Paper

Application of High-Performance Techniques for Solving Linear Systems of Algebraic Equations

Daniel Grzonka

Institute of Computer Science, Faculty of Physics, Mathematics and Computer Science,
Tadeusz Kościuszko Cracow University of Technology, Cracow, Poland

Abstract—Solving many problems in mechanics, engineering, medicine and other (e.g., diffusion tensor magnetic resonance imaging or finite element modeling) requires the efficient solving of algebraic equations. In many cases, such systems are very complex with a large number of linear equations, which are symmetric positive-defined (SPD). This paper is focused on improving the computational efficiency of the solvers dedicated for the linear systems based on incomplete and noisy SPD matrices by using preconditioning technique – Incomplete Cholesky Factorization, and modern set of processor instructions – Advanced Vector Extension. Application of these techniques allows to fairly reduce the computational time, number of iterations of conventional algorithms and improve the speed of calculation.

Keywords—Advanced Vector Extension, conjugate gradient method, incomplete Cholesky factorization, preconditioning, vector registers.

1. Introduction

Solving linear systems of algebraic equations is a problem of linear algebra, which is common in many fields of science. The basic techniques and methodologies investigates for solving such a problem can be classified into two main categories, namely direct and iterative linear system solvers. Direct methods need computationally efficient resources (e.g., large RAM memory and fast CPU), which results in an inability to achieve a proper solution in a reasonable time [1]. For most types of the engineering problems modeled by the linear systems such methodologies are able to generate the exact ideal solutions.
Iterative methods are based on approximation of the exact solutions. In each iteration, the best achieved partial results may be improved by the implemented local optimizers. Final solution vector is generated as a results of the execution of the specified maximal number of iterations of the basic algorithm or in the case of achievement of the declared accuracy. Based on the formal definitions and analysis presented in [1], the efficiency of the iterative methods comes from their main features:

– the ability to solve larger problems, especially in three-dimensions,

– the development of highly effective preconditioners can enormously improve the speed and robustness of the iterative procedures,

– the ability to solve relatively large-scale problems in mini- and microcomputers,

– the possibility to vector and parallel programming.

The main factor affecting the performance of iterative methods is the number of the equation system expressed in the matrix, which may change frequently during the calculation. The problems presented in this paper are connected with solving ill-conditioned systems. The experimental part implements one of the best known and most efficient method for solving systems of linear algebraic equations – Incomplete Cholesky Conjugate Gradient (ICCG) method. Application of preconditions leads to reduction of the number of iterations. As a result the expected accuracy and simultaneously in reasonable time is obtained.
The proposed novel technology is based on operations on vector registers, to reduce the calculation time. To achieve this, a vector processing of multiple data sets procedure, namely Single Instruction, Multiple Data (SIMD) and Advanced Vector Extension (AVX) instructions implementation of the algorithms are applied.
This paper is organized as follows. In Sections 2 and 3 the conjugate gradient method and the preconditioning of the matrix are defined. Cholesky factorization is defined in Section 4. The methods of the utilization of processor registers and cache memory are presented in Section 5. AVX and optimization methods are characterized in Sections 6 and 7, and all the proposed techniques are evaluated experimentally in Section 8. The paper ends with short conclusions.

2. Conjugate Gradient Method

One of the most popular and effective method of solving the systems of equations is the Conjugate Gradient (CG) method, introduced by Hestenes and Stiefel in year 1952 [2]. The original model was then modified as iterative method for solving large systems of linear algebraic equations [1]. CG is a type of the Krylov subspace methods and usually it is applied to a system of equations defined by using the following matrix equation:

$$Ax = b, \tag{1}$$

where A is an $n \times n$ symmetric matrix of positive real numbers. The iterations number of CG should not ex-

ceed n without the round-off error. In practice, the number of iterations decreases depending on the specified accuracy level [2].
Following the formal definitions presented in [2] the general optimization problem for GC can be specified in the following theorem:

Theorem 1. If A is symmetric and positive definite, then the problem of solving $Ax = b$ is equivalent to minimizing of the quadratic form

$$q(x) := \frac{1}{2}x^T Ax - x^T b. \tag{2}$$

The idea of CG method is to generate a new vector x_{i+1} based on the best partial solution so far, which is defined as the vector x_i. This vector is characterized by two parameters, namely direction p_i and distance α_i, that is

$$x_{i+1} = x_i + \alpha_i p_i. \tag{3}$$

The coordinates of the *search directions vector* $p = [p_1, \ldots, p_i, p_{i+1}]$ are conjugate with respect to A. It can be defined as $p_{i+1} = r_{i+1} + \beta_i p_i$, where $r_{i+1} = b - Ax_{i+1}$ [1]. The values of the parameter α are estimated by using the Eqs. (2) and (3)

$$q(\alpha_i) = \frac{1}{2}(x_i + \alpha_i p_i)^T A(x_i + \alpha_i p_i) - (x_i + \alpha_i p_i)^T b. \tag{4}$$

Next, the partial derivative of α is computed:

$$\alpha_i = \frac{p_i^T r_i}{p_i^T A p_i}. \tag{5}$$

Now an approximation x_{i+1} from Eq. (3) and computation residual vector $r_{i+1} = r_i - \alpha_i A p_i$ is possible.
The next step is the calculation of a conjugation of directions:

$$p_{i+1} = r_{i+1} + \beta_i p_i, \tag{6}$$

where $\beta_i = \frac{r_{r+1}^T r_{r+1}}{r_i^T r_i}$.

Based on the above analysis, the CG algorithm can be defined by using Algorithm 1 [3].

Algorithm 1 Conjugate gradient method

Choose the initial approximation x_0 (e.g. 0)
$r_0 = p_0 = b - Ax_0$
For $i = 0, 1, \ldots, n-2$
$\alpha_i = \frac{p_i^T r_i}{p_i^T A p_i}$
$x_{i+1} = x_i + \alpha_i p_i$
$r_{i+1} = r_i - \alpha_i A p_i$
If the stop case is true – break
$\beta_i = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$
$p_{i+1} = r_{i+1} + \beta_i p_i$
End for

3. Preconditioning

The iterative methods are less demanded on computer resources than the direct methods, but unfortunately their accuracy is usually much worse. They cannot be successfully applied for some classes of the global optimization problems with many local solutions, where the iterative methods can be trapped in local optimum. One of the possible approach is the use of preconditioner, which is not always sufficient to achieve a convergence to the global solution in a reasonable time [4].
It allows to convert the matrix A from Eq. (1) to improve the distribution of its eigenvalues and reduce the condition number. It has a direct impact on the iterative methods convergence. Therefore, the matrix A preconditioning may be the key to an effective iterative method for solving systems of equations [5].
If a small change in the input causes a large change in the output, the problem is ill-conditioned, otherwise it is well-conditioned. In case of solving systems of linear algebraic equations problems, the lower condition number, the better conditioning task. In this context, the condition number of the matrix A is defined as [2]:

$$\kappa(A) = \|A\|_2 \cdot \|A^{-1}\|_2. \tag{7}$$

To define the preconditioning algorithm first step is to reduce the condition number, and in the same time to reduce the number of iterations of the CG algorithm. To achieve this, the transformation of the linear system in Eq. (1) into the following one is provided:

$$M^{-1}Ax = M^{-1}b, \tag{8}$$

where M is a symmetric matrix of rational positive numbers.

Algorithm 2 Preconditioned conjugate gradient method

Choose the initial approximation x_0 (e.g. 0)
$r_0 = b - Ax_0$
$Mz_0 = r_0 \rightarrow z_0$
$p_0 = z_0$
For $i = 0, 1, \ldots, n-2$
$\alpha_i = \frac{z_i^T r_i}{p_i^T A p_i}$
$x_{i+1} = x_i + \alpha_i A p_i$
$r_{i+1} = r_i - \alpha_i A p_i$
If the stop case is true – break
$Mz_{i+1} = r_{i+1} \rightarrow z_{i+1}$
$\beta_i = \frac{z_{i+1}^T r_{i+1}}{z_i^T r_i}$
$p_{i+1} = z_{i+1} + \beta_i p_i$
End for

It is also assumed that M is well-conditioned, which means that $\kappa(M^{-1}A) \ll \kappa(A)$, where κ is a condition number of a matrix. The system $Mx = b$ is much simpler to solve compare with Eq. (1) [6]. However, the crucial issue here is to generate appropriate M – *preconditioned matrix*. The closer the M matrix to the original matrix A, the convergence of the method is better.
The modified CG method is called *a Preconditioned Conjugate Gradient* (PCG) method and it is defined in Algorithm 2 [3].

4. Incomplete Cholesky Factorization

In many mathematical tasks matrix is as a product of a number of other matrices. Solving linear systems of algebraic equations is the most important problem of these areas. In such cases the Lower Upper (LU) decomposition is useful.
LU is the decomposition of the matrix A as a product of the lower triangular matrix L and the upper triangular U : $A = LU$. The solution of $Ax = b$ systems reduced to two steps: solving of the $Lz = b$ with a respect to z, and solving of the $Ux = z$ with respect to x.
A special case of LU decomposition is when $U = L^T$. It is Cholesky factorization (decomposition), which can be formally defined as the distribution matrix for factors such LL^T.

Theorem 2. If A is real, symmetric and positive definite matrix, then it has a unique factorization, $A = LL^T$, in which L is lower triangular with a positive diagonal [2].

Cholesky factorization procedure is defined in Algorithm 3 [7].
Generation of the preconditioned matrix does not entangle complete factorization. The Cholesky factorization involves the solution of $Ax = b$ system, so in this case the incomplete Cholesky is used. This method returns the matrix close to A, a similar structured and characterized by lower expenditure of computing for solving the system of equations [8].

Algorithm 3 Cholesky factorization

For $k = 1, \ldots, n$
$l_{kk} = \sqrt{a_{kk}}$
  For $i = k+1, \ldots, n$
  $l_{ik} = \dfrac{a_{ik}}{l_{kk}}$
  End for i
  For $j = k+1, \ldots, n$
    For $i = j, \ldots, n$
      $a_{ij} = a_{ij} - l_{ik} l_{jk}$
    End for i
  End for j
End for k

Incomplete Cholesky Factorization (ICF) is one of the most important preconditioning strategy. This paper presents a variant of the ICF by position, as shown in Algorithm 4 [7].

Algorithm 4 Incomplete Cholesky factorization

For $k = 1, \ldots, n$
$l_{kk} = \sqrt{a_{kk}}$
  For $i = k+1, \ldots, n$
    If $a_{ik} \neq 0$
    $l_{ik} = \dfrac{a_{ik}}{l_{kk}}$
    End if
  End for i
  For $j = k+1, \ldots, n$
    For $i = j, \ldots, n$
      If $a_{ij} \neq 0$
        $a_{ij} = a_{ij} - l_{ik} - l_{jk}$
      End if
    End for i
  End for j
End for k

It should be noted that the Algorithm 4 is not always stable. As long as the M matrix is positive definite, it can be decomposed in to LL^T, it means that $M = LL^T$. In some cases, during the decomposition process, the matrix can be no longer positive definite. However, there is a solution which preserves the positive definiteness matrix during factorization process.

Algorithm 5 Stabilization of the incomplete Cholesky factorization

Start factorization with $\gamma = 0$
If during factorization process $a_{kk} < 0$
Return to initial state A
If $\gamma \leq 0$
    $\gamma = 10^{-20}$
Else: $\gamma = \gamma \cdot 10$
    End if
Correction matrix $A = D + S \cdot \dfrac{1}{1+\gamma}$,
    where: D – diagonal matrix, S – other elements
Restarting the factorization process
End if

Algorithm 5 solves the stability problem of Algorithm 4 by introducing a correction factor γ, which initially is equal 10^{-20}. In the case when the diagonal element (the second line) will be negative, calculation is interrupted, matrix returned to the initial state and all elements (except

the diagonal) are multiplied by the value of $\frac{1}{1+\gamma}$. This procedure is repeated until the matrix is positive definite [9].

The application of Algorithm 5 allows to increasing the diagonal dominance of matrix A and is one of the possibilities to stabilize the factorization process.

PCG method (Algorithm 2) requires the solution of the $Mz_{i+1} = r_{i+1}$ equation. In this case, lower triangular matrix L for forward/backward substitution method could be used.

5. Processor Registers and Cache Memory

Typical computer processor CPU (Central Processing Unit) is composed of the Execution Unit (EU), and the Control Unit (CU) main modules.

The processor does not perform operations directly on the main memory, which is time-consuming. It has a number of small, high-speed memories, called registers. They are located in the EU and are used to temporarily storage of the results and control data. Number of available registers depends on the processor architecture. The internal memory of processor benefits from fast reading and writing.

Generally, memory stores data and programs. There are various types differ in cost and performance. The most important parameter is the access time (shorter access time increases cost). Therefore a hierarchy of memory was built. The highest levels of the memory are the fastest ones, but also the most expensive and smallest. The lower ones are slower, but larger and cheaper. Figure 1 presents computer memory hierarchy.

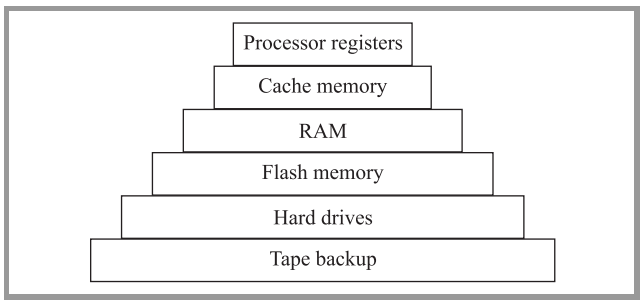

Fig. 1. Computer memory hierarchy.

Processor registers are located in the at the highest memory level. This is a static memory, and it is cleaned up in the idle mode of the computer. It has a very small capacity, e.g., 16, 32, 64, 128-bits, or 256 bits if CPU supports AVX instructions; access time is a fraction of a nanosecond. This is the fastest memory in the computer system [10].

Second in hierarchy is cache memory, which usually is a two or three-level (L1, L2 and L3) static memory with short access time. It is used to store a small amount of data, which are mostly used by the processor. Depending on the logical processor architecture, each level consists of the blocks (lines) in size 32, 64 or 128 bytes. The data between main memory and cache memory are transferred by same size blocks. Memory of the first level directly supports communication processor with main memory, while the lower-level memory (L2, L3) support the work of L1 cache. L2 and L3 – analogous to L1 memory – stores frequently used data in memory, and they are correspondingly larger. If the processor will not find the required data in L1, refers in the first instance to the L2 memory, and then – if it exists – to L3 memory. When the processor finds the requested data in the cache it is called the read hit, the opposite situation is read miss. Miss will reload the cache-line data. The new data is loaded, with completion of the cache line (up to the maximum) – because the tasks frequently cooperate with neighboring data [10], [11].

6. Advanced Vector Extension

In the Section 5, much attention has been paid to memories, including the fastest ones – CPU registers. The one type of registers is vector register that store the data processed by the SIMD architecture.

SIMD architecture is defined as systems which are processed multiple data streams based on a single instruction. Currently SIMD architecture is also used in personal computers. Processors use the extended set of SIMD instructions, such as MMX (MultiMedia eXtension), SSE (Streaming SIMD Extensions) or Advanced Vector Extension [13].

AVX is an extension of SSE instruction set that allows floating point operations on vectors of numbers using a special 256-bits processor registers (two times larger than previously used in processors that support SSE instructions). The introduction of new technology has forced changes in the architecture. Added 16 new registers are identified as YMM0, ..., YMM15. YMM registers are completely independent. It should be noted that the AVX instructions require support from the operating system. Older operating systems such as Windows XP or Windows Vista, even if the processor supports AVX instructions make impossible to use them [12], [13], [15].

AVX and previous technologies define two types of operations: packed and scalar. Scalar operations are present only on the least significant element of the vector (bits 0–63), while parallel operations on all elements of the vector in a single clock cycle [12]. The idea of operations on vectors is presented in Figure 2.

AVX has provided several new instructions, and now includes [13]:

- 19 instructions executable only on YMM registers,
- 12 multiply-accumulate instructions (MAC),
- 6 instructions support AES encryption,
- 88 instructions from the SSE instruction set, which may perform operations on vectors of floating point numbers stored in XMM/YMM registers,
- 166 instructions for 3- and 4-arguments operations.

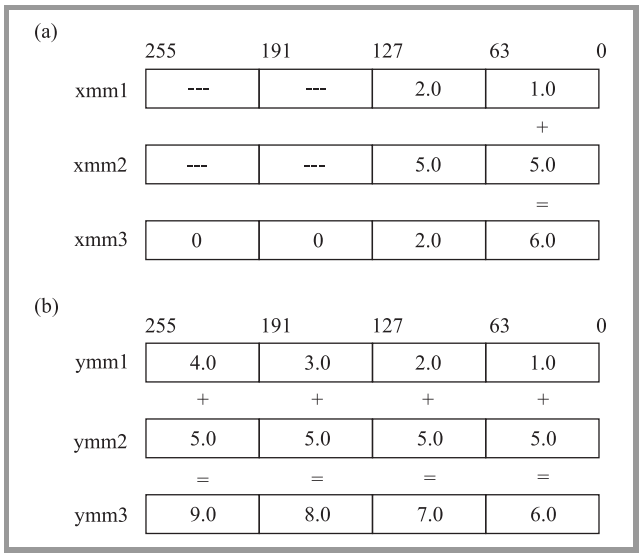

Fig. 2. Example of: (a) scalar and (b) packed multiplication.

A computer running in 32-bit mode has access to the first eight registers, in 64-bit mode to 16 registers. Due to the doubling of the size of registers, new data types are available:

- vector of eight single-precision floating-point numbers,
- vector of four double-precision floating-point numbers.

Most AVX instructions have their counterparts in special functions and data types used in C, C++, and Fortran programming languages. Using the appropriate functions and data types in C/C++ there is need to include library *immintrin.h* and compiler instruction: */arch: AVX* [13], [15].

7. Optimization Techniques

Loop unrolling is the first of the optimization techniques used in the implementation of ICF, CG and PCG methods. It allows reducing the number of hops by replicating code from loop body. Unrolled loop structure is closer to a more linear code and allows better use of the processor execution unit [14]. In implemented examples functions loops have been unrolled 8-times. This number was chosen because of the L1 cache-line size. Cache-line size of the computer where the experiment was performed is 64 bytes, while the size of one of a double is 8 bytes – thus in a cache-line fit in 8 double words.

The cache-line size is closely related to the second of the methods of optimization – data prefetching. It is realized by *void _mm_prefetch*(*char * p , int i*) function that loads a data block of size equal to the cache-line size [15]. The following example uses the prefetch function in combined with loop unrolling:

_mm_prefetch ((*const char* *) (&*vector*1[*i*+8]), *_MM_HINT_T0*).

The data are loaded from the shift of eight indexes of the double array to all levels of the cache. For single-precision data, shift will equal 16 indexes.

The application of data prefetching allows to hide the memory latency between sending and receiving a request for access to the memory. Processor must wait for data only in the first iteration of the loop [11].

The last of the optimization techniques are operations on registers (using AVX instructions). The introduction of operations on XMM/YMM registers forced to develop new types of data. In this paper two types of vector: _m256 and _m256d storing 8 float numbers and 4 numbers of double type respectively were used. Instructions for loading data into the vector registers (_mm256_load_ps/_mm256_load_pd), and unloading into RAM (_mm256_store_ps/_mm256_store_pd) require alignment of data within 32 bytes. Memory for all arrays is dynamically allocated and aligned by the function void * _aligned_malloc (size_t size, size_t alignment), where the first argument specifies the size of the allocated memory, and the second – the alignment (for instructions AVX – 32 bytes). The memory is release after performing of void _aligned_free (void *memblock) function. Static arrays have also been declared with the relevant directive: _declspec (align(#)) [11], [15].

8. Experimental Analysis

The next part of this work was the application of preconditioning in implementation of conjugate gradient method (Algorithm 2). The preconditioning method is stable variant of Incomplete Cholesky Factorization by position (Algorithms 4 and 5) and for comparison Conjugate Gradient method without preconditioning (Algorithm 1). The application is written in native C++ language.

The program consists of 16 functions, including: ICF and CG method in two versions: with and without preconditioning.

Solver has been tested on a server equipped with 16-cores, 64-bits AMD Opteron 6276 2.3 GHz processor based on Bulldozer microarchitecture. Opteron 6276 has three levels of cache, and the L1 memory is divided into data-cache (16 Kbytes) and instruction-cache (64 Kbytes). In the first-level cache is space for up 256 cache-lines 64 bytes each. Opteron 6200 series processors support MMX, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1 + SSE4.2, SSE4a, AES, ABM, AVX, FMA4, XOP instructions. The server has 64 GB DDR3 ECC memory.

For the efficiency analysis of the proposed solution system of 512 linear algebraic equations expressed in the matrix which condition number is 2186 was used. Items are mostly floating-point numbers. The desired accuracy (set a priori) of the solution for each test case was set at value 10^{-6}.

First the impact of application of the ICF to obtain a solution of the system using the CG method was examined. For both, the Conjugate Gradient method with and without

preconditioning allow to obtain the correct result with the expected accuracy.
The CG method gives the result with the expected accuracy in 79 iterations. The PCG, which use a ICF as preconditioner, reaches a result over five times faster – within 14 iterations (Table 1). For both methods the initial vector of solutions is the zero vector.

Table 1
Number of required iterations for obtaining the correct solution with the expected accuracy

Method (512×512 system)	Number of required iterations
Conjugate Gradient	79 iterations
Preconditioned CG	14 iterations

Figure 3 illustrates the process of reducing the error value with successive iterations. One can observe the PCG method is faster convergent than the CG.

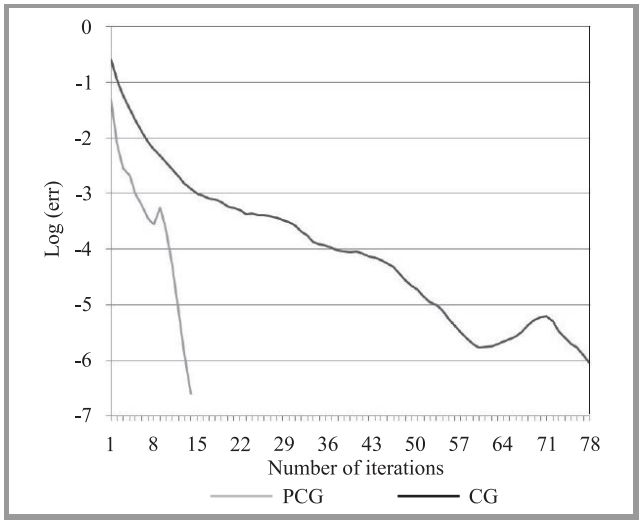

Fig. 3. Comparison of convergence of CG and PCG methods.

The second task of the experiment was to measure the computation time for the CG method with and without preconditioning in two options: standard and with using operations on YMM registers (AVX).
AVX instructions are used in: vector-matrix multiplication, scalar multiplication of vectors, calculating the Euclidean norm and the operation of the scheme: $\overline{z} = \overline{z} + (x \cdot \overline{y})$, $\overline{z} = \overline{z} - (x \cdot \overline{y})$ and $\overline{y} = \overline{z} + (x \cdot \overline{y})$.
For the PCG method additionally vector operations for the forward/backward substitution method was used, which solving systems of linear algebraic equations with triangular matrices.
Due to the small complexity of the task, calculations were repeated 500 times – in order to be able to observe changes in the time of obtaining solutions. All calculations were performed on the double precision numbers.

The results are shown in Table 2 and Figure 4. The system of equations was solved by CG over 8393 ms. Thanks to the vectorization computation task was solved three times faster compared to the solution without AVX instructions, which took 25569 ms.

Table 2
Times of obtaining solutions for all variants

Method (512×512 system)	Solution time	
	with AVX	without AVX
CG	8393 ms	25569 ms
PCG	4852 ms	14913 ms

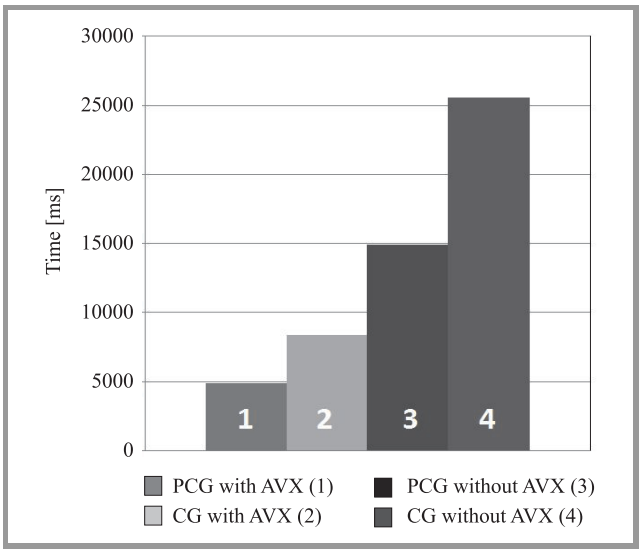

Fig. 4. Comparison of solving time for all variants.

For PCG method the expected results were received within 4852 ms for the AVX instructions and 14913 ms without.
Figure 4 illustrates the computational time differences between all solving methods.
The use of vector calculation and preconditioning technique resulted in the expected effect. Thanks to application of the AVX instructions and preconditioner, calculations were performed faster (81% less time). Even a small percentage increase in speed is important in solving large-scale complex mathematical problems.

9. Conclusions

The main aim of this research was to increase the speed of solving ill-conditioned systems of linear algebraic equations. These problems are characterized by slow convergence, and therefore, require many iterations to achieve the result with the expected accuracy. The author tried to solve this problem by using the CG, PCG and ICF methods dedicated for solving linear systems. The system matrix should be symmetric and positive-definite. The work focused on

two factors that have a significant impact on the speed of the implemented algorithms:

– the number of required iterations,

– the time-consuming operations like matrix-vector and vector-vector multiplication and forward/backward substitution method.

The obtained reduction of the number of iterations enables to increase speed of convergence. The main factor affecting the number of iterations is the condition number. In order to minimize this parameter, the ICF procedure was implemented. This variant keeps the stability of the algorithm during the decomposition process at the fair level.
In simple experimental analysis, a system of 512 equations was defined. By using Incomplete Cholesky Decomposition the number of iterations decreased more than five times without having a negative impact on the result accuracy. It is worth to note that the ICF method with its variants is widely used in various fields to solve technical issues. In practice there are no specified the universal methodology to choose the best method of preconditioning. In the other words it is impossible to determine which form of preconditioning would be best for each problem [16].
Another issue is a performance of matrix-vector operations and forward/backward substitution. They are the most time-consuming steps of obtaining solutions of equation systems. In order to accelerate this operation three closely related methods are used: loops unrolling, data prefetching, vector operations.
The loops have been unrolled hence during one iteration data required for the next one can be loaded. While information is loading, operations are parallel performed. By using these two techniques the waiting time between a requesting and receiving access for data was eliminated.
The last method use innovative technology of modern processors – 256-bits vector registers YMM. During one clock cycle are performed parallel operations on all elements of the vector.
Application of operations on the most expensive and also on the fastest computer memory, allows to obtain significant acceleration of the calculation. Reduction of the solving time of conjugate gradient method by about 67% was observed. This is due to application high-performance techniques in the application.
Application of the preconditioning technique and AVX instruction allows to solve the problem more than five times faster and effectively reduce the number of iterations.
Despite the results, it should be mentioned that not all possibilities of increasing the efficiency have been used in this study. There are many other techniques, both related to the preconditioning methods and the use of modern IT solutions, e.g., implement specific algorithms for sparse matrices, which are increasing the efficiency. It is also possible to use ready-made high-performance libraries such as BLAS, LAPACK, MKL. It is worth to consider application of adaptation solutions for multi-threaded and distributed computing, or expand the use of AVX instructions.

References

[1] M. Papadrakakis, "Solving large-scale linear problems in solid and structural mechanics", in *Solving Large-Scale Problems in Mechanics*, M. Papadrakakis, Ed. Oxford, UK: Wiley, 1993.

[2] D. Kincaid, W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing.* St. Paul, USA: Books Cole Publ., 1991.

[3] C. T. Kelly, *Iterative Methods for Linear and Nonlinear Equations*. Philadephia: SIAM, 1995.

[4] M. Benzi, "Preconditioning techniques for large linear systems: a survey", *J. Comput. Physi.*, vol. 182, pp. 418–477, 2002.

[5] H. Song, "Preconditioning techniques analysis for CG method", ECS 231 Large-Scale Scientific Computation Course, College of Engineering, University of California, Davis, 2013.

[6] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadephia: SIAM, 1997.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore: JHU Press, 1996.

[8] A. Trykozko, "Metoda elementu skończonego – programowanie" ("Finite element method – programming") – lectures, University of Warsaw, 2007 (in Polish) [Online]. Available: http://www.icm.edu.pl/∼aniat/fem/

[9] M. Suarjana and K. H. Law, "A robust incomplete factorization based on value and space constraints", *Int. J. for Numer. Methods in Engin.*, vol. 38, pp. 1703–1719, 1995.

[10] A. Piątkowska, R. Liszewski, G. Orzech, and M. Białecki, "Systemy komputerowe" ("Computer systems") (in Polish) [Online]. Available: http://cygnus.tele.pw.edu.pl/olek/doc/syko/www/

[11] S. Fialko, "Modelowanie zagadnień technicznych" ("Modeling of technical issues"), Politechnika Krakowska, 2011 (in Polish) [Online]. Available: http://torus.uck.pk.edu.pl/∼fialko/text/MZT1/MZT.pdf

[12] S. H. Ahn, "Streaming SIMD Extentions" [Online]. Available: http://www.songho.ca/misc/sse/sse.html

[13] "Intel Developer Zone" [Online]. Available: http://software.intel.com/

[14] "Opracowanie programów nauczania na odległość na kierunku studiów wyższych – informatyka" ("Study programme for a degree in computer science" (in Polish) [Online]. Available: http://wazniak.mimuw.edu.pl/

[15] "Microsoft Developer Network" [Online]. Available: http://msdn.microsoft.com/

[16] S. Fialko, "Iterative methods for solving large-scale problems of structural mechanics using multi-core computers" , Archives of Civil and Mechanical Engineering (ACME) (to be published) [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1644966513000666/

Daniel Grzonka received his B.Sc. and M.Sc. degrees in Computer Science at Cracow University of Technology, Poland, in 2012 and 2013, respectively. Actually, he is Ph.D. student at Jagiellonian University and a member of Polish Information Processing Society.

E-mail: grzonka.daniel@gmail.com
Institute of Computer Science
Faculty of Physics, Mathematics and Computer Science
Cracow University of Technology
Warszawska st 24
31-155 Cracow, Poland