Записки научных
семинаров ПОМИ
Том 258, 1999 г.

V. G. Ganzha and E. V. Vorozhtsov

PARALLEL IMPLEMENTATION OF STABILITY ANALYSIS OF DIFFERENCE SCHEMES WITH MATHEMATICA

ABSTRACT. We consider a parallel algorithm for stability investigation of the schemes of the finite difference method or the finite volume method approximating the two-dimensional Euler equations of compressible fluids on curvilinear grids. The algorithm is implemented with the computer algebra system *Mathematica 3.0*. We apply a two-level parallelization process. At the first level, the parallelization of the symbolic computation of the amplification matrix is performed by a parallel computation of the matrix rows on different processors. At the second parallelization level, we compute numerically the values of the coordinates of points of the stability region boundary. For the communication between the workstations we use a special program LaunchSlave, which uses the MathLink communication protocol. The examples of the application of the proposed parallel symbolic/numeric algorithm are presented.

1. INTRODUCTION

At present computational fluid dynamics (CFD) is changing the relative roles of experiment and computation in the engineering of aircraft and aerospace vehicles. One of the reasons for this is a drastical improvement of the accuracy and efficiency of the numerical solvers for the Euler and Navier–Stokes equations of compressible fluids. There are at present a number of techniques for the reduction of computer time when solving numerically the Navier–Stokes or the Euler equations:

(i) Parallelization of the Euler or Navier–Stokes numerical solvers [1–3]. With the advent of computers with parallel architecture, there has appeared a possibility of computer implementation of very complex solvers for multidimensional problems on curvilinear grids.

(ii) Development of more efficient numerical methods.

(iii) Development of solution-adaptive grids.

The efficiency of the methods for numerical integration of the Euler equations can be increased by developing the finite-difference or finite volume discretization schemes, which possess the maximum stability robustness within a given class of schemes.

There are several ways to solve this problem. The most popular means to extend the stability interval of a difference scheme is to use fully implicit schemes. But then one has to numerically invert matrices of very complex structure [4]. In addition, the implicit schemes are difficult to implement in the case of geometrically complex boundaries; such schemes are also difficult to parallelize.

The explicit difference schemes are much easier to implement, but the stability condition imposes restrictions on the time steps. It should be noted that all practically important CFD problems are now solved on curvilinear spatial grids. The consideration of curvilinear grids in the Fourier stability analyses of difference schemes enhances considerably the complexity of arising algebraic expressions. As a result of this, until now only a few comparatively simple difference schemes on curvilinear grids have been analyzed; a review of these studies may be found in [5]. The estimates of computational complexity of the Fourier stability analysis of difference schemes approximating the 3D Navier–Stokes equations, which were presented in [5], show that such a stability analysis may require many hours of computer time on serial computers. This stability analysis is based on the use of computer algebra systems, which prove to be the only means for obtaining the data on the stability of very complex difference schemes on curvilinear grids.

To reduce the computer time expenses when analyzing the stability of complex difference schemes it is natural to turn to the idea of the parallelization of such an analysis. The present work appears to represent the first attempt at parallelizing the Fourier stability analysis within the framework of using the computer algebra system *Mathematica 3.0* for the implementation of both symbolic and numerical stages of the algorithm for stability analysis. We consider a parallel algorithm for stability investigation of the schemes of the finite difference or the finite volume method approximating the two-dimensional Euler equations on curvilinear grids.

2. Fourier Symbol on Curvilinear Grid

We have shown in [5] that the Fourier method [6] is the most reliable and universal method for the stability investigation of difference schemes. We briefly present in this Section the general procedure of the Fourier method and its peculiarities related to the difference schemes on curvilinear grids. The curvilinear grids consisting of the quadrilateral cells have gained a very wide acceptance. They are mapped with the aid of a nondegenerate transformation

$$x = x(\xi, \eta), \quad y = y(\xi, \eta)$$

from the plane of the Cartesian spatial coordinates (x, y) onto an uniform rectangular grid in the plane of curvilinear coordinates ξ, η. Such curvilinear grids are also called the logically rectangular grids [7]. We show in Fig. 1 a C-grid around the airfoil NACA0012 as an example.

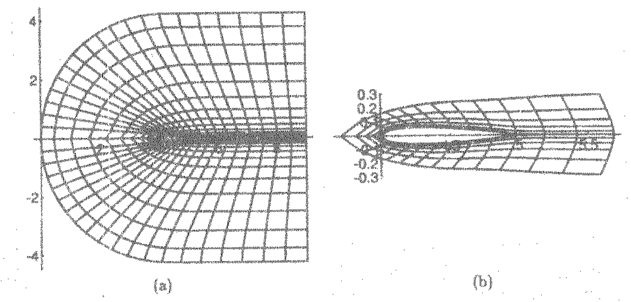

Fig. 1. The C-grid of 41 x 15 nodes around the airfoil NACA0012: (a) the complete grid; (b) the partial view of the grid near the airfoil.

In the case where the finite volume method is used for the discretization of the Euler equations on curvilinear grids there is no need to use the Euler equations in curvilinear coordinates. It is sufficient to take a small control volume at the center of which the current node (x_{jk}, y_{jk}) of the curvilinear computing mesh is located, and to transform the spatial differentiation operators in the Euler equations with the aid of the Green's theorem. We demonstrate this procedure at the example of the Euler equations governing the two-dimensional nonstationary flow of an inviscid compressible non-heat-conducting gas:

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0, \tag{1}$$

where t is the time, x, y are the spatial Cartesian coordinates, and the flux vectors $\vec{F}$ and $\vec{G}$ are as follows:

$$\begin{aligned} \vec{F}(\vec{u}) &= (\rho u, p + \rho u^2, \rho uv, pu + \rho uE)^T, \\ \vec{G}(\vec{u}) &= (\rho v, \rho uv, p + \rho v^2, pv + \rho vE)^T, \\ \vec{u} &= (\rho, \rho u, \rho v, \rho E)^T. \end{aligned} \tag{2}$$

The system (1)–(2) is completed in the following by the ideal gas equation of state

$$p = (\gamma - 1)\rho\varepsilon, \tag{3}$$

where γ is the ratio of the gas specific heats. In equations (1)–(3), ρ is the fluid density, u, v are the fluid velocity components along the x-and y-axis, $E = \varepsilon + (u^2 + v^2)/2$, ε is the specific internal energy, and p is the pressure.

Let us choose an arbitrary cell and denote its four vertices by P_1, P_2, P_3, P_4 (see Fig. 2). Let these points have the coordinates

$$P_1(x_{j,k}, y_{j,k}),\ \ P_2(x_{j+1,k}, y_{j+1,k}),$$

$$P_3(x_{j+1,k+1}, y_{j+1,k+1}),\ \ P_4(x_{j,k+1}, y_{j,k+1}).$$

Let us take the region bounded by the contour $\Gamma_{j,k} = P_1P_2P_3P_4P_1$ as the control volume $V_{j,k}$. Let $S_{j,k}$ be the area of the region bounded by the contour $\Gamma_{j,k}$. To derive the approximate formulas of the finite volume method we use the Green's theorem:

$$\iint\limits_{V_{j,k}} \Big(\frac{\partial\vec{u}}{\partial t} + \frac{\partial\vec{F}}{\partial x} + \frac{\partial\vec{G}}{\partial y}\Big)\, dx\, dy =$$

$$= \iint\limits_{V_{j,k}} \frac{\partial\vec{u}}{\partial t} dx\, dy + \oint\limits_{\Gamma_{j,k}} (\vec{F}\, dy - \vec{G}\, dx) = 0. \tag{4}$$

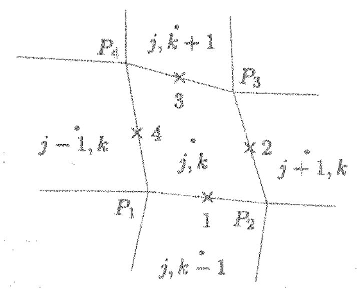

Fig. 2. The centered scheme.

One can evaluate the first integral in the right-hand side of (4) with the aid of the mean-value theorem:

$$\iint\limits_{V_{j,k}} \frac{\partial \vec{u}}{\partial t} dx\, dy \approx S_{j,k} \Big(\frac{\partial \vec{u}}{\partial t}\Big)_{j,k},$$

where the subscripts (j, k) refer to the control volume center $V_{j,k}$; that is the components of the vector $\vec{u}$ are computed at the center of the volume $V_{j,k}$. Then we obtain instead of (4) the following approximate equality:

$$\Big(\frac{d\vec{u}}{dt}\Big)_{j,k} + P\vec{u}(t) = 0, \tag{5}$$

where

$$Pu(t) = \frac{1}{S_{j,k}} \oint\limits_{\Gamma_{j,k}} (\vec{F}\, dy - \vec{G}\, dx). \tag{6}$$

There are two schemes for the approximation of the integrals in the finite-volume method: the centered scheme and the nodal scheme [8]. We use the centered scheme in the following (see Fig. 2), that is we will assume that the components of the vector $\vec{u}$ are computed at the centers of the curvilinear grid cells. The area $S_{j,k}$ can be computed within the framework of this scheme with the use of the notations of Fig. 2 as

$$S_{j,k} = \iint\limits_{V_{j,k}} dx\, dy = \oint\limits_{\Gamma_{j,k}} x\, dy =$$

$$= x_1(y_{P_2} - y_{P_1} + x_2(y_{P_3} - y_{P_2}) + x_3(y_{P_4} - y_{P_3}) + x_4(y_{P_1} - y_{P_4})$$

$$= \frac{1}{2}[(x_{j+1,k+1} - x_{j,k})(y_{j,k+1} - y_{j+1,k}) + (x_{j+1,k} - x_{j,k+1})(y_{j+1,k+1} - y_{j,k})]. \tag{7}$$

The fluxes $\vec{F}(\vec{u})$ and $\vec{G}(\vec{u})$ across the boundaries of the control volume $V_{j,k}$ are computed as the mean values. For example,

$$(\rho u)_2 = \frac{1}{2}[(\rho u)_{j,k} + (\rho u)_{j+1,k}]. \tag{8}$$

As was shown in [8] the use of the approximation of form (8) for fluxes leads in the particular case of a rectangular uniform grid to a difference scheme with central differences. Therefore, the above scheme of the finite volume method has the second order of approximation in spatial variables under the condition that the curvilinear grid is sufficiently smooth (see also [8]).

Denote by $(P_h\vec{u})_{j,k}$ the approximation of integral (6) using the formulas of type (8). In the case where no artificial dissipators are used the operator $(P_h\vec{u})_{j,k}$ has the form

$$\begin{aligned}(P_h\vec{u})_{j,k} = \frac{1}{2S_{j,k}}&[(\vec{F}_{j,k} + \vec{F}_{j,k-1})(y_{j+1,k} - y_{j,k})\\
&+ (\vec{F}_{j,k} + \vec{F}_{j+1,k})(y_{j+1,k+1} - y_{j+1,k})\\
&+ (\vec{F}_{j,k} + \vec{F}_{j,k+1})(y_{j,k+1} - y_{j+1,k+1})\\
&+ (\vec{F}_{j,k} + \vec{F}_{j-1,k})(y_{j,k} - y_{j,k+1})\\
&- (\vec{G}_{j,k} + \vec{G}_{j,k-1})(x_{j+1,k} - x_{j,k})\\
&- (\vec{G}_{j,k} + \vec{G}_{j+1,k})(x_{j+1,k+1} - x_{j+1,k})\\
&- (\vec{G}_{j,k} + \vec{G}_{j,k+1})(x_{j,k+1} - x_{j+1,k+1})\\
&- (\vec{G}_{j,k} + \vec{G}_{j-1,k})(x_{j,k} - x_{j,k+1})].\end{aligned} \tag{9}$$

The explicit schemes of the Runge–Kutta type [9] for the numerical integration of the Euler equations have a larger stability interval than many previous explicit schemes. In addition, the stability robustness of the Runge–Kutta type schemes (we will also call them the Jameson's schemes) increases with the number of stages [5, 8]. Along with this property the Jameson's schemes possess a number of other positive properties [10]. This has caused a very wide acceptance of the Runge–Kutta type schemes for the numerical solution of the aerodynamics problems.

From the viewpoint of complexity the Runge–Kutta type schemes represent a very interesting object on which one can validate the efficiency of the new approaches to the symbolic-numeric stability investigation of difference schemes. The complexity of the step operator increases in these schemes nonlinearly with the number of stages. This leads to a corresponding increase of the demands on the computer storage and speed of the processing of symbolic expressions.

The three-stage Runge–Kutta type schemes may be considered as the schemes possessing already a larger complexity than other well-known previous schemes (for example, the MacCormack scheme, which was very popular in the past, contains only two stages: the "predictor" stage and the "corrector" stage, see [5]). The number of stages three altogether ensures already a good stability robustness. In this connection, the three-stage schemes are widely used in the aerodynamic computations (see a review of the relevant works in [10]).

A three-stage Runge–Kutta scheme of the form [11]

$$\begin{aligned}&\vec{u}^{(0)} = \vec{u}^{n}; \quad \vec{u}^{(1)} = \vec{u}^{0} - \alpha_1 \tau P_h \vec{u}^{(0)}; \quad \vec{u}^{(2)} = \vec{u}^{0} - \alpha_2 \tau P_h \vec{u}^{(1)};\\ &\vec{u}^{(3)} = \vec{u}^{0} - \tau P_h \vec{u}^{(2)}; \quad \vec{u}^{n+1} = \vec{u}^{(3)},\end{aligned} \tag{10}$$

approximates the system of ordinary differential equations (5). Here τ is the time step, α_1 and α_2 are the constant weight parameters. The parameters α_1 and α_2 can be chosen from the requirement of stability robustness [5, 11]. For example, the choice $\alpha_1 = \alpha_2 = 1/2$ ensures the stability condition $0 \leq |a\tau/h| \leq 2$ in the case where scheme (10) is applied for the difference apprpximation of the scalar advection equation $u_t + au_x = 0$ [10]. We will use the values $\alpha_1 = \alpha_2 = 1/2$ in what follows.

The Fourier method is applicable only to the linear difference schemes with constant coefficients. The difference equations (10) are nonlinear, therefore, we have to linearize them before the application of the Fourier method. For this purpose we use the formulas

$$\vec{F}_{m,q} \approx \vec{F}(\vec{u}_{j,k}) + A(\vec{a}_{j,k})\delta\vec{u}_{m,q}, \ \vec{G}_{m,q} \approx \vec{G}(\vec{u}_{j,k}) + B(\vec{a}_{j,k})\delta\vec{u}_{m,q}, \tag{11}$$

where $\delta\vec{u}_{m,q} = \vec{u}_{m,q} - \vec{u}_{j,k}$, (m, q) are any values of the subscripts entering (10); the Jacobi matrices A and B are defined by formulas

$$A = \partial\vec{F}(\vec{u})/\partial\vec{u}, \quad B = \partial\vec{G}(\vec{u})/\partial\vec{u}. \tag{12}$$

Denote the linearized operator obtained by a linearization of the operator $P_h\vec{u}$ by $\bar{P}_h\vec{u}$. Substituting formulas (11) in (10) we obtain the following expression for the operator $(\bar{P}_h\vec{u})_{j,k}$:

$$\begin{aligned}(\bar{P}_h\vec{u})_{j,k} = \frac{1}{2S_{j,k}}[&A\vec{u}_{j,k-1}(y_{j+1,k} - y_{j,k})\\ &+ A\vec{u}_{j+1,k}(y_{j+1,k+1} - y_{j+1,k}) + A\vec{u}_{j,k+1}(y_{j,k+1} - y_{j+1,k+1})\\ &+ A\vec{u}_{j-1,k}(y_{j,k} - y_{j,k+1}) - B\vec{u}_{j,k-1}(x_{j+1,k} - x_{j,k})\\ &- B\vec{u}_{j+1,k}(x_{j+1,k+1} - x_{j+1,k})\\ &- B\vec{u}_{j,k+1}(x_{j,k+1} - x_{j+1,k+1}) - B\vec{u}_{j-1,k}(x_{j,k} - x_{j,k+1})].\end{aligned} \tag{13}$$

For the sake of brevity we have used here the notations $A = A(\vec{u}_{j,k})$, $B = B(\vec{u}_{j,k})$. The linearization assumes that the coefficients A and B are constant. Then we can see from (13) that the operator $\bar{P}_h$ is linear. A direct substitution of expression (13) in the three-stage scheme (10) leads to huge algebraic expressions at the stage of a symbolic computation of the Fourier symbol G. However, it is possible to simplify significantly the expression for the operator $\bar{P}_h$ under certain reasonable assumption. Rewrite expression (13) in the form

$$(\bar{P}_h\vec{u})_{j,k} = A\Lambda_1\vec{u}_{j,k} + B\Lambda_2\vec{u}_{j,k}, \tag{14}$$

where

$$\begin{aligned}\Lambda_1\vec{u}_{j,k} = \frac{1}{2S_{j,k}}[&(y_{j+1,k} - y_{j,k})\vec{u}_{j,k-1} + (y_{j+1,k+1} - y_{j+1,k})\vec{u}_{j+1,k}\\ &+ (y_{j,k+1} - y_{j+1,k+1})\vec{u}_{j,k+1} + (y_{j,k} - y_{j,k+1})\vec{u}_{j-1,k}],\end{aligned} \tag{15}$$

$$\begin{aligned}\Lambda_2\vec{u}_{j,k} = \frac{1}{2S_{j,k}}[&(x_{j,k} - x_{j+1,k})\vec{u}_{j,k-1} + (x_{j+1,k} - x_{j+1,k+1})\vec{u}_{j+1,k}\\ &+ (x_{j+1,k+1} - x_{j,k+1})\vec{u}_{j,k+1} + (x_{j,k+1} - x_{j,k})\vec{u}_{j-1,k}].\end{aligned} \tag{16}$$

Assume that a specific control volume $V_{j,k}$ is such that the following relations are satisfied for it:

$$\begin{aligned}&y_{j+1,k+1} - y_{j,k+1} = y_{j+1,k} - y_{j,k};\ y_{j+1,k+1} - y_{j+1,k} = y_{j,k+1} - y_{j,k};\\ &x_{j+1,k+1} - x_{j+1,k} = x_{j,k+1} - x_{j,k};\ x_{j+1,k+1} - x_{j,k+1} = x_{j+1,k} - x_{j,k}.\end{aligned} \tag{17}$$

It follows from Fig. 2 that the conditions (17) are the conditions for that the pairs of the sides P_3P_4 and P_1P_2, P_2P_3 and P_4P_1 are parallel. Thus, the volume $V_{j,k}$ proves to be a parallelogram under satisfaction of conditions (17). Note that these conditions are satisfied for all cells of a parallelogram grid, which is the particular case of curvilinear grid. In the case of a general curvilinear grid only some cells will have the parallelogram form or will differ little from the parallelograms.

Expression (13) simplifies greatly under satisfaction of conditions (17):

$$\begin{aligned}&(\bar{P}_h\vec{u})_{j,k} =\\ &= A_1(\vec{u}_{j,k}, x, y)\frac{\vec{u}_{j+1,k} - \vec{u}_{j-1,k}}{2} + A_2(\vec{u}_{j,k}, x, y)\frac{\vec{u}_{j,k+1} - \vec{u}_{j,k-1}}{2},\end{aligned} \tag{18}$$

where

$$\begin{aligned}A_1 &= \frac{1}{S_{j,k}}[(y_{j,k+1} - y_{j,k})A - (x_{j,k+1} - x_{j,k})B],\\ A_2 &= \frac{1}{S_{j,k}}[(x_{j+1,k} - x_{j,k})B - (y_{j+1,k} - y_{j,k})A].\end{aligned} \tag{19}$$

The linearized scheme corresponding to scheme (10) has the form

$$\begin{aligned}&\vec{u}^{(0)} = \vec{u}^n;\quad \vec{u}^{(1)} = \vec{u}^0 - \alpha_1\tau\bar{P}_h\vec{u}^{(0)};\quad \vec{u}^{(2)} = \vec{u}^0 - \alpha_2\tau\bar{P}_h\vec{u}^{(1)};\\ &\vec{u}^{(3)} = \vec{u}^0 - \tau\bar{P}_h\vec{u}^{(2)};\quad \vec{u}^{n+1} = \vec{u}^{(3)}.\end{aligned} \tag{20}$$

Eliminating the intermediate quantities $\vec{u}^{(1)}$ and $\vec{u}^{(2)}$ from (20) we obtain the two-level difference scheme of the form

$$\vec{u}^{n+1} = S(x, y, \vec{u}_0)\vec{u}^n, \qquad n = 0, 1, 2, \dots, \tag{21}$$

where the step operator S has the form

$$S(x, y, \vec{u}_0) = I - \tau\bar{P}_h + \alpha_2(\tau\bar{P}_h)^2 - \alpha_1\alpha_2(\tau\bar{P}_h)^3, \tag{22}$$

and $\vec{u}_0$ is the “frozen” solution vector whose components enter the Jacobi matrices (12). Let us substitute in (21) the solution of the form

$$\vec{u}^n_{j,k} = \vec{u}_0 e^{i(k_1 j + k_2 k - \omega n \tau)},$$

where k_1 and k_2 are the real wave numbers, ω is the wave frequency. Then we obtain from (21) the Fourier symbol on the arbitrary curvilinear grid:

$$G = I + Z + \alpha_2 Z^2 + \alpha_1\alpha_2 Z^3, \tag{23}$$

where we have denoted by Z the Fourier symbol of the operator $\bar{P}_h$,

$$Z = -i\tau(\sin k_1 A_1 + \sin k_2 A_2). \tag{24}$$

Let μ_m, $m = 1, 2, 3, 4$ be the eigenvalues of the matrix Z. Then the eigenvalues λ_m, $m = 1, 2, 3, 4$, of the matrix G (22) have the form

$$\lambda_j = 1 + \mu_j + \alpha_2\mu_j^2 + \alpha_1\alpha_2\mu_j^3,\ j = 1, 2, 3, 4. \tag{25}$$

The matrix Z represents a linear combination of the Jacobi matrices A and B:

$$Z = d_1 A + d_2 B, \tag{26}$$

where with (19) in view

$$\begin{aligned} d_1 &= -\frac{i\tau}{S_{j,k}}[(\sin k_1)(y_{j,k+1} - y_{j,k}) - (\sin k_2)(y_{j+1,k} - y_{j,k})], \\ d_2 &= -\frac{i\tau}{S_{j,k}}[-(\sin k_1)(x_{j,k+1} - x_{j,k}) + (\sin k_2)(x_{j+1,k} - x_{j,k})]. \end{aligned} \tag{27}$$

The eigenvalues μ_j of the matrix (24) are known to have the form [5]

$$\begin{aligned} \mu_1 = \mu_2 = d_1 u + d_2 v, \quad \mu_3 = \mu_1 + c(d_1^2 + d_2^2)^{0.5}, \\ \mu_4 = \mu_1 - c(d_1^2 + d_2^2)^{0.5}, \end{aligned} \tag{28}$$

where c is the sound velocity in gas.

Thus, we have found the analytical expressions for the eigenvalues $\lambda_1, \dots, \lambda_4$ of the Fourier symbol G (23) of the Runge–Kutta scheme (20) on a curvilinear grid. This has become possible owing to the fact that the matrix G is a polynomial in some other matrix Z (see (23)). As we have shown in [5] many difference schemes for the Euler equations do not possess this property, and then the expressions for λ_j cannot be found in analytic form. In addition, in the case of difference schemes approximating the Navier–Stokes equations for a compressible fluid, there is no similarity matrix T diagonalizing both the Jacobi matrices A and B and the matrices arising from the approximation of viscous terms. For such cases we have proposed several strategies of symbolic/numeric stability analysis [5, 8].

In the next two Sections, we propose a new parallel symbolic/numeric algorithm for the stability analysis of difference schemes on curvilinear grids. For the validation of our symbolic/numeric method we will use the analytical von Neumann stability condition, which can be obtained by using the analytic expressions (25) and (28). In order to obtain this condition let us at first introduce the square region

$$\Pi = \{(k_1, k_2) |\ 0 \le k_l \le 2\pi,\ l = 1, 2\} \tag{29}$$

The value 2π represents here the period of the entries of the amplification matrix G (23) in k_1 and k_2. The coefficients of the characteristic equation $\det(\lambda I - G) = 0$ may have another period, which however does not exceed the period of the entries of G, because these coefficients are polynomials in the entries of G.

It follows from (27) that all the eigenvalues μ_j, $j = 1, \dots, 4$, are purely imaginary. Then the von Neumann stability condition [6] $|\lambda_j| \le 1$, $j = 1, \dots, 4$ implies:

$$\tau \cdot \max_{\vec{k} \in \Pi} [\varphi^+(\xi_x, \xi_y, \eta_x, \eta_y, \vec{u}_0, \vec{k}),\ \varphi^-(\xi_x, \xi_y, \eta_x, \eta_y, \vec{u}_0, \vec{k})] \le C, \tag{30}$$

where $\vec{k} = (k_1, k_2)$,

$$\begin{gathered} \varphi^\pm(\xi_x, \xi_y, \eta_x, \eta_y, \vec{u}_0, \vec{k}) = \\ |\xi_x u \sin k_1 + \eta_x u \sin k_2 + \xi_y v \sin k_1 + \eta_y v \sin k_2 \pm \\ \pm [(c\xi_x \sin k_1 c\eta_x \sin k_2)^2 + (c\xi_y \sin k_1 + c\eta_y \sin k_2)^2]^{0.5}|, \end{gathered} \tag{31}$$

C is the Courant number; $C = 2$ in the case of the three-stage scheme

(20) and $\alpha_1 = \alpha_2 = 0.5$. We have introduced in (31) the notations

$$\begin{aligned} \xi_x &= \frac{y_{j,k+1} - y_{j,k}}{S_{j,k}}, \quad \eta_x = -\frac{y_{j+1,k} - y_{j,k}}{S_{j,k}}, \\ \xi_y &= -\frac{x_{j,k+1} - x_{j,k}}{S_{j,k}}, \quad \eta_y = \frac{x_{j+1,k} - x_{j,k}}{S_{j,k}}. \end{aligned} \tag{32}$$

One can show that formulas (32) represent the difference approximations of the derivatives $\partial\xi/\partial x$, $\partial\eta/\partial x$, $\partial\xi/\partial y$, $\partial\eta/\partial y$ [5, 8].

It can be shown (see also [10]) that the left-hand side of the von Neumann stability condition (30) remains the same for Runge–Kutta type schemes with different number m of intermediate stages ($m \geq 1$), only the value of the Courant number C on the right hand side of (30) is different for different Runge–Kutta schemes in the absence of added artificial dissipators; for example, $C = 4$ for the optimal five-stage Runge–Kutta scheme.

It is easy to show (cf. [5]) that the functions $\tau\varphi^{\pm}(\cdot)$ defined by (31) involve the following nondimensional variables:

$$\begin{aligned} &\kappa_1 = c\tau\xi_x, \ \kappa_2 = u\tau\xi_x, \ \kappa_3 = v\tau\xi_x, \\ &\kappa_4 = \eta_x/\xi_x, \ \kappa_5 = \eta_y/\xi_x, \ \kappa_6 = \xi_y/\xi_x. \end{aligned} \tag{33}$$

We can rewrite equation (31) in terms of the introduced variables (33) as follows:

$$\begin{aligned} \tau\varphi^{\pm}(\vec{\kappa}, \vec{k}) = |&\kappa_2 \sin k_1 + \kappa_2 \sin k_2 + \kappa_3\kappa_6 \sin k_1 + \kappa_3\kappa_5 \sin k_2 \pm \\ &\pm [(\kappa_1 \sin k_1 + \kappa_1\kappa_4 \sin k_2)^2 + (\kappa_1\kappa_6 \sin k_1 + \kappa_1\kappa_5 \sin k_2)^2]^{0.5}|, \end{aligned} \tag{34}$$

where $\vec{\kappa} = (\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6)$.

It follows from (34) and (30) that the functional form of the necessary stability condition of Jameson's schemes does not depend on the specific grid topology, and in this sense the condition (30) is universal. However, the local time steps $\tau_{j,k} = \tau(x_{jk}, y_{jk})$ in a specific grid node (x_{jk}, y_{jk}) will of course depend both on the grid topology and on grid density in the neighborhood of a given grid node, because in accordance with (30)

$$\tau_{jk} \leq \frac{C}{\max_{\vec{k}\in\Pi}\{\varphi^{\pm}[(\xi_x)_{jk}, (\xi_y)_{jk}, (\eta_x)_{jk}, (\eta_y)_{jk}, \vec{u}_0, \vec{k}]\}}. \tag{35}$$

Since $\max_{k_m} |\kappa_j \sin k_m| = |\kappa_j|$, $j = 1, \ldots, 6$, $k_m \in \Pi$, $m = 1, 2$, it follows from (35) and (34) that the local von Neumann necessary stability condition of the linearized Jameson's scheme (20) on a general curvilinear grid has the form

$$|\kappa_2| + |\kappa_2\kappa_4| + |\kappa_3\kappa_6| + |\kappa_3\kappa_5| + \\ + [(|\kappa_1| + |\kappa_1\kappa_4|)^2 + (|\kappa_1\kappa_6| + |\kappa_1\kappa_5|)^2]^{0.5} \le C. \quad (36)$$

Let us now consider also the particular case of an uniform rectangular spatial grid:

$$x_{j+1,k} = x_{j,k} + h_1,\ x_{j,k+1} = x_{j,k}, \quad y_{j+1,k} = y_{j,k}, \\ y_{j,k+1} = y_{j,k} + h_2 \quad \forall j, k, \quad (37)$$

where h_1 and h_2 are the grid steps along the x-and y-axes, respectively. We now look at the form of the variables (33) in this particular case. For this purpose we substitute (37) in (32). As a result we obtain:

$$\kappa_1 = \frac{c\tau}{h_1}, \quad \kappa_2 = \frac{u\tau}{h_1}, \quad \kappa_3 = \frac{v\tau}{h_1}, \quad \kappa_4 = 0, \quad \kappa_5 = \frac{h_1}{h_2}, \quad \kappa_6 = 0.$$

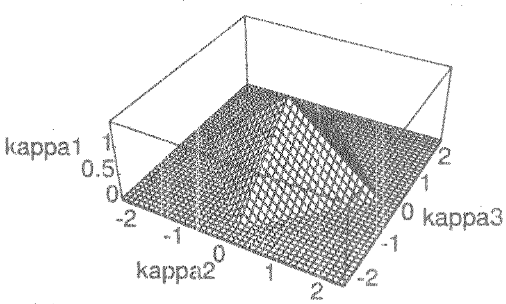

Fig. 3. The necessary stability region of Jameson's scheme (20) on an uniform rectangular grid.

Therefore, the general necessary stability condition (36) simplifies for the three-stage scheme (20) to the condition

$$|\kappa_2| + |\kappa_3|\kappa_5 + \kappa_1\sqrt{1+\kappa_5^2} \le 2. \quad (38)$$

At the specified cell aspect ratio κ_5 the equality

$$\kappa_1 = \frac{2 - |\kappa_2| - |\kappa_3|\kappa_5}{\sqrt{1+\kappa_5^2}} \quad (39)$$

thus determines a surface $\kappa_1 = f(\kappa_2, \kappa_3)$ of the necessary stability region of Jameson's scheme (20) on an uniform rectangular grid. In particular, at the apex of the pyramid (39), where $\kappa_2 = \kappa_3 = 0$, we have

$$\kappa_1 = \frac{2}{\sqrt{1+\kappa_5^2}} \tag{40}$$

see also Fig. 3.

3. Parallel Implementation of Symbolic Stages with *Mathematica*

As we have shown in Section 2, the characteristic equation corresponding to a finite volume three-stage Runge–Kutta scheme may be written in the form

$$\det(\lambda I - G) = \sum_{j=0}^{4} c_j(\vec{\kappa}, \vec{k})\lambda^{4-j} = 0, \tag{41}$$

where $\vec{\kappa} = (\kappa_1, \dots, \kappa_M)$, $M \geq 1$ is the number of nondimensional similarity parameters $\kappa_1, \dots, \kappa_M$ like the parameters (33). The symbolic computation of the entries of the amplification matrix G and of the coefficients $c_j(\vec{\kappa}, \vec{k})$ in (a) involves as a rule very large expenses of computer resourses. It is reasonable to use the data parallelism at the stage of a symbolic computation of the entries of G. There are two reasons for using parallelization at this stage. First, we reduce considerably the total computer time for the computation of the complete matrix G. Second, owing to the data distribution among several workstations we reduce the computer storage needed for each individual workstation for the symbolic computation of a group of entries of the matrix G. Note that the symbolic computation of the total matrix G on a single workstation is in some cases impossible because of a very large needed computer storage.

We have implemented a very simple parallelization strategy. Since each row of the matrix G is computed independently of other rows, we perform the computation of each row of the matrix G in parallel. That is if we have P workstations $(1 < P \leq 4)$, then we compute simultaneously P rows of matrix G by using the memory of each individual workstation only for the computation of one row. In our particular case we have a 4×4 matrix G, therefore, we use 4 workstations (see Fig. 4). After the computation of all entries of G is completed each row is recorded into a file or is returned to a monitoring workstation. Such a simple parallelization of the computation of the matrix G has reduced the computer time for

$$G = \begin{pmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{pmatrix} \begin{array}{l} \leftrightarrow \text{Workstation 1} \\ \leftrightarrow \text{Workstation 2} \\ \leftrightarrow \text{Workstation 3} \\ \leftrightarrow \text{Workstation 4} \end{array}$$

Fig. 4. Parallelization of the symbolic computation of the amplification matrix G.

calculation of G by a factor of more than 2 as compared to the case of using a single workstation.

In the process of parallelization, we have used the *Mathematica 3.0* program `MathLink` and the functions [12]

```
Queue, Wait, ExportEnvironment
```

The function `Queue[cmd]` submits `cmd` for evaluation on a remote kernel.

The function `Wait[id]` waits for the given job to finish and return its result.

The function `ExportEnvironment[s1,s2,...]` exports ownvalues and downvalues of the given symbols to all remote kernels.

Example. Let us calculate in parallel a 2×2 matrix G. We write formally G with *Mathematica* as

```
G={{a11, a12}, {a21, a22}}
```

where the `aij` are only formal notations. We make at first the matrix G to be accessible to each remote kernel by using the above command `ExportEnvironment`:

```
ExportEnvironment[{G}]
```

We further use the command `Queue` as follows: we apply it to each separate row of G:

```
joini = {Queue[G[[1]]/. {a11 -> b11, a12 -> b12}],
  Queue[G[[2]]/. {a21 -> b21, a22 -> b22}]}
```

The specific value `bij` is substituted here for `aij`. After that we send in parallel a command `Wait[...]` for the parallel computation of the rows of G: `joinG = Wait[joini]`. As a result we obtain the complete matrix G, in which each row has been computed in parallel. Thus, we have used a coarse grain parallelization at the stage of symbolic computation of the amplification matrix G. The computed matrix G corresponding to the Jameson's scheme (20) takes 1898 lines of text, with 65 symbols in each line on the average.

4. Parallel Implementation of Numerical Stages

It is in principle possible to solve the characteristic equation (41) in symbolic form for a very simple difference scheme and find the eigenvalues $\lambda_1, \ldots, \lambda_4$ of the given amplification matrix G. The information about the necessary stability region of a difference scheme is then obtained from the von Neumann necessary stability conditions [6]: $|\lambda_\alpha| \leq 1 + O(\tau), \quad \alpha = 1, \ldots, 4$. However, for sufficiently complex difference schemes the expressions for λ_α prove to be huge algebraic expressions, and it is impossible to obtain from them the necessary stability condition in analytic form.

In this connection we have to determine numerically the coordinates of points of the necessary stability region (NSR) boundary. We now describe a mathematical formulation of a problem on a search for the NSR boundary points. We will use an optimization approach, which we have described previously in [5].

Let us denote by Π_h a rectangular grid which covers the square region (29). In accordance with (35) the eigenvalues of G should be computed in all nodes of Π_h. Following [5] we introduce the functions

$$g_j(\vec{\kappa}, \vec{k}) = 1 - |\lambda_j(\vec{\kappa}, \vec{k})|^2, \qquad j = 1, \ldots, 4. \tag{42}$$

Then the von Neumann necessary stability criterion reduces to the system of inequalities

$$g_j(\vec{\kappa}, \vec{k}) \geq 0, \; j = 1, \ldots, 4.$$

Let us asume that the left-hand sides of these inequalities are periodic functions in k_1 and k_2 with periods T_1 and T_2 respectively, and let

$$\Pi_2 = \{(k_1, k_2) |\, 0 \leq k_m \leq T_m, \; m = 1, 2\}. \tag{43}$$

Let us choose the quantity κ_1 as a parameter and suppose that the functions

$$\kappa_1 = \kappa_1, \; \kappa_j = \varphi_j(\kappa_1), \; j = 2, \ldots, 6 \tag{44}$$

determine parametrically some smooth curve $\mathcal{L}$ in E^6 that intersects the boundary Γ of the necessary stability region D of a difference scheme under study. Let

$$\tilde{g}_k(\kappa_1, \vec{k}) = g_k(\kappa_1, \varphi_2(\kappa_1), \ldots, \varphi_6(\kappa_1), \vec{k}), \quad k = 1, \ldots, 6. \tag{45}$$

We now form the following binary function $f(\kappa_1)$:

$$f(\kappa_1) = \begin{cases} 1, & \tilde{g}_m(\kappa_1, \vec{k}) \geq 0 \ \forall \vec{k} \in \Pi_2, \quad m = 1 \ldots, 4 \\ -1 & \text{otherwise} \end{cases} \tag{46}$$

Then the value $\kappa_1^0 \in \Gamma$, if the function $f(\kappa_1)$ has a jump at point $\kappa_1 = \kappa_1^0$ and changes its sign.

We have determined $\kappa_1^0 \in \Gamma$ numerically as follows. Let $\kappa_1^*(k_2)$ be such a value at fixed $k_2 \in [0, T_2]$ that $\tilde{g}_k(\kappa_1^*(k_2) + \delta, k_1, k_2) \leq 0$ at least for some $k, 1 \leq k \leq 4$, and for some $k_1 \in [0, T_1]$, and any small positive $\delta > 0$. The point $\kappa_1^0 \in \Gamma$ is then determined as

$$\kappa_1^0 = \min_{k_2 \in [0, T_2]} \kappa_1^*(k_2). \tag{47}$$

This means that we at first determine $\kappa_1^*(k_2)$ at a fixed value of $k_2 \in [0, T_2]$ by varying k_1 in the interval $[0, T_1]$. In practical computations the lines $k_2 = \text{const}$ were the lines of one family of grid lines of a rectangular region Π_2 (43).

The accuracy of the computation of value $\kappa_1^0 \in \Gamma$ depends on the accuracy with which the zeroes $\lambda_j(\vec{\kappa}, \vec{k})$ of the characteristic equation (41) are computed. As was shown in [5], the arithmetic of floating-point machine numbers becomes insufficient in the case of complex entries of G. The reason for this is a rapid accumulation of roundoff errors. Two methods were previously used in [5] to reduce the efect of roundoff errors:

(i) the balancing of matrices and the arithmetic of stored orders;

(ii) the use of double precision machine arithmetic.

However, it was stressed in [5] that the double precision arithmetic can also lead to large errors when sufficiently complex difference schemes are to be analyzed.

The system *Mathematica* contains a number of numerical functions to perform the numerical computations with high accuracy: these are the functions

```
N[expr,n], SetPrecision[expr,n], SetAccuracy[expr,n].
```

It was shown in [13] that these numerical functions yield somewhat lower accuracy than that specified by the argument `n` for certain ranges of n. In this connection it was proposed in [13] to use the arithmetic of rational numbers. In this method, the trigonomeric functions `Sin[x]`, `Cos[x]` are approximated by rational numbers with the aid of the built-in *Mathematica* function `Rationalize[expr,eps]`:

```
si[xi_, n_]:= si[xi,n] =
Rationalize[N[Sin[xi], n+1], 10^-(n+1)];
co[xi_, n_]:= co[xi,n] =
Rationalize[N[Cos[xi], n+1], 10^-(n+1)]
```

The second argument `eps` of the function `Rationalize[exp,eps]` denotes the maximum difference of the rational value from the exact value rather than the number of digits. If one enters `eps=0`, then *Mathematica* tries to attain the best possible fraction.

The values of $\kappa_1, \dots, \kappa_6$ in the coefficients $c_k(\vec{\kappa}, \vec{k})$ were also specified as rational numbers. As a result, all the coefficients $c_k(\vec{\kappa}, \vec{k})$ were the rational numbers. At given rational values of the coefficients c_k, the characteristic equation (41) was solved exactly with the aid of the *Mathematica* function `Solve[]`.

Similarly to [5] we have used the bisection method to find the value $\kappa_1^*(k_2)$ at a fixed value of k_2. According to (47) the value of $\kappa_1^*(k_2)$ is determined at various $k_1 \in [0, T_1]$ as the maximum value for which still $f(\kappa_1) = 1$ holds. Since the above bisection process should be applied for any value of k_1 in $[0, T_1]$, much computer time can be required to find the value $\kappa_1^*(k_2)$. In this connection we have implemented a two-step process for determining the value κ_1^0 in accordance with (47).

Step 1. A rough computation of κ_1^0. Let us assume that we want to compute the value of κ_1^0 at first with the accuracy 10^{-n2}, where $1 \le n2 < exact$, *exact* is here user-specified desired number of correct digits in the mantissa of the result. Let us call a partition of the interval $[0, T_1]$ into 2^n equal subintervals the nth partition, $n = 0, 1, 2, \dots$. At the zeroth partition we take the interval $[0, T_1]$ itself. At the first partition we already have two intervals $[0, T_1/2]$ and $[T_1/2, T]$, etc. Let us denote by $\kappa_1^{(n)}$ the minimum value of κ_1 at the n th partition, which is obtained at fixed values of k_1 in the nodes of the nth partition, and let $\bar{\kappa}^{(n)}$ be the minimum value of κ_1 over the set of remaining nodes in the interval $[0, T_1]$. If $|\kappa_1^{(n)} - \bar{\kappa}_1^{(n)}| < 10^{-n2}$, the process of the further partition of the interval $[0, T_1]$ is terminated. It is easy to find that the nth partition adds only 2^{n-1} new nodes to the $(n-1)$th partition. This has enabled us to store the values of $\kappa_1^*(k_1)$ at the foregoing partition in the nodes of a cruder mesh on the interval $0 \le k_1 \le T_1$. This saves a lot of computing time. The value of $\kappa_1^*(k_1)$ is determined for a fixed value of $k_2 \in \Pi_2$ with the aid of the function `NSuchex[...]` written by us.

The value of $\kappa_1^*(k_2)$ in the interval $0 \le k_2 \le T_2$ is determined in a similar way by using the partition of this interval into 2^n equal subintervals. This computation is executed by the function `NaehSuche[...]`. Let us denote by `[k2min, k2max]`$\subset [0, T_2]$ the interval in which the value $\kappa_1^*(k_2)$ is located.

Step 2. Accurate computation of κ_1^0. At this step we compute the value of κ_1^0 in accordance with (48) with the high accuracy 10^{-exact}. Here we use the bisection process in the interval `[k2min, k2max]` found at Step 1. As a result we find a point $(\kappa_1^0, \varphi_2(\kappa_1^0), \ldots, \varphi_6(\kappa_1^0))$ of the necessary stability region boundary for a given difference scheme.

At the parallel implementation of the above described numerical stages of our algorithm we have used the multilevel parallelization. The schematic diagram of this parallel algorithm is shown in Fig. 5. We now describe in detail each parallelization level.

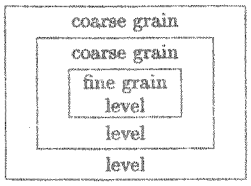

Fig. 5. Schematic description of parallel numerical algorithm.

On the first level, we use a coarse grain parallelization to compute $\kappa_1^*(k_2)$ in accordance with (47) while k_2 is varied from 0 to T_2. In the program `NaehSuche[...]`, the external Do-loop over k_2 is executed. In the process of parallelization, we split our external Do-loop into `nx` parts. Each such part is executed in parallel independently of each other. Our interval $0 \leq k_2 \leq T_2$ is partitioned at first into two subintervals $[0, T_2/2]$ and $[T_2/2, T_2]$, that is we are using only two processors. After that, each of the above two subintervals is partitioned into two smaller and equal subintervals, so that the total number of subintervals in the interval $[0, T_2]$ becomes equal to 2^2. Here we already use 4 processors.

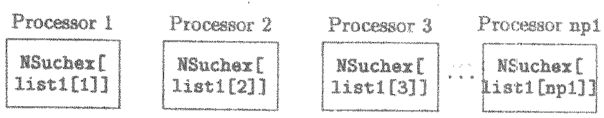

Fig. 6. Parallel process diagram.

The values of the function `NSuchex[...]` are computed within the above Do-loop in parallel. The overall parallel process on `np1` processors can be presented as in Fig. 6. This parallelization is performed with the aid of *Mathematica* function `ParCsi2[yt, np1]` written by us. This function is called in the body of the function `NaehSuche[...]` as follows:

```
......
yt = Table[ngrn, si[sci2, ngrb],...];
y = ParCsi2[yt, np1];
ymin = Min[y];
......
```

The function `ParCsi2[yt, np1]` depends on two parameters `yt` and `np1`. The parameter `np1` specifies the number of the processors used; it can be varied from 2 to `nx`, where `nx = 2*nx`, that is no complete list is assigned, but only 2, or 4, or 8, etc., elements of this list. Example: if we specify the value `np1 = 2`, then each time a list portion is taken, which consists of two function values, which will be computed in parallel on 2 processors, etc.

The parameter `yt` specifies the list

```
list1[i]={ngr1, si[k2[i], n], co[k2[i], n]}, i=1,..., np1,
```

where `ngr1` is the user-specified accuracy of the determination of the values of $\kappa_1, \ldots, \kappa_M$ at the NSR boundary; that is these values will be determined with an absolute error, which does not exceed 10^{-ngr1}.

We also transfer in parallel each time the `np1` function values `NSuchex[...]`. This gives us the possibility to fix each time the number of processors, which are used for the computation in the external Do-loop. The use of such a fixed number of processors for each parallelization level does not lead to a loss of the data. This also enables other functions from different stages, which work more quickly, to take the idle processors, at an overall parallelization of all the numerical stages of our algorithm.

The parallelization itself can be described by the *Mathematica* command

```
yy0 = Apply[yy0, ParMapOwn[Apply[NSuchex, #]&, list1]]
```

where the function `ParMapOwn[...]` sends the function `NSuchex[list1 [i]]` to the idle processors.

On the second level, we use a coarse grain parallelization too for the numerical determination of κ_1 at a fixed value of k_2. We vary the parameter k_1 within the limits $0 \leq k_1 \leq T_1$ (see above Step 1 of our numerical algorithm). Since we can do the calculations as on the first parallelization level independently for each fixed k_1, we use the same strategy as on the first level. This internal Do-loop is executed by the program `NSuchex[...]`, in which the program `Bisec[...]` is called in parallel. This program solves numerically equation $f(\kappa_1) = 0$, where the function $f(\kappa_1)$ is defined according to (46). As at the first parallelization stage, a parallel call of the function `Bisec[...]` for each fixed value of k_1 can be presented schematically for the case of `np2` processors as shown in Fig. 7. This parallel process is performed by the program `ParCsi1[xt, np2]`. We now present a fragment of our program `NSuchex[...]`, in which the program `ParCsi1[...]` is called:

```
......
xt = Table[ngn, si[csi1, ngn], co[csi1, ngn],...
x = ParCsi1[xt, np2];
xmin = Min[x];
......
```

Fig. 7. The parallel implementation of the bisection method.

The parameter `np2` specifies the number of processors, and the parameter `xt` specifies the list

```
list2[i]={ngr1, si[k1[i], n], co[k1[i], n]}, i=1,..., np2
```

We subdivide our interval $[0, T_1]$ into the subintervals as this was described above at Step 1.

In the body of the program `ParCsi1[...]`, the parallelization is performed by the command

```
xx0 = Apply[xx0, ParMapOwn[Apply[Bisec, #]&, list2]
```

that is the parallelization of this loop proceeds in portions as at the first level.

At the third level, the bisection algorithm is parallelized. A complete description of the parallel algorithm for the bisection method may be found in [14]. We only briefly describe the main idea and the peculiarities of implementation with *Mathematica*. The main idea of the parallel algorithm is rather simple. We have `np3` processors for this stage. We partition our interval $I = [a, b]$ into `p = np3 +1` subintervals rather than into two equal subintervals as this is usually done in the bisection method. We have `p` points of the partition. The function values are computed in parallel for each point. After that we search for a new interval $I_k = [a_k, b_k]$ where the function changes its sign, that is where $f(a_k)f(b_k) < 0$, and this new interval is also partitioned into `np3` intervals, and the function values are computed in parallel; the function values which were previously computed at the boundary nodes are not computed repeatedly, and the overall bisection process is repeated until the zero of the function is determined with a user-specified accuracy. At this stage we apply a fine grain parallelization, that is a very frequent parallelization in view of multiple evaluations of the function to determine its zero.

At each partition of the interval I_k we arrange the nodes of the partition into a list whose length is equal to `np3`. The computation of the function values in these nodes is executed in parallel with the aid of the command `ParMapOwn[...]`, which has the following form:

```
ParMapOwn[f_, a1, a2, ... , an, np3]:=
```

where the $f(a1), f(a2), \ldots$ are computed in parallel at a fixed number of the processors determined by the parameter `np3`. This parameter is specified in order that the above program does not send the command to compute the values of f to the idle processors, which are to be used at other stages. In this way it is ensured that the program `ParMapOwn[...]` does not impose any limitations on other programs from other parallelization levels and uses only the `np3` processors. This strategy enables us to control at each stage the number of idle processors and to distribute them more effectively between different parallelization levels of our algorithm.

We can estimate the speed-up of computation by the bisection method with the aid of estimates presented in [15]: $S_p = k_1/k_p$, where k_i is the number of the interval partitions needed in the case of using i processors, and S_p is the speed-up factor. The following estimate is further obtained in [15]: $S_p = O(\log_2 p)$. It follows from here that the above method of the parallelization of the bisection algorithm gives a considerable gain in computer time when the function whose zero is to be found is a complex expression. A more considerable gain will be achieved in the case of three spatial variables in the Euler equations; the expression for the function will then be much more complex than in the two-dimensional case.

If the function is simple then the increase in the number of processors does not give any significant advantages, because it leads to the increase in the portion of time of sending the data and retrieving the results (these operations are slow in the case of using the `MathLink`) as compared to the total time of computing the function zero by the parallel bisection method. As was noted above, the actual gain is achieved through parallelization in the case of a complex function whose values are multiply computed in parallel. Then the increase in the number of processors immediately results in a considerable speed-up of the algorithm.

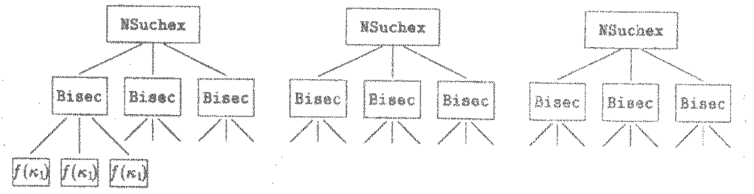

Fig. 8. The parallelization tree.

We can now present the general structure of all three parallelization stages in the form of a tree shown in Fig. 8. At stages 1,2, and 3 we use `np1`, `np2`, and `np3` processors, respectively.

For the validation of the proposed parallel numerical method we have considered the stability of scheme (20) in the particular case of an uniform rectangular spatial grid. Then we must take the values $\kappa_4 = 0$ and $\kappa_6 = 0$ in the coefficients of the characteristic equation. In order to be able to use the analytic formula (39) we have taken the values $\kappa_2 = \kappa_3 = 0$ (see the top of the pyramid in Fig. 3). We have performed a number of runs with `n2`=1 and $exact = 2$ at different values of the cell aspect ratio κ_5. We have used the following notations: κ_{1num} is the value of κ_1 on the boundary of the necessary stability region obtained by the above presented numerical method;

$$\kappa_{1ex} = \frac{2}{\sqrt{1+\kappa_5^2}};$$

and $\delta\kappa_1$ is the absolute error, $\delta\kappa_1 = |\kappa_{1ex} - \kappa_{1num}|$. It can be seen from Table 1 that the absolute error $\delta\kappa_1$ is smaller by an order of magnitude than the user-specified accuracy 10^{-2}. The proposed numerical algorithm thus computes the coordinates of points of the necessary stability region boundary with a guaranteed accuracy. This is explained by the fact that the arithmetic operations on rational numbers are exact in *Mathematica*. The *Mathematica* function `Solve[]` also finds the exact analytic expressions for the zeroes of a fourth-degree polynomial with rational coefficients.

Table 1. Values of κ_1 at point $\kappa_2 = \kappa_3 = 0$ of the stability region boundary of scheme (20) at different values of κ_5.

$\kappa_1 \setminus \kappa_5$	$\frac{1}{2}$	1	2	3
κ_{1num}	$\frac{1831}{1024}$ ≈ 1.788086	$\frac{5793}{4096}$ ≈ 1.414307	$\frac{7325}{8192}$ ≈ 0.894165	$\frac{971}{1536}$ ≈ 0.632162
κ_{1ex}	1.788854	1.414214	0.894427	0.632456
$\delta\kappa_1$	0.000768	0.000093	0.000262	0.000294

Table 2. Values of κ_1 at $\kappa_2 = \kappa_3 = \kappa_6 = 0$ on the necessary stability region boundary of scheme (20) at different values of κ_4 and κ_5.

$\kappa_1 \setminus \kappa_4, \kappa_5$	$\kappa_4 = 1, \kappa_5 = 2$	$\kappa_4 = 2, \kappa_5 = 1$	$\kappa_4 = 3, \kappa_5 = 3$	$\kappa_4 = -\frac{1}{2}, \kappa_5 = -\frac{1}{3}$
κ_{1num}	$\frac{915}{1024}$ ≈ 0.893555	$\frac{971}{1536}$ ≈ 0.632162	$\frac{409}{1024}$ ≈ 0.399414	$\frac{5331}{4096}$ ≈ 1.301514
κ_{1ex}	0.894427	0.632456	0.400000	1.301583
$\delta\kappa_1$	0.000873	0.000294	0.000586	0.000069

In Table 2 we present the numerical results obtained for the case of a curvilinear spatial grid. The notations κ_{1num} and $\delta\kappa_1$ have the same meaning as in Table 1;

$$\kappa_{1ex} = \frac{2}{\sqrt{(1 + |\kappa_4|)^2 + \kappa_5^2}}.$$

The numerical data of Table 2 also confirm the correctness of the analytical formula (36). In the last variant with $\kappa_4 = -\frac{1}{2}, \kappa_5 = -\frac{1}{3}$ the value of κ_{1num} was found at $k_1 = \frac{3\pi}{2},\ k_2 = \frac{\pi}{2}$.

In Fig. 9 we show the necessary stability region of scheme (20) in the section $\kappa_2 = 0, \kappa_4 = 1, \kappa_6 = 1$. This case also corresponds to the curvilinear grid.

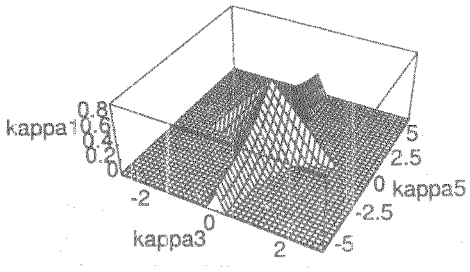

Fig. 9. The necessary stability region of Jameson's scheme on the curvilinear grid.

5. Conclusion

The purpose of this study was to investigate a possibility of achieving a speed-up of the symbolic-numeric stability analysis programs written in *Mathematica* by implementing the parallelization both at the symbolic and numerical stages of the algorithm. We have presented a parallelized version of our symbolic-numerical algorithm. We can draw a conclusion that the computer algebra system *Mathematica 3.0* can be used successfully for the development of complex parallel codes.

References

1. H. Lomax and T. H. Pulliam, *A fully implicit, factored code for computing three-dimensional flows on the ILLIAC IV. Parallel Computations* Ed. G. Rodrigue. Academic Press, New York, London, 1982, p. 217–250.
2. W. D. Gropp and E. B. Smith, *Computational fluid dynamics on parallel processors. Computers and Fluids*, **18** (1990), 289–304.
3. L. Giraud and G. Manzini, *Parallel implementation of 2D explicit Euler solvers. J. Comput. Phys.*, **123** (1996), 111–118.
4. R. W. MacCormack, *On efficient numerical methods for solving the Navier–Stokes equations in three dimensions. The Second Japan-Soviet Union Joint Symposium on Computational Fluid Dynamics, August 27-31, 1990, Tsukuba, Japan* Eds. Y. Yoshizawa and K. Oshima. Proceedings, Vol. II. Published by the Japan Society of Computational Fluid Dynamics, Tsukuba, 1991, p. 1–13.

5. V. G. Ganzha and E. V. Vorozhtsov, *Computer–Aided Analysis of Difference Schemes for Partial Differential Equations.* Wiley-Interscience, New York, 1996.
6. J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1989.
7. P. Knupp and S. Steinberg, *Fundamentals of Grid Generation.* CRC Press, Boca Raton, Ann Arbor, 1994.
8. V. G. Ganzha and E. V. Vorozhtsov, *Numerical Solutions for Partial Differential Equations: Problem Solving Using Mathematica.* CRC Press, Boca Raton, New York, London, 1996.
9. A. Jameson, W. Schmidt, and E. Turkel, *Numerical solution of the Euler equations by finite volume methods using Runge–Kutta time stepping schemes.* AIAA Paper No. 81-1259, 1981.
10. V. G. Ganzha and E. V. Vorozhtsov, *On the stability of Jameson schemes. Bridging Mind and Model: Papers in Applied Mathematics* ed. P.J. Costa. St. Thomas Technology Press, St. Paul, MN, 1993, p. 237-300.
11. A. Jameson and W. Schmidt, *Some recent developments in numerical methods for transonic flows. Computer Methods in Applied Mechanics and Engineering,* **51** (1985), 467–493.
12. R. Maeder, *Parallel Programming in Mathematica.* IMS'97.
13. V. G. Ganzha, K. L. Schaub, and E. V. Vorozhtsov, *On accurate computation of stability regions of difference schemes for fluid dynamics problems.* In : *Internat. Conf. on the Methods of Aerophysical Research, September 2-6, 1996, Novosibirsk, Russia.* Proceedings, Part I. Published by the Institute of Theoretical and Applied Mechanics, Novosibirsk, 1996, p. 96-101.
14. G. S. Schedler, *Parallel numerical methods for the solution of equations. Comm. ACM,* **10** (1967), 286–291.
15. U. Schendel, *Einführung in die Parallelnumerik.* Oldenbourg Verlag, München, Wien, 1981.

Institute of Informatics,
Technical University of Munich,
Munich 80290, Arcisstr. 21,
Germany; e-mail:
ganzha@informatik.tu-muenchen.de

Поступило 1999 г.

Institute of Theoretical and
Applied Mechanics,
Russian Academy of Sciences,
Novosibirsk 630090, Russia;
e-mail: vorozh@itam.nsc.ru