Computer Assisted Mechanics and Engineering Sciences, 1: 147–157, 1994.

Parallelization of the compact methods for the Navier-Stokes equations

Jacek Rokicki
Institute of Aeronautics and Applied Mechanics, Warsaw University of Technology, Nowowiejska 22/24, 00-665 Warsaw, Poland

Jerzy M. Floryan
Department of Mechanical Engineering, The University of Western Ontario, London, Ontario, N6A 5B9, Canada

(Received August 4, 1994)

The proposed algorithm is based on the fourth-order compact discretization schemes for the Navier-Stokes equations in streamfunction-vorticity-pressure formulation. The equations are expressed in terms of a general orthogonal curvilinear coordinate system which allows for modelling non-standard geometries. Two distinct parallelization strategies are considered. The first one relies on the domain decomposition approach, in which each subdomain is served by a different processor. In the second strategy, suitable for massively parallel computers, each processor serves a single grid point. The comparison of the performance of various computing platforms is presented, including a 2048-processor MasPar computer.

1. Introduction

Numerical solutions of the Navier-Stokes equations are computationally very demanding. The speed of the available computing hardware is insufficient for the analysis of many problems of practical interest. This motivates the search for more effective algorithms and for computing architectures that are capable of processing these algorithms at an even higher rate.

A classical line of development is to accelerate sequential processors. Tremendous progress has been achieved in this area with RISC concepts. A much higher acceleration rate can be obtained with the use of several processors working in parallel, with each processor serving a different part of the solution domain (or a different segment of the solution matrix). This concept is very attractive because there is no upper limit, at least theoretically, on the available processing speed. On the other hand, not all existing algorithms can be efficiently implemented in parallel.

The range of hardware that is presently available extends from single processor computers through architectures involving several processors (multiprocessor workstations, single processor workstation clusters) up to massively parallel computers. While the latter have a long term advantage, it is not clear which of the above configurations are capable of delivering the best processing speed at present as well as in the near future. One of the objectives of this work is to compare the performance of the computers on both extremes of the above spectrum, i.e. single processor machines versus massively parallel machines.

The accurate solutions of the Navier-Stokes equations require discretization schemes of near spectral or spectral accuracy. We present here our own algorithm that is based on a fourth-order (near spectral) compact finite-difference discretization [5]. This algorithm has been tested extensively and proven to deliver the theoretically predicted accuracy. The second reason for the selection of this algorithm is that it has been implemented with both single domain as well as domain decomposition techniques. Extensive testing for the case of multidomain implementation has shown that the time required for evaluation of a solution to a flow problem is reduced significantly by serving each subdomain with a different processor and, as a matter of fact, one can achieve an acceleration that approaches the theoretically possible maximum [4]. The third reason for the selection of this algorithm is that it is very simple and can thus be implemented on massively parallel computers in a straightforward manner.

The reader may note that compact methods are particularly suitable for parallelization because interprocessor communication is minimized. Interprocessor communication is further reduced by the fact that higher-order methods require sparser grids and fewer iterations to obtain prescribed accuracy in comparison with the classical second-order methods [1, 4]. Despite the fact that the number of arithmetic operations per grid point for a single iteration is much larger for the high-order methods, the total number of arithmetical operations necessary to obtain the prescribed accuracy of the solution remains much lower. This fact has been confirmed in numerical experiments [4].

The authors believe that the presented approach, in which a high-order algorithm (on each subdomain) is coupled with the domain decomposition method, can be further generalized to allow for the efficient solution of Navier-Stokes equations in complicated domains with moving or free boundaries.

2. Flow problem

The Navier-Stokes equations written in the streamfunction-vorticity ($\psi - \zeta$) formulation for a plane, steady, incompressible, two-dimensional flow have the form:

$$\Delta\zeta - Re(\mathbf{V}\cdot\nabla\zeta - \operatorname{curl}\mathbf{f}_e) = 0\,, \tag{1}$$

$$\Delta\psi = -\zeta\,, \tag{2}$$

$$\mathbf{V} = \mathbf{curl}\,\psi\,, \tag{3}$$

with ψ, $\frac{\partial\psi}{\partial n}$ known at the boundary. In the above, $\mathbf{f}_e$ is the external body force, $\mathbf{V}$ is the velocity vector, $\frac{\partial}{\partial n}$ stands for the derivative normal to the boundary and Re denotes the Reynolds number. Both the above field equations (1), (2), when expressed in terms of a curvilinear, orthogonal reference system ($\xi = f(x,y)$, $\eta = g(x,y)$), take the generic form

$$a\Phi_{\xi\xi} + b\Phi_{\eta\eta} - \tilde{q}\Phi_{\xi} - \tilde{s}\Phi_{\eta} = R \tag{4}$$

where $a(\xi,\eta) = {f_x}^2 + {f_y}^2$, $b(\xi,\eta) = {g_x}^2 + {g_y}^2$, $\tilde{q}(\xi,\eta) = q_* - (f_{xx} + f_{yy})$, $\tilde{s}(\xi,\eta) = s_* - (g_{xx} + g_{yy})$ and the subscripts denote the respective derivatives ((x,y) are the cartesian coordinates of the physical domain). In the case of the vorticity transport equation, one should substitute $\Phi = \zeta$, $R = -Re \cdot \operatorname{curl}\mathbf{f}_e$, $q_* = Re \cdot \psi_\eta(f_x g_y - f_y g_x)$, $s_* = -Re \cdot \psi_\xi(f_x g_y - f_y g_x)$. In the case of the equation describing streamfunction, $\Phi = \psi$, $R = -\zeta$, $q_* = s_* \equiv 0$ should be substituted into Eq. (4). It is assumed that functions f and g, responsible for the mapping, are known together with their respective derivatives.

The fourth-order compact discretization scheme for the generic equation (4), introduced by Rokicki and Floryan [5], has a functional form similar to that obtained by Dennis and Hudson [2] for a cartesian reference system, i.e.

$$-d_0\Phi_0 + (d_1\Phi_1 + \cdots + d_8\Phi_8) + B_0 = 0\,. \tag{5}$$

It has been derived by an analogous procedure (lower numerical subscripts refer to nine points of the computational molecule shown in Fig. 1). It has been shown [5] that a further generalization of the algorithm to nonorthogonal coordinate systems is not possible on the basis of 9-point computational molecules without a loss of accuracy.

The actual functional dependence of $B_0, d_0, \ldots, d_9$ on a, b, $\tilde{q}$, $\tilde{s}$, R and h can be found in the Appendix (h denotes the grid step size in the computational plane). The system of linear equations resulting from (5) is solved by the simple Gauss-Seidel relaxation procedure with the nonlinear terms being updated after every few iteration cycles [4].

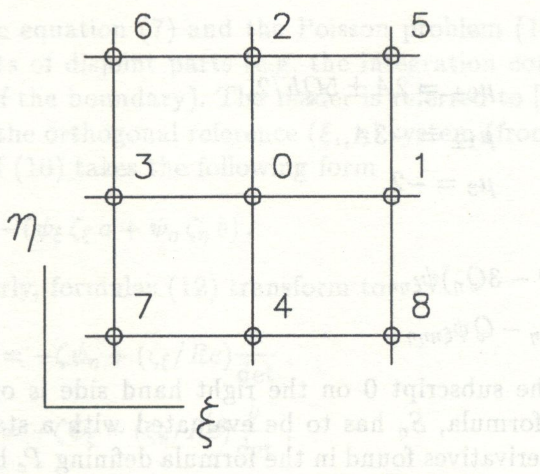

Fig. 1. Sketch of a typical computational molecule

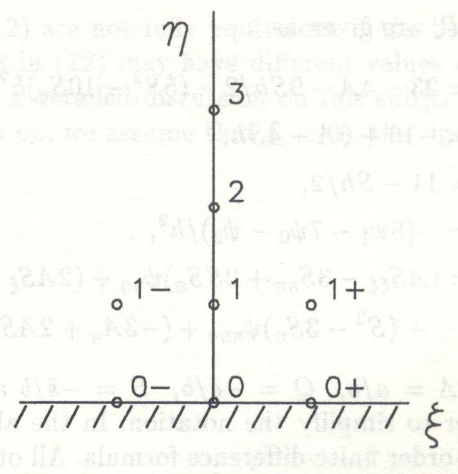

Fig. 2. Sketch of a typical computational molecule on the boundary of the solution domain

The solution algorithm considered here has a classical structure. Its main steps are listed below in order to simplify discussion in the remainder of the paper. These steps are as follows (regardless of the discretization method and the type of coordinates):

1. Initialize the vorticity field ζ in the flow region Ω and at the boundary Γ.

2. Solve Eq. (2) for the streamfunction with the Dirichlet boundary conditions.

3. Calculate the first derivatives of the streamfunction (i.e. velocity) to update the coefficients $\tilde{q}$ and $\tilde{s}$ in (4).

4. Correct the boundary value of vorticity in order to satisfy the Neumann boundary condition for streamfunction.

5. Solve the linear PDE for vorticity (1), (4) with the Dirichlet boundary conditions determined in step 4.

6. Check the convergence of vorticity ζ and streamfunction ψ. If there is no convergence proceed to step 2.

In actual calculations, only a few iterations of the discretized vorticity and streamfunction equations are to be carried out at each step of the external iteration procedure described above. If these equations were solved exactly at each step, the general algorithm would become very unstable and significant underrelaxation would be required for the vorticity boundary formula (step 5).

In order to impose the boundary condition for the normal derivative of ψ (step 4), an algebraic boundary formula for vorticity has to be employed. Contrary to the statement of Gupta [3], low order formulas, e.g. of second- or first-order accuracy, cannot be used without negatively affecting overall accuracy as shown in [4]. The fourth-order implicit boundary formula for vorticity has been developed for cartesian reference system by Rokicki and Floryan in [4]. Generalization of this formula to the curvilinear coordinate system is given in [5]. The description is limited here to the case of an impermeable, non-slip boundary corresponding to a line η = const, with the computational domain located above ($\lambda = 1$) or below ($\lambda = -1$) this line (see Fig. 2). In this case, the formula can be conveniently written with $Z = \zeta/b$ instead of ζ, which results in

$$\mu_{0+}Z_{0+} + \mu_0 Z_0 + \mu_{0-}Z_{0-} = \mu_1 Z_1 + \mu_2 Z_2 + \mu_3 Z_3 + \mu_{1+}Z_{1+} + \mu_{1-}Z_{1-} + 15P_* - 3\lambda P_c h^3 \quad (6)$$

where Z_{0+}, Z_0, Z_{0-}, which are related to the vorticity at the boundary, are considered unknown (lower subscript refers to the points in the computational molecule shown in Fig. 2). The coefficients

μ and P_* are given as

$$\mu_0 = 23 - 4A - 9Sh/2 - (5S^2 - 10S_\eta)h^2, \qquad \mu_{0\pm} = 2A \pm 5Qh/2,$$
$$\mu_1 = -16 + 6A - 4Sh, \qquad \mu_{1\pm} = -3A,$$
$$\mu_2 = 11 - Sh/2, \qquad \mu_3 = -2,$$
$$P_* = -(8\psi_1 - 7\psi_0 - \psi_2)/h^2,$$
$$P_c = (AS_{\xi\xi} - 3S_{\eta\eta} + 2SS_\eta)\psi_{\eta\eta} + (2AS_\xi + SQ - 3Q_\eta)\psi_{\xi\eta\eta} + (S^2 - 3S_\eta)\psi_{\eta\eta\eta} + (-3A_\eta + 2AS)\psi_{\xi\xi\eta\eta} - Q\psi_{\xi\eta\eta\eta},$$

where $A = a/b$, $Q = -\tilde{q}/b$, $S = -\tilde{s}/b$ and the subscript 0 on the right hand side is omitted in order to simplify the notation. In the above formula, S_η has to be evaluated with a standard second-order finite-difference formula. All other derivatives found in the formula defining P_c have to be calculated with first-order accuracy. In the special case of a rectangular grid ($A = 1$, $Q = S = 0$) the formulas above are reduced to those derived in [4].

Equation (6) written for each grid point along the boundary (except corners) results in a system of linear equations with a tridiagonal matrix of coefficients. This matrix remains diagonally dominant if (i) $23/8 > a/b$ and (ii) the step size h is sufficiently small. The numerical cost of obtaining the solution is negligibly small in comparison with the cost of a single iteration of the discretized field equations.

3. Pressure problem

Let us suppose now that the streamfunction ψ and the vorticity ζ form an exact solution of the flow problem (1–3). The corresponding pressure field p can be recovered from the momentum equation. After a suitable transformation (see [4, 7]), this momentum equation takes the following simple form

$$\nabla w = \mathbf{G} \tag{7}$$

where

$$w = \frac{|\mathbf{curl}\,\psi|^2}{2} + p, \tag{8}$$

$$\mathbf{G} = \zeta\,\nabla\psi - \frac{\mathbf{curl}\,\zeta}{Re} + \mathbf{f}_e \tag{9}$$

and **curl** denotes the vector operator introduced in Section 2. The new variable w defined above can be interpreted as the total pressure (for simplicity we set density $\rho = 1$). Equation (7) has a unique (within a constant) solution [4] provided ψ and ζ form a true solution of (1–3).

Since it is more convenient to work with the second- rather than the first-order PDE, we take the divergence of (7) and obtain the Poisson equation for the total pressure w

$$\Delta w = \zeta^2 - \nabla\psi \cdot \nabla\zeta + \operatorname{div}\mathbf{f}_e \tag{10}$$

where $\cdot$ denotes the scalar product. The Neumann and Dirichlet boundary condition for w can either be obtained by projecting (7) onto the normal unit vector $\mathbf{n}$ at the boundary Γ, or by projecting it onto the tangent unit vector $\boldsymbol{\tau}$ and integrating along the boundary, i.e.

$$\left.\frac{\partial w}{\partial \mathbf{n}}\right|_\Gamma = \mathbf{G}\cdot\mathbf{n}, \tag{11}$$

$$\left.\frac{\partial w}{\partial \tau}\right|_\Gamma = \mathbf{G}\cdot\boldsymbol{\tau}, \qquad w|_\Gamma = \int \frac{\partial w}{\partial \tau}\,dt = \int \mathbf{G}\cdot\boldsymbol{\tau}\,dt, \qquad t \in \Gamma. \tag{12}$$

The equation (7) and the Poisson problem (10), (12) are not fully equivalent if the boundary consists of disjoint parts (e.g. the integration constant in (12) may have different values on each part of the boundary). The reader is referred to [4] for a detailed discussion on this subject.

In the orthogonal reference $(\xi,\ \eta)$ system (from now on, we assume that $\mathbf{f}_e \equiv 0$), the right hand side of (10) takes the following form

$$\zeta^2 - (\psi_\xi\, \zeta_\xi\, a + \psi_\eta\, \zeta_\eta\, b). \tag{13}$$

Similarly, formulas (12) transform to

$$w_\eta = -\zeta\psi_\eta + (\zeta_\xi/Re)\frac{a}{\det}\,, \tag{14}$$

$$w_\xi = -\zeta\psi_\xi - (\zeta_\eta/Re)\frac{b}{\det}\,, \tag{15}$$

where a and b are defined as in (4) and $\det = f_x g_y - f_y g_x \neq 0$.

The Poisson equation (10) is discretized with the finite-difference scheme described in Section 2, where, in Eq. (4), Φ is interpreted as w, R denotes the right hand side of (10), i.e. (13) and $q_* = s_* \equiv 0$. The main numerical difficulty in implementing the proposed algorithm consists in calculating the first derivatives of vorticity in (15) and (14) with fourth-order accuracy. This can be accomplished by means of the standard second-order formulas supplemented by the vorticity transport equation (1) to eliminate higher-order terms (see [5]).

In the above procedure, the Dirichlet boundary conditions are used because they are easier to implement in the higher-order schemes studied here.

4. Numerical testing

We have employed two different methods to verify the accuracy of the proposed algorithm. In the first method, an artificial exact solution is compared with a corresponding numerical solution [4]. In the second method, accuracy is estimated by repeating calculations on a sequence of grids (with a diminishing grid size). The second approach is presented here. The flow between eccentric cylinders is considered in the test problem. The inner cylinder (of radius 1) rotates with the unit angular speed, the outer cylinder (of radius 2) remains at rest.

The typical vorticity pattern is presented in Fig. 3 and the estimated relative error of vorticity is displayed in Fig. 4 for a wide range of Reynolds numbers (the Reynolds number is defined here as reciprocal of the kinematic viscosity). The results shown in Fig. 4 confirm that the finite-difference algorithm introduced here is fourth-order accurate.

For this test problem, the periodicity of the pressure field had to be imposed by modifying the boundary value of the streamfunction, which makes the solution algorithm slightly more complicated. The constant value of the streamfunction is known only on one of the cylinders (e.g. on the inner one $\psi = 0$). On the other one, i.e. on the outer cylinder, the streamfunction is constant $\psi = \psi_2$, but the value of ψ_2 is not known in advance (ψ_2 describes the net mass flow between cylinders). However, for any fixed ψ_2 it is possible to obtain the flow solution (using the algorithm described in previous Sections) and, afterwards, to obtain the pressure field as described in Section 3.

The streamfunction and vorticity fields are periodic with respect to ξ but the pressure is not periodic in general and has a jump d_{pres}

$$d_{\text{pres}} = \int_\gamma \mathbf{G}\cdot\boldsymbol{\tau}\, dt, \qquad t \in \gamma, \tag{16}$$

where γ denotes an arbitrary curve surrounding the inner circle (d_{pres} does not depend on the particular choice of γ). The actual value of d_{pres} is a nonlinear scalar function of ψ_2. Therefore, the usual iteration algorithm has to be supplemented by an additional step in which ψ_2 is iteratively

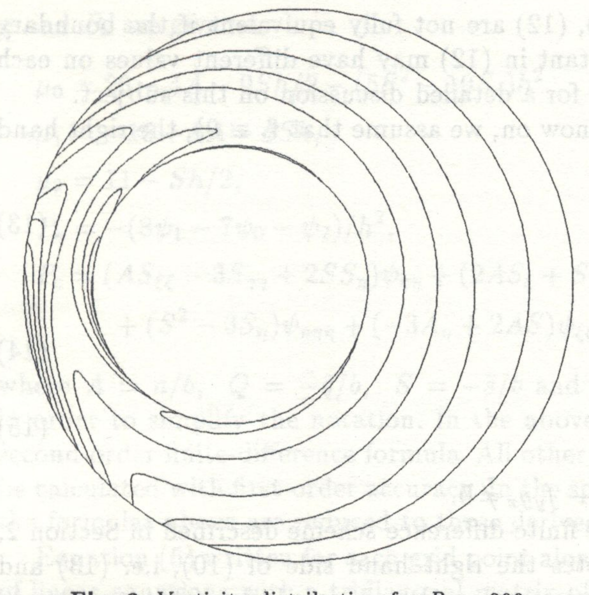
Fig. 3. Vorticity distribution for *Re* = 200

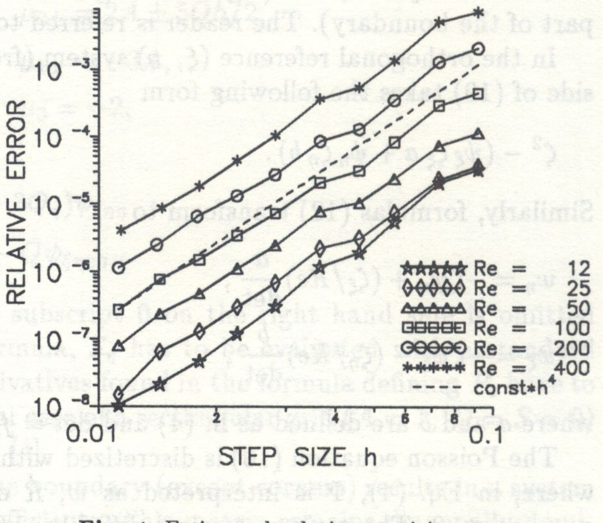

Fig. 4. Estimated relative vorticity error

modified to obtain $d_{\text{pres}} = 0$ (we used the simplest secant method). Ideally, d_{pres} can be calculated by integrating (15) along any γ. However, the most obvious choice is to take any interior line $\eta =$ const. In our tests, we calculated d_{pres} on a few such lines and took the mean value. This allowed us to obtain a good approximation of d_{pres} even from values of ψ and ζ that have not yet completely converged (it would have been a waste of computer time to calculate fully converged ψ and ζ for ψ_2 that had not yet converged).

5. Domain Decomposition

The domain decomposition methodology consists in dividing the computational domain into overlapping subdomains and solving the original problem on each subdomain separately, with the appropriate transfer of boundary information between the neighbouring subdomains. Since calculations on each subdomain can be carried out simultaneously on different processors, this procedure offers an opportunity for tremendous acceleration of calculations.

The success of domain decomposition in accelerating the overall calculations depends most crucially on the strategy of information transfer between different subdomains. Two obvious issues are (i) what type of information should be transferred and (ii) what the size of the overlap domain should be. These questions can partly be answered by identifying and analysing the transfer matrix for a simple model problem. Our analytical results show that the convergence rate for a linear problem with two subdomains is $\sim(1-\delta^3)$ for the transfer of $(\psi, \frac{\partial\psi}{\partial n})$ and $\sim(1-\delta)$ for the transfer of (ψ, ζ), where δ denotes the size of the overlap domain. These results suggest that the former type of information transfer may be impractical. Numerical experiments with the full Navier-Stokes equations confirm these predictions and show lack of convergence in the case of the transfer of $(\psi, \frac{\partial\psi}{\partial n})$. Acceleration of calculations due to multiprocessing has been investigated with the use of computational examples discussed in [4].

In our calculations, the solution domain (square) was subdivided into k equal size subdomains and multiprocessing was simulated on a single processor with the appropriate information transfer. The overlap between subdomains was characterized by δ, which in that case was half the width of the overlapping rectangle. The calculations were performed for different values of δ, including the smallest possible overlap $2\delta = h$, where h stands for the discretization step size.

It is interesting to check whether domain decomposition can influence effective accuracy of the discretization schemes. The results of our numerical experiments show no such effect, i.e. in all cases fourth-order accuracy of the solution is maintained.

The acceleration of calculations due to the use of several processors, each one serving a different subdomain, is best described by defining an acceleration factor in the form

$$A(K) = \frac{W(K)}{W(1)}$$

where $W(K)$ stands for the work effort associated with K processors (serving K subdomains) and $W(1)$ denotes the work effort of a single processor (single domain calculations). The acceleration factor cannot exceed the number of processors K which defines the theoretically possible maximum acceleration of calculations. If $A(K)$ is less than one, the application of domain decomposition brings no benefits. The work effort of a hypothetical K-processor computer can be defined as

$$W(K) = [1 + OVL(\delta)]\frac{N(K)}{K} .$$

In the above equation, $N(K)$ stands for the total number of cycles of the Navier-Stokes solver. The work effort is reduced by the factor K because each of K processors serves $1/K$ part of the total grid, but it is increased by the factor $(1 + OVL(\delta))$ because the total number of grid points increases due to the overlap between subgrids. Here, $OVL(\delta) \sim \delta \ll 1$. The work effort of a single processor computer $W(1)$ is simply equal to the number of cycles $N(1)$ of the Navier-Stokes solver.

The results shown in Fig. 5 demonstrate that it is possible to come very close to the theoretically possible maximum acceleration rate. These results also show that the advantage of multiprocessing increases with the increasing problem size.

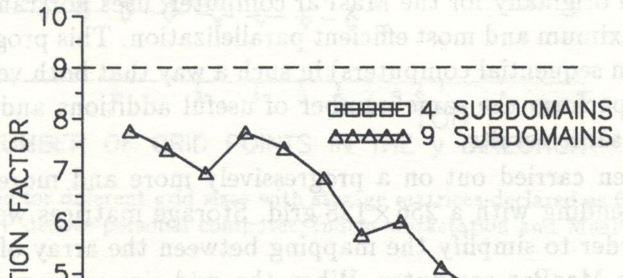

Fig. 5. Computation acceleration due to the use of several processors, each one serving a different subdomain

6. Parallel computations

Domain decomposition concepts can be applied in various ways. At one extreme, a single domain is served by a single processor. In the opposite extreme, the number of subdomains equals the number of grid points, each grid point being served by a different processor. The utility of the latter approach is illustrated below for the case of viscous flow in a periodic rectangle [4]. The formulation of the problem and the solution algorithm are as described in Section 2.

We have selected three computing platforms for testing purposes. The first one consists of a personal computer with a 486-66MHz processor, 8 MB memory, DOS-6 operating system and the Lahey EM/32 Fortran compiler version 3.01. The second one consists of an Iris-Indigo workstation with an R-4000 processor, 32 MB memory, IRIX 4.0.5F operating system and the SGI Fortran 77 compiler version 3.10. The third one uses a MasPar MP-2 massively parallel computer with an array of 64×32 (2048) processors of 64kB memory each, running under MasPar/Ultrix Operating System version 3.2.2 with the MasPar Fortran compiler version 2.2. The first two compilers are compatible with Fortran 77 standards, while the last one uses Fortran 90 concepts. All coding has been done with double precision arithmetics.

The test problem that forms the basis for the following discussion involves a steady flow of viscous incompressible fluid in a rectangular domain with the Reynolds number $Re = 10$, subject to the following boundary conditions at the upper and lower boundaries:

$$u = v = 0, \qquad \text{at} \quad y = 0, \quad 0\le x\le 2, \tag{17}$$

$$u = \sin^2(\pi x), \quad v = 0, \qquad \text{at} \quad y = 1, \quad 0\le x\le 2, \tag{18}$$

and periodic conditions in the x direction, with the length of one period equalling 2. In the above, u and v stand for the x and y components of the velocity vector, respectively. Our boundary conditions are specially selected in order to include a mix of conditions found in typical application problems. The rectangular shape of the solution domain results from the structure of the processors' array in the MasPar computer and from our desire to work with the same grid step size in the x and y directions.

The program, written originally for the MasPar computer, uses Fortran 90 array instructions in order to allow for the maximum and most efficient parallelization. This program has been translated into Fortran 77 (to use on sequential computers) in such a way that both versions are arithmetically equivalent, i.e. (i) they perform the same number of useful additions and multiplications and (ii) they deliver exactly the same results.

Calculations have been carried out on a progressively more and more refined grid, beginning with an 18×9 grid and ending with a 256×128 grid. Storage matrices were declared in multiples of 64×32 segments in order to simplify the mapping between the array of processors and the grid points in the case of the MasPar computer. When the grid size was smaller than the number of processors, some of them were performing useless work. When the grid size was exactly 64×32, all processors were performing useful work. When the declared size of matrices was larger than 64×32, the matrices were automatically partitioned (by the operating system) into 64×32 segments. Again, if the grid happened to be such that not all elements of the matrices were utilized, some processors would perform useless work during a certain part of the computational cycle.

The increase in the size of the declared matrices caused an increase in the required core memory. Tests were carried out up to the 128×64 grid in the case of the personal computer because of memory limitations. The memory available in the two remaining computers was sufficient for the in-core calculations for all the tests quoted.

The efficiency of a particular computer may depend on the overall size of the solution problem. In order to compare various machines , we divided the overall time required to carry out calculations by the number of iterations and the number of grid points. This gave us the time required to serve one grid point during one iteration. The inverse of this quantity gave us the speed of calculations. We estimated that one unit on this speed scale corresponds to roughly 4×10^7 additions and multiplications (double precision) per second. Also, we typically calculated 10^3 iterations in the tests in order to improve accuracy of time measurements.

In the first test, we used Jacobi iterations on all computers. We showed that calculations were arithmetically equivalent, i.e. the results were identical on all computers after the same number of iterations and with the same initial guess. We also showed that fourth-order accuracy was maintained on all machines and that it was possible to get convergence up to machine accuracy, regardless of the type of machine.

Figure 6 presents the results of our tests. It can be seen that the speed of calculations on single processor (sequential) machines remains essentially constant regardless of the problem size, the Indigo being about 10 times faster than the PC. We noted significant deterioration of the performance of Indigo in the case of a 256×128 grid, which required no more than 16–17 MB of core memory. The same test was repeated on a Challenger machine with an R-4400 processor and 128 MB memory and the conclusions were the same. This significant deterioration of the performance of the Silicon Graphics machines is due to the problems in the caching algorithm encountered when the declared matrix size is a power of 2 ($256 = 2^8$). The problem disappears after changing the size of the declared matrices to 257×129, for example.

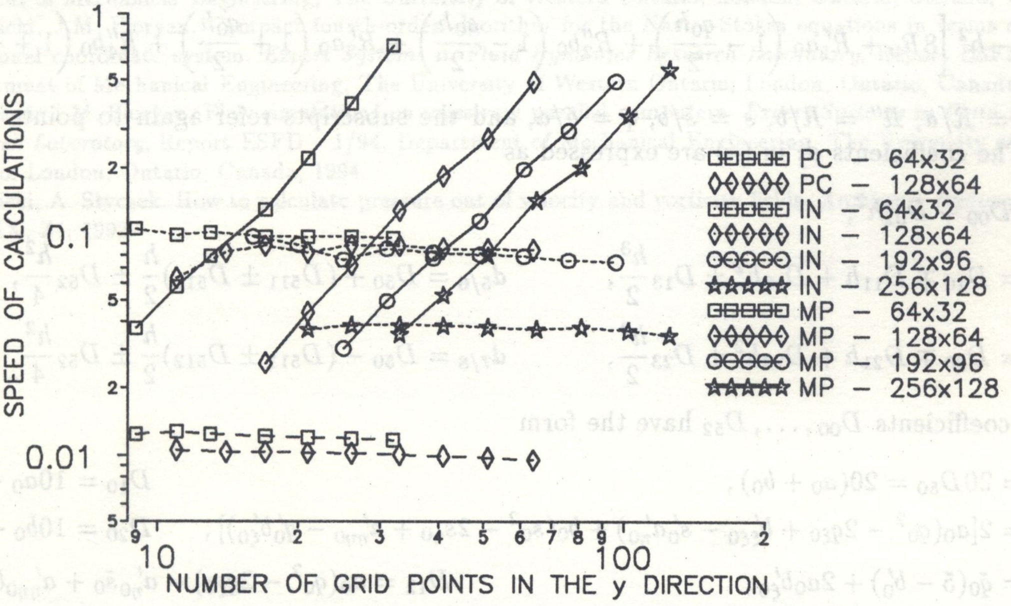

Fig. 6. Calculation speed for different grid sizes with storage matrices declared as 64×32, 128×64, 192×92 or 256×128. PC, IN and MP denote personal computer, Indigo workstation and MasPar computer, respectively

The results of the tests performed by the MasPar computer with 64×32 matrices show a linear increase in the effective speed of calculations with an increase in the number of grid points (Fig. 6). This is due to the fact that an increasing number of processors participate in the calculations as the number of grid points increases. MasPar is about 2.5 times slower than Indigo on an 18×9 grid, but it is about 8 times faster on a 64×32 grid. The increase in the size of matrices to 128×64 required their automatic partitioning (by the operating system) into four 64×32 segments for mapping into the array of processors. This caused deterioration of the performance by about 30%. Again, the speed of calculation increased with an increasing number of grid points due to better utilization of the available processors. Matrices of size 192×96 required partitioning into nine 64×32 segments, and those of size 256×128 required partitioning into sixteen 64×32 segments. The presence of these additional segments caused no additional deterioration of the performance of MasPar (see Fig. 6). If we eliminated the 256×128 grid from our considerations (due to the substandard performance of Indigo), MasPar would be at best only about 6 times faster than Indigo and only if the grid structure were such that all the processors were fully utilized. An inappropriate declaration of matrices can make MasPar five times slower than Indigo (see the case of a 64×32 grid in Fig. 6).

In the second test, we compared the efficiency of the Jacobi and Gauss-Seidel iterative procedures on the Indigo machine. This was done in order to assess loss of performance on parallel machines due to the use of a less effective algorithm. The test consisted in finding a solution to our test problem with the same grids, the same convergence criteria and the same relaxation parameters (relaxation parameters were kept constant regardless of the Reynolds number and no attempt was made to optimize them). On the average, our results show about 20% decrease in computational effort due to the use of the Gauss-Seidel rather than Jacobi iterations. This makes the Indigo workstation only 5 times slower than the MasPar computer. Computational details are described in [6].

APPENDIX

The fourth-order discretization scheme for the generic equation (4) is not restricted to equations of the Navier-Stokes type. Indeed, this discretization is valid for all PDE of generic form (4) provided $a > 0$, $b > 0$ and all a, b, $\tilde{q}$, $\tilde{s}$ are sufficiently regular functions of ξ and η. The free term B_0 in (4) is expressed as

$$B_0 = -h^2 \left[8R_0 + R_1'a_0 \left(1 - \frac{q_0 h}{2}\right) + R_2''b_0 \left(1 - \frac{s_0 h}{2}\right) + R_3'a_0 \left(1 + \frac{q_0 h}{2}\right) + R_4''b_0 \left(1 + \frac{s_0 h}{2}\right) \right]$$

with $R' = R/a$, $R'' = R/b$, $s = \tilde{s}/b$, $q = \tilde{q}/a$, and the subscripts refer again to points shown in Fig. 1. The coefficients d_j in (4) are expressed as

$$d_0 = D_{00} + D_{02}h^2,$$

$$d_{1/3} = D_{10} \mp D_{11}h + D_{12}h^2 \pm D_{13}\frac{h^3}{2}, \qquad d_{5/6} = D_{50} + (D_{511} \pm D_{512})\frac{h}{2} \pm D_{52}\frac{h^2}{4},$$

$$d_{2/4} = D_{20} \mp D_{21}h + D_{22}h^2 \pm D_{23}\frac{h^3}{2}, \qquad d_{7/8} = D_{50} - (D_{511} \pm D_{512})\frac{h}{2} \pm D_{52}\frac{h^2}{4}$$

and the coefficients $D_{00}, \ldots, D_{52}$ have the form

$$D_{00} = 20D_{50} = 20(a_0 + b_0), \qquad D_{10} = 10a_0 - 2b_0,$$

$$D_{02} = 2[a_0({q_0}^2 - 2q_{\xi 0} + b'_{\xi\xi 0} - s'_0 a'_{\eta 0}) + b_0({s_0}^2 - 2s_{\eta 0} + a'_{\eta\eta 0} - q'_0 b'_{\xi 0})], \qquad D_{20} = 10b_0 - 2a_0,$$

$$D_{11} = \tilde{q}_0(5 - b'_0) + 2a_0 b'_{\xi 0}, \qquad D_{12} = a_0({q_0}^2 - 2q_{\xi 0}) - a'_{\eta 0}\tilde{s}_0 + a'_{\eta\eta 0}b_0,$$

$$D_{21} = \tilde{s}_0(5 - a'_0) + 2b_0 a'_{\eta 0}, \qquad D_{22} = b_0({s_0}^2 - 2s_{\eta 0}) - b'_{\xi 0}\tilde{q}_0 + b'_{\xi\xi 0}a_0,$$

$$D_{13} = a_0 q_0 q_{\xi 0} + \tilde{s}_0 q'_{\eta 0} - (a_0 q_{\xi\xi 0} + b_0 q'_{\eta\eta 0}), \qquad D_{511} = 2b_0 a'_{\eta 0} - \tilde{s}_0(a'_0 + 1),$$

$$D_{23} = \tilde{q}_0 s'_{\xi 0} + b_0 s_0 s_{\eta 0} - (a_0 s'_{\xi\xi 0} + b_0 s_{\eta\eta 0}), \qquad D_{512} = 2a_0 b'_{\xi 0} - \tilde{q}_0(b'_0 + 1),$$

$$D_{52} = \tilde{q}_0 s'_0 + \tilde{s}_0 q'_0 - 2(a_0 s'_{\xi 0} + b_0 q'_{\eta 0}),$$

where $b' = b/a$, $a' = a/b$, $s' = \tilde{s}/a$, $q' = \tilde{q}/b$, lower indices ξ and η denote differentiation in the corresponding directions and subscript 0 on the right hand side corresponds to point 0 in Fig. 1. If $\xi = x$, $\eta = y$, we have $q_* = \tilde{q} = q = q'$, $s_* = \tilde{s} = s = s'$, $a = b \equiv 1$, and all derivatives of a and b vanish. In this case, the above formulas for d_j, $(j = 0, \cdots, 8)$, are reduced to those derived by Dennis and Hudson [2]. The special case of $\xi = f(x)$ and $\eta = g(y)$ is equivalent to the generation of a non-uniform rectangular grid in the original flow domain (x, y) with a uniform square grid in the computational domain (ξ, η).

All the first and second derivatives of coefficients a, b, a', b', s, s', q, q', necessary to evaluate coefficients d_j, can be calculated with the usual second-order finite-difference formulas without compromising fourth-order accuracy of the scheme (cf. [4]). All coefficients, however, have to be evaluated with fourth-order accuracy (cf. [2, 4]). This means that fourth-order formulas for the first derivative of the streamfunction are required to calculate q_* and s_* in (4). Such accuracy can be obtained by means of both the streamfunction and the vorticity grid data.

ACKNOWLEDGEMENTS

This work has been supported by the NSERC (Natural Sciences and Engineering Research Council) of Canada.

REFERENCES

[1] S.C.R. Dennis, J.D. Hudson. A difference method for solving the Navier-Stokes equations. In: C. Taylor et al., eds., *Proceedings of the First International Conference on Numerical Methods in Laminar and Turbulent Flow*, 69–80, Pentech Press, London, 1978.

[2] S.C.R. Dennis, J.D. Hudson. Compact h^4 finite-difference approximations to operators of Navier-Stokes type. *J. Comput. Phys.*, **85**: 390–416, 1989.

[3] M.M. Gupta, R.P. Manohar. Boundary approximations and accuracy in viscous flow computations. *J. Comput. Phys.*, **31**: 265–288, 1979.

[4] J. Rokicki, J.M. Floryan. Multiprocessor implementation of the compact finite-difference method for the Navier-Stokes equations. *Expert Systems in Fluid Dynamics Research Laboratory*, Report ESFD - 3/93, Department of Mechanical Engineering, The University of Western Ontario, London, Ontario, Canada, 1993.

[5] J. Rokicki, J.M. Floryan. Compact fourth-order algorithm for the Navier-Stokes equations in terms of general orthogonal coordinate system. *Expert Systems in Fluid Dynamics Research Laboratory*, Report ESFD - 5/93, Department of Mechanical Engineering, The University of Western Ontario, London, Ontario, Canada, 1993.

[6] J. Rokicki, J.M. Floryan. Flow simulations on massively parallel computers. *Expert Systems in Fluid Dynamics Research Laboratory*, Report ESFD - 1/94, Department of Mechanical Engineering, The University of Western Ontario, London, Ontario, Canada, 1994.

[7] J. Rokicki, A. Styczek. How to calculate pressure out of velocity and vorticity fields. *Archiwum Budowy Maszyn*, **XXXIX**, Z4, 1992.