Fast Fourier Transform in Planar3D model using an explicit numerical integration scheme

Nikita Mushchak1, Egor Starobnkii1, Sergei Hlopin1 and Egor Shel2,1

1 Peter the Great St. Petersburg Polytechnic University, Russia

2 LLC Gazpromneft Science and Technology Center, Russia  
Mushchak\_nd@spbpu.ru

**Abstract.** In this paper authors propose methods to speed-up calculations of a fracture propagation model using Fast Fourier Transform (FFT). We consider Planar3D model with an explicit numerical integration scheme. Current research decided to implement radix-2 Cooley-Tukey FFT algorithm in C++ code using STL containers, which provided fast calculations and give advances to work with memory and cache. We compare speed of FFT computation with other libraries (FFTW3, GSL, Eigen3). Analysis of results have been showed us comparable time of calculations. We consider a method for accelerating the calculations of a Planar3D in the framework of matrix-vector multiplication and processing of input data using a low-pass filter. In the model under consideration, the product of multiplication of the influence matrix and the distribution matrix of the opening is the matrix of the pressure distribution in the fracture is obtained. This procedure engages in substantial work from time to time throughout the program. In this paper, we implement a modified method for calculating the matrix-vector multiplication product using FFT, which allows us to speed up the calculations. A technique is used to process input data by the example of averaging lithology layers. In the presence of thin layers with high contrasts of mechanical properties, you can apply a low-pass filter, which is implemented on the basis of the library implemented by the authors. Such processed layers make it possible to obtain an increase in the computational speed when modeling the evolution of a planar hydraulic fracture model.

**Keywords:** Fast Fourier Transform, Planar3D model, low-pass filter, explicit numerical integration scheme.

1. Introduction

The first acceleration method of discrete Fourier transform (DFT) were proposed by Gauss in the 19th century [1]. FFT is widely used in engineering, sound recording and sound transmission, signal processing. One of the simplest FFT implementations was proposed by Cooley-Tukey [2]. This method requires the length of the input array equal to degree 2, represents the divide and conquer algorithm. It is also worth noting the Prime-Factor algorithm [3], which converts one-dimensional DFT to two-dimensional DFT. Bruun's FFT algorithm [4] uses the recursive polynomial-factorization approach. The Rader's FFT algorithm [5] calculates the discrete Fourier transform (DFT) for the lengths of arrays expressed by primes by repeatedly expressing the DFT as a cyclic convolution.

For frequency-domain signals, FFT can also be applied in real time [6, 7], when the signal is subject to a delay that exceeds the time required to calculate the FFT of such a signal. The asymptotic complexity of FFT is , where is the number of elements during signal sampling. FFT is used for low-pass filtering [8], high-pass filtering [9] and band-pass filtering [10].

Fast Fourier Transform allows to speed up some calculations that are bottle neck in the program code. Within the framework of this study, the implementation of FFT [11] in the form of one-header library, available in the open Git repository, is proposed. The library allows to speed up the calculation of the Planar3D model with an explicit integration method. The authors propose the matrix-vector product acceleration using FFT and technic of processing of lithology layers via low-pass filter based on current FFT realization.

There are several libraries that have FFT implementations. Nowadays acceleration of computation speed is achieved through the use of specific architectures [12, 13] and the use of parallel computing [14, 15]. In this work, the library is implemented in C++ and involves the use and compilation with gcc, g++, mvc, clang, clang++. Implementations of FFT in C ++ also have FFTW3 [16], GSL [17] and Eigen3 [18]. Libraries have proven themselves for their use in scientific research. FFTW3 uses several variants of the Cooley-Tukey FFT algorithm [2], as well as the Bluestein’s FFT algorithm [19]. The GSL library uses radix-2 and mixed-radix algorithms. Eigen takes advantage of processor architecture features.

1. Fast Fourier Transform

Discrete Fourier Transform (DFT) can be applied to sequential of complex numbers , where is number of sequence elements can be written as follows:

Note that to calculate the Fourier image for , computational operations are required, since for each the sum of terms is calculated. In Fast Fourier transform, the result is calculated in operations. There are various FFT algorithms, one of the simplest to implement is the radix-2 Cooley-Tukey algorithm [2]. This algorithm recursively divides the sequence into two equal parts and applies the discrete Fourier transform for them:

which is the DFT for even indices and for odd indices .

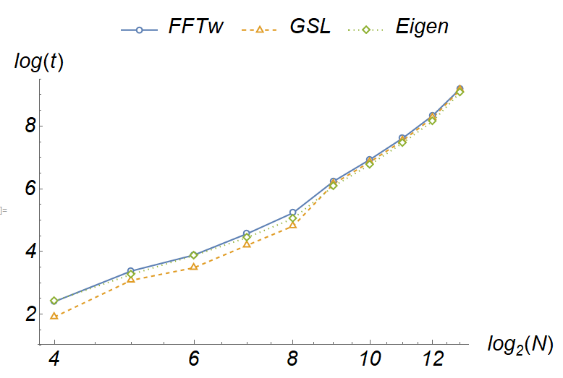
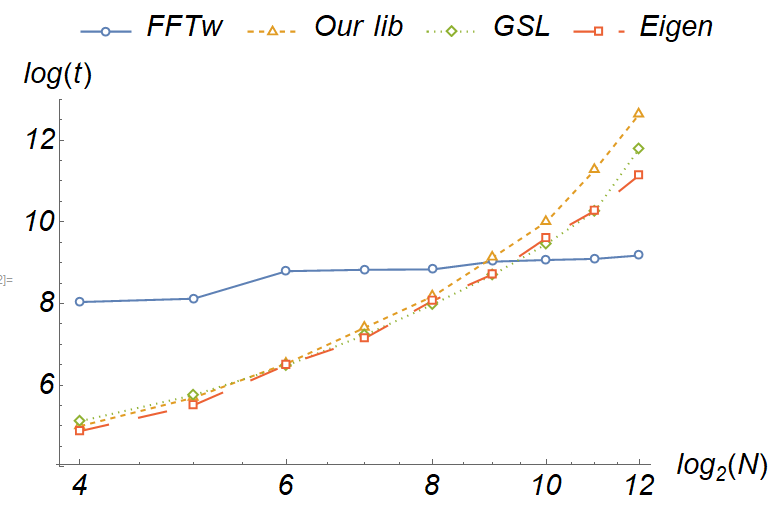
Inverse Fourier transform consists of the following steps: complex conjugation of the input vector, direct Fourier transform, repeated complex conjugation and normalization to satisfy:

()

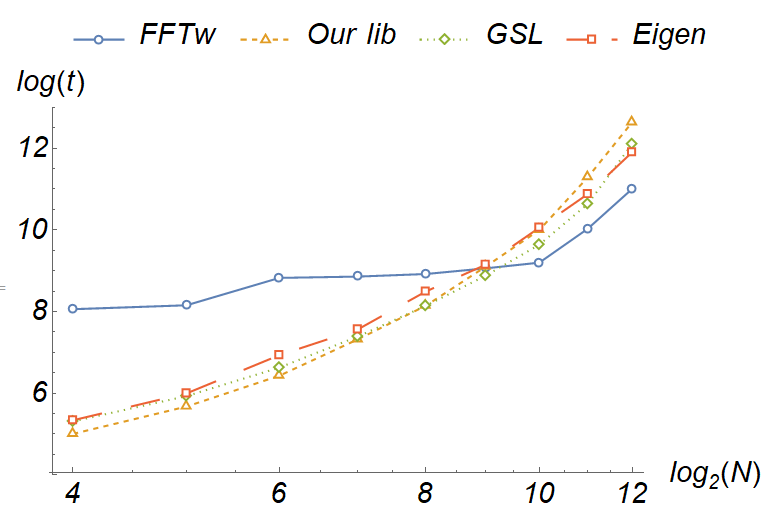
When implementing FFT in library we take into account new programming methods. The library uses standard STL containers, which are a convenient and efficient tool for programming in C++. The main advantages of containers are the automatic allocation and clearing of memory and monitoring of cache. It is most convenient to work with the STL container std::vector(), the main feature of which are the quick operations of adding an element to the end and accessing its element [20]. To store complex numbers, a vector of vectors is used, with each element of the vector containing a vector of length 2 to store the real and imaginary parts of the element. The input argument to the FFT function is a complex vector, the values ​​of which during the calculation are replaced by the values ​​of its Fourier image.

When testing the library, the calculation speed was compared with libraries of other developers: FFTW3 [16], GSL [17] and Eigen3 [18]. The same randomly generated vector was sent to input to all tested functions. The double data type was used to achieve acceptable accuracy for scientific calculations, the durations obtained during the experiments were obtained using the standard method std::chrono(). In this case, the Fourier transform is calculated 1 million times, after which the average time of calculation is found (fig.1A).

Libraries of other developers use different types of data, so conversion is required. The conversion was implemented as a sequential recording from the generated vector into the data type that each tested library works with, this operation was also performed 1 million times (fig.1B). Additionally, calculation times with data conversion and subsequent FFT calculations of the tested libraries were checked (fig.1C).



1. (B)

****

(C)

**Fig. 1.** Times of calculation FFT using different libraries (1 million times averaging) (A). Time of conversion to libraries data (1 million times averaging) (B). Time of conversion and FFT (1 million times averaging) (C).

Our library has a comparable calculation time with other libraries. It is possible to accelerate the algorithm using the implementation of radix-4 or more complicated computational schemes.

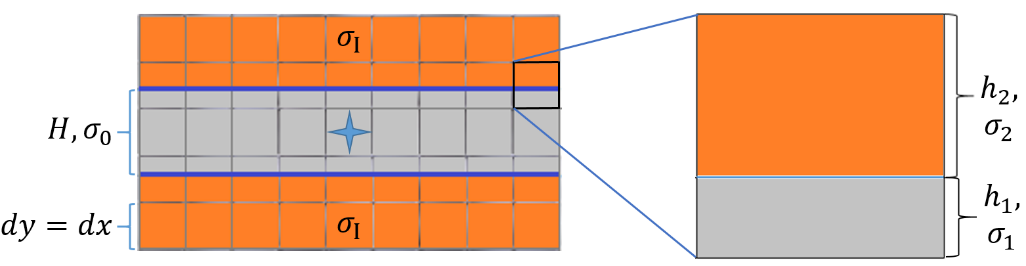
* 1. Layers averaging using FFT

The Planar3D [21, 22, 23, 24, 25] is a model of a plane fracture propagating in a multilayer medium. The assumption is used that the rock formation consists of homogeneous isotropic horizontal layers. The fracture propagates in a plane perpendicular to the minimum compressive stresses.

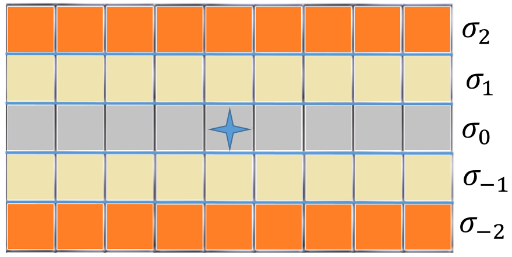
Analysis of geological data means obtaining the necessary information about the mechanical properties of the layers. One of the Planar3D [23] feature is the insensitivity to layers with a thickness of less than one grid element and, if the boundaries of the layers do not coincide with the boundaries of the cells. In this case, it is necessary to interpolate the mechanical properties of the current layer on the computational grid. As an illustration, we consider a three-layer medium (, ) fig. 4A. The stresses are found as the weighted arithmetic mean (fig.4B):

()

Where is the current number of interpolated layer and is the current number of initial layer. After applying interpolation, we obtain a five-layer medium (fig. 4 C) with stresses .



1. (B)



(C)

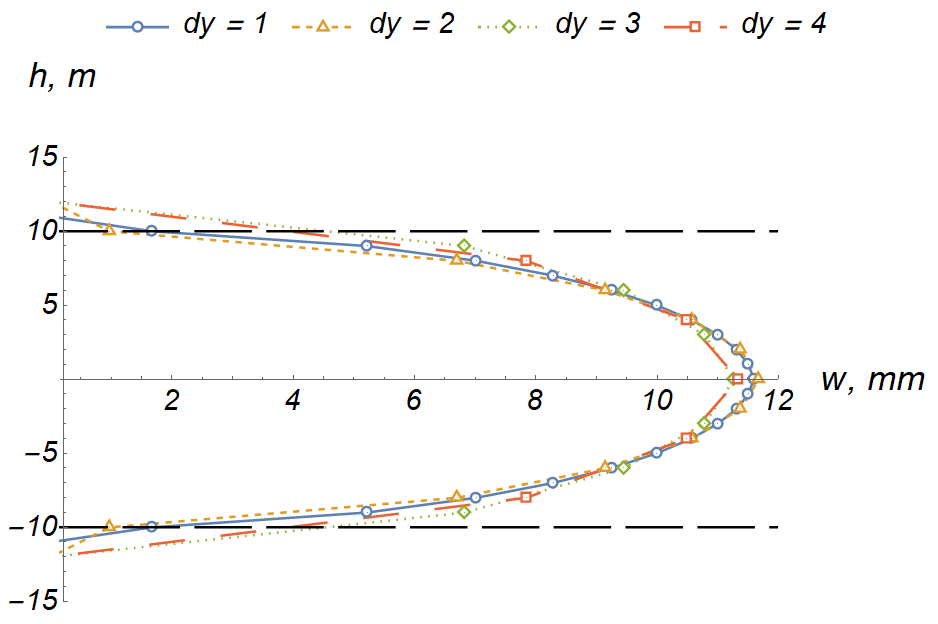
**Fig. 4.** Three-layered media (A), stress contrasts in the cell of the computational grid (B), interpolation of stress on the computational grid (C). Star it’s source of injection.

Let us consider a three-layered symmetric medium with a stress contrasts , a central layer thickness , a Newtonian fluid with a dynamic viscosity coefficient , injected 3 min with a pumping rate , flat Young modulus . The dimensions of the fracture was calculated depending on the computational cell size see table 1.

**Table 1. Time of calculations and fracture geometry by different mesh size**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mesh size, m | Length, m | Heigth, m | Opening, mm | Time of calc., s |
| 1 | 88.2303 | 22 | 11.59 | 217 |
| 2 | 91.1267 | 22.6651 | 11.6755 | 5 |
| 3 | 81.567 | 23 | 11.2662 | 1 |
| 4 | 83.5904 | 23 | 11.3372 | <1 |

### When we increase the mesh step, the height and opening of the fracture in the source of injection do not change significantly (<4%) and difference in length less than 10%, which is acceptable for engineering calculations. When we increase the mesh step, the speed of the calculation significantly accelerates. The transverse profiles of the openings are constructed depending on the various fig. 5.

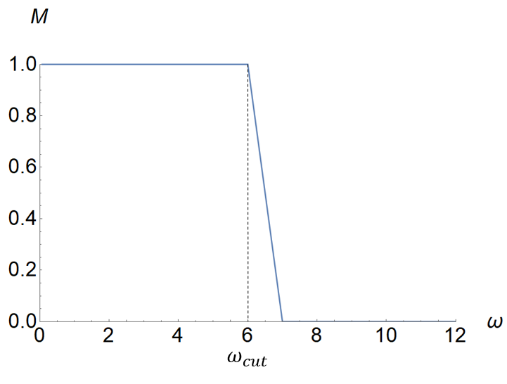
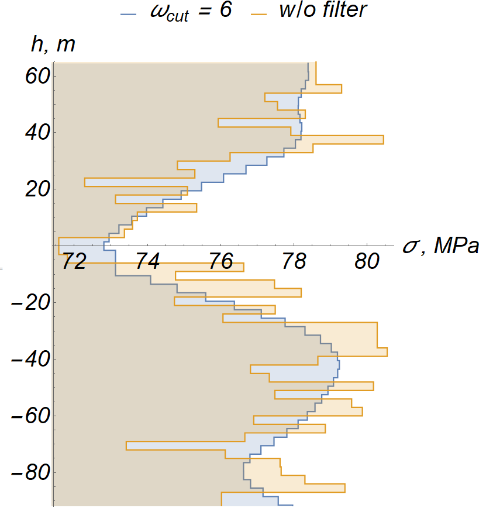


**Fig. 5.** Transverse profile of crack opening in a three-layer medium at different mesh steps , black dashed horizontal lines is layer boundaries.

For thin layers with stress contrasts greater than , a low-pass filter is used. To obtain a solution, the layer characteristic is interpolated onto a mesh with the number of elements integer, the Fourier image of the characteristic is taken, the transfer function is multiplied by vector of characteristics (fig. 6 A) (5) and IFFT is applied, followed by interpolation onto a large computational grid (fig.6 B). In this study, the length of the vector is elements.

()

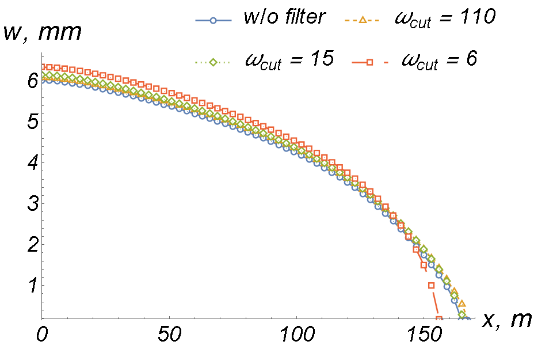
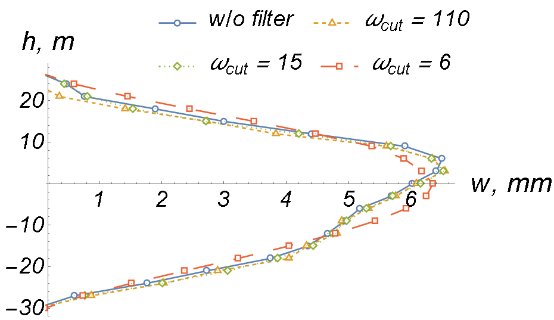
Let us consider a multilayer medium with stress contrasts (see on fig. 6), a non-Newtonian fluid with a dynamic viscosity coefficient , injected for with a pumping rate of , flat Young modulus , Carter's coefficient of leak-off , toughness ratio .

1. (B)

**Fig. 6.** Transfer function with cutoff frequency (A), layers interpolation without filter (yellow) and with low-pass filter .

The algorithm is simple and straight-forward. On (Fig. 7) the transverse and longitudinal profiles of fracture opening are shown at different cutoff frequencies . We consider the transverse profile of the opening along the vertical line passing through the injection source (Fig.7A) and the longitudinal along the horizontal line (Fig.7B).



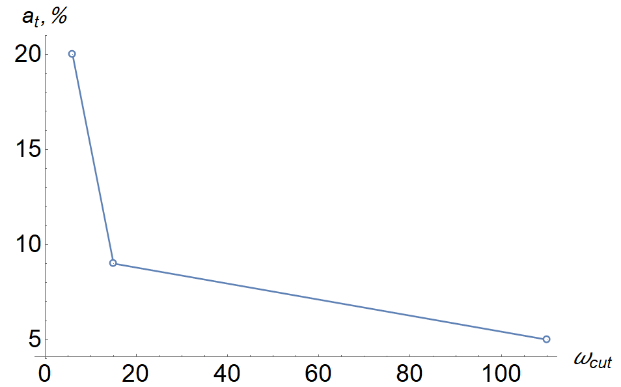
1. (B)

**Fig. 7.** Comparison of profiles (A-transverse, B-longitudinal) openings without a filter and with different .

As decreases, the medium erodes more strongly and, when = 0 the medium is homogeneous. When we take = 6, the dimensions of fracture (length, height, opening) differs by less than 10%, unlike the case without the use of a filter, which is a permissible error in engineering calculations. Also, when the cutoff frequency is reduced, the calculations of the Planar3D model accelerates (Fig. 8). Acceleration of calculations:

, ()

where is calculation time at , is the time of calculations without using a filter, which is taken as basis.



**Fig. 8.** Acceleration of calculations of a Planar3D model for various .

When the acceleration of the calculations is 20% with a difference in the fracture size of 10%. If you use interpolation on a coarser computational grid, you can get an acceleration of calculations by an order.

* 1. Matrix-vector product computation using FFT

Let us consider the technique of accelerating matrix-vector multiplication based on FFT. We assume that all calculations are carried out on a uniform mesh consisting only of square cells of the same size. In the Planar3D model, the connection between the distributions of opening and fluid pressures is carried out through a matrix depending on the computational domain. For more details, see [23]. The matrix contains the influence coefficients, which are calculated on the basis of the Green's function [26] and will be referred to below as the influence matrix .It’s depends on distance from source to point. Each element of such a matrix is a submatrix showing the influence of the concentrated force in a given cell on each other cell of the computational grid. We using a square uniform grid of size , the influence matrix of the fourth rank is , where the number of mesh elements in one direction. The pressure distribution and the openings in the fracture are matrices. It is convenient to rearrange each submatrix of the influence matrix into a vector and rebuild the resulting row matrix into a column. Similarly, the pressure matrix in a fracture is rearranged into a column vector, after which the fracture opening can be found as the product of a new influence matrix (size ) and pressure vector (size ). The calculation of pressures based on the direct multiplication of the matrix by the vector requires significant time expenditures - the asymptotic complexity of the О() algorithm, the value .

The application of the Fast Fourier Transform will reduce the asymptotic complexity of the algorithm from for the direct method of multiplication to , while the boundary conditions (BC) in the problem should be replaced by periodic ones. In hence of that, influence matrix has symmetry relative to the point source and can be rewrite as one-dimensional vector of influence coefficients. Symmetry of this vector:

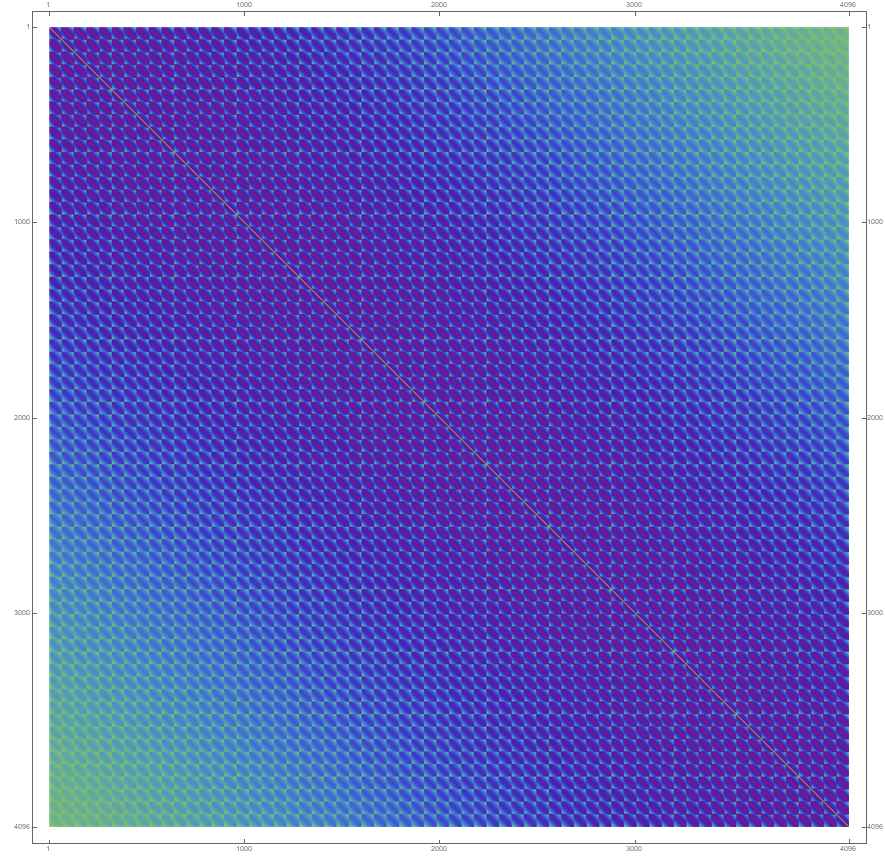
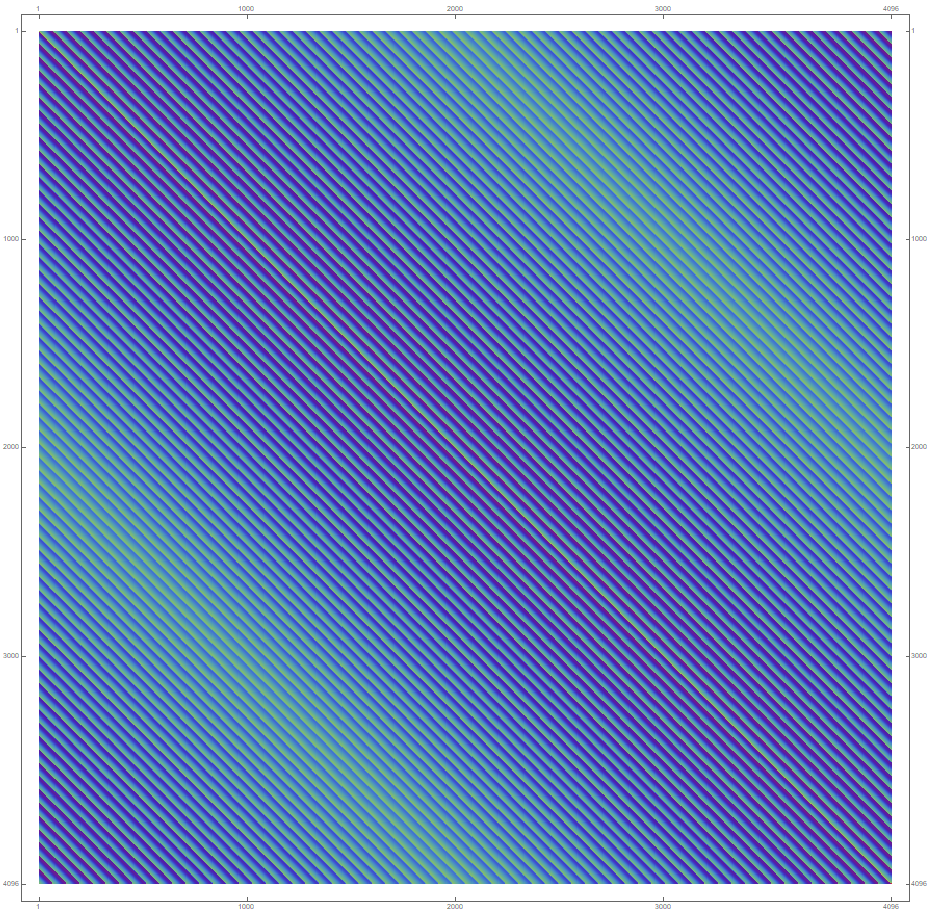
()

where is influence vector, is number of elements

It can be shown that when the size of the computational domain is of the order of two maximum dimensions of the propagating fracture introduced by modified BC, the error will be insignificant.

The matrix of influence coefficients has the form of a symmetric matrix:

(8)

**Fig.9.** Influence matrix (left) and circulant form (right).

Due to the imposition of boundary periodic conditions on the matrix , it takes the form:

(9)

Computation of matrix-vector product consist to following steps [27]:

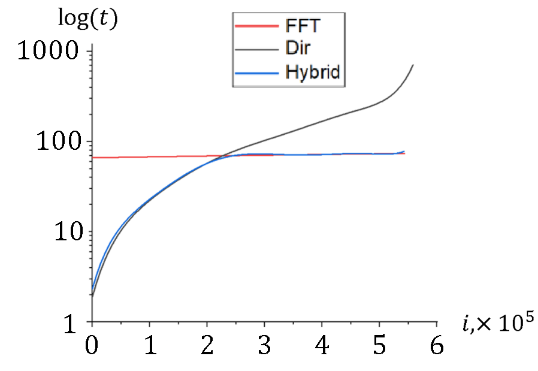
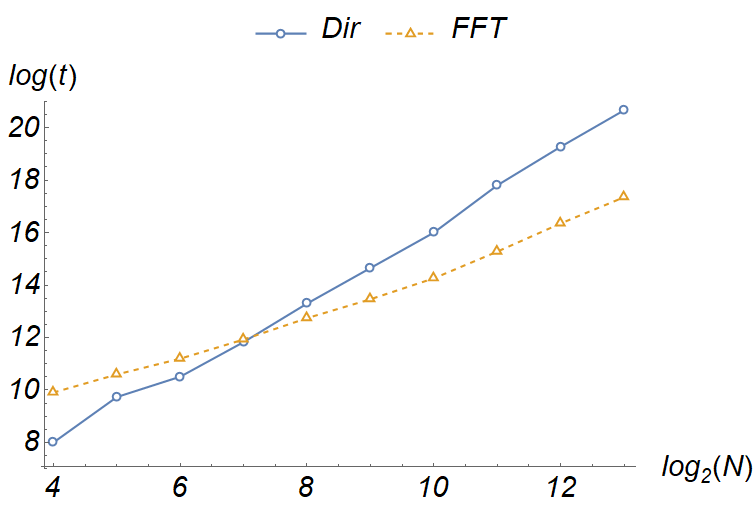
1. , where is first row of ,
2. , where is vector of opening,
3. , wise-element multiplication,
4. , obtain pressure vector using IFFT.

The advantage of this method is that the only one column vector is stored in memory, while the direct method of matrix multiplication requires storing a matrix of size .

Computer experiments were conducted to compare the calculation speed with different algorithms (using the direct method of multiplying the matrix by a vector and using the algorithm described above (see Fig. 10A). In the experiments, a random circulant matrix and a random vector were generated, after which the average execution time of the two algorithms was compared. Results have been calculated 1000 times and find average.

In the case with the number of elements of the opening vector less than 128, the speed of the modified method is inferior to the direct method of multiplication, but starting with 1024 elements manifests itself a multiple acceleration of the matrix-vector multiplication. With a vector size of 4096, the multiplication procedure is accelerated by about 20 times. It should be noted that at small vector lengths it turns out to be about two times slower than the direct method, that is, the speed gap is small and is associated with the quality of the used computer optimization of matrix multiplication.

The proposed algorithm for accelerating the product of a matrix-vector product is implemented in a Planar3D model with an explicit numerical integration scheme. As a result we compare time of calculations with difference methods of the matrix-vector multiplication on the each numerical integration step (Fig. 10.B). Three versions of the program were compared: direct, modified and hybrid method of matrix-vector multiplication. The hybrid method implies that at first the direct method of matrix multiplication is used and when a certain number of elements on the crack are reached, it is changed to a modified one. The calculations were carried out for the propagation of a fracture in a homogeneous medium. Three versions of the calculation program are written in C ++ and use the maximum optimization for compiler. To average the results, five series of calculations were performed.



1. (B)

**Fig. 10.** Time of calculation direct and modified algorithms (A), time of calculation 1-hour hydraulic fracturing with deferent methods of matrix-vector multiplication (B).

The size of the influence matrix and the opening vector in the crack depends on the number of mesh elements. In this way, faster calculations of the matrix-vector product on small crack sizes are provided. The study revealed fracture sizes (1000 elements) at which the modified method gives the advantage of (fig. 10B). The acceleration of the program execution by 2 times was obtained during the simulation of hourly hydraulic fracturing with constant injection using a hybrid method of matrix multiplication.

Thus, in this section, we apply the method of accelerating matrix-vector multiplication using FFT for matrices that are close to circulant structure. With a matrix size of 4096, an increase in the computational speed by about 20 times is obtained. When using this method in the implementation of the planar hydraulic fracturing model, the performance of the computational module has increased approximately twice.

1. Conclusions

In this paper, we propose methods for accelerating a Planar3D model of a hydraulic fracturing with an explicit numerical integration scheme using FFT. The implementation proposed to the authors is the Radix-2 Cooley-Tukey algorithm, using the std::vector() STL container. The correctness and speed of calculations was compared with well-known libraries. During the analysis, it was found that the speed of calculations is comparable with other libraries. Testing the library is represented by accelerating the multiplication of the matrix-vector product and processing the input data of lithology. In the Planar3D model, the influence matrix can be reduced to a circular form by applying periodic boundary conditions, and the use of a simple FFT-based algorithm can increase the model computation speed. The acceleration was 2 times when calculating the propagation of a crack in a homogeneous medium. FFT can also be used in processing the input data; an example of such an application with layer averaging is presented in 2.1. The use of low-pass filter allows you to accelerate the calculation by 20% within the acceptable accuracy for engineering calculations.

Acknowledgement.

The authors are grateful to A. S. Linkov, V. A. Kuzkin and N. S. Markov for useful and stimulating discussions.

This research was financed by Ministry of Education and Science of the Russian Federation within the framework of the Federal Program "Research and development in priority areas for the development of the scientific and technological complex of Russia for 2014-2020", Activity 1.2., Agreement on Grant No. 14.575.21.0146 of 26 September, 2017, unique identifier: RFMEFI57517X0146.

References

1. Heideman, Michael T., Don H. Johnson, and C. Sidney Burrus. "Gauss and the history of the fast Fourier transform." Archive for history of exact sciences 34.3 (1985): 265-277.
2. Cooley, James W., and John W. Tukey. "An algorithm for the machine calculation of complex Fourier series." Mathematics of computation 19.90 (1965): 297-301.
3. Good, Irving John. "The interaction algorithm and practical Fourier analysis." Journal of the Royal Statistical Society: Series B (Methodological) 20.2 (1958): 361-372.
4. Bruun, Georg. "z-transform DFT filters and FFT's." IEEE Transactions on Acoustics, Speech, and Signal Processing 26.1 (1978): 56-63.
5. Rader, Charles M. "Discrete Fourier transforms when the number of data samples is prime." Proceedings of the IEEE 56.6 (1968): 1107-1108.
6. Dentino, Mauro, John McCool, and Bernard Widrow. "Adaptive filtering in the frequency domain." Proceedings of the IEEE 66.12 (1978): 1658-1659.
7. Van Nee, D. J. R., and A. J. R. M. Coenen. "New fast GPS code-acquisition technique using FFT." Electronics Letters 27.2 (1991): 158-160.
8. Bellanger, M., and J. Daguet. "TDM-FDM transmultiplexer: Digital polyphase and FFT." IEEE Transactions on Communications 22.9 (1974): 1199-1205.
9. Raja, J., and V. Radhakrishnan. "Filtering of surface profiles using fast Fourier transform." International Journal of Machine Tool Design and Research 19.3 (1979): 133-141.
10. White, S. "A simple FFT butterfly arithmetic unit." IEEE Transactions on Circuits and Systems 28.4 (1981): 352-355.
11. <https://github.com/NikitaMushchak/Fast-Fourier-Transform>
12. Wang, Endong, et al. "Intel math kernel library." High-Performance Computing on the Intel® Xeon Phi™. Springer, Cham, 2014. 167-188.
13. Li, Yan, et al. "MPFFT: An auto-tuning FFT library for OpenCL GPUs." Journal of Computer Science and Technology 28.1 (2013): 90-105.
14. Gu, Liang, Xiaoming Li, and Jakob Siegel. "An empirically tuned 2D and 3D FFT library on CUDA GPU." Proceedings of the 24th ACM International Conference on Supercomputing. ACM, 2010.
15. Taboada, Jose Manuel, et al. "MLFMA-FFT parallel algorithm for the solution of large-scale problems in electromagnetics." Progress In Electromagnetics Research 105 (2010): 15-30.
16. Frigo, Matteo, and Steven G. Johnson. "FFTW: An adaptive software architecture for the FFT." Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181). Vol. 3. IEEE, 1998.
17. Galassi, Mark, et al. GNU scientific library. Network Theory Limited, 2002.
18. <http://eigen.tuxfamily.org/index.php?title=Main_Page>
19. Swarztrauber, Paul N., et al. "Bluestein's FFT for arbitrary n on the hypercube." Parallel computing 17.6-7 (1991): 607-617.
20. Pieterse, Vreda, et al. "Performance of C++ bit-vector implementations." Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists. ACM, 2010.
21. Peirce, A. "Modeling multi-scale processes in hydraulic fracture propagation using the implicit level set algorithm." Computer Methods in Applied Mechanics and Engineering 283 (2015): 881-908.
22. Khasanov, M. M., et al. "Scientific engineering as the basis of modeling processes in field development." Georesources 20.3 (eng) (2018).
23. Starobinskii, E. B., and A. D. Stepanov. "Adapting the explicit time integration scheme for modeling of the hydraulic fracturing within the Planar3D approach." Journal of Physics: Conference Series. Vol. 1236. No. 1. IOP Publishing, 2019.
24. Linkov, A. M. "The particle velocity, speed equation and universal asymptotics for the efficient modelling of hydraulic fractures." Journal of Applied Mathematics and Mechanics 79.1 (2015): 54-63.
25. Linkov, Aleksandr M. "Universal asymptotic umbrella for hydraulic fracture modeling." arXiv preprint arXiv:1404.4165 (2014).
26. Van Tiggelen, B. A. "Green function retrieval and time reversal in a disordered world." Physical review letters 91.24 (2003): 243904.
27. Strang, Gilbert. "A proposal for Toeplitz matrix calculations." Studies in Applied Mathematics 74.2 (1986): 171-176.