Research of parallel algorithms for solving three-diagonal systems of linear algebraic equations on a graphical computing device using various types of memory

X S Pogorelskih[1], **L V Loganova**[1]

[1]Samara National Research University, Moskovskoe Shosse, 34A, Samara, Russia, 443086

e-mail: sekih@yandex.ru, lloganova@yandex.ru

Abstract. In this paper, we research and compare different implementations of cyclic reduction and sweep algorithms on a graphics processing unit (GPU) using different types of device memory. As a result of the work, it was found that the algorithm of the run should be used to solve the set of the tridiagonal linear system. However, the best results are shown by a parallel version of the cyclic reduction algorithm with partial use of shared memory, when solving a single linear system on the GPU.

CEUR-WS.org/Vol-2416/paper30.pdf

1. Introduction

The applicability of systems of linear equations of tridiagonal form is very extensive[1]. Tridiagonal matrices have an extremely important role in difference methods of solving problems of mathematical physics [2]. In addition, many linear algebra problems, such as solving equations and finding eigenvalues, are solved through transformations of matrices of general form to tridiagonal ones [3]. These matrices also play an important role in the theory of orthogonal polynomials [4].

There are many algorithms and their parallel versions for solving tridiagonal systems of linear equations. The need for parallelization naturally occurs when the size or number of systems to be solved is very large. Most methods are based on simple arithmetic operations performed many times. Therefore, the implementation of parallel versions of algorithms on the GPU is a reasonable solution [5].

GPU-clusters consume less power than the CPU [6]. In addition, the development of graphics cards allows for complex calculations on personal computers. Therefore, the ability to parallelize algorithms on the GPU is interesting for researchers.

Operations in memory take a significant time of calculations on the GPU [7]. Therefore, the optimal organization of work with memory allows to achieve a significant increase in acceleration.

Global, shared, and constant GPU memory are commonly used for calculations [8]. Constant memory is small, immutable, so not suitable for large systems [9]. Therefore, in this paper we study the global and shared memory of the GPU device.

Shared memory is much faster than global memory, but its size is very limited. These features of the types of memory must be taken into account when implementing algorithms on the GPU.

2. Problem Statement and Methodology

This article investigates parallel algorithms for solving systems of linear equations of tridiagonal form.

The traditional matrix algorithm and the cyclic reduction method are usually used to solve systems of linear algebraic equations of tridiagonal form [10].

The advantage of the cyclic reduction method is a high degree of parallelism [10]. However, it also requires a large amount of computation. When implemented on a central processing unit using MPI technology, the method requires a lot of forwarding. This significantly slows down the program [11]. However, in CUDA technology shipments are not required. Therefore, the method attracts the attention of researchers to implement on the graphics card. In addition, it allows parallelization for a single system [12].

The traditional matrix algorithm Parallels for one system worse than the parallel cyclic reduction algorithm. However, there are a large number of tasks that require the solution of a set of systems. The traditional matrix algorithm is suitable for such problems. It gives a gain in acceleration due to low computational complexity.

In this paper, we investigate which algorithm for solving tridiagonal-type systems spends less time on computation. To solve this problem, we compare the he implementation of the traditional matrix algorithm, and the serial and parallel versions of the cyclic reduction algorithm using different combinations of global and shared GPU memory. Parallelization of methods is carried out using the CUDA model. Experimental research is conducted on the supercomputer "Sergey Korolev".

3. Experimental research

This work began with the implementation and research of the traditional matrix algorithm. This method is one of the most famous and popular for solving tridiagonal systems of linear arithmetic equations.

The first version of the algorithm was implemented on the CPU for further comparison in acceleration. Then an implementation of a parallel traditional matrix algorithm on the GPU using only global memory was created. The next task was to implement this method using shared memory. Since the memory size is small, the necessary factors were loaded in portions into the shared memory immediately before use. Then, after all the necessary operations, the coefficients were replaced by the next portion. To store data in shared memory, arrays of auxiliary coefficients α and β and the sought vector x of size $2 \cdot blocksize$ and arrays of initial coefficients a, b, c and k of size $blocksize$. All calculations were performed in shared memory.

The experiments were carried out for a set of systems with matrices of size $N = 4095$. This size was chosen to compare the results with the experimental versions of the cyclic reduction algorithm implemented for the size of matrices $N = 2^q - 1$, where $q \in \mathbb{N}$.

The number of equation systems varied from 5000 to 20000. If the number of systems is less than 5000, working with memory on the GPU takes much longer than the calculations themselves.

The obtained accelerations of parallel versions of algorithms relative to the serial version on the CPU are given in table 1.

Table 1. The dependence of the acceleration of parallel versions on the serial version of the traditional matrix algorithm in solving a set of systems of linear algebraic equations, depending on the number of systems with matrices of size $N = 4095$.

Number of systems	GPU with shared memory	GPU
5000	**2.19032**	1.680011
10000	**1.705628**	1.164633
15000	**1.903806**	1.274506
20000	**1.885846**	1.027733

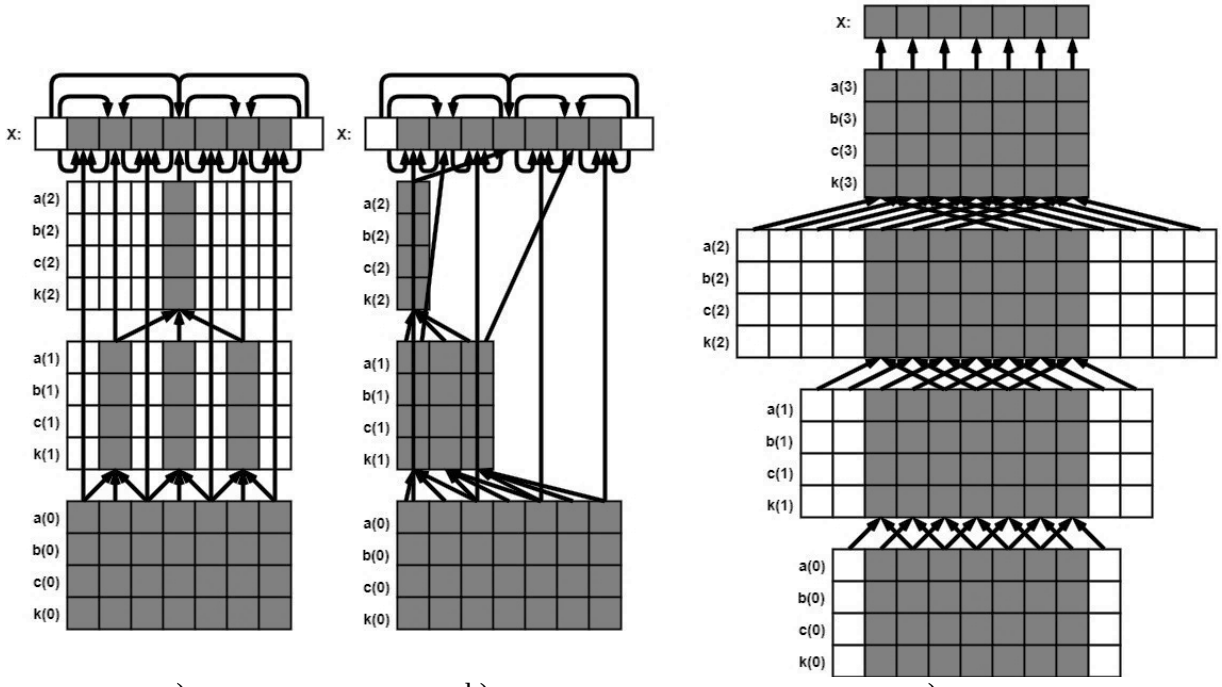
Average acceleration version on the GPU using only global memory is equal to 1.26. The average acceleration using shared memory is 1.86. Thus, the use of shared memory gives an increase in acceleration by 47%.

This result shows that the time it takes to copy data from global memory to shared memory and vice versa is justified by the high speed of calculations performed in shared memory.

The next step is the implementation of the sequential cyclic reduction algorithm (SERICR). This algorithm makes it impossible to carry out parallel calculations for one system, but is parallelized for a set of systems.

R. Hockney [13] considered in detail the cyclic reduction. It notes that this algorithm is suitable for any number of equations. However, for simplicity, the author gives the number of equations $N = 2^q - 1$, где q – where q is an integer. For convenience, he introduced the designation $N' = 2^q = N + 1$.

We present a routing diagram of this algorithm for $N' = 8$ in figure 1 (a). Cells that contain significant information are gray.



a) b) c)

Figure 1. Algorithm routing schemes for N'=8: sequential cyclic reduction SERICR (a), sequential cyclic reduction SERICR with modified indexation (b), parallel cyclic reduction PARACR (c).

In this algorithm, the coefficients of all levels of reduction are stored in memory. This requires a lot of memory. In addition, many cells do not store useful information. Therefore, the indexation of the algorithm was changed to minimize memory usage. The resulting routing scheme is shown in figure (b).

We present an algorithm with modified indexing. At the first stage new coefficients for left and right parts for reduction levels $l = \overline{1, q-1}$ with step $h = 1$ from $i = 0$ to $i = N^{(l)}$ are considered. Reduction coefficients of level $l = 0$ are assumed to be $a_i^{(0)} = a_i$, $b_i^{(0)} = b_i$, $c_i^{(0)} = c_i$ и $k_i^{(0)} = k_i$. First, we find the auxiliary coefficients:

$$\alpha_i = -\frac{a_{2i+1}^{(l-1)}}{b_{2i}^{(l-1)}};$$

$$\gamma_i = -\frac{c_{2i+1}^{(l-1)}}{b_{2i+2}^{(l-1)}};$$

Then, based on the obtained α_i and γ_i and the values of the coefficients of the left and right parts of the previous reduction level, we obtain:

$$a_i^{(l)} = \alpha_i a_{i-2^{l-1}}^{(l-1)};$$

$$c_i^{(l)} = \gamma_i c_{i+2^{l-1}}^{(l-1)};$$
$$b_i^{(l)} = b_i^{(l-1)} + \alpha_i c_{i-2^{l-1}}^{(l-1)} + \gamma_i a_{i+2^{l-1}}^{(l-1)};$$
$$k_i^{(l)} = k_i^{(l-1)} + \alpha_i k_{i-2^{l-1}}^{(l-1)} + \gamma_i k_{i+2^{l-1}}^{(l-1)}.$$

At the second stage, based on the coefficients of the left and right parts for the reduction levels $l = \overline{0, q-1}$ we find the vector of solutions x. It is assumed that $x_0 = x_{N'} = 0$. For $l = \overline{q, 1}$, for i, varying in increments $h_2 = 2^{(l-1)}$ from $i = 2^{(l-1)}$ to $i = N' - 2^{(l-1)}$, we consider:

$$x_i = \frac{k_i^{(l-1)} - a_i^{(l-1)} x_{i-2^{(l-1)}} - c_i^{(l-1)} x_{i+2^{(l-1)}}}{b_i^{(l-1)}}.$$

However, the resulting memory savings are not sufficient to use this algorithm on big data. The solution in this situation is to store only two levels of reduction and recalculate the coefficients when they are needed.

The algorithm was implemented on the central and graphics processor devices. When implementing the algorithm on the CPU, the coefficients were not recalculated, since the CPU has enough memory to store the coefficients of all levels of reduction. When implemented on the GPU recalculation was necessary. On the graphics card, the algorithm was implemented using only global memory and shared memory.

The experiments were carried out for a set of systems with matrices of size $N = 4095$. The number of systems varied from 5000 to 25000. The obtained accelerations are shown in table 2.

Table 2. The dependence of the acceleration of the sequential cyclic reduction algorithm on the GPU relative to the implementation on the CPU.

Number of systems	SERICR on a shared memory GPU	SERICR on the GPU
5000	0.316570	0.193076
10000	0.259339	0.191475
15000	0.258175	0.186748
20000	0.264068	0.190208
25000	0.253464	0.185693

GPU implementations have shown a slowdown. This is due to the need to carry out multiple recalculations of the coefficients.

The final stage of the work is the implementation of the parallel cyclic reduction algorithm. This algorithm is proposed in the work of R. Hockney [13]. Its advantage is the possibility of parallelization for one system of equations. In addition, the following reduction levels are replaced by the previous ones. This gives a noticeable gain in memory. The routing scheme of the algorithm is shown in figure 1 (c).

The algorithm was implemented on the GPU using only global memory and with full and partial loading of data into shared memory.

The experiments were carried out for a system of equations with matrices of size $N = 4095$. The number of systems varied from 5000 to 25000. The obtained accelerations are presented in table 3.

The table shows that the implementation with partial loading into shared memory shows the best results. It is 2% faster than a non-shared memory implementation and 41% faster than a fully loaded shared memory implementation. The results can be explained. Some coefficients are used only once in calculations. The time of their copying to shared memory is practically not compensated by the gain in the time of calculations. Therefore, it makes sense to load only data that is used multiple times into shared memory. In this case, the data loaded into shared memory is used twice, so the use of shared memory does not give a significant benefit.

Table 3. The dependence of the acceleration of parallel reduction algorithm implementations on the GPU relative to the sequential cyclic reduction algorithm on the CPU.

Number of systems	PARACR on the GPU with partial use of shared memory	PARACR on GPU with shared memory	PARACR on the GPU
5000	**2.006864**	1.198271	1.970711
10000	**2.021180**	1.206249	1.972713
15000	**2.008165**	1.201463	1.973943
20000	**2.007109**	1.202857	1.962254
25000	**2.017338**	1.203156	1.987762

The operating time of the traditional matrix algorithm and the cyclic reduction algorithm for solving a set of systems with matrices of size N=4095 is presented in table 4.

Table 4. The running time of the implementation of the algorithm on a GPU with a full load in the shared memory and the algorithm of parallel cyclic reduction on the GPU with a partial load in the shared memory on a set of systems with size matrices $N = 4095$.

Number of systems	Traditional matrix algorithm, c	PARACR, c
5000	**0.09**	0.59
10000	**0.19**	1.19
15000	**0.28**	1.78
20000	**0.38**	2.38
25000	**0.47**	2.96

The timeline of the algorithms when solving one system of equations is presented in Figure 2.

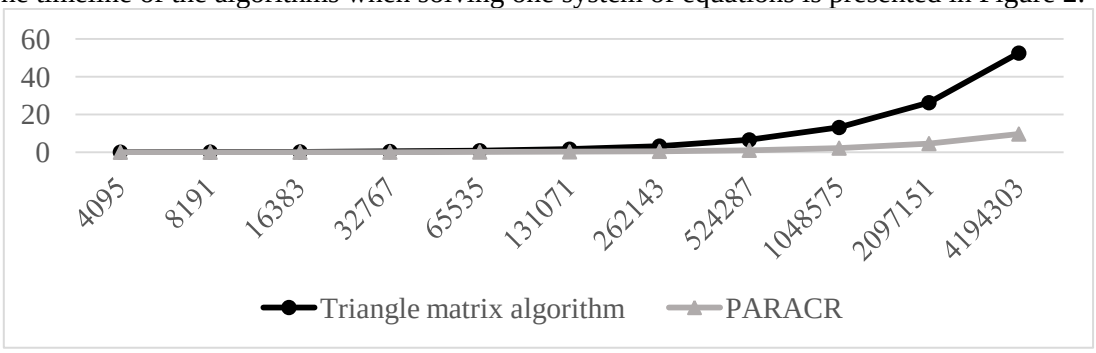


Figure 2. Schedule of implementations of traditional matrix algorithm on GPUs with full load into shared memory and parallel cyclic reduction algorithm on partial load GPUs in shared memory when solving one system (the vertical axis shows the time in seconds, the horizontal size shows the matrix size).

Thus, when solving a set of systems of linear algebraic equations, the best result is shown by the traditional matrix algorithm on a GPU with full load in the shared memory. Its running time is on average 6 times shorter than the running time of the cyclic reduction algorithm. However, when solving one system with a large matrix, it is more expedient to use a parallel cyclic reduction algorithm with partial loading into shared memory. Its implementation requires 7 times less time for calculations than the implementation of the traditional matrix algorithm.

4. Conclusion

As a result of this work, it was obtained that the use of shared memory for the traditional matrix algorithm gives an increase in acceleration by 47%. This is because data loaded into shared memory is reused.

For the cyclic reduction algorithm, it is advisable to load into the shared memory only those coefficients that are used twice in the calculations. At the same time, the acceleration is increased by 2%.

When solving a set of systems of linear algebraic equations, the best implementation of the traditional matrix algorithm is 6 times faster than the best implementation of the cyclic reduction algorithm. However, when solving a single system with a large matrix, the result is the opposite: the cyclic reduction algorithm is 7 times faster.

5. References

[1] Golovashkin D L 2002 Application of the method of counter runs for the synthesis of a parallel algorithm for solving grid equations of tridiagonal type *Computer Optics* **24** 33-39
[2] Yablokova L V, Golovashkin D L 2018 Block algorithm for the joint difference solution of the d'Alembert and maxwell's equations *CEUR Workshop Proceedings* **2212** 56-62
[3] Yablokova L V, Golovashkin D L 2018 Block algorithms of a simultaneous difference solution of D'alembert's and Maxwell's equations *Computer Optics* **42(2)** 320-327 DOI: 10.18287/ 2412-6179-2018-42-2-320-327
[4] Ilyin V P, Kuznetsov Yu I 1985 *Three-diagonal matrices and their applications* (M.: Science) p 208
[5] Yablokova L V, Golovashkin D L 2017 Application of the pyramid method in difference solution d'Alembert equations on graphic processor with the use of Matlab *CEUR Workshop Proceedings* **1902** 68-70
[6] Golovashkin D L, Loganova L V 2012 The solution to the difference equations finite difference schemes with cyclic boundary conditions on a two-dimensional grid areas using multiple graphics processing devices *Computer Optics* **36(4)** 534-540
[7] Boreskov A V 2011 *CUDA technology in the examples: an introduction to programming GPUs* (M.: DMK Press) p 232
[8] Kazennov A M 2010 Basic concepts of CUDA technology *Computer Research and Modeling* **2(3)** 295-308
[9] Boreskov A V, Kharlamov A A 2016 *Basics of CUDA technology* (Moscow: DMK Press) p 230
[10] Shinkaruk D N, Shpolyansky Yu A and Kosyakov M S 2012 Analysis of the effectiveness of the use of CUDA technology for solving systems of linear equations with three-diagonal matrices in options pricing problems *Universities. Instrument making* **10** 6
[11] Malyavko A A 2015 *Parallel programming based on OpenMP, MPI, CUDA technologies: studies. manual* (Novosibirsk: Publishing House of NSTU) p 116
[12] Yarmushkin S V, Golovashkin D L 2004 Research of parallel algorithms for solving three-diagonal systems of linear algebraic equations *Bulletin of the Samara State Technical University. Physics and Mathematics* **26** 5
[13] Hockney R, Jesshope K 1986 *Parallel computers. Architecture, programming and algorithms* (M.: Radio and communication) p 392