ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Joanna Gościk[1], Józef Gościk[2]

NUMERICAL EFFICIENCY OF ITERATIVE SOLVERS FOR THE POISSON EQUATION USING COMPUTER CLUSTER

Abstract: We present a set of numerical results which were obtained by systematic investigation of efficiency of compilers implemented on Mordor cluster (*http://mordor.wi.pb.edu.pl*) running Linux distribution CentOS 4, kernel ver. 2.6. As a generic problem the finite difference based framework for solution of the Poisson equation has been taken (with discretization on grid topologically equivalent to a Cartesian grid). The PDE converted to an algebraic system of equations is solved by adopting so-called nonstationary, Krylov type, iterative methods: conjugate gradient (CG), bi-conjugate gradient (Bi-CG), conjugate gradient squared (CGS) and bi-conjugate gradient stabilized (Bi-CGSTAB). The code was implemented using two different compilers, such as *gcc* (GNU Compiler Collection - ver. 3.4.6) and *icc* (Intel C++ Compiler - ver. 9.1). All performances reported were done with the Xeon 3.2 GHz processor that has own memory 2 GB.

Keywords: Iterative solvers, Finite difference method, Poisson equation

1. Introduction

The need to solve systems of linear algebraic equations is ubiquitous through computational physics. Such systems typically arise from the discretization of partial differential equations and for practically important, real engineering problems are very large. The demands required for the computational resources cause that often so-called direct methods are not applicable and the iterative techniques are one reasonable alternative for solution such sets of linear equations [1]. It seems that this dynamically growing application area is the main reason why the iterative techniques are so intensive investigated last years. Another well known motivation is impact of parallel architectures. Direct methods are more complex to implement in parallel than are iterative methods.

[1] Faculty of Computer Science, Bialystok Technical University, Bialystok, Poland
[2] Faculty of Mechanical Engineering, Bialystok Technical University, Bialystok, Poland

Implementation of the iterative techniques however is still not so obvious despite of the huge progress in the last years. In another words there is no preferred or the best method preferable among others. From the other side it is very well known that even the system is well conditioned (even treated by proper pre-conditioning technique) often converges slowly or even diverges [2], [3].

The paper is devoted to illuminate an early stage of progress in developing own made library of the iterative solvers implemented on cluster computer - Mordor cluster (*http://mordor.wi.pb.edu.pl*). Mainly for the comparisons reasons the reported results were obtained by sequentially executed (using one procesor) algorithms. The plan of the paper is as follows. First, we define the Poisson equation and characterize resulting from discretization by finite difference method sets of linear equations. Next, we describe the group of the interested iterative techniques which belongs to the Krylov subspace methods. In particular, we describe short characteristic of the methods, their algorithmic realization schemes and show how they are related each other. Taxonomy and especially historical chronology are given here basing on excellent review paper by Saad and Van der Vorst [4]. Finally we present efficiency of the investigated methods in terms of the CPU time consuming and rate of convergence.

2. The Poisson equation and its discretization

The Poisson equation is very well known in computational physics. In fact the equation play very important role in many branches of the scientific computing and numerical simulation. Close to the physical interpretation it describes fluid dynamic (in a stream function − vorticity formulation), heat conduction and many others. It is also a kernel for generation of structured meshes in arbitrary domains (so called body fitted mesh generation techniques) and more sophisticated area as in medical imaging where is used for electro-encephalographic source analysis. The latest new branch are requirements of the current generation of the GPU (Graphical Processor Unit) in which range of application extends traditional graphic problems to capability of more general computing needed for example for physical modelling.

In the paper as a generic problem we consider the two dimensional Poisson equation with Dirichlet boundary conditions on the rectangular domains $\Omega = \left[0, L_x; 0, L_y\right]$ (in particular on the unit square $\Omega = [0,1;0,1]$)

$$\nabla^2 \phi(x) + q(x) = 0 \quad x \in \Omega \subset \mathbb{R}^2 \tag{1}$$

with boundary conditions

$$\phi(x) = \phi_D(x) \quad x \in \partial\Omega. \tag{2}$$

We next assume that the boundary value problem (1) and (2) is solved by means of finite differences. This transforms differential equation to its finite difference form. So, after discretization, we have defined difference problem

$$L_h \cdot \phi_h = b_h \quad , \tag{3}$$

where b_h is the grid function that is a projection of the right hand side of the original differential problem on the grid and ϕ_h is the grid function which is a projection of the exact solution on the grid. The operator L_h is determined from the grid functions and depends in general on parameters called grid steps h_x, h_y. In particular we assume that L_h is linear operator acting on the regular grid 2D grid ϕ_h (it is assumed that the grid cover the whole interested region Ω uniformly, so $h_x = h_y \rightarrow h$) and is described by a set of stencils, with possibly varying entries each of size no longer than 3×3 (2×3 at the boundary and 2×2 at the corners).

Finally, the main subject of interest in the paper is a vectorial matrix form of a system of linear difference equations (3) which leads to a linear algebra problem

$$\mathbf{A}\phi = \mathbf{b} \quad , \tag{4}$$

where **A** and **b** are given and to be real, **A** is sparse, non-singular $N \times N$ matrix, **b** an N-vector (with assumption that N is large) and ϕ is the vector of the unknowns. Strictly we will try to find acceptable approximations of the Poisson equation solution by solve the equation (4). The most important is that in computational schemes the equation needs to be solved repeatedly for different source contributions. In consequence the solution of the Poisson equation is very often the most time consuming part of the overall computational scheme. Then naturally is still needing for working the most efficient solvers for this task.

3. Iterative solvers and methods

Each an iterative solver works by repeatedly apply a series of operations to an approximate solution to the linear system, with the error in the approximate solution being reduced by each application of the operations. The basic form of all iterative methods is given in Fig. 1. According to the pseudo code given in Fig. 1 an iterative scheme produces a sequence of vectors **x** which should converge to the vector **x** satisfying the system of equations (4). So apart of choosing of the most effective F function it is very important also to arrange a criterion which will decide when to stop creation a sequence of **x**. Detailed discussion about the subject can be found in [3]. In our work we decide to apply two stopping criteria which are defined in Section 4.3.

```
set x_0 to an initial estimate of x
j=0
while j < j_max and error > tolerance
  perform some operations x_{j+1} = F(x_j)
  j=j+1
```

Fig. 1. A generic iterative method.

Relating to the iterative schemes which produce successive approximation we restrict ourselves to F functions constructed on the base of the simplest Krylov subspace based iterative techniques. The methods implemented and tested in this study were Conjugate Gradient (CG), Bi-Conjugate Gradient (Bi-CG), Conjugate Gradient Squared (CGS) and Bi-Conjugate Gradient STABilised (Bi-CGSTAB).

It is also especially needed to pointed out that the algorithms for all of solver mentioned were consciously elaborated in the standard (unpreconditioned) representation.

3.1 The Conjugate Gradient (CG) method.

$r_0 \Leftarrow b - Ax_0$

$p_0 \Leftarrow r_0$

FOR $j = 0, 1, \ldots,$ until convergence DO

$\alpha_j \Leftarrow r_j^T r_j / (Ap_j)^T p_j$

$x_{j+1} \Leftarrow x_j + \alpha_j p_j$

$r_{j+1} \Leftarrow r_j - \alpha_j A p_j$

$\beta_j \Leftarrow r_{j+1}^T r_{j+1} / r_j^T r_j$

$p_{j+1} \Leftarrow r_{j+1} + \beta_j p_j$

Fig. 2. The standard (unpreconditioned) CG algorithm.

The CG method seems to be the most representative for a number of projection-type methods on Krylov subspaces. It is especially suited for symmetric positive definite matrices, for which it was originally devised in 1952 by Hestens and Stiefel [5]. However as its pointed out in [4] scarsely in the early 1970's it received the more attention in computational practice. CG is a descendant of the method of steepest

descent that avoids repeated search in the same direction by making search directions orthogonal to each other in the energy norm associated with the matrix.

For clarity of the pseudo codes description we use following, uniform notation throughout the paper: $\mathbf{x_j}$ denotes the vector $\mathbf{x}$ during the j-th iteration (consequently $\mathbf{x_0}$ is the initial guess), $\mathbf{r_j}$ is the residual which indicates how far the iterative sequence of $\mathbf{x}$ is from the solution and is defined as $\mathbf{r_j} = \mathbf{b} - \mathbf{A}\mathbf{x_j}$.

3.2 The Bi-Conjugate Gradient (Bi-CG) method.

$r_0 \Leftarrow b - Ax_0$
Choose r_0^* so that $r_0^T r_0^* \neq 0$
$p_0 \Leftarrow r_0$
$p_0^* \Leftarrow r_0^*$
FOR $j = 0, 1, \ldots,$ until convergence DO
  $\alpha_j \Leftarrow r_j^T r_j^* / (Ap_j)^T p_j^*$
  $x_{j+1} \Leftarrow x_j + \alpha_j p_j$
  $r_{j+1} \Leftarrow r_j - \alpha_j A p_j$
  $r_{j+1}^* \Leftarrow r_j^* - \alpha_j A^T p_j^*$
  $\beta_j \Leftarrow r_{j+1}^T r_{j+1}^* / r_j^T r_j^*$
  $p_{j+1} \Leftarrow r_{j+1} + \beta_j p_j$
  $p_{j+1}^* \Leftarrow r_{j+1}^* + \beta_j p_j^*$

Fig. 3. The standard (unpreconditioned) Bi-CG algorithm.

The pioneering idea of Bi-CG was fomulated in 1952 by [6], which used biothogonality relation to redure iteratively a matrix to the tri-diagonal form. Later his idea was adopted by [7] mainly to develop the algorithms which free the CG solver from its limitation of only being applicable to symmetric systems. For this the orthogonal sequence used in the CG method has been replaced by two mutually orthogonal sequences, one based on the system $\mathbf{A}$ and the other on its transpose $\mathbf{A^T}$. In result, implicitly the Bi-CG algorithm solves additionally a dual linear system $\mathbf{A^T x^* = b^*}$. Consequentely the symbols with the asterix which will appear next in the paper should be treated as a connected with the dual approximate solution.

3.3 The Conjugate Gradient Squared (CGS) method.

$\mathtt{r}_0 \Leftarrow \mathtt{b} - \mathtt{A}\mathtt{x}_0$
Choose $\mathtt{r}_0^*$ arbitrarily
$\mathtt{p}_0 \Leftarrow \mathtt{u}_0 \Leftarrow \mathtt{r}_0$
FOR $\mathtt{j} = 0, 1, \ldots,$ until convergence DO
$\alpha_j \Leftarrow \mathtt{r}_j^T \mathtt{r}_0^* / (\mathtt{A}\mathtt{p}_j)^T \mathtt{r}_0^*$
$\mathtt{q}_j \Leftarrow \mathtt{u}_j - \alpha_j \mathtt{A}\mathtt{p}_j$
$\mathtt{x}_{j+1} \Leftarrow \mathtt{x}_j + \alpha_j (\mathtt{u}_j + \mathtt{q}_j)$
$\mathtt{r}_{j+1} \Leftarrow \mathtt{r}_j - \alpha_j \mathtt{A}(\mathtt{u}_j + \mathtt{q}_j)$
$\beta_j \Leftarrow \mathtt{r}_{j+1}^T \mathtt{r}_0^* / \mathtt{r}_j^T \mathtt{r}_0^*$
$\mathtt{u}_{j+1} \Leftarrow \mathtt{r}_{j+1} + \beta_j \mathtt{q}_j$
$\mathtt{p}_{j+1} \Leftarrow \mathtt{u}_{j+1} + \beta_j (\mathtt{q}_j + \beta_j \mathtt{p}_j)$

Fig. 4. The standard (unpreconditioned) CGS algorithm.

The CGS method was proposed in 1989 by Sonneveld [8] and represents some idea of improvement of the Bi-CG solver. In computational realization it applies the updating operations for the **A** sequence and $\mathbf{A^T}$ sequence to both vectors. In concept, or theoretically this approach would double the convergence rate, but in practice convergence is not so ideal.

3.4 The Bi-Conjugate Gradient STABilized (Bi-CGSTAB) method.

The Bi-CGSTAB method was developed by Van der Vorst [9] as a hybrid of different conjugate gradient based methods (CGS, Bi-CG and not investigated in this paper GMRES method). In intention the method has been elaborated to solve general systems of equations (with **A** symmetric and non symmetric) and especially in order to avoid the often highly irregular convergence patterns of the CGS and Bi-CG.

$$
\begin{aligned}
&\mathtt{r}_0 \Leftarrow \mathtt{b} - \mathtt{A}\mathtt{x}_0 \\
&\text{Choose } \mathtt{r}_0^* \text{ arbitrarily} \\
&\mathtt{p}_0 \Leftarrow \mathtt{r}_0 \\
&\text{FOR } \mathtt{j} = 0, 1, \ldots, \text{until convergence DO} \\
&\quad \alpha_\mathtt{j} \Leftarrow \mathtt{r}_\mathtt{j}^\mathtt{T}\mathtt{r}_0^* / (\mathtt{A}\mathtt{p}_\mathtt{j})^\mathtt{T}\mathtt{r}_0^* \\
&\quad \mathtt{s}_\mathtt{j} \Leftarrow \mathtt{r}_\mathtt{j} - \alpha_\mathtt{j}\mathtt{A}\mathtt{p}_\mathtt{j} \\
&\quad \omega_\mathtt{j} \Leftarrow (\mathtt{A}\mathtt{s}_\mathtt{j})^\mathtt{T}\mathtt{s}_\mathtt{j} / (\mathtt{A}\mathtt{s}_\mathtt{j})^\mathtt{T}\mathtt{A}\mathtt{s}_\mathtt{j} \\
&\quad \mathtt{x}_{\mathtt{j}+1} \Leftarrow \mathtt{x}_\mathtt{j} + \alpha_\mathtt{j}\mathtt{p}_\mathtt{j} + \omega_\mathtt{j}\mathtt{s}_\mathtt{j} \\
&\quad \mathtt{r}_{\mathtt{j}+1} \Leftarrow \mathtt{s}_\mathtt{j} - \omega_\mathtt{j}\mathtt{A}\mathtt{s}_\mathtt{j} \\
&\quad \beta_\mathtt{j} \Leftarrow \left(\mathtt{r}_{\mathtt{j}+1}^\mathtt{T}\mathtt{r}_0^* / \mathtt{r}_\mathtt{j}^\mathtt{T}\mathtt{r}_0^*\right) \cdot (\alpha_\mathtt{j} / \omega_\mathtt{j}) \\
&\quad \mathtt{p}_{\mathtt{j}+1} \Leftarrow \mathtt{r}_{\mathtt{j}+1} + \beta_\mathtt{j}\left(\mathtt{p}_\mathtt{j} - \omega_\mathtt{j}\mathtt{A}\mathtt{p}_\mathtt{j}\right)
\end{aligned}
$$

Fig. 5. The standard (unpreconditioned) Bi-CGSTAB algorithm.

4. A comparison of solvers

All results reported in this section are done with the Xeon 3.2 GHz processor on Mordor cluster (*http://mordor.wi.pb.edu.pl*) running Linux distribution CentOS 4, kernel ver. 2.6. The codes were generated using two different compilers: gcc (GNU Compiler Collection - ver. 3.4.6) and icc (Intel C++ Compiler - ver. 9.1). For each solver we report on the elapse CPU time with a systematic investigation of the impact of

- the compiler optimization options,
- the grid size (nx,ny).

An attention is also directed on comparison of the linear solvers in terms of their convergence rate.

4.1 The solvers test case

To compare the convergence and overall behaviour of the iterative procedure of the tested solvers they were used to solve a two-dimensional Poisson problem with Dirichlet boundary conditions. The problem has been chosen taking into account two aspects, namely physical connections with the cases encountered in computational fluid dynamic (CFD) as well as relative simplicity of the geometry and boundary conditions which provide to case which have an analytical (exact) solution.

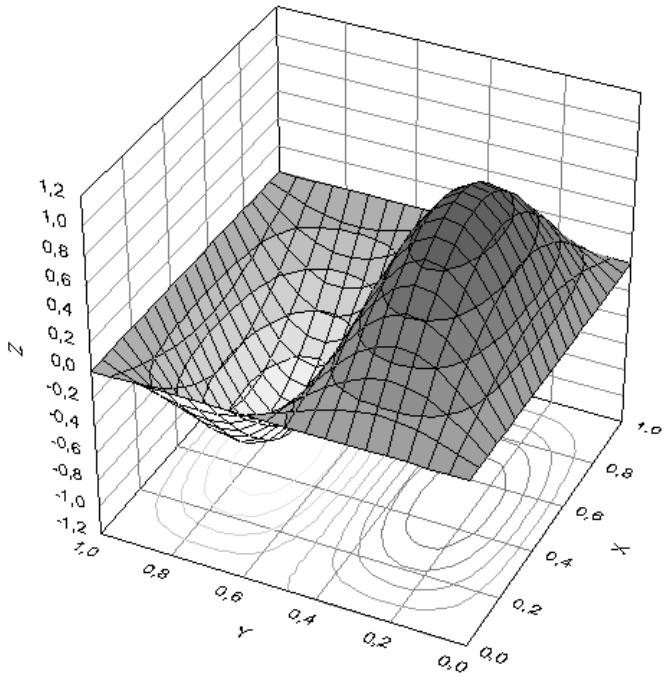

Fig. 6. Analytical solution.

The test case was a finite difference discretization of the Poisson equation applied to a rectangular domain ($0 \le x \le L_x$, $0 \le y \le L_y$) within is defined the two dimensional equation

$$\partial^2_{xx}\phi + \partial^2_{yy}\phi + q = 0 \tag{5}$$

where ϕ is treated as a continuous scalar field and q is a sinusoidally varying source term given by

$$q(x,y) = \left[\left(\frac{\pi}{L_x}\right)^2 + \left(\frac{2\pi}{L_y}\right)^2\right] \cdot \sin\left(\frac{\pi}{L_x}\cdot x\right) \cdot \sin\left(\frac{2\pi}{L_y}\cdot y\right) \tag{6}$$

For the Dirichlet problem which is defined imposing $\phi = 0$ on all boundaries, the test problem (5), (6) has the exact solution

$$\phi(x,y) = \sin\left(\frac{\pi}{L_x}\cdot x\right) \cdot \sin\left(\frac{2\pi}{L_y}\cdot y\right) \tag{7}$$

which is also additionaly presented in Fig. 6

4.2 Assessment of the CPU time according to compilers optimization options

Test of influence of the compiler optimization options affecting the code speed is presented in Tables 1 and 2. The details of the compiler features which enhance an application performance are described in [10] for *gcc* compiler and in [11] for *icc* compiler.

Table 1. gcc (version 3.4.6) performance.

Compiler options	Iterative solver			
	CG	BiCG	CGS	BiCGSTAB
-O0[1]	73.61	117.81	136.76	122.88
-O	44.47	71.06	82.54	76.58
-O1	44.81	71.13	82.36	76.54
-O1 -O	44.37	71.13	82.40	76.30
-O2	44.93	71.70	83.44	76.91
-O3	44.92	74.84	93.01	81.79
-Os	46.10	74.35	85.44	78.65

The elapsed CPU times given in Tables 1 and 2 are documented by solution of the biggest problem. Timings were made using the C `clock()` function which provide accuracy to 1/1000th of second. The test cases were run to fulfill stopping criteria (see eq. (8)).

Table 2. icc (version 9.1) performance.

Compiler options	Iterative solver			
	CG	BiCG	CGS	BiCGSTAB
-O1	35.95	59.41	71.01	58.95
-O2[1]	35.66	59.79	71.24	59.70
-O3	36.18	59.62	71.41	59.64
-fast	27.31	45.77	54.99	44.47
-O2 -ipo	36.48	60.78	71.73	60.63

Summing up the results we can formulate the following general conclusions. When using *gcc* practically we should not expect any increase in performance by

[1] default compiler option

proper choosing of optimizing compiler options. Only one recommendation should be formulated to no using *gcc* compiler without specification of any optimizing options. In other words that means that it should be omitted using the default compiler option [10].

Almost the same remark we can write characterizing results obtained by using *icc*. We found also that the execution time is relative more sensitive to the chosen compiler options. There is also one remarkable exception when the option `-fast` is used. By analyzing the times given in the Table 2 we can expect then a significant increase in performance of the code (the CPU time elapsed is shorter in a $20 \div 40\%$).

However the general conclusion is that the involving *icc* compiler generates codes which are considerably faster than those resulted from *gcc*.

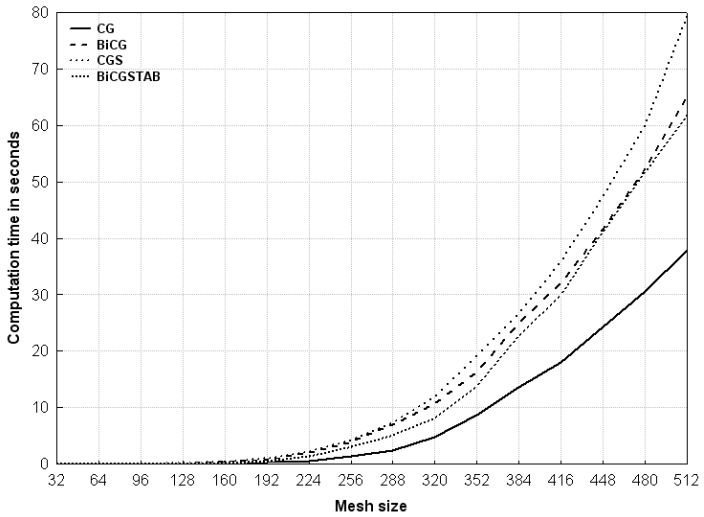

Fig. 7. Computation time.

Taking into account those observations for investigate the impact of the grid size on the CPU time, we performed a series of systematic calculation using *icc* with `-fast` option (which generate a fastest code). Results are given in Figure 7 in a form of variation of the solution time as the number of equations is increased. For the test problem at hand, a symmetric sparse linear system, the iterative methods we have tested confirm superior recommendation of the CG method for their solve. In that context is obvious that other methods took longer to solve the same problem.

4.3 Convergence of the solvers

A stopping criteria is based on the norm of the current residual related to the norm of the vector **b**, initial right hand side vector of the equation (4), and is defined as

$$\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} < \varepsilon \tag{8}$$

where ε is a tolerance chosen from the $\left\langle 10^{-6}, 10^{-1} \right\rangle$. It is worth to mention, that the calculation is also stopped, if the left hand side of the equation (8) does not become smaller than ε within the chosen maximum number of iterations. At our calculations we took that the maximum number of iterations permitted each algorithm to perform has been set as a 1000. With no exception for each run we took $\mathbf{x_0} = 1$ as an initial guess.

To compare the convergence rates of the four iterative solvers, runs were made using 5^2 and 512^2 meshes, with the residual at each iteration being printed out for plotting, the abscissa being as iteration number. Figures 8 and 9 show the convergence of the solvers for the 5^2 mesh and the 512^2 mesh respectively. As is clearly shown the solution on the mesh 5^2 fails the convergence due to small h. The very nice and representative for the Krylov space methods is Figure 9 containing convergence rate history for 512^2 mesh.

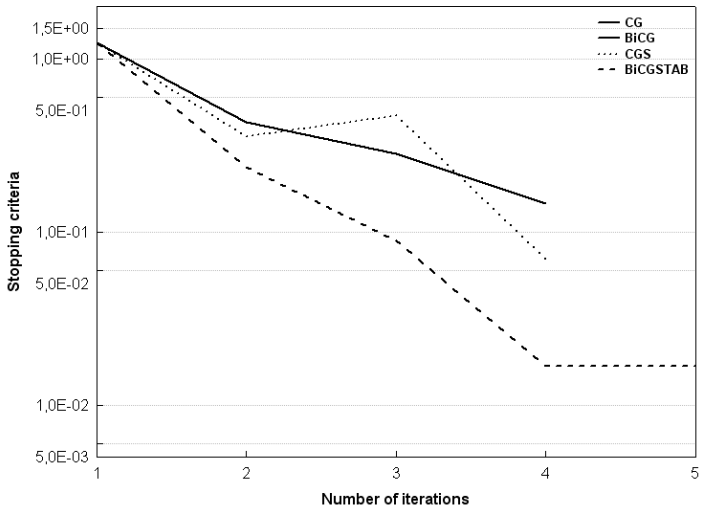

Fig. 8. Convergence plot for mesh 5^2.

First of all it exhibits an irregular convergence with typical [2], a non-monotonic reduction in residual. The CG solver shows an initial period approximately linear convergence, after which the rate of convergence increases. The Bi-CGSTAB solver has a more irregular rate of convergence than their CG or Bi-CG counterparts.

However in general the rate of convergence is highest among the four tested solvers. The worst of all rate exhibits the CGS method. In an initial period the CGS solver shows even some tendency to divergence. After this initial period, the rate of convergence increases and finally is near close those observed for CG and Bi-CG methods.

Finally we should pointed out that the poor in general behavior of the convergence rate of the Krylov type solvers once again confirms that they should be implemented with segments which guaranties a proper preconditioning.

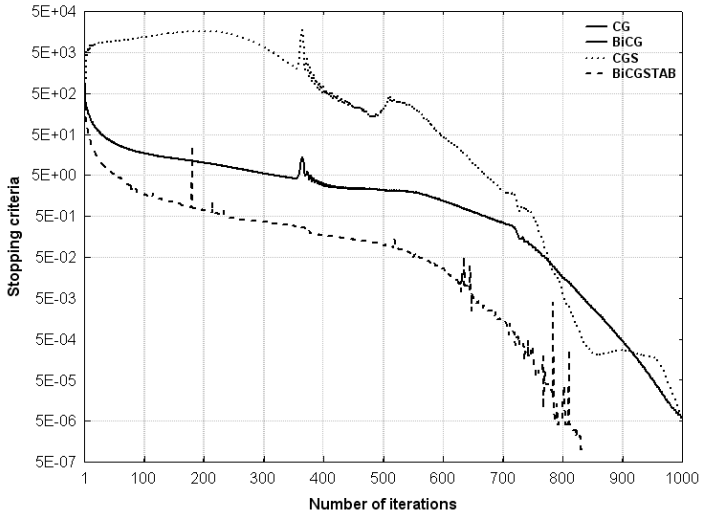

Fig. 9. Convergence plot for mesh 512^2.

5. Conclusion

The presented paper documents the first step in realization of the project leading in intention to develop numerically efficient software addressed to solution of the Poisson equation using parallel computer with distributed memory. Because results were obtained consequently in the sequential environment without any special tuning of

the source they will be helpful in reliable investigation of performance parallelisation. In the next step we will consider in details implementation analyzed solvers on Mordor cluster by appropriate construction of

- data structures for sparse matrices,
- data parallel algorithms for sparse matrix vector multiplies,
- reduction operators for inner product computation.

Bibliography

[1] Saad, Y.: Iterative Methods for Sparse Linear Systems, Second Edition, SIAM, Philadelphia, Pa, 2003.

[2] Van der Vorst, H.A.: The Iterative Krylov Methods for Large Linear Systems, Cambridge University Press, Cambridge, 2003.

[3] Barrett, R., Berry, M., Chan, T., Demmel, J., June, D., Dongarra, J., Eijkhout, V., Pozo, R., Romine, Ch., Van der Horst, H.: Templates for the solution of linear systems: Building blocks for iterative methods, Second Edition, SIAM Publication, 2003.

[4] Saad, Y., Van der Vorst, H.A.: Iterative solution of linear systems in the 20-th Century, Journal of Computational and Applied Mathematics, Vol. 123, No. 1-2, pp. 1-33, 2000.

[5] Hestens, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Stand, Vol. 49, No. 6, pp. 409-436, 1952.

[6] Lanczos, C.: Solution of systems of linear equations by minimized iterations, J. Res. Nat. Bur. Stand, Vol. 49, No. 1, pp. 33-53, 1952.

[7] Fletcher, R.: Conjugate gradient methods for indefinite systems, In Edited by G.A.Watson, Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974, Springer Verlag, New York, pp. 73-89, 1975.

[8] Sonneveld, P.: CGS: A fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput, Vol. 10, pp. 36-52, 1989.

[9] Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput, Vol. 13, pp. 631-644, 1992.

[10] Stallman, R.M., GCC Developer Community: Using the GNU Compiler Collection, GNU Press, 2004.

[11] Intel C++ Compiler Optimization Applications, Intel Corporation, 1996-2006.

EFEKTYWNOŚĆ NUMERYCZNA ITERACYJNYCH TECHNIK ROZWIĄZANIA RÓWNANIA POISSONA NA KLASTRZE KOMPUTEROWYM

Streszczenie: Przedstawiono wstępne wyniki badania efektywności sekwencyjnego przetwarzania danych w algorytmach rozwiązywania dużych układów równań liniowych na klastrze obliczeniowym Mordor (*http://mordor.wi.pb.edu.pl*) zarządzanym przez system operacyjny Linux (dystrybucja CentOS 4, wersja jądra 2.6). Szczególną uwagę zwrócono na wpływ doboru opcji optymalizacyjnych w dostępnych kompilatorach na wydajność obliczeniową kodu komputerowego. Jako bazowe do rozważań przyjęto duże układy równań liniowych z macierzą współczynników o strukturze rzadkiej. Takie układy równań generowane są w procedurze numerycznego rozwiązania równania Poissona, którego aproksymację otrzymuje się na gruncie metody różnic skończonych (dyskretyzacja na uporządkowanej siatce różnicowej w kartezjańskim układzie współżędnych prostokątnych). Cząstkowe równanie różniczkowe przekształcone do postaci układu równań liniowych rozwiązano z wykorzystaniem czterech metod iteracyjnych typu Kryłowa: gradientów sprzężonych (CG), gradientów bisprzężonych (Bi-CG), kwadratowego gradientu sprzężonego (CGS) oraz stabilizowaną metodą wzajemnie sprzężonych gradientów (Bi-CGSTAB). Metody te wdrożono generując własne oprogramowanie oraz zaimplementowano z wykorzystaniem dwóch różnych kompilatorów *gcc* (GNU Compiler Collection - wesja 3.4.6) oraz *icc* (Intel C++ Compiler - wersja 9.1). Wyniki wszystkich testów efektywności obliczeniowej uzyskano rozwiązując sformułowane zagadnienie testowe przy użyciu jednego procesora Xeon 3.2 Ghz wchodzącego w skład jednego węzła obliczeniowego z pamięcią własną 2GB.

Słowa kluczowe: Metody iteracyjne, Metoda różnic skończonych, równanie Poissona