INFORMATION TECHNOLOGY, COMPUTER SCIENCE AND MANAGEMENT

UDC 004.272.22
https://doi.org/10.23947/2687-1653-2023-23-3-329-339

Original article

Model of a Parallel-Pipeline Computational Process for Solving a System of Grid Equations

Vladimir N. Litvinov[1] ✉, **Nelli B. Rudenko**[2], **Natalya N. Gracheva**[2]
[1]Don State Technical University, Rostov-on-Don, Russian Federation
[2] Azov-Black Sea Engineering Institute, Don State Agrarian University, Zernograd, Russian Federation
✉ LitvinovVN@rambler.ru

Abstract
Introduction. Environmental problems arising in shallow waters and caused by both natural and man-made factors annually do significant damage to aquatic systems and coastal territories. It is possible to identify these problems in a timely manner, as well as ways to eliminate them, using modern computing systems. But earlier studies have shown that the resources of computing systems using only a central processor are not enough to solve large scientific problems, in particular, to predict major environmental accidents, assess the damage caused by them, and determine the possibilities of their elimination. For these purposes, it is proposed to use models of the computing system and decomposition of the computational domain to develop an algorithm for parallel-pipeline calculations. The research objective was to create a model of a parallel-conveyor computational process for solving a system of grid equations by a modified alternating-triangular iterative method using the decomposition of a three-dimensional uniform computational grid that takes into account technical characteristics of the equipment used for calculations.
Materials and Methods. Mathematical models of the computer system and computational grid were developed. The decomposition model of the computational domain was made taking into account the characteristics of a heterogeneous system. A parallel-pipeline method for solving a system of grid equations by a modified alternating-triangular iterative method was proposed.
Results. A program was written in the CUDA C language that implemented a parallel-pipeline method for solving a system of grid equations by a modified alternating-triangular iterative method. The experiments performed showed that with an increase in the number of threads, the computation time decreased, and when decomposing the computational grid, it was rational to split into fragments along coordinate z by a value not exceeding 10. The results of the experiments proved the efficiency of the developed parallel-pipeline method.
Discussion and Conclusion. As a result of the research, a model of a parallel-pipeline computing process was developed using the example of one of the most time-consuming stages of solving a system of grid equations by a modified alternating-triangular iterative method. Its construction was based on decomposition models of a three-dimensional uniform computational grid, which took into account the technical characteristics of the equipment used in the calculations. This program can provide you for the acceleration of the calculation process and even loading of program flows in time. The conducted numerical experiments validated the mathematical model of decomposition of the computational domain.

Keywords: parallel algorithm, computational process, grid equations

Acknowledgements: the authors would like to thank the editorial board of the journal and the reviewer for their professional analysis and recommendations for correcting the article.

Funding information. The research was done with the support of the Russian Science Foundation (project No. 21-71-20050).

For citation. Litvinov VN, Rudenko NB, Gracheva NN. Model of a Parallel-Pipeline Computational Process for Solving a System of Grid Equations. *Advanced Engineering Research (Rostov-on-Don).* 2023;23(3):329–339. https://doi.org/10.23947/2687-1653-2023-23-3-329-339

Научная статья

Разработка модели параллельно-конвейерного вычислительного процесса для решения системы сеточных уравнений

В.Н. Литвинов[1] ✉, **Н.Б. Руденко**[2], **Н.Н. Грачева**[2]
[1] Донской государственный технический университет, г. Ростов-на-Дону, Российская Федерация,
[2] Азово-Черноморский инженерный институт, ДГАУ, г. Зерноград, Российская Федерация
✉ LitvinovVN@rambler.ru

Аннотация
Введение. Экологические проблемы, возникающие на мелководных водоёмах и вызываемые как природными, так и техногенными факторами, ежегодно наносят существенный ущерб аквасистемам и прибрежным территориям. Своевременно определить эти проблемы, а также пути их устранения возможно с использованием современных вычислительных систем. Но проведённые ранее исследования показали, что ресурсов вычислительных систем, использующих только центральный процессор, недостаточно для решения больших научных задач, в частности, по прогнозированию крупных экологических происшествий, оценке нанесенного ими ущерба и определению возможностей их устранения. Для этих целей предлагается использовать модели вычислительной системы и декомпозиции расчётной области для разработки алгоритма параллельно-конвейерных вычислений. Целью данной работы является создание модели параллельно-конвейерного вычислительного процесса для решения системы сеточных уравнений модифицированным попеременно-треугольным итерационным методом с использованием декомпозиции трёхмерной равномерной расчётной сетки, учитывающей технические характеристики используемого для расчетов оборудования.
Материалы и методы. Разработаны математические модели вычислительной системы и расчётной сетки. Модель декомпозиции расчётной области выполнена с учётом характеристик гетерогенной системы. Предложен параллельно-конвейерный метод решения системы сеточных уравнений модифицированным попеременно-треугольным итерационным методом.
Результаты исследования. На языке CUDA C написана программа, реализующая параллельно-конвейерный метод решения системы сеточных уравнений модифицированным попеременно-треугольным итерационным методом. Проведённые эксперименты показали, что с увеличением числа потоков время вычислений уменьшается и при декомпозиции расчётной сетки рациональным является разбиение на фрагменты по координате z на величину, не превышающую 10. Результаты экспериментов подтвердили эффективность разработанного параллельно-конвейерного метода.
Обсуждение и заключения. По итогам проведенных исследований разработана модель параллельно-конвейерного вычислительного процесса на примере одного из самых трудоёмких этапов решения системы сеточных уравнений модифицированным попеременно-треугольным итерационным методом. Её построение основано на моделях декомпозиции трёхмерной равномерной расчётной сетки, учитывающей технические характеристики используемого в расчетах оборудования. Применение программы позволит ускорить процесс расчёта и равномерно по времени загрузить программные потоки. Проведенные численные эксперименты подтвердили математическую модель декомпозиции расчётной области.

Ключевые слова: параллельный алгоритм, вычислительный процесс, сеточные уравнения

Благодарности: авторы выражают благодарность редакционной коллегии журнала и рецензенту за профессиональный анализ и рекомендации для корректировки статьи.

Финансирование. Работа выполнена при поддержке Российского научного фонда (проект № 21–71–20050).

Для цитирования. Литвинов В.Н., Руденко Н.Б., Грачева Н.Н. Разработка модели параллельно-конвейерного вычислительного процесса для решения системы сеточных уравнений. *Advanced Engineering Research (Rostov-on-Don).* 2023;23(3):329–339. https://doi.org/10.23947/2687-1653-2023-23-3-329-339

Introduction. Recently, a number of serious environmental problems have been observed in the Rostov region. These include, in particular, the eutrophication of waters of the Sea of Azov and the Tsimlyansk reservoir, which causes the growth of harmful and toxic species of phytoplankton populations [1]. Engineering works in the waters of rivers and seas cause pollution of adjacent territories, changes in the population structure of biota, and deterioration of reproduction conditions of valuable and commercial fish. Climate change in the south of Russia has led to an increase in the number of cases of flooding of some territories in the area of the Taganrog Bay and the floodplain of the Don River caused by up and down surges. In the last decade, during the summer period, almost complete drainage of the Don riverbed was observed several times, which led to a complete stop of navigation. To predict the occurrence and development of such cases, to plan ways to address their consequences, to assess the damage caused by them, modern software systems built using high-precision mathematical models, numerical methods, algorithms and data structures are needed [2].

Mathematical models used in predicting natural and man-made disasters are based on systems of partial differential equations, such as Poisson, Navier-Stokes, diffusion-convection-reaction, and thermal conductivity equations. The numerical solution to such systems causes the need for operational storage of large amounts of data (in arrays of various structures) and the solution to systems of grid equations of high dimension exceeding 10^9. The amount of RAM required to store arrays of data when numerically solving only one Poisson equation for a three-dimensional domain with a dimension of 103×103×103 by an alternately triangular iterative method is more than 64 GB. In the case of numerical solution to combined tasks, hundreds of gigabytes of RAM are required, which can be accessed only when using supercomputer systems.

An earlier study has shown that the resources of a computing system using only the CPU are not enough to solve such scientific problems [3]. Increasing the GPU power and video memory made it possible to use video adapter resources for calculations [4]. The GPU utilization depends on the application of parallel algorithms to solve computationally intensive problems of aquatic ecology [5–7]. To partially solve the problems of lack of memory and computing power on workstations, you can install additional video adapters in PCI-E X16 slots directly and in PCI-E X1 slots using PCI-E X1–PCI-E X16 adapters. Thus, the number of video adapters installed on one workstation can be increased to 12 [8–11].

Heterogeneous computer systems that provide sharing CPU and GPU resources are becoming increasingly popular in the scientific community. Application of such systems makes it possible to reduce the computation time of scientific problems [12–14]. However, the utilization of a heterogeneous computing environment involves the modernization of mathematical models, algorithms and programs that implement them numerically. A heterogeneous system provides organizing the calculation process in parallel mode. At the same time, fundamental differences in the construction of software systems using CPU and GPU together should be taken into account.

Materials and Methods. We describe the proposed mathematical models of the computational system, the computational grid, as well as the method of decomposition of the computational domain.

Let D be a set of technical characteristics of a computing system, then

$$D = D^1 \bigcup D^2 \bigcup D^3, \tag{1}$$

where D^1 — a subset of the characteristics of the central processing units (CPU) of a computing system; D^2 — a subset of the characteristics of video adapters (GPU) of a computing system; D^3 — a subset of RAM characteristics.

$$D^1 = \left\langle d^{1,1}, d^{1,2}, d^{1,3}, d^{1,4} \right\rangle, \tag{2}$$

where $d^{1,1}$ — total number of CPU cores; $d^{1,2}$ — number of streams simultaneously processed by one CPU core; $d^{1,3}$ — clock rate, MHz; $d^{1,4}$ — CPU bus frequency, MHz.

$$D^2 = \bigcup_{k_{GPU} \in K_{GPU}} D^2_{k_{GPU}} = \left\{ d^2 \mid \exists k_{GPU} \in K_{GPU},\, d^2 \in D^2_{k_{GPU}} \right\}, \tag{3}$$

where $K_{GPU} = \{1, ..., N_{GPU}\}$ — multiple video adapter indices; N_{GPU} — number of computer system video adapters; k_{GPU} — video adapter index. Each video adapter is represented as a tuple

$$D^2_{k_{GPU}} = \left\langle d^{2,1}_{k_{GPU}}, d^{2,2}_{k_{GPU}} \right\rangle, \tag{4}$$

where $d^{2,1}_{k_{GPU}}$ — amount of video memory of the video adapter with index k_{GPU}, GB; $d^{2,2}_{k_{GPU}}$ — number of streaming multiprocessors.

$$D^3 = \left\langle d^{3,1}, d^{3,2} \right\rangle, \tag{5}$$

where $d^{3,1}$ — total amount of RAM, GB; $d^{3,2}$ — clock rate of RAM, MHz.

Let S — a set of software streams involved in the computing process, then

$$\begin{aligned} &S = S^1 \bigcup S^2, \\ &S^1 = \left\{1, ..., N_{S^1}\right\}, \\ &S^2 = \bigcup_{k_{GPU} \in K_{GPU}} S^2_{k_{GPU}}, \quad S^2_{k_{GPU}} = \left\{1, ..., N_{S^2_{k_{GPU}}}\right\}, \end{aligned} \tag{6}$$

where S^1 — a subset of program streams implementing the calculation process on the CPU; S^2 — a subset of CUDA streaming blocks implementing the calculation process on GPU streaming multiprocessors; N_{S^1} — number of CPU program streams involved; $S^2_{k_{GPU}}$ — a subset of CUDA streaming blocks that implement the calculation process on GPU streaming multiprocessors with index k_{GPU}; $K_{GPU}=\{1,...,N_{GPU}\}$ — multiple GPU indices; N_{GPU} — number of GPU in the computing system; $N_{S^2_{k_{GPU}}}$ — number of CUDA stream blocks involved that implement the calculation process on GPU stream multiprocessors with index k_{GPU}.

Let E be a set of identifiers of program streams. Then, in order to identify program flows in the computing system, we assign tuple e of two elements to each element of the set of program flows S:

$$\forall s \in S \;\; \exists e \in E: \; e=\langle n_d, n_t \rangle, \tag{7}$$

where n_d — index of a computing device in a computing system; n_t — index of the CPU program stream or GPU streaming unit.

$$n_d = \begin{cases} 0, & s \in S^1 \\ K_{GPU}, & s \in S^2 \end{cases}, \tag{8}$$

$$n_t = \begin{cases} K_{S^1}, & s \in S^1 \\ K_{S^2_{k_{GPU}}}, & s \in S^2_{k_{GPU}} \end{cases}. \tag{9}$$

Let us take the computational domain with the following parameters: l_x — characteristic size on axis Ox; l_y — on axis Oy; l_z — on axis Oz. We compare a uniform computational grid of the following type to the specified area:

$$\begin{aligned} W = \{ & x_i = ih_x, y_j = jh_y, z_k = kh_z; \\ & i=\overline{0,n_x-1}, j=\overline{0,n_y-1}, k=\overline{0,n_z-1}; \\ & (n_x-1)h_x = l_x, (n_y-1)h_y = l_y, (n_z-1)h_z = l_z \}, \end{aligned} \tag{10}$$

where h_x, h_y, h_z — steps of the computational grid in the corresponding spatial directions; n_x, n_y, n_z — number of nodes of the computational grid in the corresponding spatial directions.

Then, we represent the set of nodes of the computational grid in the form

$$\begin{aligned} & G=\{g_{i,j,k}, i=\overline{0,n_x-1}, j=\overline{0,n_y-1}, k=\overline{0,n_z-1}\}, \\ & g_{i,j,k}=\langle x_i, y_j, z_k \rangle, \end{aligned} \tag{11}$$

where $g_{i,j,k}$ — computational grid node.

The number of nodes of the computational grid N_G is calculated from formula

$$N_G = n_x \cdot n_y \cdot n_z. \tag{12}$$

By the subsection of the computational grid $G^{k_1} \subset G$ (hereinafter — subsection), we mean a subset of the nodes of the computational grid G.

$$G=\bigcup_{k_1\in K_{k_1}} G^{k_1} = \{ g^{k_1} \mid \exists k_1 \in K_{k_1}, g^{k_1} \in G_{k_1} \}, \; \bigcap_{k_1 \in K_{k_1}} G^{k_1} = \varnothing, \tag{13}$$

where $K_{k_1}=\{1,...,N_{k_1}\}$ — multiple indices of subsections G^{k_1} of computational grid G; N_{k_1} — number of subsections G^{k_1}; $K_{k_1}, N_{k_1} \subset N$; N — set of natural numbers; k_1 — index of subsection G^{k_1}.

Since $G^{k_1} \subset G$, then

$$G^{k_1} = \{ g^{k_1}_{i,\tilde{j},k}, i=\overline{0,n_x-1}, \tilde{j}=\overline{0, n_y^{k_1}-1}, k=\overline{0,n_z-1} \}, \tag{14}$$

where $g^{k_1}_{i,\tilde{j},k}$ — node of subsection k_1; sign $\sim$ indicates belonging to the subsection; $\tilde{j}$ — node index of subsection k_1 by coordinate y; $n_y^{k_1}$ — number of nodes in subsection k_1 by coordinate y.

$$g_{i,\tilde{j},k}^{k_1} = \langle x_i, y_j, z_k \rangle,$$
$$x_i = ih_x, y_j = \left(\sum_{b_1=1}^{k_1-1} n_y^{b_1} + \tilde{j} \right) \cdot h_y, z_k = kh_z, \tag{15}$$

where $n_y^{b_1}$ — number of nodes by coordinate y of b_1-th subsection.

Under the block of computational grid G^{k_1,k_2} (hereinafter — block), we mean a subset of the computational grid nodes of subsection G^{k_1}.

$$G^{k_1} = \bigcup_{k_2 \in K_{k_1,k_2}} G^{k_1,k_2} = \left\{ g^{k_1,k_2} \mid \exists k_2 \in K_{k_1,k_2}, g^{k_1,k_2} \in G^{k_1,k_2} \right\}, \bigcap_{k_2 \in K_{k_1,k_2}} G^{k_1,k_2} = \varnothing, \tag{16}$$

where $K_{k_1,k_2} = \left\{ 1,..., N_{k_1,k_2} \right\}$ — multiple indices of block G^{k_1,k_2} of subsection G^{k_1}; N_{k_1,k_2} — number of blocks G^{k_1,k_2}; $K_{k_1,k_2}, N_{k_1,k_2} \subset N$; k_2 — index of block G^{k_1,k_2} of subsection G^{k_1}.

Since $G^{k_1,k_2} \subset G^{k_1}$, then

$$G^{k_1,k_2} = \left\{ g_{i,\hat{j},k}^{k_1,k_2}, i = \overline{0, n_x - 1}, \hat{j} = \overline{0, n_y^{k_1,k_2} - 1}, k = \overline{0, n_z - 1} \right\}, \tag{17}$$

where $g_{i,\hat{j},k}^{k_1,k_2}$ — node of block k_1, k_2; sign $\wedge$ indicates belonging to the block; $\hat{j}$ — node index of block k_1, k_2 by coordinate y; $n_y^{k_1,k_2}$ — number of nodes in block k_1, k_2 by coordinate y.

$$g_{i,\hat{j},k}^{k_1,k_2} = \langle x_i, y_j, z_k \rangle,$$
$$x_i = ih_x, \quad y_j = \left(\sum_{b_1=1}^{k_1-1} \sum_{b_2=1}^{N_{k_1,k_2}} n_y^{b_1,b_2} + \hat{j} \right) \cdot h_y, \quad z_k = kh_z, \tag{18}$$

where $n_y^{b_1,b_2}$ — number of nodes of block b_1, b_2.

By a fragment of the computational grid G^{k_1,k_2,k_3} (hereinafter — fragment), we mean a subset of the nodes of the computational grid of block G^{k_1,k_2} of subsection G^{k_1}.

$$G^{k_1,k_2} = \bigcup_{k_3 \in K_{k_3}} G^{k_1,k_2,k_3} = \left\{ g^{k_1,k_2,k_3} \mid \exists k_3 \in K_{k_3}, g^{k_1,k_2,k_3} \in G^{k_1,k_2,k_3} \right\},$$
$$\bigcap_{k_3 \in K_{k_3}} G^{k_1,k_2,k_3} = \varnothing, \tag{19}$$

where $K_{k_1,k_2,k_3} = \left\{ 1,..., N_{k_1,k_2,k_3} \right\}$ — multiple indices of fragments G^{k_1,k_2,k_3} of block G^{k_1,k_2} of subsection G^{k_1}; N_{k_1,k_2,k_3} — number of fragments G^{k_1,k_2,k_3}; $K_{k_1,k_2,k_3}, N_{k_1,k_2,k_3} \subset N$; k_3 — index of fragment G^{k_1,k_2,k_3} of block G^{k_1,k_2} of subsection G^{k_1}.

Each index k_3 of fragment G^{k_1,k_2,k_3} is assigned a tuple of indices $\langle k_4, k_5 \rangle$, designed to store the fragment coordinates in the plane xOz, where k_4 — fragment index by coordinate x; k_5 — fragment index by coordinate z.

$$k_3 = k_4 + K_{k_4} \cdot k_5, \tag{20}$$

where k_4 — fragment index by coordinate x; k_5 – fragment index by coordinate z.

Number of fragments G^{k_1,k_2,k_3} of block G^{k_1,k_2} is calculated from the formula

$$K_{k_3} = K_{k_4} \cdot K_{k_5}, \tag{21}$$

where K_{k_4} — number of fragments along axis Ox; K_{k_5} — number of fragments by coordinate z.

Since $G^{k_1,k_2,k_3} \subset G^{k_1,k_2}$, then

$$G^{k_1,k_2,k_3} = \left\{ \breve{g}_{\breve{i},\hat{j},\breve{k}}, \quad \breve{i} = \overline{0, \breve{n}_x - 1}, \quad \hat{j} = \overline{0, \hat{n}_y - 1}, \quad \breve{k} = \overline{0, \breve{n}_z - 1} \right\}, \tag{22}$$

where $\breve{g}_{\breve{i},\hat{j},\breve{k}}$ — fragment node; sign $\smile$ indicates belonging to the fragment; $\breve{i},\breve{k}$ — fragment node indices by coordinates x, z; $\breve{n}_x$, $\breve{n}_z$ — number of nodes of the computational grid in the fragment by coordinates x, z; $\breve{l}_x$, $\breve{l}_z$ — fragment dimensions by coordinates x, z.

$$\begin{gathered}\breve{g}_{\breve{i},\hat{j},\breve{k}}=\langle x_i, y_j, z_k\rangle,\\ x_i=\left(\sum_{b=1}^{k_4-1}\breve{n}_b+\breve{i}\right)h_x,\ y_j=\hat{j}h_y,\ z_k=\left(\sum_{b=1}^{k_5-1}\breve{n}_b+\breve{k}\right)h_z,\end{gathered} \tag{23}$$

where $\breve{n}_b$ — number of nodes of b-th fragment.

We introduce a set of comparisons of the computational grid blocks to program flows M^1

$$M^1=\bigcup_{k_1\in K_{k_1}}\left(\bigcup_{k_2\in K_{k_2}} M^1_{k_1,k_2}\right), \tag{24}$$

where $M^1_{k_1,k_2}$ — element of the set M^1.

Let $M^1_{k_1,k_2}$ — mapping block G^{k_1,k_2} to program stream s_{k_1,k_2}, then

$$M^1_{k_1,k_2}=\left\langle G^{k_1,k_2}, s_{k_1,k_2}\right\rangle, \tag{25}$$

where $s_{k_1,k_2}\in S$ — program flow, computing block G^{k_1,k_2}.

In the process of solving hydrodynamic problems on three-dimensional computational grids of large dimension, high-performance computing systems and huge amounts of memory for data storage are needed. The resources of one computing device are not enough for computing and storing a three-dimensional computational grid with all its data. To solve this problem, various methods of decomposition of computational grids followed by the use of parallel calculation algorithms in heterogeneous computing environments are proposed [15].

For the decomposition of the computational grid, it is required to take into account the performance of computing devices involved in calculations. By performance, we mean the number of nodes of the computational grid calculated using a given algorithm per unit of time.

Assume that all computing devices are used for calculations. Then, the total performance of the computing system P_Σ is calculated from the formula

$$P_\Sigma=P_{CPU}\cdot N_{S^1}+\sum_{b=1}^{N_{GPU}}P^b_{GPU}\cdot N^b_{S^2}, \tag{26}$$

where P_{CPU} — performance of a single CPU stream; N_{S^1} — number of program streams implementing the calculation process on the CPU; P^b_{GPU} — GPU performance with index b on a single streaming multiprocessor; $N^b_{S^2}$ — number of CUDA streaming blocks implementing the calculation process on GPU streaming multiprocessors.

Then, the number of nodes of the computational grid n^b_y in the subsection by coordinate y for each GPU with index b can be calculated from the formula

$$n^b_y=\left\lfloor\frac{P^b_{GPU}}{P_\Sigma}\right\rfloor\cdot n_y. \tag{27}$$

In the process of calculating by formula (27), we get the remainder — a certain number of nodes of the computational grid. These nodes will be located in RAM. The number of remaining nodes n^b_y by coordinate y is calculated from the formula:

$$n^b_{yL}=n_y-\sum_{b=1}^{N_{GPU}}n^b_y. \tag{28}$$

To calculate the number of nodes by coordinate y in the blocks of the computational grid processed by GPU streaming multiprocessors, we use the formulas:

$$n^b_{yGT} = \left\lfloor \frac{n^b_y}{N^b_{s^2} - 1} \right\rfloor, \quad b = \overline{1, N^b_{s^2} - 1},$$
$$n^b_{yGTL} = n^b_y - \sum_{b=1}^{N^b_{s^2}-1} n^b_{yGT}, \tag{29}$$

where n^b_{yGT} — number of nodes by coordinate y in computational grid blocks processed by GPU streaming multiprocessors with index b, except for the last block; n^b_{yGTL} — number of nodes by coordinate y in the last block of the computational grid processed by GPU streaming multiprocessors with index b.

To calculate the number of nodes of the computational grid by coordinate y in blocks processed by software streams implementing the calculation process on the CPU, we use the formulas

$$n_{yCT} = \left\lfloor \frac{n^{CPU}_y}{N_{s^1} - 1} \right\rfloor,$$
$$n_{yCTL} = n^{CPU}_y - n_{yCT} \cdot \left(N_{s^1} - 1\right), \tag{30}$$

where n_{yCT} – number of nodes of the computational grid by coordinate y, processed by CPU program streams, except the last stream; n_{yCTL} — number of nodes of the computational grid by coordinate y, processed by CPU program streams, in the last stream.

Calculate the number of the computational grid fragments by coordinate y:

$$N^f_y = N_{s^1} + \sum_{b=1}^{N_{GPU}} N^b_{s^2}. \tag{31}$$

Let the number of fragments N^f_x and N^f_z be specified by coordinates x and z, respectively. Then, the number of nodes of the computational grid by coordinate x is calculated using the formulas

$$n^f_x = \left\lfloor \frac{n_x}{N^f_x - 1} \right\rfloor,$$
$$n^{fL}_x = n_x - n^f_x \cdot \left(N^f_x - 1\right), \tag{32}$$

where n^f_x — number of nodes of the computational grid by coordinate x in all fragments, except the last fragment; n^{fL}_x — number of nodes of the computational grid by coordinate x in the last fragment.

Similarly, the number of nodes of the computational grid is calculated by coordinate z:

$$n^f_z = \left\lfloor \frac{n_z}{N^f_z - 1} \right\rfloor,$$
$$n^{fL}_z = n_z - n^f_z \cdot \left(N^f_z - 1\right), \tag{33}$$

where n^f_z — number of nodes of the computational grid by coordinate z in all fragments, except the last node; n^{fL}_z — number of nodes of the computational grid by coordinate z in the last fragment.

Let us describe a model of the parallel-pipeline method. Suppose it is necessary to organize a parallel process of computing some function F on M^1, and the calculations in each fragment G^{k_1,k_2,k_3} depend on the values in neighboring fragments, each of which has at least one of the indices by coordinates x, y, z, and one less than the current one (Fig. 1).

To organize the parallel-pipeline method, we introduce a set of tuples A, that specify correspondences a between the identifiers of program streams e, processing fragments G^{k_1,k_2,k_3}, to the step numbers of the parallel-pipeline method r

$$\forall e \in E \;\; \exists a \in A: \; a = \left\langle e, G^{k_1,k_2,k_3}, r \right\rangle, \tag{34}$$

where $r = \overline{1, N_r}$ — step number of the parallel-pipeline method;

N_r — number of steps of the parallel-pipeline method, calculated from the formula

$$N_r = N_x^f N_z^f + N_y^f - 1. \qquad (35)$$

The full load of all calculators in the proposed parallel-pipeline method starts with step $r_{100START} = N_y^f$ and ends at step $r_{100STOP} = N_x^f N_z^f$. At the same time, the total number of steps with a full load of calculators N_{rPAR} will be

$$N_{rPAR} = r_{100STOP} - r_{100START} + 1 = N_x^f N_z^f - N_y^f + 1. \qquad (36)$$

The calculation time of some function F by the parallel-pipeline method is written as

$$T_{M^1} = \sum_{r=1}^{N_r} \max(\mathrm{T}_a), \qquad (37)$$

where T_a — vector of values of time spent on processing fragments in parallel mode.

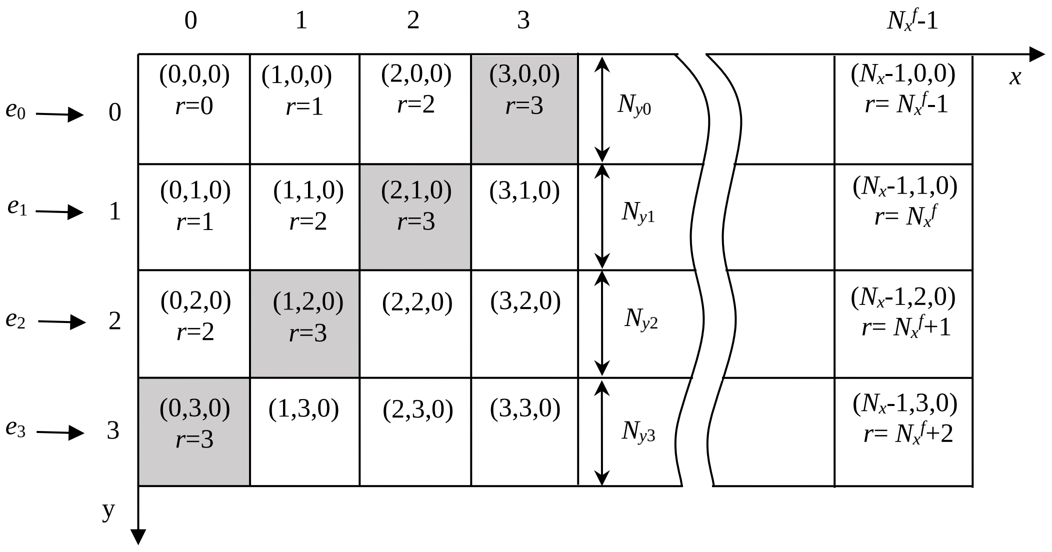

Fig. 1. Parallel-pipelined computing process

Research Results. The computational experiments were carried out on K-60 high-performance computing system of the Keldysh Applied Mathematics Institute, RAS. A GPU section was used, each node of which was equipped with two Intel Xeon Gold 6142 v4 processors, four Nvidia Volta GV100GL video adapters and 768 GB of RAM.

The computational experiment consisted of two stages — preparatory and basic. At the preparatory stage, the correctness of the decomposition of the computational domain into subsections, blocks and fragments was checked by step-by-step comparison of values in the nodes of the initial grid and in fragments obtained as a result of decomposition. Then, the operation of the flow control algorithm during which the time spent on calculating 1, 8, 16 and 32 fragments of the computational grid with a dimension of 50 nodes by spatial coordinates x, y, z, and the same number of CPU streams N_{S^1} was checked by the iterative alternating-triangular method in parallel mode. Ten repetitions were performed with the calculation of the arithmetic mean T_a and standard deviation σ. Based on the data obtained, time $T_a^1 = T_a / N_{S^1}$, spent by each stream on processing one fragment of the computational grid and acceleration $E = T_a^1(N_{S^1}) / T_a^1(1)$, equal to the ratio of the processing time $T_a^1(N_{S^1})$ of one fragment N_{S^1} by streams to the corresponding processing time by one stream $T_a^1(1)$ was calculated. The experimental data are given in Table 1. The experiment showed that the standard deviation had the smallest value in the case of using 32 parallel CPU streams and was 0.026 ms, i.e., using 32 parallel CPU streams when calculating 32 fragments of the computational grid gave a more uniform time load of program streams, which generally increased the efficiency of the computing node. At the same time, the average value of the calculation of one fragment was 4.14 ms. The dependence of acceleration E on the number of streams turned out to be linear $E = 0.603 + 0.804 N_{S^1}$, with a coefficient of determination equal to 0.99. We have found that with an increase in the number of streams, the acceleration of the developed algorithm increases. This indicates the efficient use of the subsystem when working with memory.

Table 1

Results of the preparatory stage of the computational experiment

N_{S^1}	$\max(T_a)$, ms	σ, ms	T_a, ms	E
1	3.38	0.141	3.38	1.00
8	3.66	0.042	0.46	7.39
16	3.94	0.028	0.25	13.73
32	4.14	0.026	0.13	26.13

At the basic stage of the computational experiment, a three-dimensional computational domain having dimensions of 1,600; 1,600; 200 by spatial coordinates x, y and z, accordingly, was divided into 32 fragments of 50 nodes for each of the coordinates x and y. The division into fragments by coordinate z is given in Table 2. For each decomposition option with a tenfold repetition, the processing time of the entire computational grid was measured by the proposed parallel-conveyor method, and its average value T_{pm} was calculated. Acceleration E_{pm} was calculated as ratio T_{pm} to time T_{sm} of the calculation by sequential version of the algorithm, equal to 6.963 ms. Regression equation $E_{pm} = 7.35 + 1.97\ln(N_z^f)$ with a determination coefficient equal to 0.94 was obtained. Analysis of the results of the basic stage of the computational experiment showed a significant slowdown in growth E_{pm} at $N_z^f > 10$. Therefore, we conclude that splitting into fragments by coordinate z by an amount not exceeding 10 is optimal.

Table 2

Results of the main stage of the computational experiment

N_z^f	n_z^f	T_{pm}, ms	E_{pm}
1	200	1033.20	6.74
2	100	779.00	8.94
4	50	651.90	10.68
8	25	588.35	11.84
20	10	550.22	12.66

Discussion and Conclusion. As a result of the conducted research, a model of a parallel-pipeline computing process was developed by the example of one of the most intensive stages of solving a system of grid equations by a modified alternating-triangular iterative method. Its construction was based on decomposition models of a three-dimensional uniform computational grid, taking into account the technical characteristics of the equipment used in the calculations.

The results obtained under the computational experiments validated the effectiveness of the developed method. The correctness of the decomposition of the computational domain into subsections, blocks and fragments was also confirmed. The operation of the flow control algorithm was verified. At the same time, it was revealed that the standard deviation had the smallest value in the case of using 32 parallel CPU streams and is 0.026 ms, i.e., using 32 parallel CPU streams when calculating 32 fragments of the computational grid gave a more uniform time load of program streams. Here, the average value of the calculation of one fragment was 4.14 ms.

The results of processing the measurements of the calculation time by the proposed parallel-conveyor method showed a significant slowdown in the growth of acceleration when divided into fragments by coordinate z at $N_z^f > 10$. It was found that splitting into fragments by coordinate z by an amount not exceeding 10 was optimal.

References

1. Shiganova TA, Alekseenko E, Kazmin AS. Predicting Range Expansion of Invasive Ctenophore *Mnemiopsis leidyi* A. Agassiz 1865 under Current Environmental Conditions and Future Climate Change Scenarios. *Estuarine, Coastal and Shelf Science*. 2019;227:106347. https://doi.org/10.1016/j.ecss.2019.106347

2. Sukhinov AI, Chistyakov AE, Nikitina AV, Filina AA, Lyashchenko TV, Litvinov VN. The Use of Supercomputer Technologies for Predictive Modeling of Pollutant Transport in Boundary Layers of the Atmosphere

and Water Bodies. In book: L Sokolinsky, M Zymbler (eds). *Parallel Computational Technologies*. Cham: Springer; 2019. P. 225–241. 10.1007/978-3-030-28163-2_16

3. Rodriguez D, Gomez D, Alvarez D, Rivera S. A Review of Parallel Heterogeneous Computing Algorithms in Power Systems. *Algorithms*. 2021;14(10):275. https://doi.org/10.3390/a14100275

4. Abdelrahman AM Osman. GPU Computing Taxonomy. In ebook: Wen-Jyi Hwang (ed). *Recent Progress in Parallel and Distributed Computing*. London: InTech; 2017. http://dx.doi.org/10.5772/intechopen.68179

5. Parker A. GPU Computing: The Future of Computing. In: *Proceedings of the West Virginia Academy of Science*. Morgantown, WV: WVAS; 2018. Vol. 90 (1). 10.55632/pwvas.v90i1.393

6. Nakano Koji. Theoretical Parallel Computing Models for GPU Computing. In book: Ç Koç (ed). *Open Problems in Mathematics and Computational Science*. Cham: Springer; 2014. P. 341–359. 10.1007/978-3-319-10683-0_14

7. Bhargavi K, Sathish Babu B. GPU Computation and Platforms. In book: Ganesh Chandra Deka (ed). *Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing*. Hershey, PA: IGI Global; 2016. P.136–174. 10.4018/978-1-4666-8853-7.ch007

8. Ebrahim Zarei Zefreh, Leili Mohammad Khanli, Shahriar Lotfi, Jaber Karimpour. 3-D Data Partitioning for 3-Level Perfectly Nested Loops on Heterogeneous Distributed System. *Concurrency and Computation: Practice and Experience*. 2017;29(5):e3976. https://doi.org/10.1002/cpe.3976

9. Fan Yang, Tongnian Shi, Han Chu, Kun Wang. The Design and Implementation of Parallel Algorithm Accelerator Based on CPU-GPU Collaborative Computing Environment. *Advanced Materials Research*. 2012;529:408–412. https://doi.org/10.4028/www.scientific.net/AMR.529.408

10. Varshini Subhash, Karran Pandey, Vijay Natarajan. A GPU Parallel Algorithm for Computing Morse-Smale Complexes. *IEEE Transactions on Visualization and Computer Graphics*. 2022. P. 1–15. 10.1109/TVCG.2022.3174769

11. Leiming Yu, Fanny Nina-Paravecino, David R Kaeli, Qianqian Fang. Scalable and Massively Parallel Monte Carlo Photon Transport Simulations for Heterogeneous Computing Platforms. *Journal of Biomedical Optics*. 2018;23(1):010504. https://doi.org/10.1117/1.JBO.23.1.010504

12. Fujimoto RM. Research Challenges in Parallel and Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation*. 2016;26(4):1–29. https://doi.org/10.1145/2866577

13. Qiang Qin, ChangZhen Hu, TianBao Ma. Study on Complicated Solid Modeling and Cartesian Grid Generation Method. *Science China Technological Sciences*. 2014;57:630–636. 10.1007/s11431-014-5485-5

14. Seyong Lee, Jeffrey Vetter. Moving Heterogeneous GPU Computing into the Mainstream with Directive-Based, High-Level Programming Models. In: *Proc. DOE Exascale Research Conference*. Portland, Or; 2012.

15. Thoman P, Dichev K, Heller Th, Iakymchuk R, Aguilar X, Hasanov Kh, et al. A Taxonomy of Task-Based Parallel Programming Technologies for High-Performance Computing. *Journal of Supercomputing*. 2018;74(2):1422–1434. https://doi.org/10.1007/s11431-014-5485-5

Received 17.07.2023
Revised 14.08.2023
Accepted 18.08.2023

About the Authors:

Vladimir N. Litvinov, Cand.Sci. (Eng.), Associate Professor of the Mathematics and Informatics, Don State Technical University (1, Gagarin sq., Rostov-on-Don, 344003, RF), ResearcherID, ScopusID, ORCID, LitvinovVN@rambler.ru

Nelli B. Rudenko, Cand.Sci. (Eng.), Associate Professor, Associate Professor of the Mathematics and Bioinformatics Department, Azov-Black Sea Engineering Institute, Don State Agrarian University (21, Lenina St., Zernograd, 347740, RF), ScopusID, ORCID, nelli-rud@yandex.ru

Natalya N. Gracheva, Cand.Sci. (Eng.), Associate Professor of the Mathematics and Bioinformatics Department, Azov-Black Sea Engineering Institute, Don State Agrarian University (21, Lenina St., Zernograd, 347740, RF), ScopusID, ORCID, grann72@mail.ru

Claimed Contributorship:

VN Litvinova: basic concept formulation, research objectives and tasks, development of algorithms for performing a computational experiment, calculation analysis.

NB Rudenko: analysis of research results, drawing conclusions, text preparation.

NN Gracheva: writing program code for performing a computational experiment, preparing illustrations, finalizing the text, correction of the conclusions.

Conflict of interest statement: the authors do not have any conflict of interest.

All authors have read and approved the final manuscript.

Поступила в редакцию 17.07.2023
Поступила после рецензирования 14.08.2023
Принята к публикации 18.08.2023

Об авторах:

Владимир Николаевич Литвинов, кандидат технических наук, доцент кафедры математики и информатики Донского государственного технического университета (344003, РФ, г. Ростов-на-Дону, пл. Гагарина, 1), ResearcherID, ScopusID, ORCID, LitvinovVN@rambler.ru

Нелли Борисовна Руденко, кандидат технических наук, доцент, доцент кафедры математики и биоинформатики Азово-Черноморского инженерного института, ДГАУ (347740, РФ, г. Зерноград, ул. Ленина, 19), ScopusID, ORCID, nelli-rud@yandex.ru

Наталья Николаевна Грачева, кандидат технических наук, доцент кафедры математики и биоинформатики Азово-Черноморского инженерного института, ДГАУ (347740, Ростовская область, г. Зерноград, ул. Ленина, 19), ScopusID, ORCID, grann72@mail.ru

Заявленный вклад соавторов:

В.Н. Литвинов — формирование основной концепции, цели и задачи исследования, разработка алгоритмов для выполнения вычислительного эксперимента, проведение расчетов.

Н.Б. Руденко — анализ результатов исследований, формирование выводов, подготовка текста.

Н.Н. Грачева — написание программного кода для выполнения вычислительного эксперимента, подготовка иллюстраций, доработка текста, корректировка выводов.

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Все авторы прочитали и одобрили окончательный вариант рукописи.