). Note that for small blocks the addition of the extra rows can result in a considerable increase in problem size. However, for large blocks the increase is negligible.

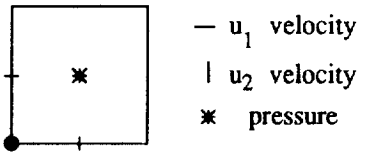


Fig. 1. Assignment of unknowns to a cell using a staggered grid.

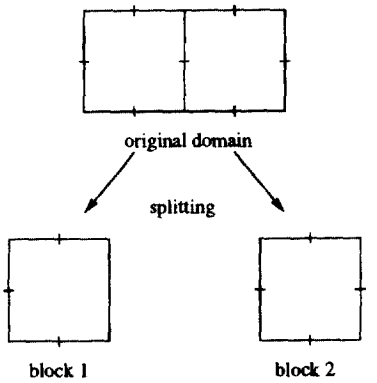


Fig. 2. Decomposition in blocks.

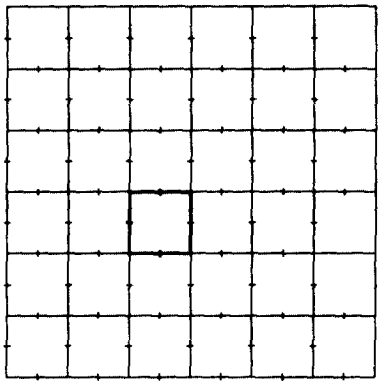

Fig. 3. Block 1 with auxiliary cells.

### 2.2. *Inner products and Gram–Schmidt orthogonalization*

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 are summed to form the full inner product. To obtain the full inner product global communication is required. We describe two possible communication strategies.

*Inner product I*: We assume each processor to be a leaf of a binary tree (Fig. 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.

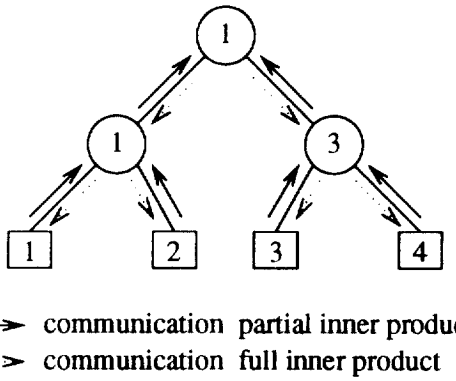


Fig. 4. Communication pattern for inner product I.

*Inner product II*: To explain the second communication strategy we consider a small 1D torus (Fig. 5). In 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 two 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. It appears that the speed of inner product II is comparable to the fastest Cray native inner product. Therefore, and also because it is easier to analyse, we will use inner product II. Note that inner product II can also be used on machines without a native inner product.

Communication time is 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.

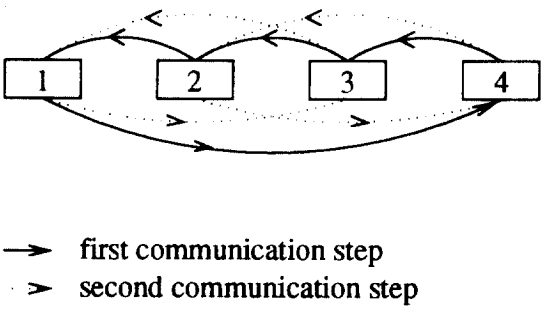


Fig. 5. Communication pattern for inner product II.

Using the 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 [6]. Therefore, the Modified Gram–Schmidt (MGS) method is preferred, which is stable with respect to rounding errors [6]. However, the inner products are calculated sequentially when MGS is used in the original GMRES method (see Section 1). So clustering of the inner product communications is impossible. Since for the Cray T3D, the latency is relatively small, we use in our T3D specific code the MGS method for stability reasons. On a computer with a relative large latency, it is better to use an adapted GMRES method ([11,3]) where a parallel (clustered) variant of the MGS method can be used.

### *2.3. 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, due to inherent recursiveness of the obvious algorithms. 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 [5].

## 3. Parallel ILU-preconditioning

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 $U$, which are used in the ILU-preconditioner. 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 matrix $P$ that is nonsymmetric if the underlying boundary-fitted coordinate system is non-orthogonal. For details see [29] and [38]. The discretization stencil of the pressure equation consists of 9 points. Due to the structured grid approach the matrix $P$ has only nine non-zero diagonals [38]. We use the same preconditioner (RILU(0.975)) as in [39]. Our parallelization only depends on the non-zero structure of $L + U$, so it can also be combined with other preconditioners (as mentioned in the introduction), provided that the non-zero structure is the same. Note that for the RILU-preconditioner the non-zero structure of $L + U$ is identical to the non-zero structure of $P$. Fig. 6 displays the stencils of $P$, $L$, and $U$.

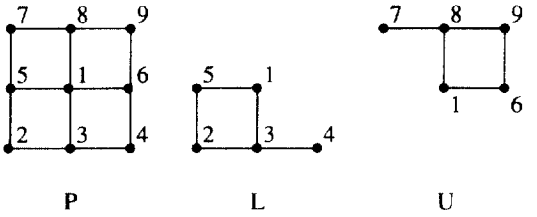


Fig. 6. The stencils of $P$, $L$, and $U$.

*Momentum equations*: 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 nine non-zero diagonals and their non-zero structure is the same as that of $P$. Both $M_{21}$ and $M_{12}$ contain four non-zero diagonals. As preconditioner we use RILUD_2($\alpha$) with $\alpha = 0.95$ [39]. Here the off-diagonal parts of $M$ are the same as those 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 Eq. (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) and (3), and that encountered in the solution algorithm of the pressure equation. For this reason, we restrict ourselves in the next section to the parallelization of the pressure preconditioner.

### 3.2. Staircase parallelization of the pressure preconditioner

Parallelization of the construction of $L$, $U$, and the solution of the triangular systems is comparable. So we only consider the parallel implementation of the solution of $Lx = b$. The algorithm is first explained for a matrix originating from a 5-point stencil. Then it is adapted for matrices based on a 9-point stencil.

The ideas for the staircase parallelization come from [5]. In [27] a comparable parallelization is given, where the solution of a lower triangular system is done with a block wave-front forward sweep. To avoid too many communication start ups square blocks are used. A related approach is to calculate the unknowns on a diagonal of the grid. For a 5-point stencil it is easy to see that these unknowns only depend on the unknowns corresponding to the previous diagonal. Therefore all components of $x$ corresponding to the same diagonal can be computed indepen-

dently. This technique is used for vectorization ([1] and [39]) and parallelization ([32] and [13]). A drawback of these alternatives is that on average only one half of the processors are active simultaneously.

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 $x_d$-direction is denoted by $n_d$. 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 $S_k = \{(i,j) \mid i \in [(k-1)\cdot n_x + 1, k \cdot n_x], j \in [1, n_y]\}$.

*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 (Fig. 7).

*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. Then 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 one extra communication is necessary for a 9-point stencil.

The torus communication network of the Cray T3D is well adapted for this kind of communication, because only nearest neighbour communication takes place. Per message one real is sent, so this approach is only useful on computers with a low latency. For this reason we use Cray specific shared arrays (as in the inner product) instead of PVM subroutines. The auxiliary cells are used to store information from neighbouring processors (Fig. 3).

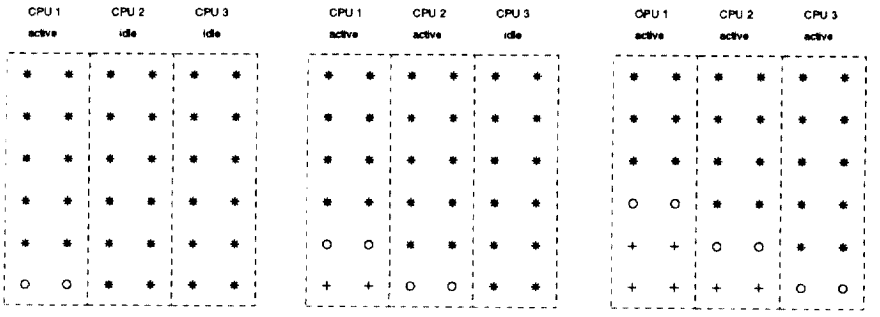


Fig. 7. The first stages of the staircase parallel solution of the lower triangular system $Lx = b$. The symbols denote the following: * nodes to be calculated, o nodes being calculated, and + nodes that have been calculated.

## 4. A performance model for the linear solver

A model is presented 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. The models are based on the theory given in [19,20], and [13].

### 4.1. Preliminaries

Many different aspects of parallel computer architecture play a role in determining the execution speed of a parallel algorithm. We only take into account the number of floating point operations that the processor can realistically be expected to perform per second, the communication latency, and the bandwidth.

We assume that the latency and bandwidth of the 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,

$$R_l = \text{latency in seconds} \times f \text{ and}$$
$$R_b = \frac{f}{\text{floating point numbers transported per second}}.$$

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

In an 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. We assume that 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:

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

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

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

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

$$S_e(p) = S_0(p) \times \frac{1}{1 + \max\{0, R_l R_M(p) + R_b R_L(p) - W_{ov}(p)/W(p)\}}, \tag{7}$$

where $R_M(p) = M(p)/W(p)$ and $R_L(p) = L(p)/W(p)$ are, respectively, the start-up and transmission costs as fractions of the calculation costs.

### 4.2. 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 $n_1^{\text{tot}}(p) \times n_2^{\text{tot}}(p) = n_1\sqrt{p} \times n_2\sqrt{p}$, 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 [17].

#### 4.2.1. 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. 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

$$R_M(p) = \frac{2}{17n_x n_y} = \frac{2}{17n_1 n_2}. \tag{8}$$

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

$$R_L(p) = \frac{2n_y}{17n_x n_y} = \frac{2}{17n_x} = \frac{2\sqrt{p}}{17n_1}. \tag{9}$$

The calculation of the matrix vector product in the $n_x - 2$ interior nodes can be done independently of the communication, so $W_{\text{ov}}(p)$ is given by

$$W_{\text{ov}}(p) = 17(n_x - 2)n_y. \tag{10}$$

After substituting Eqs. (8)–(10) into Eq. (7) and measuring the values of $R_l$ and $R_b$ one can predict the speedup of the matrix vector product.

#### 4.2.2. Inner product

The Cray T3D computer is composed of pipelined RISC processors. 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 [19]. 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 for a vector of length $n$ is

$$f_i(n) = \frac{2n}{t_{is} + \frac{2n}{f_{im}}}. \tag{11}$$

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 5.2.2).

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 the theoretical speedup is

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

The communication strategy of inner product II is used (see Section 2). 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

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

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

*4.2.3. The parallel ILU-preconditioner*

We also 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 nine flops 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 $9(p-1)n_y \simeq 9pn_x$ 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 other values of $n_2$ the theoretical speedup $S_0(p)$ lies between these bounds.

To account for communication delays we note that two messages (one to the left and one to the right) per horizontal line are communicated 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 one floating point number, we have

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

Only the communication to the left neighbouring processor can be overlapped by calculations.

## 5. Timing experiments and analysis

In Section 5.1 time measurements of our CFD code running on a Cray T3D are given. These results are compared with our predictions in Section 5.2. For other measurements on the Cray T3D we refer to [21,42], and [34].

### 5.1. Experiments with the parallel code

Our test problem is the curved channel problem as described in [39]. The solution process was artificially interrupted after 19 iterations of full GMRES for the momentum equations and 26 iterations for the pressure equation to allow for direct comparison of timings on different sized grids. The measurements were done at the Cray T3D computer at the Edinburgh Parallel Computing Center. A restricted number of results are given to illustrate the parallel properties of our ILU-preconditioned GMRES method and to compare our model for wall clock time prediction with experiments. Other measurements are given in [36].

In Table 1 we present the wall clock time to solve the momentum equations. The results are only presented in a certain band. To explain this, consider the $32 \times 8$ grid. For the parallelization the computational domain is split into strips parallel to the $x_2$-axis. This means that if eight processors are used the grid on one processor consists of $4 \times 8$ grid points. It is impossible to reduce this further, so no measurements are done for the $32 \times 8$ grid using a larger number of processors. On the other hand using one processor one cannot solve grid sizes larger than $256 \times 64$ due to memory limitations. With respect to the solution of the system we see that the wall clock time for two processors is approximately the same as for one processor. We cannot explain this phenomenon.

To investigate scalability of our approach we show the total time used per variable for the momentum equations in Fig. 8 (matrix construction) and Fig. 9 (system solution). 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. In these figures the grid size is given by $4n_2 \cdot \sqrt{p/2} \times n_2 \cdot \sqrt{p/2}$, where $p$ is the number of processors. A constant amount of time per variable means that the considered algorithm is scalable, with isoefficiency function equal to $Kp$ ([20], Section 4.4.3). It appears that both algorithms are scalable

Table 1
Measured momentum solution times in msec

| $p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| $32 \times 8$ | 123 | 147 | 115 | 108 | | | | | |
| $64 \times 16$ | 432 | 453 | 291 | 204 | 180 | | | | |
| $128 \times 32$ | 1609 | 1643 | 906 | 536 | 373 | 337 | | | |
| $256 \times 64$ | 6200 | 6354 | 3404 | 1812 | 1064 | 835 | 651 | | |
| $512 \times 128$ | | 24526 | 12312 | 6706 | 3656 | 2224 | 1503 | 1303 | |
| $1024 \times 256$ | | | | 25475 | 13311 | 7297 | 4435 | 3023 | 2592 |
| $2048 \times 512$ | | | | | | 26755 | 14756 | 8822 | 6199 |

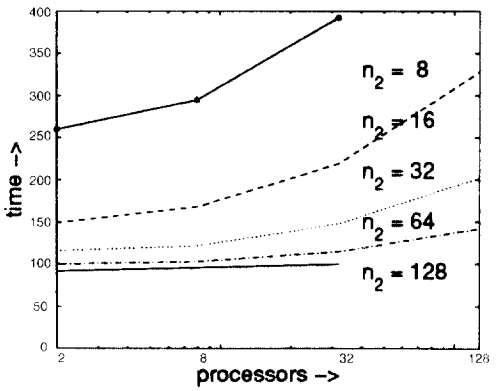


Fig. 8. Measured total time per variable in μs for the matrix construction.

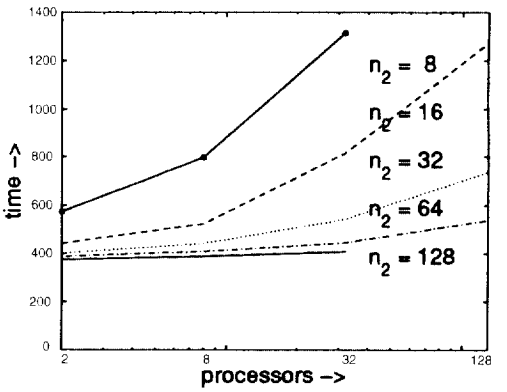


Fig. 9. Measured total time per variable in μs for the solution of the system.

when $n_2$ is large enough. 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. 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 4.2 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.

Table 2 contains the Megaflop rates for the inner product. 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 2), and 12.5 for the solution of the systems. It appears that memory access is the most time consuming part. For the inner

Table 2
Inner product performance in Megaflops per second

| $p$ | Measured | | | Predicted | | |
|---|---|---|---|---|---|---|
| | 1 | 16 | 128 | 1 | 16 | 128 |
| #elem per proc | | | | | | |
| 32 | 10 | 34 | 194 | 5 | 33 | 187 |
| 1024 | 28 | 331 | 2346 | 28 | 365 | 2572 |
| 8192 | 33 | 496 | 3875 | 33 | 505 | 3949 |
| 65536 | 33 | 529 | 4212 | 33 | 531 | 4231 |

product this leads to an expected rate of 4288 Mflop/s on 128 processors. For long vectors the observed rate (see Table 2) is very close to this value. In [21] it is observed that the complex inner product has a performance of 60 Megaflops per second. This is in accordance with our measurements, when we note that in a complex inner product the number of computations per memory access is two times as high than for the real inner product. In [21] a Megaflop rate of 2500 is measured when 128 processors are used. Note that this is much less than the expected value $60 \times 128 = 7680$.

### 5.2. Analysis of the timing experiments

The measurements given in Section 5.1 are analysed using the models specified in Section 4. First some parameters are estimated: start-up time, transmission time and latency. Thereafter the Megaflop rates of the inner product are predicted and compared with the rates observed. Finally we predict the total times per grid point for the solution of the pressure equation and compare them with the observed values.

#### 5.2.1. Parameters of the Cray T3D

Before we give quantitative results it is helpful to consider the T3D machine on a qualitative level. Important characteristics of the T3D machine are the high bandwidth of the inter-processor communication channels, the 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.

The start-up time, transmission time and latency are obtained from the measurements of the inner product. Applying model Eq. (11) to the data we find a maximum flop rate of $f_{im} = 33.4$ Megaflops per second and a start-up time of $t_{is} = 12$ μs. We estimate the communication latency $t_l$ from the time measurements of the inner product with and without communication. 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 16/300 μs per real (300 MB/sec per link per two processors). To eliminate the fixed overhead we consider $t(p) - t(2)$. When we fit the model $t(p) - t(2) = t_l \log_2 p + t_b \log_2 p - t_l - t_b$ to the measurements we find a latency of 4 μs. This value compares well with the values given in [31] p. 725. The Megaflop rates for the inner product predicted with these parameters are given in Table 2. There is a good correspondence with the observed rates.

*5.2.2. 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 time per grid point is then approximately

$$t = k\left(\frac{17}{f_m} + \frac{27}{f_p} + \frac{k}{f_i} + \frac{k}{f_u}\right). \tag{12}$$

When $k$ is very large it appears from Eq. (12) 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 Eq. (12) are based on Eq. (11). From the experiments we observe that the Megaflop rates on two processors are approximately 50% of the megaflop rates on one processor. Therefore we use the following formulae:

$$f_m = \frac{17(n_x-2)}{14 + \frac{17(n_x-2)}{5}},\quad f_p = \frac{9(n_x-2)}{14 + \frac{9(n_x-2)}{5}},\quad f_i = \frac{2n_xn_y}{12 + \frac{2n_xn_y}{16}},\quad f_u = \frac{2n_xn_y}{12 + \frac{2n_xn_y}{10}}.$$

Note that the maximum flop rate for the vector update is 2/3 of that for the inner product, due to one extra memory access per element. The maximum flop rates for the matrix vector and back substitution are obtained from [36]. The Megaflop rates are multiplied by the respective efficiencies ($E_e(p) = S_e(p)/p$, [20] p. 120) and substituted into Eq. (12) to predict the total time per grid cell (Table 3). There is a good correspondence between the predictions and the measurements (Table 4). 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. From Fig. 9 we conclude that the time per unknown is constant if $n_2$ is large enough. We see that the time per unknown is constant (or decreases for increasing number of processors) on the diagonals of Table 4 (except column 1). This implies that for all choices of $n_2$, the isoefficiency function is $Kp^2$.

In Figs. 10 and 11 we present the percentage of the total time for the various parts of GMRES. Fig. 10 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

Table 3
Predicted total time per cell in μs for the solution of the pressure equation

| $p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| 32 × 8 | 435 | 568 | 919 | 2151 | | | | | |
| 64 × 16 | 382 | 432 | 549 | 861 | 2011 | | | | |
| 128 × 32 | 363 | 384 | 429 | 537 | 826 | 1931 | | | |
| 256 × 64 | 355 | 364 | 384 | 427 | 529 | 806 | 1886 | | |
| 512 × 128 | | 356 | 365 | 384 | 426 | 524 | 795 | 1860 | |
| 1024 × 256 | | | | 365 | 384 | 425 | 522 | 789 | 1846 |
| 2048 × 512 | | | | | | 384 | 425 | 520 | 785 |

Table 4
Measured total time per cell in μs for the solution of the pressure equation

| $p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|---|
| 32 × 8 | 259 | 645 | 1204 | 2510 | | | | | |
| 64 × 16 | 185 | 402 | 597 | 965 | 1969 | | | | |
| 128 × 32 | 179 | 340 | 395 | 518 | 830 | 1694 | | | |
| 256 × 64 | 163 | 319 | 331 | 380 | 487 | 833 | 1521 | | |
| 512 × 128 | | 306 | 311 | 338 | 373 | 478 | 740 | 1443 | |
| 1024 × 256 | | | | 317 | 335 | 375 | 469 | 731 | 1444 |
| 2048 × 512 | | | | | | 354 | 374 | 492 | 722 |

takes 65% of the time for a small grid size and 45% for a large grid size. In Fig. 11 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.

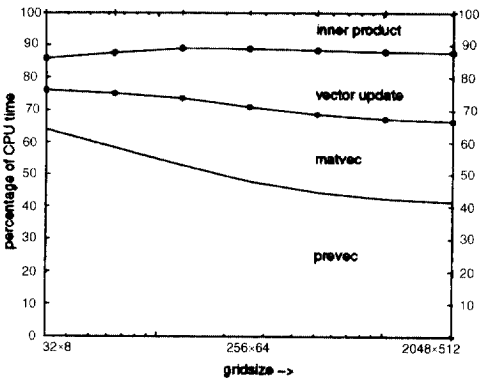


Fig. 10. The percentage of time used by the various parts (eight processors).

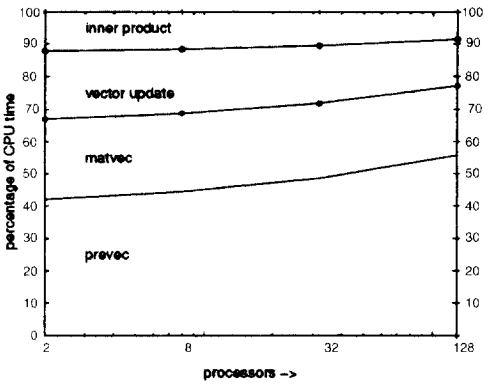


Fig. 11. The percentage of time used by the various parts (grid size $256 \cdot \sqrt{p/2} \times 64 \cdot \sqrt{p/2}$).

## 6. Conclusions

A scalable parallel implementation of the ILU-preconditioned GMRES method is given. 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 and full GMRES is used, 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 [5]. 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. A 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. Finally, we observe a loss of efficiency (50%) when we run the program on two or more processors. We are unable to explain this loss.

## Acknowledgements

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 three month stay in Edinburgh for the second author.

## References


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

[2] O. Axelsson, G. Lindskog, On the eigenvalue distribution of a class of preconditioning methods, Numer. Math. 48 (1986) 479–498.

[3] Z. Bai, D. Hu, L. Reichel, A Newton basis GMRES implementation, IMA J. Num. Anal. 14 (1994) 563–581.

[4] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, Templates for the solution of linear systems: Building blocks for iterative methods, SIAM Philadelphia, 1994.

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

[6] Å. Björck, Solving linear least squares problems by Gram–Schmidt orthogonalization, BIT 7 (1967) 1–21.

[7] E.F.F. Botta, K. Dekker, Y. Notay, A. van der Ploeg, C. Vuik, F.W. Wubs, P.M. de Zeeuw, How fast the Laplace equation was solved in 1995, Appl. Num. Meth. 24 (1997) 439–455.

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

[9] A.M. Bruaset, A survey of preconditioned iterative methods, Pitman Research Notes in Mathematics Series 328, Longman Scientific and Technical, Harlow, 1995.

[10] E. de Sturler, Incomplete Block LU preconditioners on slightly overlapping subdomains for a massively parallel computer, Appl. Num. Math. 19 (1995) 129–146.

[11] E. de Sturler, H.A. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Appl. Num. Math. 18 (1995) 441–459.

[12] S. Doi, On parallelism and convergence of incomplete LU factorizations, Appl. Num. Math. 7 (1991) 417–436.

[13] J.J. Dongarra, L.S. Duff, D.C. Sorensen, H.A. van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM, Philadelphia, 1991.

[14] I.S. Duff, G.A. Meurant, The effect of ordering on preconditioned conjugate gradient, BIT 29 (1989) 635–657.

[15] V. Eijkhout, Analysis of parallel incomplete point factorizations, Lin. Alg. Appl. 154/156 (1991) 723–740.

[16] M.J. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comp. 18 (1997) 838–853.

[17] J. Gustafson, Reevaluating Amdahl's law, Comm ACM 31 (1988) 532–533.

[18] I. Gustafsson, A class of first order factorization methods, BIT 18 (1978) 142–156.

[19] R.W. Hockney, C.R. Jesshope, Parallel Computers 2: architecture programming and algorithms, Adam Hilger, Bristol, 1988.

[20] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to parallel computing; design and analysis of algorithms, Benjamin/Cummings Redwood City, 1994.

[21] P.M. Meijer, S. Poedts, J.P. Goedbloed, A. Jakoby, A parallel semi-implicit method for 3D nonlinear magnetohydrodynamics, in: B. Hertzberger, G. Serazzi (Eds.), High-Performance Computing and Networking, International Conference and Exhibition, Milan, Italy, 3–5 May 1995, Lecture Notes in Computer Science 919, Springer, Berlin, 1995, pp. 170–175.

[22] J.A. Meijerink, H.A. Van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comp. 31 (1977) 148–162.

[23] Y. Notay, DRIC: a dynamic version of the RIC method, J. Num. Lin. Alg. 1 (1994) 511–532.

[24] G. Radicati, Y. Robert, Parallel conjugate gradient-like algorithms for solving sparse non-symmetric linear systems on a vector multiprocessor, Parallel Comp. 11 (1989) 223–239.

[25] Y. Saad, ILUM: a multi-elimination ILU preconditioner for general sparse matrices, SIAM J. Sci. Comput. 17 (1996) 830–847.

[26] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.
[27] Y. Saad, M.H. Schultz, Parallel implementation of preconditioned conjugate gradient methods, in: W.E. Fitzgibbon (Ed.), Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling, 1986, pp. 108–127, SIAM, Philadelphia.
[28] M.K. Seager, Parallelizing conjugate gradient for the cray X-MP, Paral. Comp. 3 (1986) 35–47.
[29] A. Segal, P. Wesseling, J. van Kan, C.W. Oosterlee, K. Kassels, Invariant discretization of the incompressible Navier–Stokes equations in boundary fitted co-ordinates, Int. J. Num. Meth. Fluids 15 (1992) 411–426.
[30] P. van Beek, R.R.P. van Nooyen, P. Wesseling, Accurate discretization on non-uniform curvilinear staggered grids, J. Comp. Phys. 117 (1995) 364–367.
[31] A. van der Ploeg, E.F.F. Botta, F.W. Wubs, Nested grids ILU-decomposition (NGILU), J. Comp. Appl. Math. 66 (1956) 515–526.
[32] A. van der Ploeg, R. Keppens, G. Tóth, Block incomplete LU-preconditioners for implicit solution of advection dominated problems, in: B. Hertzberger, P. Sloot (Ed.), High-Performance Computing and Networking, International Conference and Exhibition, Vienna, Austria, 28–30 April 1997, Lecture Notes in Computer Science 1225, Springer, Berlin, 1997, pp. 421–430.
[33] H.A. van der Vorst, Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, Par. Comp. 5 (1987) 45–54.
[34] M. van Gijzen, Parallel iterative solution methods for linear finite element computations on the CRAY T3D, in: B. Hertzberger, G. Serazzi (Eds.), High-Performance Computing and Networking, International Conference and Exhibition, Milan, Italy, 3–5 May 1995, Lecture Notes in Computer Science 919, 1995, Springer, Berlin, pp. 723–728.
[35] 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.
[36] R.R.P. van Nooyen, C. Vuik, P. Wesseling, Parallelism in GMRES applied to the computation of incompressible flows, Report 96-109, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1996.
[37] R.R.P. van Nooyen, C. Vuik, P. Wesseling, Some parallelizable preconditioners for GMRES, Report 96-50, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1996.
[38] C. Vuik, Solution of the discretized incompressible Navier-Stokes equations with the GMRES method, Int. J. for Num. Meth. Fluids 16 (1993) 507–523.
[39] C. Vuik, Fast iterative solvers for the discretized incompressible Navier–Stokes equations, Int. J. for Num. Meth. Fluids 22 (1996) 195–210.
[40] T. Washio, K. Hayami, Parallel block preconditioned based on SSOR and MILU, Num. Lin. Alg. Appl. 1 (1994) 533–553.
[41] T. Washio, K. Oosterlee, Experiences with robust parallel multilevel preconditioners for BiCGSTAB, Technical Report 949, Arbeitspapiere der GMD, Sankt Augustin, 1995.
[42] F. Willien, E. Piault, F.-X. Roux, Parallel preconditioners on MIMD computers applied to petroleum industry, in: B. Hertzberger, G. Serazzi (Ed.), High-Performance Computing and Networking, International Conference and Exhibition, Milan, Italy, 3–5 May 1995, Lecture Notes in Computer Science 919, Springer, Berlin, 1995, pp. 287–292.
[43] S. Zeng, C. Vuik, P. Wesseling, Numerical solution of the incompressible Navier–Stokes equations by Krylov subspace and multigrid methods, Adv. Comp. Math. 4 (1995) 27–49.
[44] M. Zijlema, A. Segal, 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 (1995) 209–225.