UDC 519.6

# HYBRID ALGORITHM NEWTON METHOD FOR SOLVING SYSTEMS OF NONLINEAR EQUATIONS WITH BLOCK JACOBI MATRIX

*A.N. Khimich*[0000-0001-9284-139X], *V. A. Sydoruk*[0000-0003-0210-6020], *A.N. Nesterenko*[0000-0001-6174-1812]

V.M. Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine, Kyiv, Ukraine

.

**Abstract**. Systems of nonlinear equations often arise when modelling processes of different nature. These can be both independent problems describing physical processes and also problems arising at the intermediate stage of solving more complex mathematical problems. Usually, these are high-order tasks with a big count of un-knows, that better take into account the local features of processes or things that are modelled. Also, more accurate discrete models allow for more accurate solutions.

Usually, the matrices of such problems have a sparse structure. Often the structure of sparse matrices is one of next: band, profile, block-diagonal with bordering, etc. In many cases, the matrices of the discrete problems are symmetric and positively defined or half-defined.

The solution of systems of nonlinear equations is performed mainly by iterative methods based on the Newton method, which has a high convergence rate (quadratic) near the solution, provided that the initial approximation lies in the area of gravity of the solution. In this case, the method requires, at each iteration, to calculates the Jacobi matrix and to further solving systems of linear algebraic equations. As a consequence, the complexity of one iteration is $O(n^3)$.

Using parallel computations in the step of the solving of systems of linear algebraic equations greatly accelerates the process of finding the solution of systems of nonlinear equations.

In the paper, a new method for solving systems of nonlinear high-order equations with the Jacobi block matrix is proposed. The basis of the new method is to combine the classical algorithm of the Newton method with an efficient small-tile algorithm for solving systems of linear equations with sparse matrices. The times of solving the systems of nonlinear equations of different orders on the nodes of the SKIT supercomputer are given.

**Key words:** systems of nonlinear equations, hybrid algorithm, sparse matrices, systems of linear algebraic equations, high-performance computing.

## Introduction

Systems of nonlinear equations (SNE) often arise when modelling processes of different nature. These can be both stand-alone problems that describe physical processes and problems that arise in the intermediate stage of solving more complex mathematical problems. As a rule, these are problems of ultrahigh orders that allow better consideration to take into account the local characteristics of the process or phenomenon (occurrence) being modelled. Also, more accurate discrete models allow for more accurate approximations.

On the other hand, the matrices of such problems have a sparse structure. Most often it is band, profile, block-diagonal with bordering, etc. In many cases, the matrices of discrete problems are symmetric and positively defined or half -definite.

The solution SNE is carried out (Systems of nonlinear equations are solved) mainly by iterative methods based on Newton's method, which has a high convergence rate (quadratic) near the solution, provided that the initial approximation lies in the region of convergence of the solution. In this case, the method requires at each iteration the computation of the Jacobi matrix and further solution of systems of linear algebraic equations (SLAE). As a consequence, the complexity of one iteration is $O(n^3)$.

Parallelizing the calculation of the SLAE solution greatly accelerates the process of finding the SNE solution.

## Statement of the problem with approximate initial data

Let a system of *n* nonlinear equations be given

$$f(x) = 0 , \tag{1}$$

where $x = (x_1, x_2, \dots, x_n)^T$, $f(x) = (f_1(x), f_2(x), \dots, f_n(x))^T$ – is the *n*-dimensional vector of the desired solution and the *n*-dimensional vector-function, respectively.

Problem (1) is some approximation to the exact system of nonlinear equations $\varphi(x)=0$, and for these vector functions the inequality holds:

$$\|f(u)-\varphi(u)\|\leq \Delta$$

for any $n$-dimensional vector $u$.

To solve problem (1) initial approximation $x^{(0)}$ and required accuracy to get an approximation to the solution of the system $\varepsilon$ are given and the area in which the solution is sought is determined $D=\{a_i \leq x_i \leq b_i, \quad i=1,2,\ldots,n\}$. Herewith the initial approximation belongs to a given area $x^{(0)} \in D$. The lower index in the formulas denote the component numbers of the vectors, and the upper index will denote the iteration numbers.

If $A=\left\{\frac{\partial f_i}{\partial x_j}\right\}_{i,j=1}^{n}$ is the Jacobi matrix of system (1) (or some approximation to it), then the iterative process of Newton's method of finding the solution at a given initial approximation is written in the form

$$A^{(k)}w^{(k)}=-f\left(x^{(k)}\right) \tag{2}$$

where $w^{(k)}=x^{(k+1)}-x^{(k)}$ correction, $k$ = 0, 1, ... – is the iteration number. The next approximation to the solution is calculated by the formula:

$$x^{(k+1)}=x^{(k)}+w^{(k)} \tag{3}$$

As can be seen from formula (2) it is necessary to solve a system of linear algebraic equations, calculating the values of the vector function and the Jacobi matrix, at each iteration.

To obtain a solution of the system of nonlinear equations (1) with a given accuracy $\left\|x^{(k)}-x\right\| \leq \varepsilon$, the iterative process must be completed if the condition

$$\left\|f\left(x^{(k+1)}\right)\right\| \leq \frac{\varepsilon}{\left\|\left(A^{(k+1)}\right)^{-1}\right\|}, \tag{4}$$

where $A^{-1^{(k)}}$ is a matrix inverse to the Jacobi matrix calculated on the $k$-th iteration, and the relationship $\frac{\varepsilon}{\left\|A^{-1^{(k)}}\right\|}$ will be denoted as $\Delta$ [1]. Since to check condition (4) at each iteration it is necessary to rotate the Jacobi matrix, which requires a significant number of arithmetic operations, then on the first iterations check the more economical condition of the end of iterations $\left\|f\left(x^{(k)}\right)\right\| \leq \varepsilon$, and after its execution, we check conditions (4).

After completing the iterative process, the error of the obtained approximation to the solution of the problem with approximate data relative to the exact solution of the system with accurate data is calculated:

$$\|x_k - x\| \leq \varepsilon + \left\|A_k^{-1}\right\| \Delta,$$

where $x$ is the exact solution of the exact SNE.

It follows that when solving systems of nonlinear equations, most of the arithmetic operations are performed when calculating the values of the vector function, solving the corresponding SLAE and calculating the inverse matrix to find the decoupling estimate. In the general case, the complexity of Newton's iterative method is $O(n^3)$. To reduce the execution time of one iteration for solving SLAE, we will use a small-tile hybrid factorization algorithm, where $-f\left(x^{(k)}\right)$ is the right part of SLAE.

## Small-tile hybrid factorization algorithm of the sparse block-diagonal matrix with a border

Consider the problem of solving a system of linear algebraic equations

$$\tilde{A}\tilde{x}=\tilde{b}$$

with the symmetric positive-definite sparse matrix of order $n$.

The ideological prerequisite for the use of hybrid calculations in the processing of sparse matrices of arbitrary structure is the preliminary rearrangement of the structure of the original matrix by the method of parallel sections, which leads the matrix to a block-diagonal view with a border [2, 3]

$$A = P^T \tilde{A} P = \begin{pmatrix} D_{11} & 0 & 0 & \cdots & 0 & C_{1p} \\ 0 & D_{22} & 0 & \cdots & 0 & C_{2p} \\ 0 & 0 & D_{33} & & 0 & C_{3p} \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & & D_{p-1p-1} & C_{p-1p} \\ C_{p1} & C_{p2} & C_{p3} & \cdots & C_{pp-1} & D_{pp} \end{pmatrix},$$

$$x = P^T \tilde{x}, b = P^T \tilde{b},$$

where $P$ – permutation matrix, and blocks $D_{ii}$, $C_{pi}$, $C_{ip}$, $i = \overline{1, p-1}$, $D_{pp}$ retain a sparse structure, $p$ – the number of diagonal blocks in the matrix, $p \geq 3$.

It is natural for the method of parallel sections that the size of the last diagonal block is much smaller than the orders of the other diagonal blocks. Also, the structure of the block-diagonal matrix with a border allows to carry out parallel and independent processing of blocks. $D_{ii}$, $C_{pi}$, $C_{ip}$, $i = \overline{1, p-1}$.

Given the structure of the resulting matrix for data storage, it is advisable to use a block distribution. Moreover, the number of blocks should be chosen taking into account the number of processors and graphics accelerators that are involved in the calculations.

In Fig. 1 shows the profile of matrix fill by nonzero elements. As a result of the application of the method of parallel sections, the structure of the matrix takes the form shown in Fig. 2.

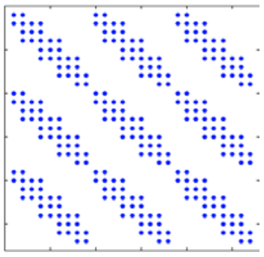

Fig. 1. The profile of the original matrix

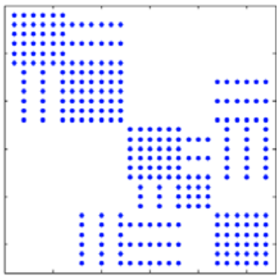

Fig. 2. Profile of block-diagonal matrix

As is well known, the most effective direct method of solving such a problem is the Kholetsky method. To solving of the system consists of solving the subtasks: triangular decomposition of the matrix of the system, to solving two SLAE with triangular matrices (7) and (8):

$$A = LL^T \tag{6}$$

$$Ly = b \tag{7}$$

$$L^T x = y \tag{8}$$

Consider a small-tile hybrid factorization algorithm of a sparse block-diagonal matrix with a border $A$. This algorithm better takes into account the profile or sparse structure of diagonal blocks, blocks with a border.

Divide matrix $A$ into blocks of dimension $c \times c$.

Further, to factorization a block-diagonal matrix, we apply the algorithm proposed in [4] for dense matrices.

To factorization the matrix during the $k$-th step, we use the following relations

$$A^{\kappa} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}$$

where the dimensions of the blocks $A_{11}$ – $c\times c$, $A_{21}$ – ($n$-$kc$)c, $A_{22}$ – ($n$-$kc$)($n$-$kc$), blocks and take into account the structure of the blocks $D_{ii}$, $C_{pi}$, $D_{pp}$.

From here we get the algorithm by which the decomposition is carried out during the $k$ step

$$A_{11} = L_{11} * L_{11}^T; \tag{9}$$

$$L_{21} = A_{21} * \left(L_{11}^T\right)^{-1}; \tag{10}$$

$$\tilde{A}_{22} = A_{22} - L_{21} * L_{21}^T. \tag{11}$$

Note that the implementation (9) - (10) at each step modifies only the blocks $D_{ii}$, $C_{pi}$, $i = \overline{1, p-1}$, $D_{pp}$.

To implement the algorithm we will use the following data distribution: in GPU($i$) $i = \overline{1, p-1}$ are containing blocks $D_{ii}$, $C_{pi}$ and block $A_{pp}^{(i)}$ the same size as $D_{pp}$; in GPU($p$) the $D_{pp}$ block is stored.

In fig. 3 show the block distribution of data at the $k$-th step of factorization of the block-diagonal matrix with a border, taking into account the above-proposed decomposition.

The parallelization of triangular factorization calculations is that the factorization of blocks $A_{11}$ and the modification of $A_{21}$ and $A_{22}$ can be done independently in each CPU($i$) and GPU($i$) $i = \overline{1, p-1}$.

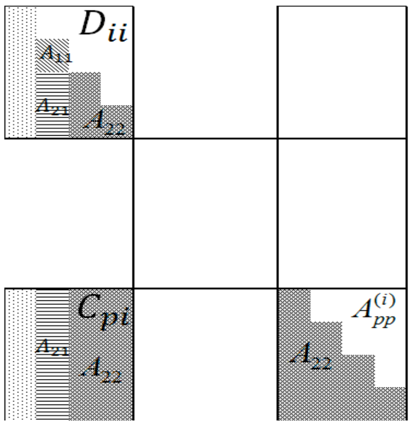

Fig. 3. Data decomposition on GPU($i$)

At each step in all pairs of CPU($i$) and GPU($i$) $i = \overline{1, p-1}$ execut:

- in CPU($i$), $i = \overline{1, p-1}$ we factorize $A_{11}$ with $D_{ii}$:

$$A_{11} = L_{11} * L_{11}^T;$$

- in GPU($i$), $i = \overline{1, p-1}$ we modify the column of blocks $L_{21}$:

$$L_{21} = A_{21}\left(L_{11}^T\right)^{-1};$$

- in GPU($i$), $i = \overline{1, p-1}$ we modify the blocks of the matrix $A_{22}$ with $A_{pp}^{(i)}$ by the formula:

$$\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T.$$

- In the CPU($p$), using multi-assembly, we modify $D_{pp}$:

$$D_{pp}^{*} = D_{pp} - \sum_{i=1}^{p-1} A_{pp}^{(i)}.$$

We factorize the block $D^*_{pp}$, thereby completing the process of factorization of the matrix.

We will assume that the orders of all diagonal blocks are approximately equal to

$$q_i \approx q = \frac{n-s}{p-1},$$

where $s$ is the order of the last diagonal block. Then the following theorems are valid.

**Theorem 1.** The number of operations, which performed on one GPU to find the factorization of a sparse block-diagonal matrix with a border is estimated by the value

$$N_p \approx \frac{q^3}{3} + sq^2 = \frac{q^2}{3}(q+3s).$$

**Proof.** We introduce the following notation $l = \frac{q}{c}$ the number of rows of tiles in the diagonal block.

Since (9) - (11) are performed in parallel and independently in all $p$-1 pairs of CPU and GPU and the maximum number of operations falls on step (11), the estimate of the number of operations performed on one GPU is determined by the complexity of step (11).

The number of operations required to perform (11) can be estimated by the value

$$N_p \approx 2c\left(\sum_{k=1}^{l}\frac{(q-k)^2}{2} + \sum_{k=1}^{l}(q-kc)s + l\frac{s^2}{2}\right). \quad (12)$$

Let's write the first term from (12)

$$2c\sum_{k=1}^{l}\frac{(q-k)^2}{2} = c\left(\sum_{k=1}^{l}(q-k)^2\right) = c\left(\sum_{k=1}^{l}q^2 - \sum_{k=1}^{l}2qkc + \sum_{k=1}^{l}k^2c^2\right). \quad (13)$$

Here

$$\sum_{k=1}^{l}q^2 = lq^2,\ \sum_{k=1}^{l}2qkc = 2qc\sum_{k=1}^{l}k = qcl^2,\ \sum_{k=1}^{l}k^2c^2 = c^2\sum_{k=1}^{l}k^2 = \frac{c^2l^3}{3}.$$

We substitute the value of $l$ in the formula and obtain

$$\sum_{k=1}^{l}q^2 = \frac{q^3}{c},\ \sum_{k=1}^{l}2qkc = \frac{q^3}{c},\ \sum_{k=1}^{l}k^2c^2 = \frac{q^3}{3c}.$$

From here

$$2c\sum_{k=1}^{l}\frac{(q-k)^2}{2} = \frac{q^3c}{3c} = \frac{q^3}{3}. \quad (14)$$

Let's write the second term from (12).

$$2c\sum_{k=1}^{l}s(q-kc) = 2cs\left(\sum_{k=1}^{l}q + \sum_{k=1}^{l}kc\right) = 2cs\left(ql - \frac{l^2c}{2}\right) = 2cs\left(\frac{q^2}{c} + \frac{q^2c}{2c^2}\right) = 2cs\left(\frac{q^2}{2c}\right) = sq^2. \quad (15)$$

The last term in (12) can be neglected because the order of the last diagonal block is small.

Substituting (14) and (15) in (13) we obtain

$$N_p \approx \frac{q^3}{3} + sq^2 = \frac{q^2}{3}(q+3s).$$

Theorem proved.

**Theorem 2.** Acceleration (speed up) of small-tile hybrid algorithm $LL^{\mathrm{T}}$ - decomposition of a sparse block-diagonal matrix with a border $A$, is

$$S_p \approx (p-1)\left(1 + \frac{3(p-1)s^2}{2q^2(q+3s)}\left(\tau_{opp} + \left(\frac{2p}{(p-1)} + \frac{4qc}{s^2} + \frac{4c}{(p-1)s}\right)\tau_{opg}\right)\right)^{-1},$$

where $c$ is the size of the tile

**Proof.** To prove the theorem, we will use the methodology for estimating the efficiency and acceleration of hybrid algorithms, given in [5].

We will find the amount of data that are transferred between processors during the execution of the algorithm. Because the ring communication topology is used and the processors exchange $s^2$ machine words, the total amount of data that the processors exchange is $\frac{(p-1)s^2}{2}$.

Before execution of a multi-collection operation in all CPU and GPU pairs except the last one must exchange data of volume $s^2$. A similar exchange also executed in the last pair of CPU and GPU.

Also in the process of factorization of diagonal blocks $D_{ii}$ all pairs of CPU and GPU except the last one exchange data of volume $2qc$. The last pair of CPU and GPU exchanges data of volume $2sc$.

The total amount of data exchanged by all CPU and GPU pairs is equal to $2qc(p-1)+ps^2+2sc$.

We will use the value of $N_p$ from Theorem 1 when calculating $T_1$ and $T_p$.

$$T_1 \approx \frac{(p-1)N_p}{n_o}t_g, \quad T_p \approx \frac{N_p}{n_o}t_g + \frac{(p-1)s^2}{2}t_{opp} + \left(2qc(p-1)+ps^2+2sc\right)t_{opg}.$$

Substituting all the values found in the formula for calculating the acceleration factor $S_p$, we obtain

$$S_p \approx \frac{(p-1)\frac{N_p}{n_o}t_g}{\frac{N_p}{n_o}t_g + \frac{(p-1)s^2}{2}t_{opp} + \left(2qc(p-1)+ps^2+2sc\right)t_{opg}}.$$

Dividing the numerator and denominator by $\frac{N_p}{n_o}t_g$ we obtain

$$S_p \approx \frac{(p-1)}{1+\frac{1}{N_p}\left(\frac{(p-1)s^2}{2}\tau_{opp} + \left(2qc(p-1)+ps^2+2sc\right)\tau_{opg}\right)}.$$

We take out for brackets $\frac{(p-1)s^2}{2}$

$$S_p \approx \frac{(p-1)}{1+\frac{3(p-1)s^2}{2q^2(q+3s)}\left(\tau_{opp} + \left(\frac{2p}{(p-1)} + \frac{4qc}{s^2} + \frac{4c}{(p-1)s}\right)\tau_{opg}\right)}.$$

Theorem proved.

According to the above algorithms, programs were developed and the following numerical experiments were performed.

## Numerical experiments

Computational experiments to solving the SNE were performed on the nodes of the supercomputer SKIT [6] with the following characteristics:

1. 2 CPUs Intel Xeon E5-2600 with a frequency of 2.6 GHz;
2. integrated with the general data storage of a cluster complex of 200 TB;
3. data network between Infiniband FDR 56 Gbps;
4. 128 GB of RAM;
5. 3 accelerators NVidia Tesla M2075.

With a given initial approximation $x^0$=0,5 in the region $D=\{-1000 \le x_i \le 1000\}$, $i=0,1,2,\dots,n-1$ the following system of nonlinear equations with the Jacobi block matrix was solved by Newton's method:

- the first block contains the following $m$ equations:

$$4x_1^2 - x_2^2 - x_{m+1}^2 + x_{m+2} - 3 = 0,$$

$$-2x_{j-1}x_j + 5x_j^2 - x_{j+1}^2 + x_{m+j-1} - x_{m+j}^2 + x_{m+j+1} - 3 = 0, \quad j = 2, \dots, m\text{-}1$$

$$-2x_{m-1}x_m + 5x_m^2 + x_{2m-1} - x_{2m}^2 - 3 = 0 .$$

- following $N$-2 ($K$=2,…, $N$-1) blocks each contain $m$ equations:

$$-2x_{Km-2m+j}x_{Km-m+j} + x_{Km-2m+2} + 5x_{Km-m+j}^2 - x_{Km-m+j+1}^2 - x_{Km+j}^2 + x_{Km+j+1} - 3 = 0 , \quad j = 1,$$

$$x_{Km-2m+j-1} - 2x_{Km-2m+j}x_{Km-m+j} + x_{Km-2m+j+1} -$$

$$-2x_{Km-2m+j}x_{Km-m+j} + 6x_{Km-m+j}^2 - x_{Km-m+j+1}^2 + x_{Km+j-1} - x_{Km+j}^2 + x_{Km+j+1} - 4 = 0 , \qquad j = 2, \ldots, m\text{-}1$$

$$x_{Km-m-1} - 2x_{Km-m}x_{Km} - 2x_{Km-1}x_{Km} + 6x_{Km}^2 + x_{Km+m-1} - x_{Km+m}^2 - 3 = 0$$

- last $m$ equations:

$$-2x_{(N-2)m+1}x_{(N-1)m+1} + x_{(N-2)m+2} + 5x_{(N-1)m+1}^2 - x_{(N-1)m+2}^2 - 3 = 0 ,$$

$$x_{(N-2)m+j-1} - 2x_{(N-2)m+j}x_{(N-1)m+j} + x_{(N-2)m+j+1} -$$

$$-2x_{(N-1)m+j}x_{(N-1)m+j} + 6x_{(N-1)m+j}^2 - x_{(N-1)m+j+1}^2 - 3 = 0 , j = 2, \ldots, m\text{-}1$$

$$x_{(N-1)m-1} - 2x_{(N-1)m}x_{Nm} - 2x_{Nm-1}x_{Nm} + 6x_{Nm}^2 - 3 = 0$$

where $m$ is the number of rows in the block, $N$ is the number of blocks. Then $N = [(n-1)/m+1]$ or $n$=$Nm$.

When software implementation of algorithms on computers of hybrid architecture it is advisable to use the functions of optimized software libraries. In particular, such libraries include Intel MKL [7], CUBLAS [8].

The following library's functions are used to implement the main computational stages of algorithms:

- dpotrf – finding LLT decomposition of a dense matrix. The function is performed on the CPU;
- cublasDsyrk – performs s-rank modification of a dense matrix. The function is performed on the GPU;
- cublasDgemm – finds the multiplication of two dense matrices. The function is performed on the GPU;
- cublasDtrsm – solving a triangular system with many right parts.. The function is performed on the GPU.

The algorithm uses functions from the MPI [9] and CUDA libraries [10] to perform communications:

- Mpi_reduce – global reduction function with saving the result in the specified processor.
- cudaMemcpyAsync – a function that performs asynchronous data copying between the CPU and the GPU. Running the function in multiple cudaStream streams reduces the execution time of the program because copying is performed against the background of calculations and does not cause a stop in the calculations.

Calculations on the GPU are also performed simultaneously in different cudaStream streams.

Here are some of the results.

Table 1 shows the time (sec.) Execution of one iteration of Newton's method using a parallel variant (8 CPU) of the fine-tile factorization algorithm of the block-diagonal matrix with a frame. Tile size is 128.

Table 1.

| Matrix order / number of threads | 15000 | 20000 | 25000 |
|---|---|---|---|
| 1 | 0.184481 | 0.2459 | 0.3074 |
| 2 | 0.16771 | 0.2236 | 0.2795 |
| 3 | 0.0819 | 0.1108 | 0.1336 |
| 4 | 0.0631 | 0.0842 | 0.0988 |
| 5 | 0.0519 | 0.0630 | 0.0772 |
| 6 | 0.0445 | 0.0539 | 0.0637 |

| | | | |
|---|---|---|---|
| 7 | 0.0393 | 0.0474 | 0.0540 |
| 8 | 0.0354 | 0.0427 | 0.0468 |

From the table, we can conclude that the increase in the order of the system which is solving and the transformation of the matrix for different numbers of processors contribute to a uniform load of processors by calculations. Another factor that contributes to such a reduction in calculation time is the adjustment of the tile size. With the right choice of tile size, the effect of caching calculations can be achieved, which can manifest itself as super-acceleration on certain parameter configurations.

In fig. 4. the dependence of the time of execution of calculations of the hybrid variant (8 CPU + 1 GPU) of the fine-tile algorithm on the number of flows and the size of the tile is shown. The order of the matrix is 10050.

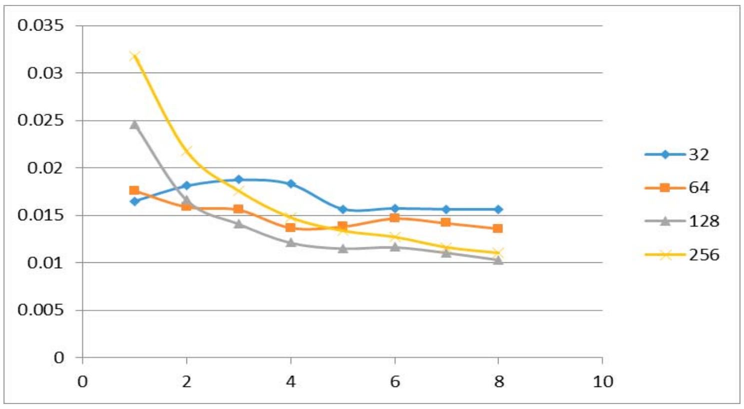

Fig. 4. GPU computation time, depending on tile size and number of threads.

## Conclusions

A new method for solving systems of large-orders nonlinear equations with Jacobi block matrices is considered. Its ideological basis is a combination of the classical algorithm of Newton's method with an efficient fine-tile algorithm for solving systems of linear equations with sparse matrices. The times of solving SNE of different orders on the nodes of the SKIT supercomputer are given. A significant reduction in the solution time of systems of nonlinear equations is obtained. This algorithm allows you to adjust the dimension of the block with which the calculations are performed at each step of the algorithm. This can have a computational caching effect when the blocks are completely stored in the GPU's fast memory. Also, this block structure allows you to work with inseparable data sets on the GPU, which reduces the number of index operations and checks, which are quite expensive on a graphics accelerator.

**References:**

1. Nesterenko, A.N. & Khimich, A.N. & Yakovlev, M.F. (2006) To the problem of solving of non-linear systems on multi-processor distributed memory computing system. *Gerald of computer and information technologies*. 10. pp. 54-56.
2. Dzhordzh A. Chyslennoe reshenye bol□shykh razrezhennykh system uravnenyy / A. Dzhordzh, Dzh. Lyu. – M.: Myr, 1984. – 334 s.
3. Pysanetsky S. Tekhnolohyya razrezhennykh matryts / Pysanetsky S. – M.: Myr, 1988. — 410 s.
4. Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra: A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures.Parallel Computing, Volume 35, Issue 1, P. 38-53, 2009, ISSN:0167-8191.
5. O.V. Popov. Doslidzhennya efektyvnosti paralel□nykh alhorytmiv dlya komp□yuteriv hibrydnoyi arkhitektury. Materialy Mizhnarodnoyi naukovoyi shkoly-seminaru «Pytannya optymizatsiyi obchyslen□ (POO XLII)», (Zakarpat·s□ka obl., Mukachivs□kyy r.-n, smt. Chynadiyevo, 21–25 veresnya 2015), Kyyiv: 2015, S. 16.
6. Rezhym dostupu: http://icybcluster.org.ua
7. Intel® Math Kernel Library (Intel® MKL) – Rezhym dostupu: https://software.intel.com/en-us/intel-mkl
8. CUDA CUBLAS_Library – Santa Clara: Nvidia, 2010. – 254 c.
9. W. Gropp Using MPI-2: Advanced Features of the Message-Passing Interface / W. Gropp, E. Lusk, and R. Thakur. – Cambridge: MIT Press, 1999. – 382 p.
10. Boreskov A.V. Osnovy raboty s tekhnolohyey CUDA / Boreskov A.V., Kharlamov A.A. – M.: DMK Press, 2010. – 232 s.

***About the authors*:**

*A.N. Khimich,*
Head's assistant,
number of scientific publications in Ukrainian publications - 110,
https://orcid.org/0000-0001-9284-139X.

*V. A. Sydoruk*,
Senior Research Assistant,
number of scientific publications in Ukrainian publications - 30,
http://orcid.org/0000-0003-0210-6020.

*A.N. Nesterenko*,
Research Scientist,
number of scientific publications in Ukrainian publications - 32,
https://orcid.org/0000-0001-6174-1812.

***Place of work of the authors*:**

V.M. Glushkov Institute of Cybernetics of NAS of Ukraine,
40, Acad. Glushkov Av., 03187, Kyiv
tel. (066)-367-85-40
e-mail: alla.nest1958@gmail.com