Parallel Implementation of Algorithm for Calculation of Gas Flow through Porous Media with Heat Sources

N. Lutsenko, T. Miroshnichenko, D. Odyakova, D. Kharitonov

Abstract—A parallel computational algorithm for researching of two-dimensional (plane and axially symmetric) unsteady gas flows through a porous media of complex shape with energy sources has been developed. It has been shown that object-oriented approach made it possible to develop a distributed computational algorithm and to automate definition assignment of the object geometry without essential changes of code. Calculations have been carried out on the multiprocessor computer MPC-15K. The parallel algorithm efficiency has been analyzed.

Index Terms—Parallel algorithms, computational modeling.

I. INTRODUCTION

A lot of natural and anthropogenic disasters lead to the emergence of heat-evolution sources. These sources often appear in the porous media. It is necessary to restrict efficiently and do away the heat sources because of their hazard for life. The only efficient method of disaster elimination is a gas cooling of the type media [1]. Mathematical model of the gas flow through a porous media is constructed on the common ideas of multiphase media mechanics. The problem at hand has next features. The self-heating porous media is open both from the top and from the bottom to the atmosphere. For such problem an original calculating method founded on the combination of implicit and explicit finite-difference schemes [2-3] was introduced. Note that due to the scheme step time limit the calculation can be quite long. It is most noticeable when the distance between grid points is getting smaller. Therefore, the computational algorithm was parallelized. This paper describes the model and analysis of the parallel computational algorithm for the two-dimensional (plane and axially symmetric) unsteady gas flow through a porous heat-evolution media with a step-curve upper part. Conversion process of the sequential algorithm to an object-oriented representation and its parallelization have been studied. The parallel algorithm efficiency has been analyzed. The results of computations have been presented.

II. MATH MODEL

Let us examine a porous heat-evolution media of height \( H \) bounded by impermeable non-heat-conducting walls. Suppose that the heat evolution into solid phase is taking place. The cold gas is injected under pressure in the lower part of the object, passing through the object and going into the atmosphere. Due to heat exchange gas is getting warmer. The distinction of the model is that gas consumption and velocity are unknown quantities. Mathematical model of the process can be written as:

\[
(1 - a)\rho_c c_v \frac{\partial T}{\partial t} = - \alpha (T - \bar{T}) + Q_s (1 - a)C + (1 - a)\lambda \Delta T,
\]

\[
apc \left( \frac{\partial T}{\partial t} + (\bar{v} \cdot \nabla)T \right) = \alpha (T - \bar{T}) + \alpha \left( \frac{\partial P}{\partial t} + (\bar{v} \cdot \nabla)P \right) +
\]

\[
a^2 \frac{\partial}{\partial \xi} \left( \frac{\partial P}{\partial \xi} \right) + \alpha \nabla \cdot \left( \frac{\partial P}{\partial \xi} \right) = -\text{grad} P - \rho \bar{g} - a \frac{\partial^2 P}{\partial \xi^2},
\]

\[
p = \rho RT, \quad \frac{\partial C}{\partial t} = -k_c C.
\]

With boundary conditions:

\[
p_{\xi = 0} = p_0(t), \quad T_{\xi = 0} = T_{\xi = 0}(t), \quad \frac{\partial P}{\partial \xi} \bigg|_{\xi = 0} = \frac{\beta}{\lambda}(T_{\xi = 0} - T_{\xi = H}),
\]

\[
v_{\xi = 0} = 0, \quad p_{\xi = H} = p_s, \quad \frac{\partial T}{\partial \xi} \bigg|_{\xi = H} = \frac{\beta}{\lambda}(T_{\xi = 0} - T_{\xi = H} - \Delta T),
\]

\[
\frac{\partial T}{\partial n} \bigg|_{\omega} = 0, \quad \frac{\partial T}{\partial n} \bigg|_{\omega} = 0, \quad v_{\xi = 0} = 0.
\]

Where \( a \) – porosity, \( c_v \) – heat capacity of condensed phase, \( c_p \) – gas heat capacity at constant pressure, \( c_{ij} \) and \( c_{ij} \) – constants in Satherland formula, \( g \) – acceleration of gravity, \( G \) – surface of side walls, \( k_i \) – permeability coefficient of condensed phase, \( k_2 \) – coefficient defining the heat-evolution decrease, \( n \) – side walls surface normal, \( p \) – gas pressure, \( Q_0 \) – constant defining the rate of heat-evolution, \( R \) – gas constant, \( t \) – time, \( T \) – condensed phase temperature, \( T_{\xi = 0} \) – gas temperature, \( \bar{\xi} \) – Euler coordinate, \( \bar{v} \) – gas velocity, \( \alpha \) – constant defining the rate of interphase heat exchange, \( \beta \) – heat-transfer
coefficient, $\lambda$ – heat conductivity of condensed phase, $\mu$ – gas dynamic viscosity, $\rho$ – gas density, $\rho_c$ – condensed phase density, $\chi$ – factor of apparent mass having taken into account the inertial interaction of phases at their accelerated relative motion.

III. SEQUENTIAL COMPUTATIONAL ALGORITHM

We are going to solve the equation system (1) numerically, using the finite-difference approximation. According to [3-4] the momentum and energy equations turn into explicit finite-difference equations from which gas and solid phase temperatures and filtration velocity are resolved. The continuity equation transforms into implicit finite-difference equation from which gas pressure is resolved. The sequential program is implemented as follows. Input and output data are presented by 6 different 3D static arrays of gas temperature, gas pressure, gas density, solid phase temperature, vertical and horizontal filtration velocity accordingly.

The preliminary stage consists of constants calculation and reading of arrays initial values from files. The calculation stage implements time cycle where values of the current time layer are calculated from values of the previous layer. Unknown quantities are determined as follows: having “horizontal” index fixed; for each “vertical” index four explicit equations (for the gas temperature, solid medium temperature and components of gas filtration velocity) are solved; then the implicit equation is solved using sweep method (for gas pressure); then the gas density is trivially resolved; after that computations are moved to next horizontal index. Note that for different horizontal coordinates the sweep method algorithm depends upon the penetrability of a porous media upper part. If the upper part is open to atmosphere, the forward sweep method is applied. If the upper part is impermeable, the backward sweep method is applied. Having all inner values computed, boundary conditions are calculated. The computational cycle is repeated with time step shift for each time layer through the whole required time period. In the final stage the result array values are written to 6 files and overhead information to 7th file.

Notice that general algorithm complexity is estimated as $O\left(\frac{1}{h^2}\right)$, where $h$ is a dimensional step and $\tau = rh^2$ - a time step. But it has been found out empirically, if $h$ decreases then $r$ is usually possible to increase. Hence, the complexity of algorithm could come up to $O\left(\frac{1}{h^2}\right)$.

IV. PARALLELIZATION

The computational algorithm features (in particular forward and backward sweep method, complex nested loops) prevent automatic and semiautomatic tools to parallelize the code. Consideration of most popular decomposition methods reveals that loop exchange and loop index conversion techniques don’t fit the computational algorithm because of the sweep method and complex regions. Functional decomposition is also inefficient because of the too small calculation time for the variables in a mesh. The most natural method for the problem is a geometric decomposition method, widely used for calculations on uniform grids. Computational algorithm used allows geometric decomposition only by horizontal (radial) coordinate, as the sweep method prevent it on the vertical (axial) coordinate. The parallel version of the computational algorithm developed “manually” have been based on the source sequential program with the use of MPI communication library. The only stage subjected to parallelizing is computational, since preliminary and final stages don’t spend too much time. The object-oriented technique used for parallel program development is rarely used in a computational programming because of incidental executable code slow down. Nevertheless, this approach has a number of advantages compensating speed loss in the long run. The object-oriented technique allows to represent a program as a set of objects, interacting by programmer defined rules, each of object implements own subtask in the developing system. Competently developed inheritance tree allows to distribute the unwieldy mathematical formulae between objects, minimizing risk of difficult-to-locate error appearance. Such approach facilitates the changes in computational formulae, for example in case of math model complication or simplification. Changes are isolated within the functions covering certain computation type like boundary conditions or calculation of temperature, pressure, density in inner points. The object-oriented technique considerably simplifies the modification of data representation and placement in computer memory. It allows efficiently manage the cache loading and computation domains distribution between nodes of multiprocessor computer as well.

When the source program was converted to the object-oriented representation, following modifications were made:

- The functions responsible for calculations and data reading and recording functions were grouped in classes corresponding different types of calculations (“bar” with open or close upper part, edge zones).
- Formulae were converted to relative coordinates instead of absolute coordinates. In absolute coordinates, all three indexes should be given in absolute value, i.e. from the origin, to access array cell. But in relative coordinates the value in the cell is expressed by formula dependent on values of adjacent cells addressed relating to calculated cell index. This form of the formulae allows computing the grid of any dimension and configuration.
- Data structures were restructured so that the proper cell values of the initial data arrays were united. Thus the values needed to recalc are in adjacent memory addresses. It reduces the number of cache pages load-unload procedure calls.

Calculations in the program are geometrically decomposed by horizontal coordinate $m$. Along with it,
in order to ensure the even load of computational nodes, it is preliminary solved the problem of finding domains that are comparable with each other by a computational load. For this the calculation domain was broken by \( m \in [0,m_{\text{max}}] \) coordinate into \( k \) parts in the following way: ascending sequence of \( k+1 \) natural numbers \( \{m_j | 0 \leq j \leq k, m_0 = 0, m_k = m_{\text{max}} \} \) is constructed so that the calculation domain for \( j \)-th node is defined by the coordinate values \( m \in [m_{j-1}, m_j) \). Taking to account that computational load is nearly proportional to area, the calculation domain area function \( S(m) \) has been used for the load estimation while the specified sequence being constructed. Since the calculation load of node \( j \) is proportional to \( (m_j - m_{j-1}) \) (and in the ideal the area of calculation domain should be equal to \( S(m_{\text{max}}) \)), the simple iteration algorithm allows to calculate the values of sequence \( m_j \) knowing \( m_{j-1} \) from \( j=1 \) to \( j=k \) using condition:

\[
S(m_{j-1}) - S(m_{j-1}) < S(m_j) - S(m_{j-1})
\]

This algorithm doesn't give the best solution in sense of quadratic deviation, but its low computational complexity is the factor determining its usage.

The peculiarity of object-oriented version of the parallel program implementation is the support of an arbitrary computational domain configuration. The configuration is specified in text form in the settings file as the sequence of open and closed “bars” that are described by its heights and widths. At the beginning of the program, the computational domain configuration is read by the main node and the list of objects controlling calculations on vertical (axial) coordinate is formed. Every object in the list is assigned a node number, which should carry out the calculations, then nodes pick out their “own” objects in their lists to make the computational iterations.

The preliminary and final stages of the parallel program coincide with respective ones of the sequential program and are carried out on the main node. At the beginning of the parallel computation stage, \( m_j \) are calculated and the initial values of computational domains are sent to the respective nodes. Next, in the time iterations cycle every node completes the calculation stage algorithm within the coordinates of its computational domain and exchanges the boundary cells with adjacent nodes at the end of each iteration. At the end of the parallel computation stage every node sends calculated values to the main node.

V. TESTING

The parallel program was tested calculating plane and axially symmetric gas flows. Results were obtained exactly the same as ones of the sequence program regardless of computational nodes number used. Example of the physical quantities distribution in the porous heat-evolution object is illustrated on fig. 1. Note that all calculations have been executed as nondimensional.

![Fig. 1. Distribution of the solid medium temperature, gas temperature, gas pressure and density, vertical and horizontal filtration velocity in the porous heat-evolution object.](image1)

![Fig. 2. Dependence of the working time of parallel program \( t \) from numbers of processor \( N \) for grid with \( h=1/40 \) (A) and \( h=1/100 \) (B).](image2)

Theoretical dependence of the parallel program working time from numbers of processors cores is expressed by formula

\[
T = \frac{C_1}{Nh^2} + \frac{C_2}{h}(\text{sign}(N - 2) + 1),
\]

where \( N \) - numbers of processor, \( h \) - dimensional step, \( C_1 \) is responsible for calculation complexity of the mesh, \( C_2 \) is a constant defining the communication environment efficiency. Having only two processors the single-side data exchange occurs. When the number of processors is greater than two, every node except outside left and right makes both sides data exchange. Plots on the fig. 2 are quite good confirmed by theoretical formula. It shows that program has been parallelized perfectly well and allows to identify the optimal number of processors for the problem at hand. It can be seen from the plots that optimal number of processors for the problem, after which the increase of CPU number doesn't make sense, increases with mesh refinement. For a quite crude mesh
with dimensional step $h=1/40$ it is reasonable to use 4 CPU, but for $h=1/10$ optimal number of CPU is 8.

VI. CONCLUSION

The parallel computational algorithm for the two-dimensional (plane and axially symmetric) unsteady gas flows through a porous heat-evolution media in the gravity field has been developed. The conversion process of the sequential program to object-oriented representation and consequent parallelization based on the geometric decomposition technique have been examined in detail. It is shown that object-oriented representation allows to create parallel programs without essential changing of the initial sequential program. The calculations of heat-evolution porous media with a step-curved upper part have been carried out. Efficiency of developed parallel algorithm has been demonstrated.

REFERENCES