www.dergipark.gov.tr
ISSN:2148-3736
El-Cezerî Fen ve Mühendislik Dergisi
Cilt: 5, No: 1, 2018 (1-10)

El-Cezerî Journal of Science and Engineering
Vol: 5, No: 1, 2018 (1-10)

ECJSE

**Research Paper / Makale**

# Reducing The Space And Time Complexity By The Use of Triangular Matrices

**Murat Erşen BERBERLER[a], Zeynep Nihan BERBERLER[b]**

[a,b]Faculty of Science, Department of Computer Science, Dokuz Eylul University, 35160, Izmir, TURKEY
[a]murat.berberler@deu.edu.tr, [b]zeynep.berberler@deu.edu.tr

**Received/Geliş:** 31.07.2017 **Revised/Düzeltme:** 23.10.2017 **Accepted/Kabul:** 24.10.2017

**Abstract:** Matrices are commonly used data structures in computer science. There do not exist available structures in programming languages for the special type of matrices such as triangular matrix. If it is required to use a triangular matrix as a data structure, then softwares are coded with inefficient space and time complexity due to the lack of data structure of a programming language. In this paper, transformation and inverse tranformation formulae to be used for representing the triangular matrices as a one dimensional array are gathered and an increase in the amount of efficiency of a program in terms of space and time complexity is objected.

***Keywords:*** Triangular matrix, One dimensional array, Space complexity, Time complexity.

# Üçgen Matrisler Kullanılarak Yer ve Zaman Karmaşıklığının Azaltılması

**Özet:** Matrisler bilgisayar bilimlerinde sıklıkla kullanılan veri yapılarıdır. Üçgensel matris gibi özel matris türleri için programlama dillerinde hazır yapılar bulunmamaktadır. Bir veri yapısı olarak üçgensel matris kullanmak gerekiyorsa programlama dilinden kaynaklanan bu eksiklik nedeniyle bellek ve zaman karmaşıklığı yönünden etkin olmayan yazılımlar kodlanmaktadır. Bu çalışmada üçgensel matris tipindeki yapıların bir boyutlu dizi olarak temsil edilmesinde kullanılan dönüşüm ve ters dönüşüm formülleri elde edilerek programların yer ve zaman karmaşıklığı açısından etkinliğinin arttırılması amaçlanmaktadır.

***Anahtar Kelimeler:*** Üçgensel matris, Bir boyutlu dizi, Yer karmaşıklığı, Zaman karmaşıklığı.

## 1. Introduction

A square matrix $A=\left[a_{ij}\right]$ is called *upper triangular* if the elements below the main diagonal are zero, that is $a_{ij}=0$ for $i>j$. It is called *lower triangular* if $a_{ij}=0$ for $i<j$. Triangular matrices are of critical importance for computer science since those type of matrices are all used for representation and solution of many problems existing in literature. As an example, the $LU$-factorization of a matrix used to efficiently solve a linear system, representation of symmetric matrices with less space [1,2].

There are many applications of the triangle matrices such as graph theory [3-5], computer games [6], big data processing [7-12], solutions of linear equation systems [13-18].

*How to cite this article*
Berberler, M.E., Berberler, Z.N., "*Reducing The Space And Time Complexity By The Use of Triangular Matrices*" El-Cezerî Journal of Science and Engineering, 2018, 5(1); 1-10.

*Bu makaleye atıf yapmak için*
Berberler, M.E., Berberler, Z.N., *"Üçgen Matrisler Kullanılarak Alan ve Zaman Karmaşıklığının Azaltılması"* El-Cezerî Fen ve Mühendislik Dergisi 2018, 5(1); 1-10.

When it is required to use triangular type matrices, since the data structures for triangular matrices are not existing in programming languages, there occur two disadvantages. The first one is if the triangular matrices are stored with square matrix data structures in a wasteful manner, then the increase in the space complexity due to the use of much more space than required decreases the efficiency of the program. The second one is that if jagged arrays are created by the use of pointers for the structure of triangular matrices, then this time the increase in the time complexity due to the use of pointer arithmetic often decreases the efficiency of the program.

The paper proceeds as follows. Representation of the triangular matrices as a one dimensional array are deeply investigated, mathematical analysis for space and time complexity is performed, and transformation and inverse transformation formulae are derived throughout section 2. Section 3 includes computational experiments performed in order to demonstrate the efficiency of the derived formulae. Finally, section 4 concludes the paper.

## 2. Method

The method proposed for the representation of triangular matrices takes the method as a base that is used for representing the two dimensional matrices as a one dimensional array, and the underlying idea for storage of matrices in memory and accessing the contents of the cells are the same [2-3]. The direct relation between a two dimensional $m \times n$ matrix and a one dimensional array with $m \times n$ elements can be represented as given in Figure 1. It is clear that if the transformation formulae are found, then an array can be used instead of matrix as a data structure and vice versa.

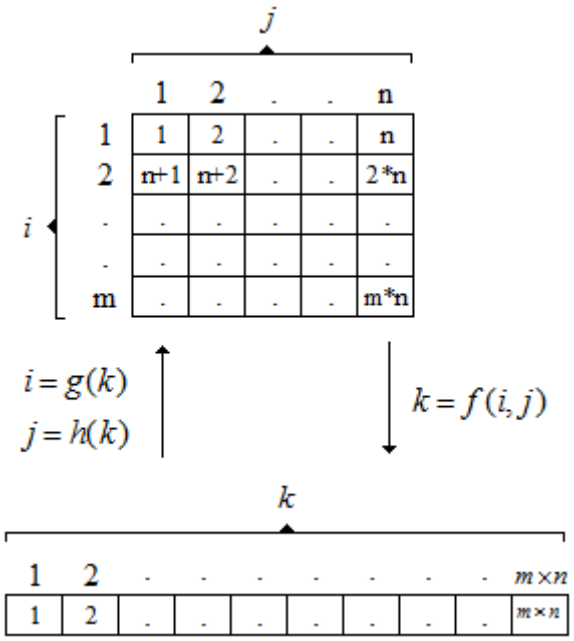


*Figure 1. A two dimensional matrix and a one dimensional array with elements*

### *2.1 Representation of matrix data structure as a one dimensional array and transformation formula*

For the simplicity of both introducing the method and the occurrence of triangular matrices, matrices are all considered as square matrices throughout the rest of the paper. The cells of the matrix are labeled in increasing order from left to right and from top to bottom as seen in Figure 2 for the ease of understanding which cell of the matrix is for the which element of the array.

| | | j | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| i | 1 | 1 | 2 | 3 | 4 | 5 |
| | 2 | 6 | 7 | 8 | 9 | 10 |
| | 3 | 11 | 12 | 13 | 14 | 15 |
| | 4 | 16 | 17 | 18 | 19 | 20 |
| | 5 | 21 | 22 | 23 | 24 | 25 |

*Figure 2. 5x5 square matrix*

Simply the method is based on the projection of a plane on a line. That is, by taking the index numbers of the row vectors of the matrix into consideration, the indices are sorted successively to obtain a unique vector. For this purpose, the row vectors of the matrix should be isolated from each other, thus if the column index value $j$ is subtracted from the current value of each cell, then the state holds as it can be seen in Figure 3.

| | | j | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| i | 1 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 5 | 5 | 5 | 5 | 5 |
| | 3 | 10 | 10 | 10 | 10 | 10 |
| | 4 | 15 | 15 | 15 | 15 | 15 |
| | 5 | 20 | 20 | 20 | 20 | 20 |

*Figure 3. The column index value j had subtracted from the current value of each cell*

Thereafter, in order to obtain the corresponding values of the cells in each row, it is clear to see that the formula $(i-1)\times n$ that is dependent on $i$ can be used. As a last step, when the index value $j$ that is subtracted at first is added to $(i-1)\times n$, the following formula that will be used in vector projection of two dimensional array is obtained, that is, $k=(i-1)\times n+j$.

### *2.2 The formulae of transformation from array data structure to two dimensional matrix data structure*

The projection function $k=(i-1)\times n+j$ obtained in the previous subsection is defined in the set of positive natural numbers and since it is a bijective function, it is quite possible to find the inverse tranformation formulae.

It is clear that in order to find the value of the row index $i$, $k$ should be divided by the dimension of the matrix $n$. The reason is that while $k$ is being evaluated, $i$ times $k$ is added, therefore the number of $i$s used for reaching $k$ should be found by division. The problem encountered here is that the result of division is not an integer, hence the value $k/n$ is rounded up to get an integer value as $i=\lceil k/n \rceil$.

On the other hand, the value of the column index $j$ is found by $j \equiv k \pmod{n}$ since it should be detected for the related row that how many times $j$ is added to get $k$. Thus, the modulo operation is taken into consideration. The problem encountered here is the result obtained after the modulo operation is zero. Since there does not exist a zero indexed column in the matrix, when $j$ is zero due to the equivalence $j \equiv n \pmod{n}$, the value $n$ itself will be taken instead of zero. Without making any adjustments, the correct result can be obtained by typing: $j = ((k-1) \bmod n) + 1$.

### *2.3 The representation of triangular matrix data structure as a vector and transformation formulae*

The triangular matrices are the matrix types that are often used in computer science. There do not exist any special type data structures available in programming languages for the use of those type matrices. Therefore, the triangular matrices are commonly stored in square matrix data structures and the unnecessary increment of the space complexity leads inefficient program design. As an example, if a triangular matrix is stored in a square matrix of dimension $n \times n$, it takes a space of $n(n+1)/2$ units, but $n(n-1)/2$ units of space remain inert and this leads an increase of the space complexity as a rate of $\geq 50\%$ for the values of $n \geq 50$ with respect to the formula $\left(50 - \frac{50}{n}\right)\%$. To avoid such types of disadvantages, the transformation formulae can be generated between two dimensional array and a vector by the repetition of the same steps detailed in subsection 2.1. The cells of the matrix are labeled in the increasing order of counting numbers from left to right and from top to bottom as seen in Figure 4.

| | | *j* | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| *i* | 1 | 1 | | | | |
| | 2 | 2 | 3 | | | |
| | 3 | 4 | 5 | 6 | | |
| | 4 | 7 | 8 | 9 | 10 | |
| | 5 | 11 | 12 | 13 | 14 | 15 |

*Figure 4. Triangular matrix*

The row vectors of the matrix should be isolated from each other. This is achieved by subtracting the value of column index $j$ from the value of each cell as it is seen in Figure 5.

| | | *j* | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| *i* | 1 | 0 | | | | |
| | 2 | 1 | 1 | | | |
| | 3 | 3 | 3 | 3 | | |
| | 4 | 6 | 6 | 6 | 6 | |
| | 5 | 10 | 10 | 10 | 10 | 10 |

*Figure 5. The column index value j had subtracted from the current value of each cell*

The same values are obtained at each row of the final matrix, and it is clear that those values are the sum of the integers from 1 to $i-1$. As a consequence, the formula that gives the value of the vector index $k$ is determined as $k=\frac{i(i-1)}{2}+j$.

The inverse transformation formulae for the triangular matrices are not easy to find as it is for the matrices considered in subsection 2.2. The following method can be used to reach a result.

Let $i$ be the row index of $k$ .Whereat it can be written as $\frac{i(i-1)}{2}<k\le\frac{i(i+1)}{2}$. When this inequality is simplified we have that

$$(i-1)*i<2k\le i*i(+1)$$

$$i^2-i-2k<0 \qquad\qquad i^2+i-2k\ge 0$$

$$i_1=\frac{1-\sqrt{1+8k}}{2} \qquad\qquad i_2=\frac{-1+\sqrt{1+8k}}{2}$$

The roots of the inequalities $i_1$ and $i_2$ are of the same real values with opposite signs, thereof the positive valued root $i_2$ is preferred and used for the row index $i$. The resulting value is ceiling to get an integer, that is $i=\left\lceil\frac{\sqrt{1+8k}-1}{2}\right\rceil$.

The most difficult part is to find a transformation formula of the column index $j$ for the triangular matrices. To obtain this formula, it will be useful to carefully examine Figure 6.

| $k$ | $i$ | $j$ | $m$ |
|---|---|---|---|
| 1 | 1 | **1** | - |
| 2 | 2 | **1** | - |
| 3 | 2 | **2** | - |
| 4 | 3 | **1** | 3 |
| 5 | 3 | **2** | 3 |
| 6 | 3 | **3** | 0≡3 |
| 7 | 4 | **1** | 6 |
| 8 | 4 | **2** | 6 |
| 9 | 4 | **3** | 6 |
| 10 | 4 | **4** | 6 |

*Figure 6. The value of the column index j changes depending on the value of the row index i*

The value of the column index $j$ that is in bold in Figure 6 changes depending on the value of the row index $i$. The reason is that the value $j$ is obtained when the modulo of $k$ is taken due to the

value in the column $m$ and here the value $m=\frac{(i-1)i}{2}$ used in the modulo operation depends on the index $i$. The first row in which $m$ equals zero, and the second and the third rows in which $m$ equals one are all left blank since the modulo operation is meaningless. Therefore, the column index $j$ for the first three values should be given as predefined.

As it can be seen in the sixth row of Figure 6, a special case occurs. When $k=6$, the value $j\equiv 6(\bmod 3)\equiv 0$ is obtained and the value 0 is updated with 3. This special case is only valid for $k=6$ and this can be proven as follows.

*Proof.* All values of $i$ that are used in the equation $m=\frac{(i-1)i}{2}$ and satisfying the equivalence of $k(\bmod m)\equiv 0$ are investigated. As it is seen in Figure 6, $k$ is ranging between $(m+1)\le k\le(m+i)$. Thus, the remainders that are found by $k(\bmod m)$ should be in the interval $1\le k(\bmod m)\le i$, that is, a division with remainder zero is not expected. However, as it is in the case of otherwise, it is examined that whether it is possible to choose such a value $i$ that determines the upper bound of $m+i$ so that $k(\bmod m)\equiv 0$, that is, $(m+i)(\bmod m)\equiv 0$. If $m$ is rewritten in terms of $i$, we get $m=\frac{(i-1)i}{2}$ and so $(\frac{(i-1)i}{2}+i)\left(\bmod \frac{(i-1)i}{2}\right)\equiv 0$. Then, $\exists c\in \mathrm{Z}$ should be found that satisfies $(\frac{(i-1)i}{2}+i)=c\,\frac{(i-1)i}{2}$. This leads an equation of $(1-c)i^2+(1+c)i=0$ with $\Delta=(1+c)^2$ yielding two different roots;

$$i_1=\frac{-(1+c)-(1+c)}{2(1-c)} \qquad i_2=\frac{-(1+c)+(1+c)}{2(1-c)}=0.$$

If $i=i_2=0$, then the equation $(1-c)i^2+(1+c)i=0$ is valid for all $c\in$Z, but this case contradicts with Figure 6. Henceforth, $i=i_1=\frac{-(1+c)-(1+c)}{2(1-c)}=\frac{c+1}{c-1}$. It is clear that $i=\frac{c+1}{c-1}$ takes an integer value only for $c=3$.

Consequently, the formula that will be used to evaluate the value of index $j$ is as follows:

$$j=\begin{cases} 1, & \text{if } k=1 \text{ or } k=2;\\ 2, & \text{if } k=3;\\ k\left(\bmod\left(\frac{(i-1)i}{2}\right)\right), & \text{if } k>3. \end{cases}$$

In a simpler way that j can be found as follows: if j is extracted from the formula $k=\frac{i(i-1)}{2}+j$, then $j=k-\frac{i(i-1)}{2}$ is obtained.

### *2.4 The representation of modified triangular matrix data structure as a vector and transformation formulae*

If the definition of a triangular matrix is modified as if it does not include the main diagonal, then the transformation formulae should be updated as follows.

| | | *j* | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| *i* | 1 | | | | | |
| | 2 | 1 | | | | |
| | 3 | 2 | 3 | | | |
| | 4 | 4 | 5 | 6 | | |
| | 5 | 7 | 8 | 9 | 10 | |

*Figure 7. Modified triangular matrix*

In Figure 7, a new matrix is obtained by shifting down the matrix in Figure 4 to one row below. If the proposed method is applied, then the desired formulae are as follows.

The transformation formula from matrix data structure to vector is $k = \frac{(i-2)(i-1)}{2} + j$.

The transformation formulae from vector to matrix are;

$$i = \left\lceil \frac{\sqrt{1+8k}-1}{2} \right\rceil + 1 \text{ and } j = \begin{cases} 1, & \text{if } k = 1 \text{ or } k = 2; \\ 2, & \text{if } k = 3; \\ k\left( \bmod\left( \frac{(i-2)(i-1)}{2} \right) \right), & \text{if } k > 3. \end{cases}$$

In a simpler way that j can be found as follows: if j is extracted from the formula $k = \frac{(i-2)(i-1)}{2} + j$, then $j = k - \frac{(i-2)(i-1)}{2}$ is obtained.

## 3. Computational Experiments

Computational experiments are performed with the matrices which are generated with the use of jagged array by taking the dimension $n$ of the matrix in the interval $10000,11000,\ldots,20000$ to test the efficiency of the transformation formulae proposed in section 2.3 in practical. These tests are carried out with the accession to four different lower triangular matrices that are taken as a basis in the derivation of the formulae. The access order of the cells for each of the accession method is illustrated in the following figure.

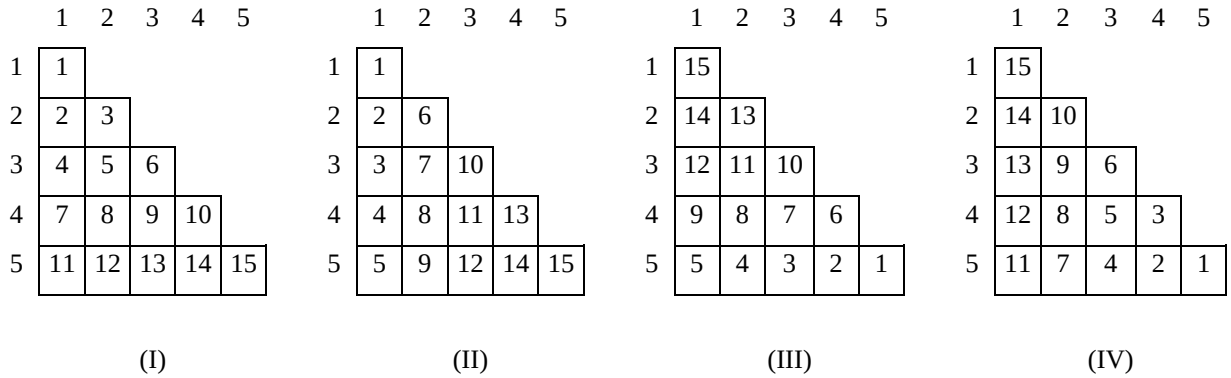


*Figure 8. Four different lower triangular matrices*

In the method of projection from matrix to vector ($m2v$), in the transformation formula $k = \frac{(i-1)i}{2} + j$ there are four different operations performed and these operations increases the accession time to cells. In order to avoid this adverse effect, the formula can be partitioned into $k = \frac{(i-1)i}{2}$ and $k = k + j$. Therefore, especially as it is in the accession types of I and III, when the row index $i$ is the outer loop and the column index $j$ is the inner index, the time loss is significantly reduced. On the other hand, since the operation division by 2 corresponds to the operation of shifting the dividend to the right for once, it is preferred in the formula and thus the results are equivalent to the times obtained by the jagged array approximation and sometimes even better results are obtained.

In the accession types of II and IV in which the method of projection from matrix to vector ($m2v$) is noticeably superior to the method of jagged array ($ja$), the column index $j$ is the outer loop and the row index $i$ is the inner loop. At first, this case is also disadvantageous for $m2v$, and this adverse case is seen when the times are examined within the following table. On the other hand, in the accession types of II and IV, pointer arithmetic should be applied frequently due to the nature of the method $ja$, the times are more than the times of method $m2v$.

*Table 1. CPU times*

| | **I ≈ III** | | **II ≈ IV** | |
|---|---|---|---|---|
| *n* | *m2v* | *ja* | *m2v* | *ja* |
| 10000 | 0,109 | 0,109 | 0,530 | 0,546 |
| 11000 | 0,140 | 0,140 | 0,639 | 0,655 |
| 12000 | 0,156 | 0,156 | 0,780 | 0,795 |
| 13000 | 0,187 | 0,187 | 0,920 | 0,952 |
| 14000 | 0,218 | 0,218 | 1,076 | 1,092 |
| 15000 | 0,249 | 0,265 | 1,232 | 1,263 |
| 16000 | 0,280 | 0,280 | 1,435 | 1,451 |
| 17000 | 0,327 | 0,327 | 1,606 | 1,638 |
| 18000 | 0,358 | 0,374 | 1,825 | 1,856 |
| 19000 | 0,405 | 0,405 | 2,012 | 2,059 |
| 20000 | 0,436 | 0,453 | 2,246 | 2,278 |

The results of the computational experiments show that the method *m2v* proposed in this paper is more efficient than the method *ja* . The times in Table 1 denote the *CPU* times in terms of second, and the source codes used in computational tests are available in *http://kisi.deu.edu.tr/murat.berberler/m2v/*.

## 4. Conclusion

The programmers generally use the standard data structures that a programming language allows instead of special cases. However, if it is required to use a data structure specific for the program and if the language does not allow to use, then the programs designed with standard data structures require much more space and time complexity. This is not acceptable since it yields inefficient codes. Therefore, by the use of mathematical approximations, space and time complexity can be reduced. In this paper, a general method is proposed for how to derive transformation formulae between matrix and vector data structures for the triangular matrix types that can be represented by vector, and mathematical analysis is performed.

## 5. References

[1] B. Kolman, David R. Hill, “Elementary Linear Algebra”, Prentice Hall (2000).
[2] R. Prather, “Discrete Mathematical Structures for Computer Science” Houghton Mifflin Company, Boston (1976).
[3] W. Kocay and D.L. Kreher, “Graphs, Algorithms and Optimization”, CRC Press Company, Florida (2005).
[4] Bakhadly, B.R., "Orthogonality Graph of The Algebra of Upper Triangular Matrices", Operators and Matrices, Vol. 11, Num. 2, 2017, pp. 455-463.
[5] Alvarado, F.L., Pothen, A., Schreiber, R., "Highly Parallel Sparse Triangular Solution", Graph Theory and Sparse Matrix Computation, 1993, pp. 141-157.
[6] Ju, T., Losasso, F., Schaefer, S., Warren, J., "Dual contouring of Hermite data", SIGGRAPH '02 Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002, pp. 339-346.
[7] Qiu, R.C., Antonik, P. "Smart Grid using Big Data Analytics", John Wiley & Sons Ltd., 2017.
[8] Lourenço, A., Fred, A.L.N., Jain, A.K., "On the Scalability of Evidence Accumulation Clustering",2010 International Conference on Pattern Recognition, 782-785.
[9] Gilbert, J.R., "Graphs and Sparse Matrices: There and Back Again", SIAM Combinatorial Scientific Computing, October 11, 2016.
[10] Silva, D.A.O., "Efficient Evidence Accumulation Clustering for large datasets/big data", Master Thesis, TecnicoLisboa, Electrical and Computer Engineering, 2015.
[11] Silva, D.A.O., Fred, A.L.N., "Efficient Evidence Accumulation Clustering for large datasets/big data", Proc INSTICC International Conf. on Pattern Recognition Applications and Methods - ICPRAM, Rome, Italy, Vol. 0, pp. 367 - 374, February, 2016.
[12] Liu, J., Liang, Y., Ansari, N., "Spark-based Large-scale Matrix Inversion for Big Data Processing", IEEE Access, Volume: 4, 1-10.
[13] Park, S.C., Draayer, J.P., "Fast sparse matrix multiplication", Computer Physics Communications, Volume 70, Issue 3, July 1992, Pages 557-568.
[14] Storjohann, A., "Computing Hermite and Smith normal forms of triangular integer matrices", Linear Algebra and its Applications, Vol. 282, Issue 1998, pp. 25-45.
[15] Mahajan, M., Jayalal Sarma M.N., "Rigidity of a simple extended lower triangular matrix", Information Processing Letters, Vol. 107, 2008, pp. 149–153.
[16] Alvarado, F.L., Pothen, A., Schreiber, R., "Optimal Parallel Solution of Sparse Triangular Systems", RIACS Tehnical Report 90.36, September, 1990.

[17] Yang, B., Liu, H., Chen, Z., "Accelerating Linear Solvers for Reservoir Simulation on GPU Workstations", 24th High Performance Computing Symposium, Volume 48, Number 4, pp. 1-8, California, USA, 3 - 6 April 2016.
[18] Liu, W., Li, A., Hogg, J., Duff, I.S., Vinter, B., "A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves", Proceedings of the 22nd International Conference on Euro-Par 2016: Parallel Processing , Volume 9833, pp. 617-630.