algorithms

MDPI

Article

Vectorized Numerical Algorithms to Solve Internal Problems of Computational Fluid Dynamics

Konstantin Volkov

Department of Mechanical Engineering, Kingston University, London SW15 3DW, UK; k.volkov@kingston.ac.uk

Abstract: The opportunities provided by new information technologies, object-oriented programming tools, and modern operating systems for solving boundary value problems in CFD described by partial differential equations are discussed. An approach to organizing vectorized calculations and implementing finite-difference methods for solving boundary value problems in CFD is considered. Vectorization in CFD problems, eliminating nested loops, is ensured through the appropriate data organization and the use of vectorized operations with arrays. The implementation of numerical algorithms with vectorized mesh structures, including access to internal and boundary mesh cells, is discussed. Specific examples are reported and the implementation of the developed computational algorithms is discussed. Despite the fact that the capabilities of the developed algorithms are illustrated by solving benchmark CFD problems, they enable a relatively simple generalization to more complex problems described by three-dimensional equations.

Keywords: computational fluid dynamics; boundary value problem; vectorized algorithm; programming

Citation: Volkov, K. Vectorized Numerical Algorithms to Solve Internal Problems of Computational Fluid Dynamics. *Algorithms* **2024**, *17*, 50. https://doi.org/10.3390/a17020050

Academic Editor: Bogdan Dumitrescu

Received: 29 November 2023
Revised: 8 January 2024
Accepted: 19 January 2024
Published: 23 January 2024

1. Introduction

Modern computational technologies are represented by general-purpose and subject-specific software packages. Research and modeling of multiphysics processes in technological systems is inextricably linked with the use of such technologies. The following ways to further improve performance are possible: parallelization at the algorithm level, which will lead to a reduction in the number of synchronization operations and lower costs for thread dispatching; and implementation on a specialized device (for example, GPGPU).

The development of models and algorithms for performing multivariant analysis of various structures at the stages of exploratory research, in which the efficiency of obtaining results plays an important role, is becoming quite important [1,2]. At the same time, a significant part of the computational resources is devoted to solving systems of linear algebraic equations (SAE), where about 40% of processor time is spent on solving SAE. The creation of algorithms that increase the efficiency of solving SAE is one of the main directions for increasing the performance of software packages. Vectorization of calculations is one of the promising optimization methods, along with the implementation of parallel computing algorithms [3,4].

Modern programming languages (for example, C++) enable creating vectorized algorithms, the effectiveness of which is evident with the appropriate hardware. Vectorization is one of the ways to perform parallel computing, in which the program is modified in a certain way to perform several similar operations simultaneously [5,6]. This approach potentially leads to a significant acceleration of similar calculations over large datasets. To improve program execution speed, MATLAB 5 uses loop vectorization, pre-allocation of arrays in memory, removal of unnecessary variables and functions, and memory defragmentation [7,8]. MATLAB capabilities for implementing parallel computing remain beyond the scope of many studies [9]. MATLAB-vectorized computations using large arrays may be good candidates for GPU acceleration [10,11].

Many problems in CFD, when mathematically modeled, are reduced to boundary value problems for systems of differential equations. Meanwhile, the numerical solution

of the Cauchy problem is well formalized and provided by reliably developed methods for finding an approximate solution, and a wide range of these methods (Euler, Runge–Kutta, and others) are presented in many packages of computational mathematics (for example, MATLAB). Boundary value problems represent a more difficult object to study. Solving many of them requires an individual approach and selection of the most effective solution method [12,13].

The speed of an algorithm is critical in determining its reliability, particularly in real-time applications. To ensure that the algorithm runs as quickly as possible, it is important to use vectorized mathematical functions for fast operations on large arrays of data without the need for explicit loops. Programming languages (MATLAB/GNU Octave or Python) that implement vectors and matrices may have easy means for vectorization. Many operations are implemented in a way that eliminates the overhead of loops, pointer indirection, and per-element dynamic type checking. Loops in MATLAB are slower compared to languages such as C/C++; one of the reasons is MATLAB's dynamically typed nature. During each iteration, MATLAB has to perform a series of checks, such as determining the type of variable, resolving its scope, and checking for invalid operations. In C/C++, arrays consist of one data type, which the compiler knows in advance. This is why loops in MATLAB are often slower than in C/C++, and nested loops can significantly slow down the performance.

Many CFD solution procedures are extensively used in solutions of Euler or Navier–Stokes equations. However, these procedures are computationally demanding. One reason for this is the slow convergence rates of the numerical relaxation algorithms that are commonly used. It is shown that the convergence rates are greatly enhanced by using multigrid methods with a conventional scalar processor [14].

The structure of numerical algorithms and their implementation on vector computers are widely discussed. There are many examples of vectorized numerical algorithms that enable improving performance of code running on modern CPUs. These examples include adaptive quadrature codes process [15], solution of linear algebraic equations [16,17], integration of ordinary differential equations [18], and convergence acceleration techniques [19]. Applications of various vectorized algorithms for solving CFD problems are reported in [20–24]. Modern GPUs provide additional possibilities for acceleration of computer codes, but their use requires algorithm adaptation to the memory structure [25].

The Godunov method is widely used for solving systems of unsteady equations of gas dynamics. In this method, at each iteration of calculations, the Riemann problem is solved on each face of each cell of the computational mesh to determine the fluxes through these faces. Due to the fact that the dimensions of the computational meshes used for calculations amount to tens and hundreds of millions of cells, the effective use of numerical methods with Riemann solvers requires the use of supercomputers and the use of various approaches regarding parallelization of calculations. The most low-level approach used to increase supercomputer application performance is code vectorization.

Numerical methods based on solving the Riemann problem on the decay of an arbitrary discontinuity are demanding on computing resources. The instruction sets of modern programming languages have a number of features that allow vectorization to be applied to the software context of the Riemann solver, which leads to significant speedup of the solver. Using an example of an exact Riemann solver, a practical approach to vectorizing program code for solving CFD problems, including simple linear sections and also nested loops, is considered. The exact Riemann solver has a compact implementation and accommodates a number of features of the software context that require their own techniques when performing vectorization. Examples are provided and implementation of the developed computational algorithms is discussed.

2. Governing Equations

Unsteady supersonic flows of inviscid compressible gas are considered. These flows are described by Euler equations. Finite volume method and time-marching scheme are applied to discretization of Euler equations. Euler equations are written for three independent

variables, time, and two Cartesian coordinates. Generalization of governing equations for three Cartesian coordinates is straightforward.

Governing equations are written in conservative form as

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

where t is the time, x and y are the Cartesian coordinates. Equation of state of perfect gas is

$$p = (\gamma - 1)\rho\varepsilon.$$

Total energy per unit volume is found as

$$e = \rho\varepsilon + \frac{1}{2}\rho(u^2 + v^2).$$

The vector of conservative variables, $\boldsymbol{U}$, and the vectors of fluxes, $\boldsymbol{F}$ and $\boldsymbol{G}$, have the form

$$\boldsymbol{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \quad \boldsymbol{F} = \begin{pmatrix} \rho u \\ \rho u u + p \\ \rho u v \\ (e + p)u \end{pmatrix}, \quad \boldsymbol{G} = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v v + p \\ (e + p)v \end{pmatrix}.$$

Here, ρ is the density, u and v are the velocity components in Cartesian directions x and y, p is the pressure, e is the specific total energy and ε is the specific internal energy, and γ is the ratio of specific heat capacities at constant pressure and constant volume. Euler equations and relevant initial and boundary conditions do not contain any non-dimensional parameters. Euler equations written in dimensional and non-dimensional variables have the same form.

Steady state flows are described with Euler Equation (1). In this case, it is assumed that flow is hyperbolic (velocity magnitude is larger than speed of sound, $u^2 + v^2 > c^2$) in one of the Cartesian directions. For example, hyperbolicity in x direction takes place if $u^2 > c^2$.

3. Numerical Method

For simplicity, a two-dimensional Cartesian mesh is applied. Mesh step sizes in x and y directions are uniform. The value of the vector of conservative flow variables in the center of control volume is $\boldsymbol{U}_{i,j}$, $i = 1, 2, \ldots, M$ and $i = 1, 2, \ldots, N$. A number of control volumes in x and y directions are M and N. The time step size is Δt. The mesh function is constant over control volume.

The numerical algorithm involves reconstruction of flow variables on the faces of control volumes using the averaged values of the vector of flow variables in the centers of control volumes. The reconstruction step is applied to physical variables. The Riemann problem is solved on each face of the control volume. Solutions of the Riemann problem on left and right faces of the control volume are $\boldsymbol{U}_{i-1/2,j}$ and $\boldsymbol{U}_{i+1/2,j}$. Application of the explicit time marching scheme and Godunov method leads to the numerical scheme

$$\frac{\boldsymbol{U}_{i,j}^{n+1} - \boldsymbol{U}_{i,j}^{n}}{\Delta t} + \frac{\boldsymbol{F}_{i+1/2,j}^{n} - \boldsymbol{F}_{i-1/2,j}^{n}}{\Delta y} + \frac{\boldsymbol{G}_{i,j+1/2}^{n} - \boldsymbol{G}_{i,j-1/2}^{n}}{\Delta y} = 0. \tag{2}$$

The superscript $n = 0, 1, \ldots$ is related to the time step.

The flux on the faces of control volume is linearized and found as

$$\boldsymbol{F}_{i+1/2,j} = \frac{1}{2}\left(\boldsymbol{F}_{i,j}^{n} + \boldsymbol{F}_{i+1,j}^{n}\right) + \frac{1}{2}|A|_{i+1/2,j}^{n}\left(\boldsymbol{F}_{i,j}^{n} - \boldsymbol{F}_{i+1,j}^{n}\right),$$

where $A = (\partial \boldsymbol{F}/\partial \boldsymbol{U})^{-1}$ is Jacobian.

The stability condition for Equation (1) is

$$C = \max |\lambda_{\pm}| \frac{\Delta x}{\Delta y} \leqslant 1.$$

Eigenvalues of the Jacobian are

$$\lambda_{\pm} = \frac{uv \pm c(u^2 + v^2 - c^2)^{1/2}}{u^2 - c^2}.$$

The speed of sound is

$$c = \left[(\gamma - 1)h - \frac{1}{2}(\gamma - 1)(u^2 + v^2) \right]^{1/2},$$

where h is the total enthalpy.

A three-step Runge–Kutta scheme is applied to integration in time direction. Godunov method uses the analytical solution of the Riemann problem. The exact solution consists of two waves, a shock wave or a simple rarefaction, and a tangential discontinuity. These discontinuities are separated from each other by uniform flow zones. Numerical scheme (2) is the scheme of the first order of accuracy. Higher-order accuracy schemes are created using piece-wise distributions of flow quantities in the control volume and interpolation procedure to integrate Euler equations in marching direction.

4. Addressing Mesh Cells

A structured mesh covering the computational domain has a structure similar to a two-dimensional array. This allows to address the cells (or nodes) with two indices, i and j. A mesh node with indices (i, j) has neighbors that are addressed, giving them increments or decrements.

4.1. Vectorization of Loops

Work with mesh structures is usually carried out with cyclic algorithms and nested loops. Using a pseudo programming language, this algorithm is written as

—one-dimensional case

```
for i=1 to nx step 1
  begin
    U(i)=...
  end
```

—two-dimensional case

```
for i=1 to nx step 1
  for j=1 to ny step 1
    begin
      U(i,j)=...
    end
```

Here, `nx` and `ny` are number of nodes in x and y directions. This approach involves calculating an index expression to address the data and store the result in `U` array.

In solution of CFD problems in rectangular fluid domain, two-dimensional data structures correspond to Cartesian coordinates of mesh nodes and flow quantities, such as velocity components, pressure, and temperature. Multidimensional arrays in computer memory are stored as linear sequential structures, and the hardware capabilities of modern computer processor units make it possible to ensure high performance of computational algorithms with appropriate data organization.

Vectorization allows to work with data as a single structure, which not only makes the design of a computational algorithm compact, avoiding nested loops, but also increases

the efficiency of calculations. Vectorization of calculations is organized with modern object-oriented programming languages like C++ or MATLAB.

4.2. Addressing to Internal Cells

Implementation of computational algorithms requires addressing to current cell or node and values in neighboring cells or nodes included in the finite-difference template. The traditional way to address flow quantities is to specify two indices defining line and column of two-dimensional array.

Using one-dimensional array of indices, it is possible to specify a subset of cells of mesh (Figure 1). For example, denoting by **pC** the vector (set) of indices of all internal cells (Figure 1a), subsets of cells that are above (Figure 1b) or below a current layer of cells are addressed. To address neighboring cells, it is necessary to add or subtract one from all components of the vector **pC**

$$\mathbf{pU} = \mathbf{pC} + 1, \quad \mathbf{pD} = \mathbf{pC} - 1.$$

Using similar procedure, the index vectors for left and right cells of computational domain are defined

$$\mathbf{pL} = \mathbf{pC} - m, \quad \mathbf{pR} = \mathbf{pC} + m.$$

The integer value m determines the number of cells in the column of computational domain.

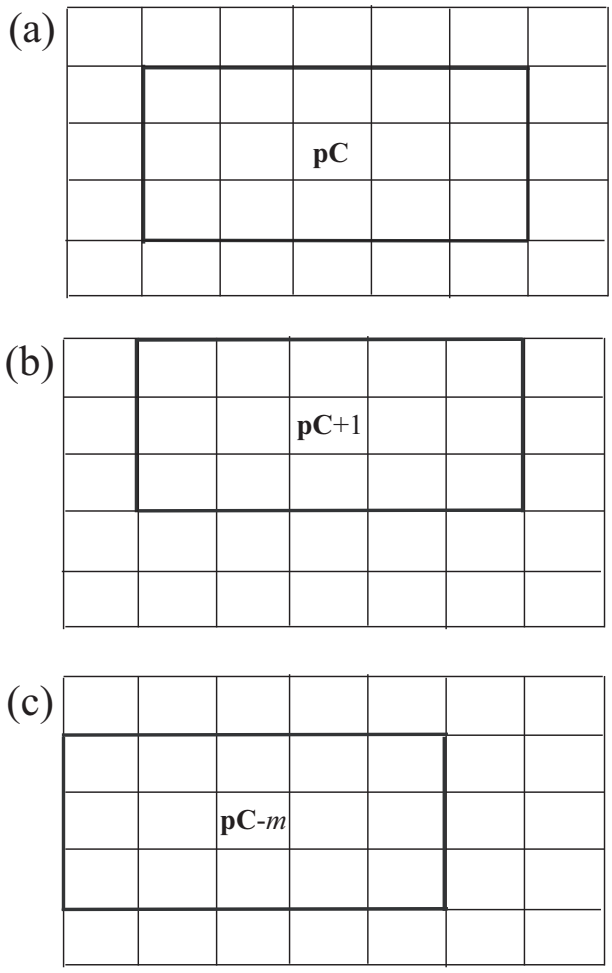

Figure 1. Addressing subsets of central cells (**a**), upper cells (**b**) and left cells (**c**) in a rectangular computational domain.

4.3. Addressing to Boundary Cells

To set boundary conditions, ghost cells are used, forming with internal cells an extended computational domain. Internal cells are highlighted in bold in Figure 2, and addressing them is provided using the index vector **pC**.

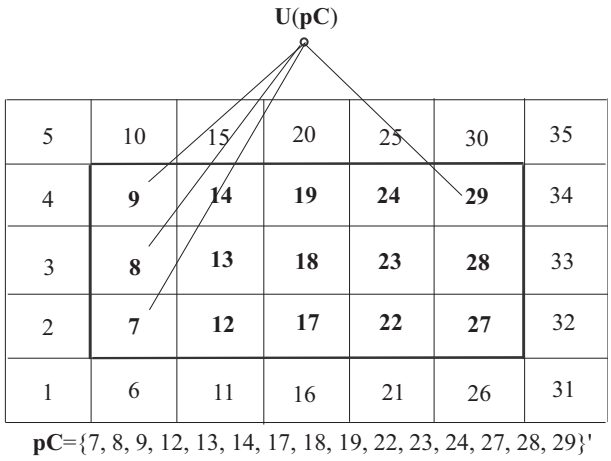

Figure 2. Example of extended computational domain.

The method of addressing ghost cells is shown in Figure 3. For this purpose, index vectors **pBL**, **pBU**, **pBR**, **pBD** are designed, corresponding to left, upper, right, and lower boundaries of computational domain. The method of their design is similar to index vectors of internal cells (corner cells are ignored).

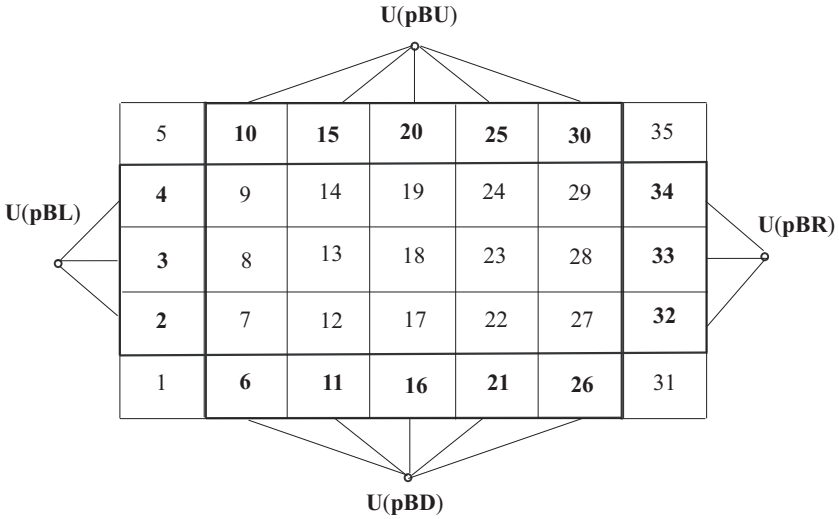

Figure 3. Examples of index vectors corresponding to ghost cells.

4.4. Flux Computation

The face separating two neighboring cells is selected as the object to which the flux is assigned. Linking the flux to a face allows to avoid re-calculating it when implementing conservation laws for the control volume. The same flux value is involved in constructing balance relations for two neighboring cells (the flux entering a cell equals the flux leaving the neighboring cell), which ensures the conservativeness of the numerical scheme.

When introducing ghost cells, all control volumes located inside the computational domain have neighbors, and the calculation of fluxes for all faces is carried out using a single algorithm. To calculate fluxes, two sets of index vectors are formed (Figure 4). One of these vectors, **pCFL**, defines the set of all cells located to the left of the face (Figure 4a), and

another index vector, **pCFR**, provides a wildcard addressing operation for the set of all right cells (Figure 4b). It is defined as follows

$$\mathbf{pCFR} = \mathbf{pCFL} + m.$$

The vector operation of numerical fluxes is provided by

$$\mathbf{F}(\mathbf{pC}) = \mathbf{F}(\mathbf{U}(\mathbf{pCFL}), \mathbf{U}(\mathbf{pCFR})). \tag{3}$$

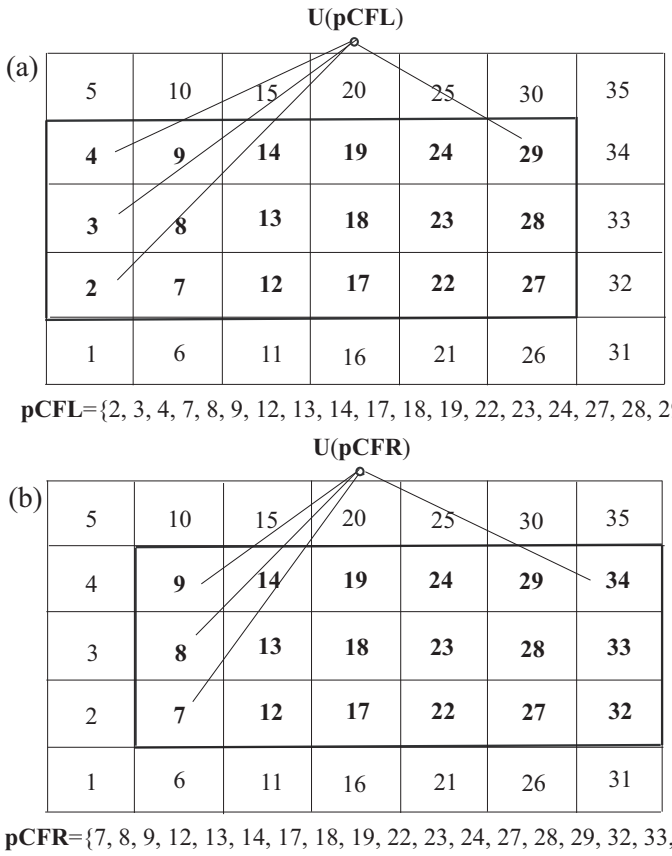

Figure 4. Examples of sets of left and right cells to calculate fluxes on the left (**a**) and right (**b**) faces.

The sequence of calculations written in ordinary index expressions has the form

$$\mathbf{F}_{i+1/2,j} = \mathbf{F}(\mathbf{U}_{i+1,j}, \mathbf{U}_{i,j}). \tag{4}$$

Indices i and j refer to the center of the cell, and the index $i + 1/2$ corresponds to the face separating cells with indices i and $i + 1$.

There is a difference between expressions (3) and (4). While relation (3) defines the entire set of threads, relation (4) requires a double round of calculations on indices i and j. The advantages of the expression-based approach (3) are manifested when using computing environments that allow the execution of vectorized operations.

Since the flux is assigned to the cell face, and flow quantities are assigned to cell center (Figure 5), it is necessary to determine the correspondence of index expressions for calculating the fluxes with the indices that define the cells of the computational domain. Even though a flux contains fewer elements than an extended set of cells, it makes sense to store it in a structure of the same size (although a number of elements are not used, but addressing convenience and clarity prevail over memory-saving considerations).

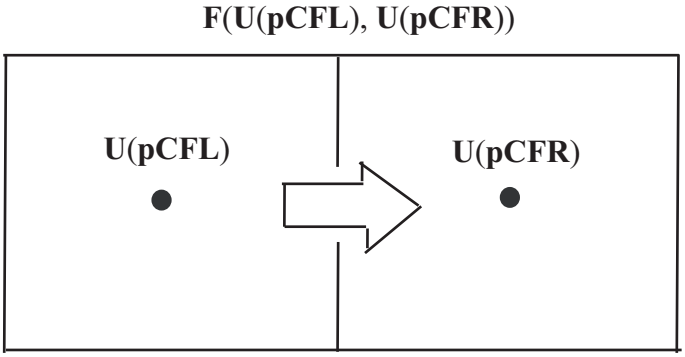

Figure 5. Vectorized flux computation (arrow shows direction of flux).

An index vector is designed to determine the set of fluxes through the cell faces using a layout shown in Figure 6.

5	10	15	20	25	30	35
4	**9**	**14**	**19**	**24**	**29**	34
3	**8**	**13**	**18**	**23**	**28**	33
2	**7**	**12**	**17**	**22**	**27**	32
1	6	11	16	21	26	31

pC={2, 3, 4, 7, 8, 9, 12, 13, 14, 17, 18, 19, 22, 23, 24, 27, 28, 29}'

Figure 6. Example of the link between fluxes and vector structure.

With this data organization, vectorized balance relationships are written for each cell of computational domain. Addressing fluxes through a set of cell edges is shown in Figure 7. For all computational cells defined by the index vector **pC**, fluxes through their left faces are determined by the index vector **pC** − m, and fluxes through the right faces are determined by the index vector **pC**. The value m represents the number of cells in the extended cell area column and determines the shift from a two-dimensional matrix structure to a linear arrangement of elements.

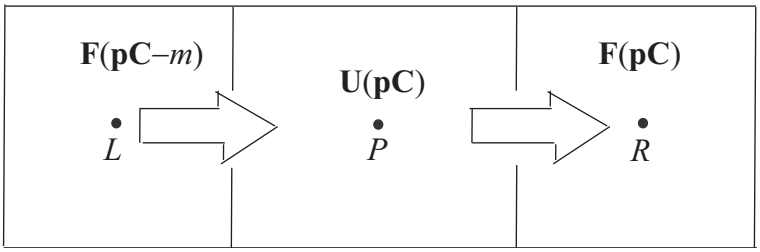

Figure 7. Calculation of fluxes through cell faces (arrows show direction of flux).

5. Discretization of Derivatives

Consider a finite-volume discretization of the derivative of a mesh function **U** defined on a sequential set of N equally spaced nodes. This dataset is represented using a linear array containing N elements.

To calculate the derivative, the function `y = diff(x,n)` is defined, which returns finite difference of order n that satisfies the recurrent relation `diff(x,n) = diff(x,n-1)`. If `x` is one-dimensional array

```
x = [x(1) x(2) ... x(n)],
```

then `diff(x)` is a vector of finite differences between neighboring elements

```
diff(x) = [x(2)-x(1) x(3)-x(2) ... x(n)-x(n-1)].
```

The number of elements of vector `x` is one less than the number of elements of vector `diff(x)`. The derivative approximation of order n is `diff(y,n)./diff(x,n)`, where "./" operation denotes element-wise division of vectors.

The derivative is calculated in another way, using the method of addressing mesh data structures. The integration interval $x_0 \leqslant x \leqslant x_N$ is divided into N equal segments, and denote the length of each of them as $\Delta x = x_i - x_{i-1}$ (Figure 8a). A number of all intervals varies from 1 to $N-1$. An array of indices $\mathbf{I}$, in which the sequential increase in indices corresponds to the order of its elements, is

$$\mathbf{I} = \{1, 2, \ldots, P-1, P, P+1, \ldots, N-1, N\}' = 1 : N.$$

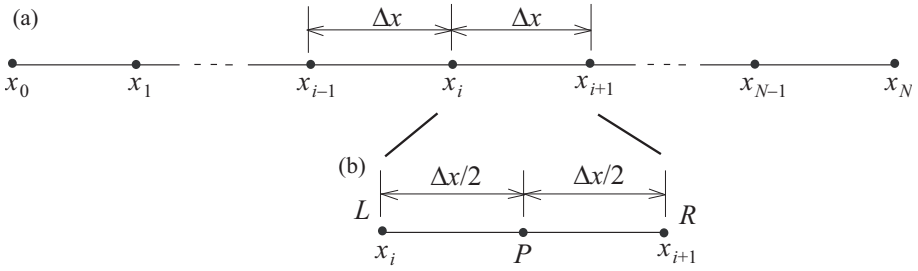

Figure 8. Partition of the integration interval in x direction (**a**) and finite-difference template (**b**).

The finite-difference template is shown in Figure 8b, where P is the midpoint of each segment, and the left and right boundaries of the interval are L and R, respectively (there are $N-1$ points P, L, and R in total). The arrays of indices for the left and right elements of mesh structure are introduced

$$\mathbf{L} = \{1, 2, \ldots, P-1, P, P+1, \ldots, N-2, N-1\}' = 1 : N-1;$$
$$\mathbf{R} = \{2, 3, \ldots, P-1, P, P+1, \ldots, N-1, N\}' = 2 : N.$$

The vector subtraction operation is defined as

$$\Delta\mathbf{U} = \mathbf{U}(\mathbf{R}) - \mathbf{U}(\mathbf{L}),$$

where

$$\begin{pmatrix} \Delta\mathbf{U}(1) \\ \Delta\mathbf{U}(2) \\ \vdots \\ \Delta\mathbf{U}(N-1) \end{pmatrix} = \begin{pmatrix} \mathbf{U}(\mathbf{R}(1)) - \mathbf{U}(\mathbf{L}(1)) \\ \mathbf{U}(\mathbf{R}(2)) - \mathbf{U}(\mathbf{L}(2)) \\ \vdots \\ \mathbf{U}(\mathbf{R}(N-1)) - \mathbf{U}(\mathbf{L}(N-1)) \end{pmatrix}.$$

Then, the formula for vectorized calculation of derivative has the form

$$\frac{d\mathbf{U}(x)}{dx} = \frac{\mathbf{U}(\mathbf{R}) - \mathbf{U}(\mathbf{L})}{\Delta x} + O(\Delta x^2).$$

In a pseudo programming language that supports vectorized operations, the algorithm of derivative calculation is represented as the following sequence of steps.

```
% indexes of intervals and boundary nodes
iI=1:N-1; iL=iI; iR=iI+1;
% mesh step
dx=1/(N-1);
% derivative y'(x)
yp=(y(iR)-y(iL))/dx;
```

The presented method of addressing mesh data structures underlies the calculation of derivatives of mesh functions and the construction of vectorized algorithms for solving boundary value problems.

For the left and right boundaries, index vectors $\boldsymbol{I}_L$ and $\boldsymbol{I}_R$ are introduced, having dimension M, whose components are equal to 1 if the left/right boundary condition is specified for the component k of the solution vector, and 0 if there is no left/right boundary condition for the k component of the solution vector.

Two vectors $\boldsymbol{L}$ and $\boldsymbol{R}$ are formed. Their components contain, at the places determined by the index vectors $\boldsymbol{I}_L$ and $\boldsymbol{I}_R$, left and right boundary values, respectively, and the remaining components have undefined (arbitrary) values. Arbitrarily specified components of the vectors $\boldsymbol{L}$ and $\boldsymbol{R}$ do not take part in the calculations. Two integer numbers, M_L and M_R, equal to the number of left and right boundary conditions (obviously, $M_L + M_R = M$), are introduced.

6. Boundary Value Problem

A system of ordinary differential equations of order M, resolved with respect to derivatives, is considered

$$\frac{d\boldsymbol{U}}{dx} + F\boldsymbol{U} = \boldsymbol{H}, \tag{5}$$

where $\boldsymbol{U}$ is column vector of flow quantities, F is matrix of coefficients, $\boldsymbol{H}$ is column vector of right-hand sides. To correctly solve a system of Equation (5), it is necessary that the number of boundary conditions be equal to the order of the system. In the general case, the boundary conditions between the initial x_0 and final x_N nodes of the integration interval are distributed in an arbitrary ratio.

A vector of physical variables $\boldsymbol{U}_i$ is assigned to each node of the computational domain x_i ($i = 0, \ldots, N$). Vectors $\boldsymbol{U}_0$ and $\boldsymbol{U}_N$, corresponding to the beginning and end of the integration interval, contain left and right boundary values.

For each point P, Equation (5) is written in discrete form

$$\frac{\boldsymbol{U}_R - \boldsymbol{U}_L}{\Delta x} + F_P\boldsymbol{U}_P = \boldsymbol{H}_P. \tag{6}$$

The matrix of coefficients F_P and the vector of right-hand side $\boldsymbol{H}_P$ corresponding to node P as the half-sum of values related to points L and R are found as

$$F_P\boldsymbol{U}_P = \frac{1}{2}(F_L\boldsymbol{U}_L + F_R\boldsymbol{U}_R), \quad \boldsymbol{H}_P = \frac{1}{2}(\boldsymbol{H}_L + \boldsymbol{H}_R).$$

Substitution into Equation (6) provides

$$\boldsymbol{U}_R - \boldsymbol{U}_L + \frac{\Delta x}{2}(F_L\boldsymbol{U}_L + F_R\boldsymbol{U}_R) = \frac{\Delta x}{2}(\boldsymbol{H}_L + \boldsymbol{H}_R). \tag{7}$$

To linearize the relationship (7), the frozen coefficient method is used.

7. Vectors of Computational Variables

Vectors $\boldsymbol{U}_0$ and $\boldsymbol{U}_N$, at some generally arbitrary places, contain quantities corresponding to the boundary conditions, which makes it difficult to formalize the computational procedure with vectors of physical variables.

Vectors of computational variables $\boldsymbol{W}_i$ are introduced (their number is one less than physical vectors $\boldsymbol{U}_i$). They contain only unknowns. Each computational vector $\boldsymbol{W}_i$ contains part of the unknowns from the vector $\boldsymbol{U}_{i-1}$ and part of the unknowns from the vector $\boldsymbol{U}_i$. Obviously, any component of vectors $\boldsymbol{U}_i$ is included in the set of vectors $\boldsymbol{W}_i$ at most once. In this case, the known components of vectors $\boldsymbol{U}_0$ and $\boldsymbol{U}_N$ corresponding to boundary conditions are not included in vectors $\boldsymbol{W}_i$.

The common scheme for constructing computational vectors $\boldsymbol{W}_i$ is considered. The construction with the vector $\boldsymbol{W}_N$ is the first step. The components of the vector $\boldsymbol{U}_N$ that do not correspond to boundary conditions are transferred to the vector $\boldsymbol{W}_N$ to the same places. The components remaining in the vector $\boldsymbol{W}_N$ are filled with the components of the vector $\boldsymbol{U}_{N-1}$ from the same positions. Formally, this procedure is written using the index vector $\boldsymbol{I}_R$

$$\boldsymbol{W}_N^j = \left(1 - \boldsymbol{I}_R^j\right)\boldsymbol{U}_N^j + \boldsymbol{I}_R^j \boldsymbol{U}_{N-1}^j. \tag{8}$$

Moving to the left from the right boundary (indices move from N to 1), the vector $\boldsymbol{W}_i$ is represented as

$$\boldsymbol{W}_i^j = \left(1 - \boldsymbol{I}_R^j\right)\boldsymbol{U}_i^j + \boldsymbol{I}_R^j \boldsymbol{U}_{i-1}^j. \tag{9}$$

The method for generating the vector $\boldsymbol{W}_1$ differs from the others. The still unclaimed components of the vector $\boldsymbol{U}_1$ take their positions, and the remaining places are filled with components of vector $\boldsymbol{U}_0$ that do not correspond to the boundary conditions. To do this, these components are taken in increasing order of their indices and placed on the free places of vector $\boldsymbol{W}_1$ in increasing order of indices of these places. The representation of vector $\boldsymbol{W}_1$ has the form

$$\boldsymbol{W}_1^j = \left(1 - \boldsymbol{I}_R^j\right)\boldsymbol{U}_1^j + \boldsymbol{I}_R^j \boldsymbol{U}_*^j. \tag{10}$$

The vector $\boldsymbol{U}_*$ is obtained from the vector $\boldsymbol{U}_0$ by rearranging the components according to the described principle. The formal representation of the vector $\boldsymbol{U}_*$ is not essential for further presentation and is not provided here.

8. Transition Formulas

The vector of physical variables is expressed in terms of vectors of computational variables. Consider the agreement that the computational vector is formed taking into account the form of specifying the right boundary conditions

$$\boldsymbol{U}_i = T_L^i \boldsymbol{W}_i + T_R^i \boldsymbol{W}_{i+1}, \tag{11}$$

where $i = 2, \ldots, N-1$. Relation (11) formalizes a simple shift in components, which is conducted in such a way that the components of computational vectors fall into place in the physical vectors.

The right permutation matrix T_R contains the components of index vector $\boldsymbol{I}_R$ on the main diagonal, and the remaining components of this matrix are zero

$$T_R^i = \boldsymbol{I}_R E = \begin{pmatrix} I_R^1 & 0 & \cdots & 0 \\ 0 & I_R^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_R^M \end{pmatrix}.$$

The left permutation matrix T_L contains zeros on the main diagonal where the corresponding components of index vector $\boldsymbol{I}_R$ are equal to one, and ones otherwise, and the remaining components of this matrix are equal to zero

$$T_L^i = E - T_R^i = \begin{pmatrix} 1 - I_R^1 & 0 & \cdots & 0 \\ 0 & 1 - I_R^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 - I_R^M \end{pmatrix}.$$

The permutation matrices T_L^i and T_R^i are the same for all internal nodes of computational domain (for $i = 1, \ldots, N-1$).

For the right boundary (for $i = N$), the representation of the vector of physical variables has the form

$$\boldsymbol{U}_N = T_L^N \boldsymbol{W}_N + T_R^N \boldsymbol{R}. \tag{12}$$

The permutation matrices T_L^N and T_R^N remain the same as at the internal nodes.

The representation of vector of physical variables on the left boundary (at $i = 0$) has a special form. To form the corresponding expression, several auxiliary matrix objects are introduced. Note that a square matrix having a single non-zero element equal to 1 in row l and in column k, when multiplied by a column vector, moves the element of vector k to position l. The representation of vector of physical variables at node $i = 0$ is related to the form of computational vector at $i = 1$. The structure of this relationship has the form

$$\boldsymbol{U}_0 = T_L^0 \boldsymbol{L} + T_R^0 \boldsymbol{W}_1. \tag{13}$$

The first term on the right side is the set of components of vector of variables that are determined by the boundary values at the left end of the interval. The second term represents those vector components that are determined during the solution process. The left permutation matrix has the form

$$T_L^0 = E - \boldsymbol{I}_L E.$$

The matrix T_R^0 provides the substitution of those variables that are not included in the boundary conditions and therefore are in the vector $\boldsymbol{W}_1$.

To form the matrix T_R^0, the vector $\boldsymbol{W}_1$ is filled with those components of the vector $\boldsymbol{U}_1$ for which right boundary conditions are not specified (these components are formed in the same way as in other vectors). As a result, $M_L = M - M_R$ components are determined. To form the remaining M_R components, the first zero element I_L^j of the vector $\boldsymbol{I}_L$ and the first non-zero element I_R^k of vector $\boldsymbol{I}_R$ are found. The pair of indices (j, k) determines the transition of an element from $\boldsymbol{U}_0$ to $\boldsymbol{W}_1$. Then, the next pair of indices is determined and the process is repeated. The constructed set of M_R pairs of indices allows to form a matrix T_R^0, which contains units in places indicated by pairs of indices (the first number of the pair indicates the row, and the second number of the pair indicates the column). Its remaining elements are equal to zero. The described procedure allows to connect the physical and computational vectors at node $i = 0$ by a matrix relation.

9. Numerical Scheme

In the computational variables, the difference scheme for Equation (5) takes the form

$$\boldsymbol{U}_i - \boldsymbol{U}_{i-1} + \frac{\Delta x}{2}(F_{i-1}\boldsymbol{U}_{i-1} + F_i \boldsymbol{U}_i) = \frac{\Delta x}{2}(\boldsymbol{H}_{i-1} + \boldsymbol{H}_i). \tag{14}$$

Considering the representation of vector of physical variables through the vector of computational variables, substitution into Equation (14) provides

$$\begin{aligned}
&\left(-T_L^{i-1} + \frac{\Delta x}{2} F_{i-1} T_L^{i-1}\right)\boldsymbol{W}_{i-1} + \\
&\left[T_L^i - T_R^{i-1} + \frac{\Delta x}{2}\left(F_{i-1} T_R^{i-1} + F_i T_L^i\right)\right]\boldsymbol{W}_i + \\
&+ \left(T_i^R + \frac{\Delta x}{2} F_i T_R^i\right)\boldsymbol{W}_{i+1} = \frac{\Delta x}{2}(\boldsymbol{H}_{i-1} + \boldsymbol{H}_i).
\end{aligned} \tag{15}$$

In matrix form, the system of difference equations has the form

$$A\,\boldsymbol{U} = \boldsymbol{B}, \tag{16}$$

where A is square matrix of coefficients of $MN \times MN$, $\boldsymbol{U}$ and $\boldsymbol{B}$ is column vector of unknowns and column vector of right-hand sides of $M \times N$. To solve a system of Equation (16), one of the methods for solving systems of linear equations is used, in particular, the traditional block Thompson method.

Relation (15) is written as

$$A_i \boldsymbol{W}_{i-1} + B_i \boldsymbol{W}_i + C_i \boldsymbol{W}_{i+1} = \boldsymbol{D}_i. \tag{17}$$

Here,

$$A_i = -T_L^{i-1} + \frac{\Delta x}{2} A_{i-1} T_L^{i-1};$$
$$B_i = T_L^i - T_R^{i-1} + \frac{\Delta x}{2}\left(A_{i-1} T_R^{i-1} + A_i T_L^i\right);$$
$$C_i = T_i^R + \frac{\Delta x}{2} A_i T_R^i;$$
$$\boldsymbol{D}_i = \frac{\Delta x}{2}(\boldsymbol{B}_{i-1} + \boldsymbol{B}_i).$$

Vectors $\boldsymbol{W}_0$ and $\boldsymbol{W}_{N+1}$ are taken to be vectors $\boldsymbol{L}$ and $\boldsymbol{R}$, respectively. For $i = 1$ and $i = N$

$$B_1 \boldsymbol{W}_1 + C_1 \boldsymbol{W}_2 = \boldsymbol{D}_1 - A_1 \boldsymbol{L};$$
$$A_N \boldsymbol{W}_{N-1} + B_N \boldsymbol{W}_N = \boldsymbol{D}_N - C_N \boldsymbol{R}.$$

The running relations are constructed according to the following dependence

$$\boldsymbol{W}_{i-1} = K_{i-1} \boldsymbol{W}_i + \boldsymbol{P}_{i-1}.$$

Substitution leads to equation

$$\boldsymbol{W}_i = -(A_i K_{i-1} + B_i)^{-1} C_i \boldsymbol{W}_{i+1} + (A_i K_{i-1} + B_i)^{-1}(\boldsymbol{D}_i - A_i \boldsymbol{P}_{i-1}).$$

As a result, the representations for K_i and $\boldsymbol{P}_i$ for $i > 1$ have the form

$$K_i = -(A_i K_{i-1} + B_i)^{-1} C_i;$$
$$\boldsymbol{P}_i = (A_i K_{i-1} + B_i)^{-1}(\boldsymbol{D}_i - A_i \boldsymbol{P}_{i-1}).$$

The matrix K_1 and the vector $\boldsymbol{P}_1$ are calculated using the relations

$$K_1 = -B_i^{-1} C_i;$$
$$\boldsymbol{P}_1 = B_i^{-1}(\boldsymbol{D}_i - A_i \boldsymbol{L}).$$

The vector $\boldsymbol{W}_N$ is found by the formula (forward step)

$$\boldsymbol{W}_N = (A_N K_{N-1} + B_N)^{-1}(\boldsymbol{D}_N - C_N \boldsymbol{R} - A_N \boldsymbol{P}_{N-1}).$$

Using the found values of the sweep coefficients, the following vector values are calculated, moving along the computational domain from left to right (reverse step)

$$\boldsymbol{W}_i = K_i \boldsymbol{W}_{i+1} + \boldsymbol{P}_i.$$

To fill the physical vectors, transition formulas are used, described by the relations (11)–(13).

10. Results and Discussion

Vectorized algorithms are applied to solving benchmark CFD problems. The benchmark problems allow to validate the reconstruction procedure for the presence of phase

errors, the occurrence of non-physical oscillations, and smoothing of the solution in the region of sharp gradients. The problem of wave convection serves to validate the reconstruction step of the numerical algorithm, and the shock tube problem allows to validate the evolution step.

10.1. Wave Convection

Equation of wave convection is a simple model of convective vortex transport. When studying wave processes, the case when the initial data are a sinusoidal (or cosine) distribution is interesting. The linear transport equation transports such a wave without distortion, while difference schemes that implement this equation can distort the wave, changing its amplitude and shape. The wave convection equation (scalar transport equation) is written as [26,27]

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0, \tag{18}$$

where $a = \text{const}$. The initial condition includes Gaussian wave, square wave, triangular wave, or semi-elliptical wave. The boundary condition is

$$u(x,0) = u_0(x).$$

The general solution of transport equation for $a = \text{const}$ is

$$u = u_0(x - at).$$

To validate the reconstruction step of the numerical algorithm, the initial condition for Equation (18) is specified in the form

$$u(x,0) = \exp\left(-x^2/16\right).$$

The exact solution is

$$\widehat{u}(x,t) = \exp\left[-(x-t)^2/16\right].$$

The numerical error is estimated as

$$\delta = \frac{1}{N}\left[\sum_{i=1}^{N}(u_i - \widehat{u}_i)^2\right]^{1/2},$$

where N is the number of mesh nodes.

To discretize time derivatives, the three-step Runge–Kutta method is used, and UDS-3 scheme (upwind difference scheme) and STVD-4 and STVD-6 schemes (symmetric TVD scheme) are used to calculate fluxes.

The results of numerical calculations (solid line) are shown in Figure 9 in comparison with the exact solution (symbols ■). The results demonstrate that upwind scheme is too dissipative. It underestimates the maximum value of the unknown and leads to oscillations of the solution. Two numerical schemes of TVD type of second and third orders provide similar results. Increasing the order of numerical scheme makes it possible to achieve a higher accuracy with a smaller number of nodes but requires an increase in number of operations on each time layer (Figure 10).

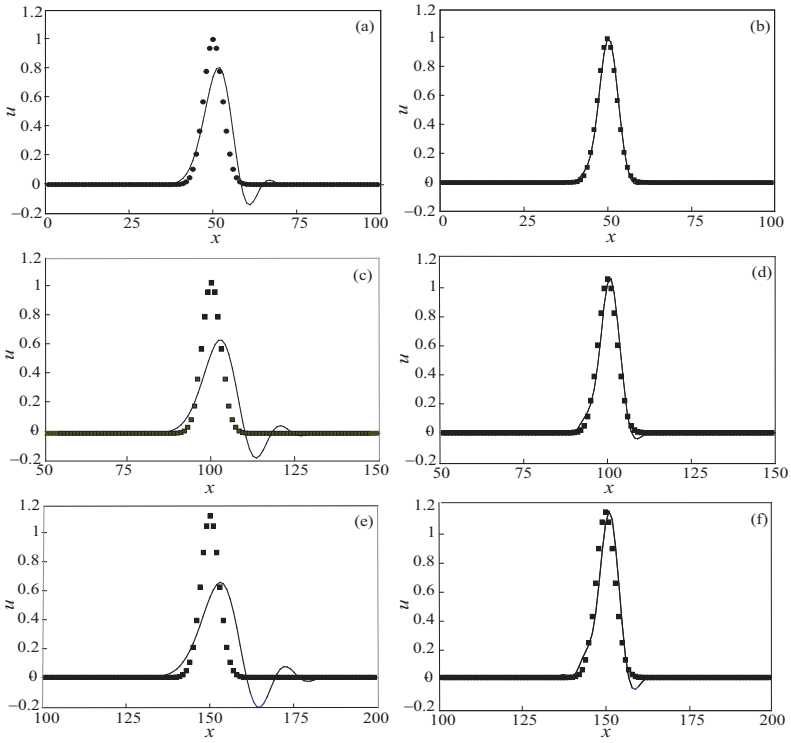

Figure 9. Solution of wave equation at times 45 (**a**,**b**), 90 (**c**,**d**), 135 (**e**,**f**) with upwind scheme (**left**) and TVD scheme of 2nd order (**right**). Symbols ■ correspond to the exact solution.

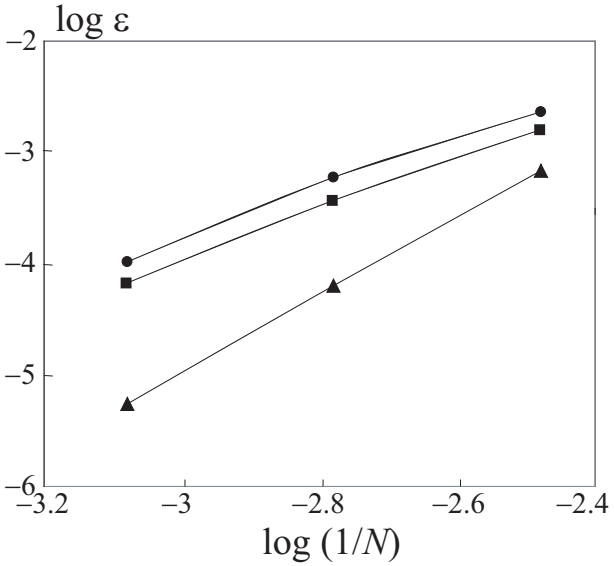

Figure 10. Dependence of numerical error on number of nodes. Symbols •, ■, ▲ correspond to upwind scheme and TVD schemes of 2nd and 3rd orders.

10.2. Shock Tube

In a shock tube (Sod problem), two regions are occupied by a gas (a diaphragm is located at $x = 0$). When diaphragm is broken, two gases interact with each other. The problem has exact solution. It consists of a shock or rarefaction wave travelling to the left region, a shock or rarefaction wave travelling to the right region, and contact discontinuity. A wave front of rarefaction wave represents a fan of flow characteristics.

The initial conditions correspond to two different solutions. Flow is subsonic in case 1, and flow is supersonic in case 2. Working substance is air; it is treated as ideal gas ($\gamma = 1.4$).

In case 1, $\rho(x,0) = p(x,0) = 2$ if $x \leqslant 0$, and $\rho(x,0) = p(x,0) = 1$ if $x > 0$, $u(x,0) = 0$. The mesh contains 100 cells (mesh step size is $\Delta x = 1.0$). To reach a steady state, 100 time steps are applied (time step is $\Delta t = 0.00058$). The Courant number is fixed at 0.59.

In case 2, $\rho(x,0) = p(x,0) = 20$ if $x \leqslant 0$, and $\rho(x,0) = p(x,0) = 1$ if $x > 0$, $u(x,0) = 0$. The mesh contains 1000 cells (mesh step size is $\Delta x = 0.1$). To reach a steady state, 100 time steps are applied (time step is $\Delta t = 0.00012$). The Courant number is fized at 0.54.

The distributions of flow quantities are shown in Figure 11. The graphs presented show numerical solutions in a short period of time when initial discontinuity starts to break and the largest disturbance from the exact Riemann solver is observed. Solid line corresponds to the exact solution of Riemann problem, dotted line corresponds to calculation with Godunov scheme, symbols • correspond to calculation with third-order numerical scheme, and circles correspond to calculation with the scheme using a piece-parabolic approximation of flow quantities in a control volume. In this case, Godunov scheme requires approximately 3.8 times more computational time than schemes with approximate solution of Riemann problem.

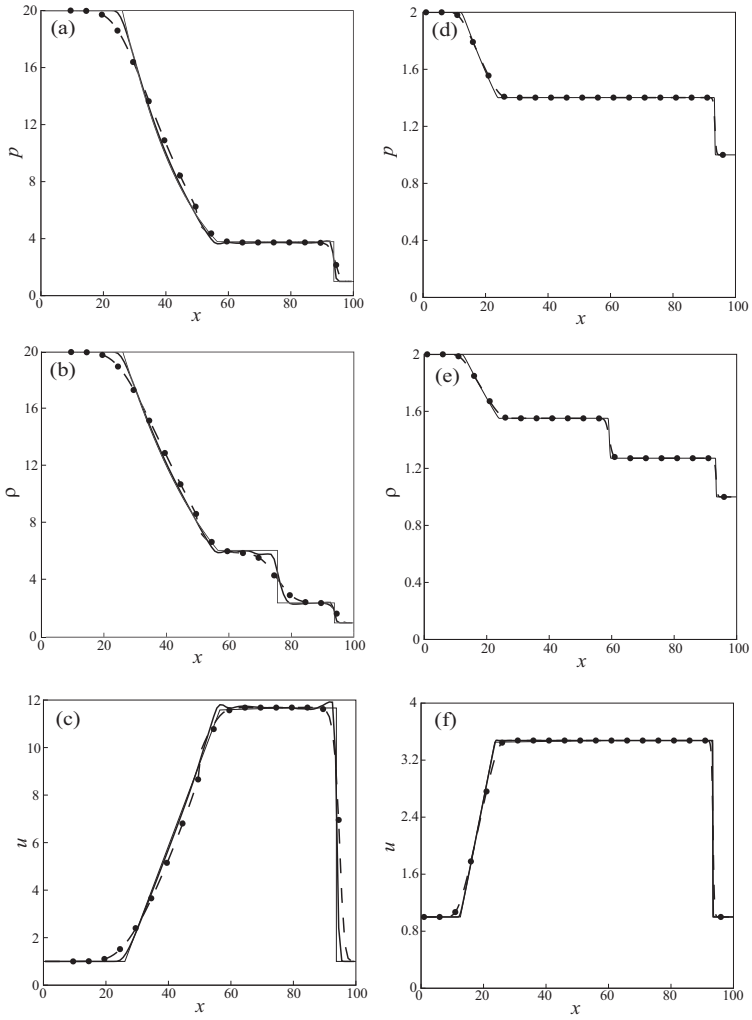

Figure 11. Distributions of pressure (**a**,**b**), density (**c**,**d**) and velocity (**e**,**f**) in Sod problem in case 1 (**left**) and case 2 (**right**). Solid lines show exact solutions, dotted lines show numerical solutions with Godunov scheme, and symbols • correspond to calculations with numerical scheme of 3rd order.

The solution of shock tube problem [28] has a monotonic density profile. Solution of Lax problem [29] provides verification of properties of numerical schemes. Various numerical schemes correctly predict horizontal parts of density distributions. The distributions computed with second-order scheme have a slight increase in density in 1–2 mesh

cells to the right of contact discontinuity. The contact discontinuity is less smoothed in the numerical solutions computed with high-order accuracy schemes.

10.3. Flow Around Expansion Corner

A Prandtl–Meyer flow around expansion corner is considered [30]. The corner angle is $\theta = 6.25°$. Corner point is located at $x = 0$. Initial flow is parallel to lower wall (Figure 12). Total pressure and total temperature are fixed on inflow boundary ($p_0 = 10^5$ Pa, $T_0 = 290$ K). Flow speed is 682 m/s, and Mach number is 2.0.

Solutions on a structured mesh with 65×41 nodes (symbols •) and an unstructured mesh with 2600 nodes (solid line) are shown in Figure 13 for the fixed Courant number 0.54. Distributions of flow quantities for various Courant numbers are presented in Figure 14. The maximum velocity in the computational domain is about 711 m/s. If Courant number equals 0.1, oscillations of numerical solution occur, corresponding to the location of reversed flow. Distributions of velocity magnitude computed on different meshes are shown in Figure 15.

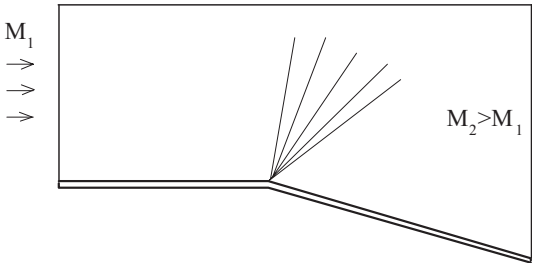

Figure 12. Prandtl–Meyer flow.

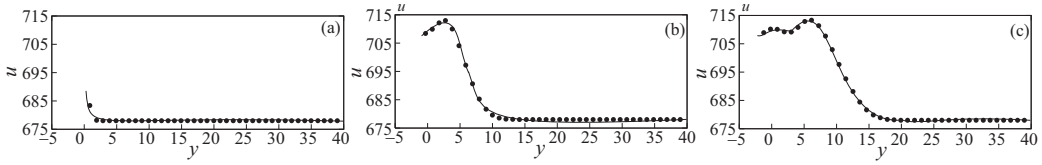

Figure 13. Velocity profiles at $x = 10.32$ (**a**), 23.51 (**b**), 33.36 (**c**). Symbols • corresponds to the numerical solution, ans solid line corresponds to data computed in [30].

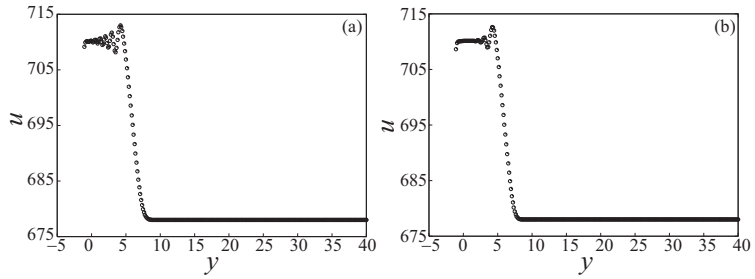

Figure 14. Velocity profiles at $x = 23.51$ and Courant numbers 0.1 (**a**) and 0.6 (**b**).

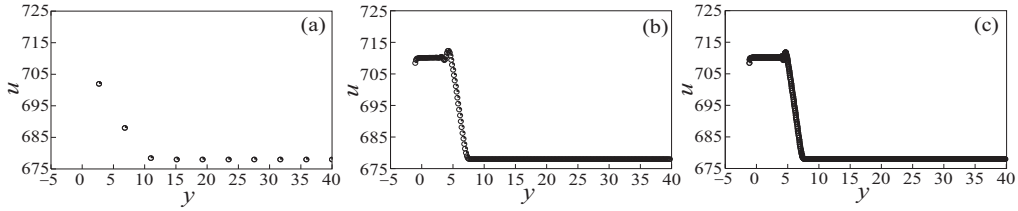

Figure 15. Velocity profiles at $x = 23.51$ computed on meshes with 26×12 (**a**), 310×310 (**b**), 610×810 (**c**) nodes.

10.4. Nozzle Flow

An unsteady one-dimensional flow of inviscid compressible gas in a convergent-divergent nozzle is considered. Air ($\gamma = 1.4$) is a working gas. The cross-sectional area of the nozzle varies as (Figure 16)

$$S(x) = 1 + 2x^2, \quad \text{where} \quad -1/3 \leqslant x \leqslant 1.$$

The numerical solution is compared with the exact solution in overexpanded and underexpanded flow regimes.

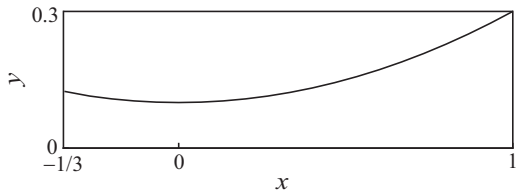

Figure 16. Nozzle profile.

Total pressure and total temperature are fixed on inlet boundary ($p_0 = p_{00}$, $T_0 = T_{00}$). The third quantity is found from the characteristic relations. Specification of boundary conditions on outlet boundary depends on flow regime (flow speed), overexpanded or underexpanded. Flow regime on outlet boundary depends on nozzle pressure ratio

$$\frac{p_\infty}{p_{00}} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}.$$

A static pressure on outlet boundary equals atmospheric pressure ($p = p_\infty$) if outflow is subsonic. Overexpanded flow regime occurs if $p_\infty/p_{00} > 0.528$ (pressure is restored to atmospheric pressure through a nozzle shock wave). No nozzle shock wave occurs if $p_\infty/p_{00} < 0.528$ (the flow speed continuously increases in streamwise direction). Calculations are carried out on a mesh containing 35 cells, with a step of $\Delta x = 0.038$. Initial conditions assume that $u = 0$, $p_0 = p_{00}$, $T_0 = T_{00}$.

The results of numerical simulation, processed in the form of pressure dependencies on streamwise coordinates, are shown in Figure 17 (Courant number is 0.88) and Figure 18 (Courant number is 0.18). The symbols ■ and □ correspond to Godunov-type schemes of first and second orders (figure a), symbols ▷ correspond to Roe scheme (figure b), symbols ◁ correspond to TVD scheme of second order (figure c), and symbols • correspond to TVD scheme of third order (figure d). The computed results show that numerical schemes spread the discontinuity over 1–2 mesh cells, preserving the monotonic nature of numerical solution.

In the case of the presence of a nozzle shock wave (Figure 17), the third-order TVD numerical scheme provides results that are close to the exact solution. When using the Godunov scheme, it takes 0.0352 s of estimated time (fourteen time steps) to achieve a steady state; the Godunov scheme of second order takes 0.0348 s (eleven steps); the Roe scheme takes 0.02 s (five steps); and the TVD schemes of second and third orders take 0.021 s (four steps) and 0.023 s (four steps).

Numerical schemes provide similar results in overexpanded flow regimes. The Godunov scheme takes 0.0068 s (twelve time steps) to achieve a steady state; the Godunov scheme of second order takes 0.0082 s (sixteen steps); the Roe scheme takes 0.01 s (nineteen steps); and the TVD schemes of second and third orders take 0.007 s (thirteen steps) and 0.0072 s (eleven steps).

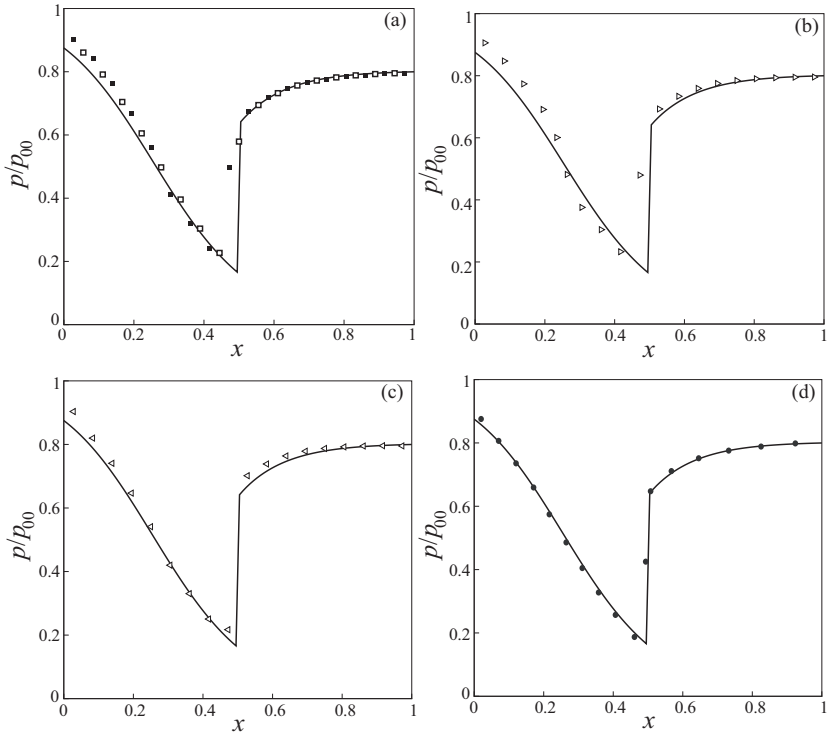

Figure 17. Pressure distributions in streamewise direction computed with Godunov scheme (**a**), Roe scheme (**b**), TVD scheme of second order (**c**), TVD scheme of second order (**d**) at $p_{00} = 10^8$ Pa, $T_{00} = 300$ K, $p_\infty = 8 \times 10^5$ Pa. Symbols show numerical solution, and solid lines correspond to the exact solution.

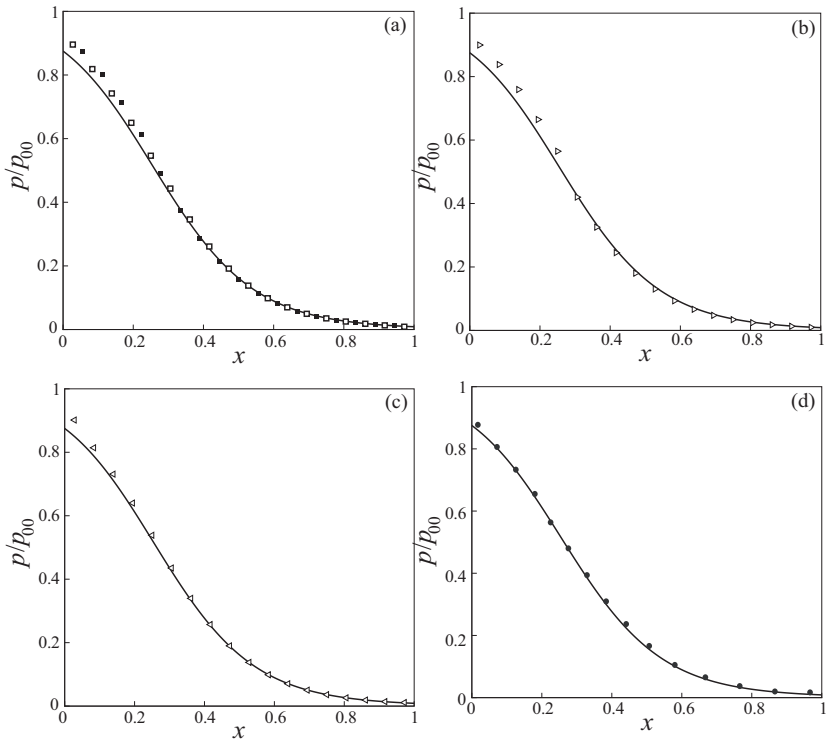

Figure 18. Pressure distributions in streamewise direction computed with Godunov scheme (**a**), Roe scheme (**b**), TVD scheme of second order (**c**), TVD scheme of second order (**d**) at $p_{00} = 10^6$ Pa, $T_{00} = 300$ K, $p_\infty = 8 \times 10^5$ Pa. Symbols show numerical solution, and solid lines correspond to the exact solution.

11. Conclusions

Using loops is a common and natural tendency when performing repetitive operations in programming. However, when working with a large number of iterations, such as millions or billions of rows, using loops can lead to poor performance and long run times. In these cases, implementing vectorization can greatly improve efficiency and avoid the frustration of waiting for slow processes to complete. Vectorized operations enable the use of efficient pre-compiled functions and mathematical operations on arrays and data sequences.

An approach to the construction and implementation of vectorized algorithms for solving CFD benchmark problems has been developed. The capabilities of the developed approach are demonstrated by solving a number of CFD problems. Speed is important because small differences in runtime can accumulate and become significant when repeated over many function calls. For example, an incremental 30 microseconds of overhead, when repeated over 1 million function calls, can result in 30 s of additional runtime.

The Riemann solver has a function that processes the right and left decay profiles of a discontinuity in the same way, with minor changes. Using a simple change in variables, which consisted of changing the sign of the velocity value, it was possible to merge the code for two subtrees. This allowed us to halve the volume of calculation code and reduce execution time by 45%.

The developed methods for vectorizing a complex software context can be used to optimize other computational problems. In particular, a library is being developed aimed at vectorizing arbitrary planar cycles, that is, cycles in which there are no interiteration dependencies.

Funding: The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of World-class Research Center program: Advanced Digital Technologies (contract No. 075-15-2020-903 dated 16 November 2020).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Soukov, S.A. Methods for improving and evaluating the performance of unstructured CFD algorithms. *Math. Model. Comput. Simul.* **2023**, *15*, 717–724. [CrossRef]
2. Gorobets, A.; Soukov, S.; Bogdanov, P. Multilevel parallelization for simulating compressible turbulent flows on most kinds of hybrid supercomputers. *Comput. Fluids* **2018**, *173*, 171–177. [CrossRef]
3. Mavriplis, D.J. Progress in computational fluid dynamics discretizations algorithms and solvers for aerodynamic flows. *AIAA J.* **2021**, *9*, 5374–5397. [CrossRef]
4. Mastrone, M.N.; Concli, F. CFD simulations of gearboxes: Implementation of a mesh clustering algorithm for efficient simulations of complex system's architectures. *Int. J. Mech. Mater. Eng.* **2021**, *16*, 12. [CrossRef]
5. Alberty, J.; Carstensen, C.; Funken, S.A. Remarks around 50 lines of Matlab: Short finite element implementation. *Numer. Algorithms* **1999**, *20*, 117–137. [CrossRef]
6. Persson, P.-O.; Strang, G. A simple mesh generation in MATLAB. *SIAM Rev.* **2004**, *42*, 329–345. [CrossRef]
7. Koko, J. Vectorized MATLAB codes for linear two-dimensional elasticity. *Sci. Program.* **2007**, *15*, 157–172. [CrossRef]
8. Funken, S.; Praetorius, D.; Wissgott, P. Efficient implementation of adaptive P1-FEM in MATLAB. *Comput. Methods Appl. Math.* **2011**, *11*, 460–490. [CrossRef]
9. Rahman, T.; Valdman, J. Fast MATLAB assembly of FEM matrices in 2D and 3D: Nodal elements. *Appl. Math. Comput.* **2013**, *219*, 7151–7158. [CrossRef]
10. Koko, J. Fast MATLAB assembly of fem matrices in 2d and 3d using cell array approach. *Int. J. Model. Simul. Sci. Comput.* **2016**, *7*, 1650010. [CrossRef]
11. Walker, S.W. FELICITY: A MATLAB/C++ toolbox for developing finite element methods and simulation modelling. *SIAM J. Sci. Comput.* **2018**, *40*, C234–C257. [CrossRef]
12. Cermak, M.; Sysala, S.; Valdman, J. Efficient and flexible Matlab implementation of 2D and 3D elasto-plastic problems. *Appl. Math. Comput.* **2019**, *355*, 595–614.

13. Tsega, E.G. A finite volume solution of unsteady incompressible Navier—Stokes equations using MATLAB. *Numer. Comput. Methods Sci. Eng.* **2019**, *1*, 117–131.
14. Volkov, K.N.; Emelyanov, V.N.; Karpenko, A.G.; Teterina, I.V. Simulating flows of viscous incompressible fluid on graphics processors using the splitting scheme and multigrid method. *Comput. Math. Math. Phys.* **2019**, *59*, 136–149. [CrossRef]
15. Shampine, L.F. Vectorized adaptive quadrature in MATLAB. *J. Comput. Appl. Math.* **2008**, *211*, 131–140. [CrossRef]
16. Guo, H.; Yin, Z.-H.; Yuan, L. A block SPP algorithm for multidimensional tridiagonal equations with optimal message vector length. *J. Algorithms Comput. Technol.* **2009**, *3*, 229–245. [CrossRef]
17. Kűhn, M.J.; Holke, J.; Lutz, A.; Thies, J.; Rőhrig-Zőllner, M.; Bleh, A.; Backhaus, J.; Basermann, A. SIMD vectorization for simultaneous solution of locally varying linear systems with multiple right-hand sides. *J. Supercomput.* **2023**, *79*, 14684–14706. [CrossRef]
18. Guglielmi, N.; Hairer, E. An efficient algorithm for solving piecewise-smooth dynamical systems. *Numer. Algorithms* **2022**, *89*, 1311–1334. [CrossRef]
19. Qu, J.; Faney, T.; de Hemptinne, J.-C.; Yousef, S.; Gallinari, P. PTFlash: A vectorized and parallel deep learning framework for two-phase flash calculation. *Fuel* **2023**, *331*, 125603. [CrossRef]
20. Shang, J.J.S. Landmarks and new frontiers of computational fluid dynamics. *Adv. Aerodyn.* **2019**, *1*, 5. [CrossRef]
21. Ma, Y.; Sclavounos, P.D. Support vector machines model of the nonlinear hydrodynamics of fixed cylinders. *J. Offshore Mech. Arct. Eng.* **2021**, *143*, 051701. [CrossRef]
22. Sofos, F.; Stavrogiannis, C.; Exarchou-Kouveli, K.K.; Akabua, D.; Charilas, G.; Karakasidis, T.E. Current trends in fluid research in the era of artificial intelligence: A review. *Fluids* **2022**, *7*, 116. [CrossRef]
23. Sharma, P.; Chung, W.T.; Akoush, B.; Ihme, M. A review of physics-informed machine learning in fluid mechanics. *Energies* **2023**, *16*, 2343. [CrossRef]
24. Nagesha, K. Machine learning algorithms to study the relative roles of jet convection and natural convection in the presence of surface roughness elements on enhancement of jet impingement heat transfer. *Int. J. Therm. Sci.* **2024**, *198*, 108847. [CrossRef]
25. Dobrov, Y.; Karpenko, A.; Malkovsky, S.; Sorokin, A.; Volkov, K. Simulation of high-temperature flowfield around hypersonic waverider using graphics processor units. *Acta Astronaut.* **2023**, *204*, 745–760. [CrossRef]
26. Suresh, A.; Huynh, H.T. Accurate monotonicity-preserving schemes with Runge–Kutta time stepping. *J. Comput. Phys.* **1997**, *136*, 83–99. [CrossRef]
27. Balsara, D.S.; Shu, C.-W. Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy. *J. Comput. Phys.* **2000**, *160*, 405–452. [CrossRef]
28. Sod, G.A. A survey of several finite difference methods of systems of nonlinear hyperbolic conservation laws. *J. Comput. Phys.* **1978**, *27*, 1–31. [CrossRef]
29. Lax, P.D. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Commun. Pure Appl. Math.* **1954**, *7*, 159–193. [CrossRef]
30. Struchkov, A.V.; Kozelkov, A.S.; Volkov, K.N.; Kurkin, A.A.; Zhuckov, R.N.; Sarazov, A.V. Numerical simulation of aerodynamic problems based on adaptive mesh refinement method. *Acta Astronaut.* **2020**, *172*, 7–15. [CrossRef]