ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 5, pp. 547–554. © Pleiades Publishing, Ltd., 2019.

High-performance Processing of Covariance Matrices Using GPU Computations

K. Yu. Erofeev[1*], E. M. Khramchenkov[2], and E. V. Biryal'tsev[3***]**

(Submitted by A. M. Elizarov)

[1] *Kazan (Volga region) Federal University, Kazan, 420018 Russia*

[2] *Kazan Branch of Joint Supercomputer Center, Russian Academy of Sciences, Kazan, 420111 Russia*

[3] *ZAO Gradient, Kazan, 420045 Russia*

Received January 10, 2019; revised February 20, 2019; accepted February 20, 2019

Abstract—Practical applicability of many statistical algorithms is limited by large sizes of corresponding covariance matrices. These limitations can be significantly weakened due to effective use of the structure of covariance matrices, properties of the autocorrelation function, and advantages of the architecture of modern GPUs. This paper presents GPU implementations of the algorithms for inversion of a covariance matrix and solution of a system of linear equations whose coefficient matrix is a covariance matrix. Inversion of close to sparse covariance matrices is also considered in the work. For all the cases considered, significant accelerations were obtained in comparison with Octave mathematical software and ViennaCL computational library. For example, implemented algorithm of solution of a linear system was 6 times faster as compared with the implementation of Octave on the CPU and 3 times faster as compared with the ViennaCL implementation on the GPU for general matrices. The performance of inversion of a covariance matrix was 14 times faster than inversion algorithm of Octave on the CPU and 6 times faster than ViennaCL inversion algorithm on GPU.

DOI: 10.1134/S1995080219050068

Keywords and phrases: *covariance matrices, Toeplitz matrices, high-performance computing, GPU.*

1. INTRODUCTION

At the present time use of covariance matrices is widespread in various fields of science. For example, in the economic statistics, covariance matrices find their use in analysis of multidimensional time series [1]. Covariance matrices are used in biostatistics to process data of functional magnetic resonance imaging [2]. Covariance matrices are used in data analysis to predict weather [3]. Many applications of statistical signal processing, in particular, to microseismic problems [4], are built around covariance matrices. In the modern world, data volumes are rapidly growing and simultaneously requirements for processing time become more tight. For the number of areas discussed above, this means the need to perform operations with covariance matrices of a large size more quickly. For example, the practical application of the method described in [4] is limited by size of a covariance matrix and time required for its inversion. In these conditions, methods for effective work with large covariance matrices become especially relevant. The architecture of modern graphics processing units (GPUs) is suited for performing operations on matrices; therefore, nowadays GPUs are widely used in optimizing matrix calculations. There are many libraries for effective work with general matrices, but there are no open access libraries optimized for working with covariance matrices. At the same time usage of special

*E-mail: krllerof@gmail.com

**E-mail: ekhramch@gmail.com

***E-mail: igenbir@ya.ru

structure of covariance matrices and properties of the autocorrelation function can significantly reduce time and memory costs of matrix computations. Covariance matrices have Toeplitz structure and contain only $O(n)$ unique elements instead of $O(n^2)$ in case of general matrices. Toeplitz matrix A can be presented as follows:

$$A = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & \cdots & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & \ddots & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{pmatrix}.$$

Specialized algorithms for processing of Toeplitz matrices have been known for a long time. One of the first examples were algorithms of solution of linear systems with a Toeplitz coefficient matrix which required $O(n^2)$ operations instead of $O(n^3)$ for general linear systems. Algorithms were proposed by Levinson [5] and Bareiss [6], the issues of numerical stability were considered in [7] for the Levinson algorithm and in [8] for the Bareiss algorithm. Comparing these two algorithms, it should be noted that the Levinson algorithm only requires $O(n)$ additional memory space, while the Bareiss algorithm requires $O(n^2)$. In turn, Bareiss algorithm shows better numerical stability for ill-conditioned coefficient matrices. To date, asymptotically faster $O(n \log n)$ algorithms for solving the Toeplitz systems of equations have been developed [9]. However, due to the known problems with their numerical stability, as well as significant constants hidden behind O-notation in logarithmic complexity estimates, this paper presents GPU implementation of improved Levinson algorithm described in [10].

The inversion of general matrices of can be performed in $O(M(n))$, where $M(n)$ is complexity of multiplication of $n \times n$ matrices, which in practice almost always equals $O(n^3)$. The first inversion algorithm for Toeplitz matrices working in $O(n^2)$ was proposed by Trench [11]. At the moment asymptotically faster algorithms were developed. For example, algorithm described in [12] requires $O(n \log^2 n)$ operations but it is worse parallelized than Trench algorithm. Numerical stability of the Trench algorithm is shown in [13], and the algorithm itself is presented in a simplified form, prepared for effective implementation. Also, ideas allowing the inverse Toeplitz matrix to be stored in a sparse form are presented there.

2. LEVINSON ALGORITHM FOR SOLVING LINEAR SYSTEMS

Solution of linear systems of equations using the Levinson algorithm is performed in two stages. At the first stage, the first and last columns of the matrix inverse Toeplitz matrix are obtained. At the second stage, the system of equations is solved using these obtained columns.

If T is a coefficient matrix of a linear system expressed by covariance matrix of two sequences L and S, i.e. Toeplitz matrix of $n \times n$ size. Let us define matrix T_k, which can be obtained from the matrix T by deleting all rows and columns with indexes $i > k$. It should be noted that if T is Toeplitz matrix, then T_k is also a Toeplitz matrix. A covariance matrix due to its Toeplitz structure is completely specified by the first raw and the last column. In order to use this fact for convenient writing let us designate a_i for $i > 0$ as covariance of a sequence L shifted on i counts with S sequence, and a_{-i} for $i > 0$ as covariance of a sequence L with S sequence shifted on i counts. Then the first row of the matrix T is a vector $(a_0, ...a_{(n-1)})$, and the first column of the matrix T is a vector $(a_0, ..., a_{-(n-1)})^T$.

Let's consider x_k^1 and x_k^k such that they are the solutions of equations $T_k x_k^1 = e_1, \quad T_k x_k^k = e_k$, where e_k is a vector of corresponding size filled with zeroes and with 1 on the i-th position. Now we consider following correlation

$$T_{k+1} \begin{bmatrix} x_k^1 & 0 \\ 0 & x_k^k \end{bmatrix} = \begin{bmatrix} e_1 & y_k^1 \\ y_k^k & e_k \end{bmatrix},$$

where $y_k^k = [a_1 ... a_k]x_k^1$, $y_k^1 = [a_{-1} ... a_{-k}]x_k^k$. We introduce a matrix G such that

$$G_k = \begin{bmatrix} 1 & y_k^1 \\ y_k^k & 1 \end{bmatrix}.$$

Then vectors x_{k+1}^1 and x_{k+1}^{k+1} will be obtained as follows

$$[x_{k+1}^1 x_{k+1}^{k+1}] = \begin{bmatrix} x_k^1 & 0 \\ 0 & x_k^k \end{bmatrix} G_k^{-1} \tag{1}$$

Using recursion scheme (1) and starting from $x_1^1 = x_1^k = 1/a_0$, we can calculate the first and the last column of the matrix T^{-1}.

The second stage of the algorithm consists in solution of the linear system using the first and the last column of the matrix T^{-1}. Let us consider the following linear system

$$Tz = b. \tag{2}$$

Defining x_k and z_k as the solutions of linear systems

$$T_k x_k = e_k, \quad T_k z_k = b_k. \tag{3}$$

From (2) and (3) we obtain following expression

$$T_{k+1} \begin{bmatrix} z_k \\ 0 \end{bmatrix} = \begin{bmatrix} b_k \\ d_k \end{bmatrix},$$

where $d_k = [t_{k+1,1} ... t_{k+1,k}]z_k$. In this case, the recursive scheme for solving the linear system will be as follows:

$$z_{k+1} = \begin{bmatrix} z_k \\ 0 \end{bmatrix} + (b_{k+1} - d_k)x_{k+1}, \quad z_1 = \frac{b_1}{a_0}. \tag{4}$$

Beginning the recursion (4) from z_1 we will obtain the solution of the linear system.

3. EFFICIENT STORAGE OF DIRECT AND INVERSE COVARIANCE MATRICES

The covariance matrix has a Toeplitz structure, which means that in the worst asymmetric case it has only $2n - 1$ unique values, instead of n^2 unique values in case of a general matrix. Algorithms for operations on general matrices require ordinary dense matrices as arguments, therefore, the use of efficient methods for storing covariance matrices is impossible as long as operations on them are performed by standard methods. This paper presents methods that are specific to covariance matrices, so we are free to choose how to represent them in computer memory.

In the implementation of the algorithms presented in this paper, the covariance matrices are stored as a continuous array of unique elements. This approach to storing covariance matrices has two advantages. The first and most obvious is space saving. Modern GPU with a memory of 16GB can accommodate a covariance matrix with a side $n \approx 4.5 \times 10^4$ if it is stored in the classical way and with a side of $n \approx 10^9$ if only unique elements are stored. The second advantage is a significant increase in data locality. When using this method of storage, operation of accessing the element $a_{i,j}$ of the matrix is replaced by the operation of accessing the element c_{i-j+n} of the actually stored sequence of unique values. During inversion covariance matrix loses its Toeplitz structure while remaining persymmetric.

A matrix that is inverse for Toeplitz matrix has $n^2/2$ unique elements. But as it outlined in [13], a matrix inverse to a covariance matrix is completely determined by its first column and row, while the remaining elements can be expressed as a function of the elements of the first column and the first row. Thus the above-mentioned method of storage of direct covariance matrices can also be applied to inverse ones, which allows obtaining the advantages of space saving and data locality.

4. MATRIX INVERSION: TRENCH ALGORITHM

The Trench algorithm is applied to normalized covariance matrices, i.e. to those who have ones on the main diagonal, in other words to the correlation matrices. With this in mind, we introduce the following notation:

L_k is the correlation matrix which is obtained from the original matrix by deleting first $n-k$ rows and columns;

a is the vector consisting of the elements of the first row of the original matrix, starting with the second element;

r is the vector consisting of elements of the first column of the original matrix, starting from the second element;

$B_k = L_k^{-1}$;

e_k is the vector consisting of elements of the first column of the matrix B_k starting from the second element and multiplied by some coefficient;

g_k is the vector consisting of elements of the first row of the matrix B_k starting from the second element and multiplied by some coefficient;

e is the vector consisting of elements of the first column of the matrix L_{-1} starting from the second element;

g is the vector consisting of elements of the first row of the matrix L_{-1} starting from the second element;

According to the previous section, vectors e and g and element $B_{1,1}$ completely determines the matrix B. Let us take the following initialization values for the recursion:

$$\lambda_1 = 1 - a_1 r_1, \quad e_1 = -a_1, \quad g_1 = -r_1.$$

Then, using relations

$$q_k = -(a_{k+1} + e_k[a_k \dots a_1]), \quad w_k = -(r_{k+1} + r_k[g_k \dots g_1]),$$

$$e_{k+1} = \begin{bmatrix} e_k + \frac{q_k}{\lambda_k}[g_k \dots g_1] \\ \frac{w_k}{\lambda_k} \end{bmatrix}, \quad [g_{k+1} \dots g_1] = \begin{bmatrix} \frac{q_k}{\lambda_k} \\ [g_k \dots g_1] + \frac{w_k}{\lambda_k} e_k \end{bmatrix}, \quad \lambda_{k+1} = \lambda_k - \frac{w_k q_k}{\lambda_k}, \qquad (5)$$

vectors e, g and $L_{1,1}^{-1}$ can be obtained from (5) as

$$L_{1,1}^{-1} = \frac{1}{\lambda_n}, \quad e = \frac{e_n}{\lambda_n}, \quad g = \frac{g_n}{\lambda_n},$$

and the elements of L_{-1} can be found from the following equation:

$$L_{i+1,j+1}^{-1} = L_{i,j}^{-1} + \frac{1}{\lambda_n}(ge' - [e_{1,k} \dots e_{1,1}][g_{1,k} \dots g_{1,1}]')_{i,j}.$$

It should be noted that there is no need to store the entire matrix during the algorithm, since the inverse matrix to the covariance matrix is completely determined by the first row and column, and two arrays can be used to store the entire sequence of vectors e_k and g_k.

5. INVERSION OF CLOSE TO SPARSE AUTOCORRELATION MATRICES USING FROBENIUS FORMULA

In a large number of practical applications, for example in search of a known in advance pattern that does not repeat in a signal [4], the value of the elements of the autocorrelation matrix tend to zero with distance from the main diagonal. This is due to that while correlation is a measure of similarity, the signal is usually most similar to itself, little less similar to itself shifted by one, etc. This fact can be used to speed up the calculation of the inverse covariance matrix using the Frobenius formula. The formula of Frobenius is as follows:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BH^{-1}CA^{-1} & -A^{-1}BH^{-1} \\ -H^{-1}CA^{-1} & H^{-1} \end{bmatrix},$$

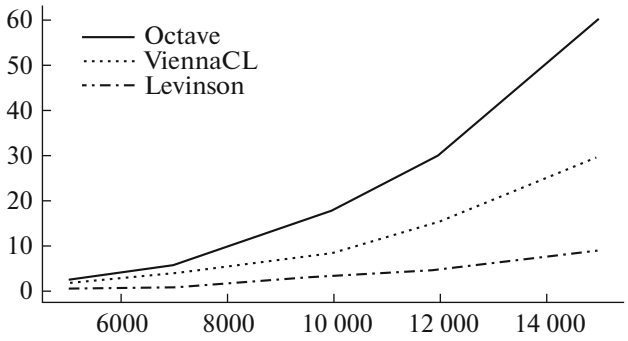

Fig. 1. Comparison of the speedup of solving a linear system.

where $H = D - CA^{-1}B$. Frobenius formula gives opportunity to reduce inversion of a matrix with $n \times n$ size to inversion of two $n/2 \times n/2$ matrices and ten multiplications of $n/2 \times n/2$ matrices.

In practice one of the most well-known libraries of matrix algebra on GPU, ViennaCL, requires approximately a $3n^2$ memory to invert $n \times n$ matrix. In accordance with this, a matrix with a side of $n \approx 2.5 \times 10^4$ can be inverted on a modern GPU with 16GB memory. In the meantime using Frobenius formula with ViennaCL we memory usage can be reduced to $6(n/2)^2$, if only A_{-1}, H_{-1}, current calculating block and temporary matrix are stored. Thus, a matrix with a side of 36000 can be inverted on a GPU with 16 GB memory. It should be noted that using Frobenius formula recursively to A_{-1} and H_{-1} memory usage can be brought into proximity with $2(n/2)^2$ at the cost of time for matrix inversion.

The Frobenius formula can also be used to accelerate the inversion of the covariance matrix, if we take into account the properties of the autocorrelation function, namely, the fact that with the distance from the main diagonal, the elements of the autocorrelation matrix tend to zero. We can consider the blocks C and B close to zero, in this case, even closer to zero we can consider those terms of the Frobenius formula, which contain multiplications both on the block C and on the block B at the same time. Then the condensed Frobenius formula for approximate matrix inversion will take the form

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \approx \begin{bmatrix} A^{-1} & -A^{-1}BD^{-1} \\ -D^{-1}CA^{-1} & D^{-1} \end{bmatrix}.$$

Thus, we replace the full inversion of matrix of size $n \times n$ by the inversion of two matrices of size $n/2 \times n/2$ and four multiplications of matrices of size $n/2 \times n/2$, which gives a gain in speed and also a fairly good approximation to the inverse matrix if the autocorrelation function has assumed properties. The sufficiency of such approximation should be decided separately for each specific practical problem. An even rougher approximation is possible if we consider all terms in which C and B are found to be zeros. Such an approximation allows us to reduce the inversion of an $n \times n$ matrix only to the inversion of two $n/2 \times n/2$ matrices

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \approx \begin{bmatrix} A^{-1} & -A^{-1}BD^{-1} \\ -D^{-1}CA^{-1} & D^{-1} \end{bmatrix}.$$

It is also worth noting that further application of the operation shown above will lead us to the case when the inversion of the matrix is reduced to practically computationally free inversion of the elements of its main diagonal.

6. PERFORMANCE TESTS

The code for the Levinson algorithm and the Trench algorithm was written using C++ and the VexCL vector expression library, which allows short and scalable coding of calculations on the GPU [15, 16]. Approximate matrix inversion using the Frobenius formula was implemented using the ViennaCL

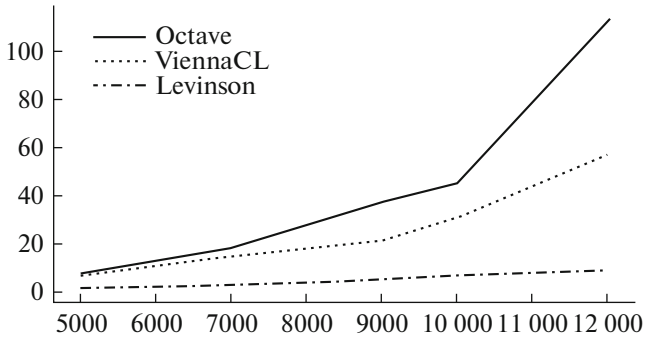

Fig. 2. Comparison of the speedup of matrix inversion using the Trench algorithm.

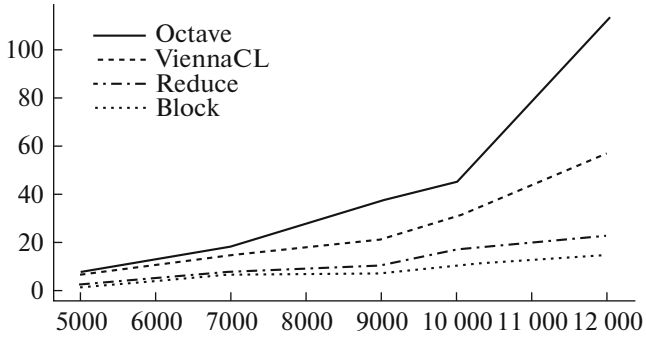

Fig. 3. Comparison of the speedup of the approximate matrix inversion using the Frobenius formula.

GPU library [17]. Computational experiments described in this section were carried out on a high-performance workstation equipped with Tesla C2070 professional graphics card. The x-axis on the graphs shows the size of the matrix side, and the y-axis shows the time for the corresponding operation in seconds.

Figure 1 shows three timelines for solving the Toeplitz linear system, made using the Octave package on the central processor, using the ViennaCL matrix algebra library on the GPU and the implementation of the Levinson algorithm on the GPU presented in this paper. From the tests, we can conclude that the implementation presented in the work is 6 times faster than the implementation of the linear system solution in Octave and 3 times faster than the ViennaCL GPU implementation.

Figure 2 shows three plots of the inversion time of a Toeplitz matrix made using the Octave package on the central processor, using the ViennaCL matrix algebra library on the GPU and using the implementation of the Trench algorithm on the GPU presented in this paper. It's worth to mention that proposed implementation performs better at larger matrix sizes and computational time increase insignificantly as we increase matrix size. From the tests, we can conclude that the implementation presented in the work is 14 times faster than the implementation of Octave and 7 times faster than the implementation of ViennaCL. Again it should be noted that proposed algorithm demonstrates far better scalability as computational time almost does not increase with increase of the matrix size.

Figure 3 shows four plots of matrix inversion time, performed using the Octave package on the central processor, using the ViennaCL matrix algebra library on the GPU and using the presented implementation of the approximate matrix inversion using the Frobenius formula in two forms.

The line marked as Reduce corresponds to the approximate inversion that was performed using the formula

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \approx \begin{bmatrix} A^{-1} & -A^{-1}BD^{-1} \\ -D^{-1}CA^{-1} & D^{-1} \end{bmatrix}.$$

The line marked as Block corresponds to the approximate inversion that was performed using the formula

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \approx \begin{bmatrix} A^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix}.$$

7. CONCLUSION

In this paper, the implementations of the Levinson algorithm for solution of linear systems and the Trench algorithm for inverting Toeplitz matrices are presented. The developed implementations are intended for GPGPU high-performance computing. The tests showed a threefold increase in performance for the Levinson algorithm, compared with the current implementation of solving general systems of equations by the ViennaCL library. The inversion of the Toeplitz matrices based on the Trench algorithm turned out to be 7 times faster than the current implementation of the inversion general matrices of the ViennaCL library.

The paper also describes the implementation and performance test for the approximate inversion algorithm for covariance matrices that are close to sparse using the Frobenius formula. The possible acceleration of this algorithm depends on the properties of the autocorrelation function for each specific application. On the test matrices, it was possible to achieve a double acceleration of the matrix inversion compared with the algorithm from the ViennaCL library.

Also, a method for efficient storage of matrices which are inverse to Toeplitz matrices was considered. It was possible to achieve a reduction in the used memory from n^2 elements to $2n - 1$ elements, as well as obtain significantly increase of the data locality. It should be noted that data locality is one of the important factors for improving performance when computing on GPUs.

Thus, the proposed implementations of algorithms for covariance matrices can be used in scientific high-performance software that are demanding in terms of execution time and memory usage.

FUNDING

The work was partially funded by the state assignment to the Joint Supercomputer Center of the Russian Academy of Sciences for scientific research 065-2019-0016 (reg. no. AAAA-A19-119011590098-8) and was partially funded by the RFBR grant nos. 18-07-00964 and 18-47-160010.

REFERENCES

1. M. W. Robbins and T. J. Fisher, "Cross-correlation matrices for tests of independence and causality between two multivariate time series," J. Business Econ. Statist. **33**, 459–473 (2015).
2. S. Efromovich and E. Smirnova, "Statistical analysis of large cross-covariance and cross-correlation matrices produced by fMRI images," J. Biometr. Biostatist. **5** (2), 1 (2014).
3. M. Jun, "Non-stationary cross-covariance models for multivariate processes on a globe," Scand. J. Statist. **38**, 726–747 (2011).
4. D. E. Demidov, E. V. Mokshin, and E. V. Birialtsev, "Determination of moment tensor and location of microseismic events under conditions of highly correlated noise based on the maximum likelihood method," Geophys. Prospect. **33**, 437–449 (2017).
5. N. Levinson, "The Wiener RMS error criterion in filter design and prediction," J. Math. Phys. **25**, 261–278 (1947).
6. E. H. Bareiss, "Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices," Numer. Math. **13**, 404–424 (1969).
7. H. Krishna and Y. Wang, "The Split Levinson algorithm is weakly stable," SIAM J. Numer. Anal. **30**, 1498–1508 (1993).
8. A. W. Bojanczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet, "On the stability of the Bareiss and related Toeplitz factorization algorithms," SIAM J. Matrix Anal. Appl. **16**, 40–57 (1995).
9. S. Chandrasekeran, M. Gu, X. Sun, J. Xia, and J. Zhu, "A superfast algorithm for Toeplitz systems of linear equations," SIAM J. Matrix Anal. Appl. **29**, 1247–1266 (2007).
10. G. Heinig and K. Rost, "Fast algorithms for Toeplitz and Hankel matrices," Linear Algebra Appl. **435**, 1–59 (2011).

11. W. F. Trench, "An algorithm for the inversion of finite Toeplitz matrices," SIAM J. Appl. Math. **12**, 525–522 (1964).
12. G. S. Ammar, "Classical foundations of algorithms for solving positive definite Toeplitz equation," Calcolo **33**, 99–113 (1996).
13. S. Zohar, "Toeplitz matrix inversion: the algorithm of W. F. Trench," J. ACM **16**, 592–601 (1969).
14. V. Y. Pan, Z. Q, Chen, and A. Zheng, "The complexity of the matrix eigenproblem," in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999,* pp. 507–516.
15. D. Demidov, K. Ahnert, K. Rupp, and P. Gottschling, "Programming CUDA and OpenCL: a case study using modern C++ libraries," SIAM J. Sci. Comput. **35**, 453–472 (2013).
16. D. Demidov, K. Ahnert, and M. Mulansky, "Solving ordinary differential equations on GPUs," in *Numerical Computations with GPUs* (Springer, Cham, 2010).
17. K. Rupp, Ph. Tillet, F. Rudolf, J. Weinbub, A. Morhammer, T. Grasser, A. Jangel, and S. Selberherr, "ViennaCL—linear algebra library for multi—and many-core architectures," SIAM J. Sci. Comput. **38**, 412–439 (2016).