ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Joanna Gościk[1], Józef Gościk[2]

NUMERICAL EFFICIENCY OF THE CONJUGATE GRADIENT ALGORITHM - SEQUENTIAL IMPLEMENTATION

Abstract: In the paper we report on a second stage of our efforts towards a library design for the solution of very large set of linear equations arising from the finite difference approximation of elliptic partial differential equations (PDE). Particularly a family of Krylov subspace iterative based methods (in the paper exemplified by the archetypical Krylov space method - Conjugate Gradient method) are considered. The first part of the paper describes in details implementation of iterative algorithms for solution of the Poisson equation which formulation has been extended to the three-dimensional. The second part of the paper is focused on the performance measurement of the most time-consuming computational kernels of iterative techniques executing basic linear algebra operations with sparse matrices. The validation of prepared codes as well as their computational efficiency have been examined by solution a set of test problems on two different computers.

Keywords: Iterative solvers, Finite difference method, Poisson equation, Performance of sequential code

1. Introduction

At this time, the search for high performance in a large scale computation is of growing importance in scientific computing. This is very important due to recent attention which has been focused on distributed memory architectures with particular interest in using small clusters of powerful workstations. However, the appearing of modern multiprocessors platforms demands to resolve a lot of tasks which are addressed to efficient porting solvers on three different architectures: a sequential machine, a shared memory machine and on a distributed memory machine. So the key problem is to understand how the computation aspects of using linear algebra solvers (in our case - iterative) can affect the optimal execution of code on machines with different architecture.

For understanding the implication of design choices for systems, and for studying increasingly complex hardware architecture and software systems very important

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

tools are more or less standardized benchmarks. However running for example very well known Linpack benchmark [1] on complex architectures is not so obvious and easy in interpretation. The problem was discovered when the execution time of Linpack was studied systematically on different machines (the results and main problems arise when we tried to estimate performance of the benchmark will be pulished elsewhere). This observation in our opinion should be interpreted by misunderstanding the impacts of different architecture designs. Among them are characteristic differences in latency between different levels of caches and memory, introduction of simultaneous multithreading and single-chip multiprocessors, to mention of few.

In the paper we try to explore some of the main challenges in implementation and estimation computational efficiency of the executing sequential code. The special attention has been paid on credible study of performance in terms of number of equations. The paper report on continuation of our research described in [2]. So, we were mainly interested in building and improvement of the software for solving very large set of linear equations arising from the finite difference approximation of the elliptic PDE. At the stage the iterative solver module has been rearranged in a such way which allows flexible operating on separate basic algebra operations as inner products of two vectors, updating of vectors and the matrix-vector multiplication.

This paper continues as follows. In Section 2 we present the scope of the mathematical formulation of the elliptic PDE as well as results of calculations made by making use raw, not tuned codes. Our main goal was to prepare validated iterative solver for systems with large sparse matrices of coefficients. In Section 3 we describe in details our data structures and results of testing performance on two platforms (including one which is installed in WI Bialystok - Mordor2). In Section 4 we present our results for timing and estimating performance of running different parts of the solver routines. Finally, in Section 5 we summarize our efforts and formulate direction of further research.

2. Software for the Poisson equation solution

In mathematical formulation we confined ourselves to the one of the simplest elliptic boundary value problem described by the Poisson equation in a classical form (for the constant coefficients)

$$\nabla^2 u(x) = f(x) \quad \forall (x) \in \Omega \tag{1}$$

where u is a continuous function, ∇^2 is the Laplace operator and f is specified in domain Ω forcing (source). The problem (1) is considered with homogenous Dirichlet boundary conditions

$$u = 0 \quad \forall (x) \in \partial\Omega \tag{2}$$

The essential difference of the work in the relation to the previous stage [2] is extension of the consideration on any descryption in space. Taking respectively 1D, 2D and 3D domains we worked out three different codes (Pois1D, Pois2D and Pois3D) dedicated for solution of these problems. For the purpose of the codes validation, three different problems have been resolved. However the special attention is also directed on linear algebra problems arising from approximation technique. Each continuous problem has been discretized by standard second order difference method on structured grid. Thus our numerical schemes generate a class - in general very large systems - of linear equations which can be expressed as

$$A\_{D(h)} \cdot \hat{u} = f_h \tag{3}$$

where $A\_{D(h)}$ is the matrix of coefficients which structure depends on finite difference operator used, $\hat{u}$ is the vector of nodal unknowns, and f_h is the right - hand side vector.

The system of equations (3) is solved using the conjugate gradient solver with or without diagonal scaling (preconditioning). The iteration were performed till the relative norm of residual [2] dropped to 10^{-8}.

The accuracy of the schemes were analyzed by convergence of a global measure of the error, which a discrete norm (L2 - norm) has been chosen as

$$\|e\| = \left(\sum_J e_J^2 / N_{max} \right)^{1/2}, \tag{4}$$

where N_{max} is the total number of grid points and J stands for summation index as a i, ij, ijk for 1D, 2D and 3D respectively.
The error (4) is defined as

$$e = u - \hat{u} \tag{5}$$

and overall accuracy is found in each case by a systematic grid refinement study.

In the next, we present our numerical experiments which were carried out for three representative test problems. Our implementation is in C++.

2.1 Case 1D

As a first test case we consider (1,2) in 1D domain. In this case the problem in fact reduces to ordinary differential equation, which consequently for general Laplace operator form we defined as a

$$-\frac{d^2u}{dx^2} = f(x) \quad \forall (x) \in \Omega \tag{6}$$

where Ω is defined as the open section $(0,1)$. For homogenous Dirichlet boundary conditions $u(x=0)=0$ and $u(x=1)=0$ the problem has the exact solution

$$u(x) = (1-x)\cdot x\cdot e^{x}$$

with the source term defined as

$$f(x) = (3+x)\cdot x\cdot e^{x}. \tag{7}$$

For finite difference based solution of the Case 1D we use uniform grid describing the computational domain Ω. The grid nodes are equally spaced, $h=1/(n_x-1)$, where n_x is the number of grid points in x-direction. The grid coordinates are defined as

$$x_i = (i-1)\cdot h, \quad \text{for} \quad 1 \le i \le n_x\ .$$

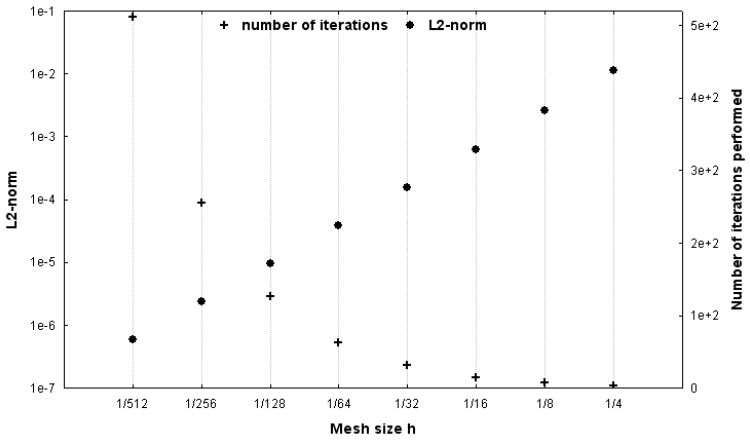


Fig. 1. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 1D.

We solved the problem (6, 7) on a series of meshes with h = 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256 and 1/512. The results in a form of norm (4) for the sequence of meshes are presented in Fig. 1.

2.2 Case 2D

As a second test case we consider the Poisson equation defined for 2D. Laplace operator modifies to the form

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y) \quad \forall (x,y) \in \Omega, \tag{8}$$

where Ω is defined as a unit square $(0,1)\times(0,1)$, and the exact solution has a form

$$u(x,y) = \sin(\pi x) \cdot \sin(2\pi y) .$$

For taken form of the exact solution and boundary conditions kind of (2) the source term is defined as

$$f(x,y) = 5\pi^2 \cdot \sin(\pi x) \cdot \sin(2\pi y) . \tag{9}$$

For finite difference based solution of the Case 2D we use uniform grid describing

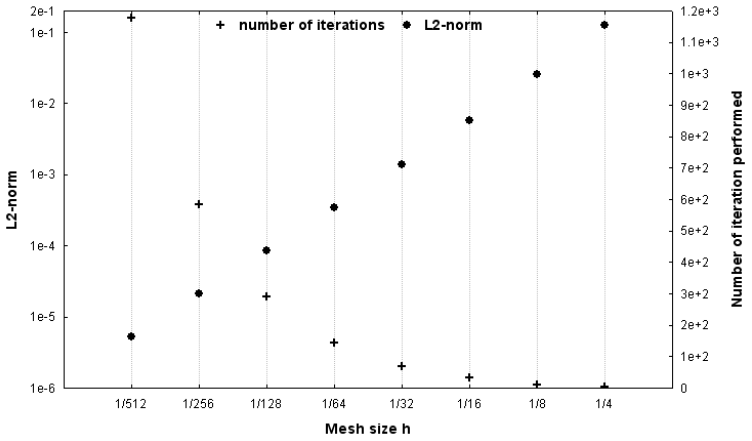


Fig. 2. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 2D.

the computational domain Ω. The grid nodes are equally spaced, $h = 1/(n_x - 1) = 1/(n_y - 1)$, where n_x and n_y is the number of grid points in x-, and y-direction, respectively. For unifirm mesh, the grid coordinates are defined as

$$x_i = (i-1) \cdot h, \quad y_j = (j-1) \cdot h \quad \text{for} \quad 1 \le i,j \le n (= n_x = n_y) .$$

The results for the same sequence of meshes in the 2D case as for the case 1D (uniform spacing along x- and y-direction) are presented in Fig. 2.

2.3 Case 3D

Finally, we successfully extended finite difference schemes approximation to 3D domains, so the generic Poisson equation has a form

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} = f(x,y,z) \quad \forall (x,y,z) \in \Omega \tag{10}$$

where Ω is defined as the unit cube $(0,1) \times (0,1) \times (0,1)$. For testing purpose we again took a test problem which has an exact solution

$$u(x,y,z) = \sin(\pi x)\cdot\sin(\pi y)\cdot\sin(\pi z)\,.$$

For the homogenous Dirichlet boundary conditions (2) - corresponding source term is defined as

$$f(x,y,z) = 3\pi^2\sin(\pi x)\cdot\sin(\pi y)\cdot\sin(\pi z)\,. \tag{11}$$

Also in this case, for defining the computational domain, we use an uniform, rectangular finite difference grid. The grid nodes are again equally spaced, $h = 1/(n_x - 1) = 1/(n_y - 1) = 1/(n_z - 1)$, where in general n_x, n_y, n_z are the number of grid points in x-, y-, and z-direction, respectively. For uniform spacing ($n_x = n_y = n_z = n$) the grid coordinates are defined as

$$x_i = (i-1)\cdot h, \quad y_j = (j-1)\cdot h, \quad z_k = (k-1)\cdot h, \quad 1 \le i,j,k \le n\,.$$

The problem Case 3D has been resolved by using different discrete problem with $h = 1/4, 1/8, 1/16, 1/32, 1/64, 1/128$. Results are given in Fig. 3.

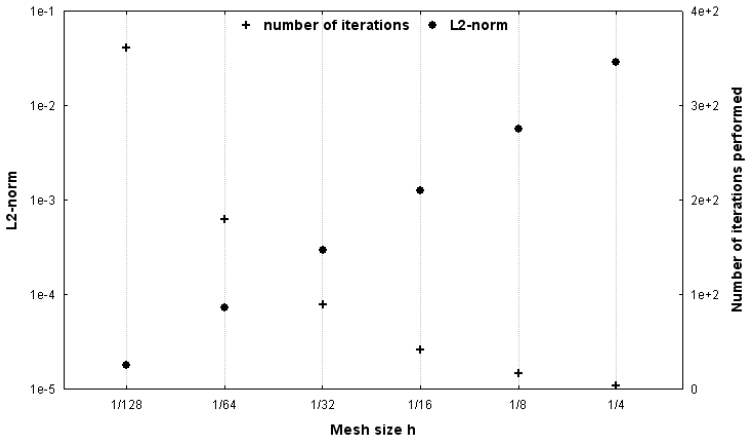


Fig. 3. Convergence plot (in L2-norm) and number of iterations performed versus mesh size for the test Case 3D.

A log-log plots (Fig. 1, Fig. 2, Fig. 3) of the error in L2 - norm versus h is approximately a straight line with a constant slope when h is sufficiently small. This observation confirm correctness of the worked out numerical schemes. At the same Figures we also present the total number of iterations performed for each run. Points related to performed iterations versus mesh size illustrate also the typical behavior of CG iterative algorithm for which number of iterations grows as the problem size increases. Moreover, they are also clear confirmation that CG like algorithm has a finite termination property - that means that at N-th iteration the algorithm will terminate

(for our boundary value problem N = n-2). For Case 1D this property is perfectly fulfilled. For 2D and 3D cases the number of iterations needed for convergence is likely much lower than N.

3. Performance and preliminary timing experiments

The obtained results very well validate the codes Pois1D, Pois2D and Pois3D. In this way - especially by comparisons with analytical solutions - we ensure that we operate on validated software. On this base we can redirect the main attention to linear algebra problems arising from approximation technique. It is worth to notice, that in the previous Section we have illustrated convergence plots as well iteration counts only in the sense of the maximum of iteration needed to obtain convergence.

Before we describe in details iterative solvers performance lets take a look on the linear algebra demands. As was pointed earlier, if we will consider a standard second order finite difference discretization of (1) on regular grid in 1D, 2D, and 3D dimensions we obtain linear system of equations (3). Data structure for storing grid points is not important because uniform mesh subdivision has been assumed. Quite different situation is when we must decide about data structure for storing coefficients of the matrix A. Strictly, after discretization on uniform mesh, the A is a matrix which is symmetric, positive definite and has a block-tridiagonal structure where the block have order N. In details, depending on the problem under consideration, the coefficient matrix is described in Tab. 1

Table 1. Coefficient matrix forms in diagonal format.

A_{1D}	has a 3-diagonal structure, in which non-zero coefficients (2) lie on the main diagonal and coefficients equal (-1) lie on the two off-diagonals;
A_{2D}	has a 5-diagonal structure, in which all the non-zero coefficients lie on: the main diagonal (4), the two neighboring diagonals (-1), and two other diagonals (-1) removed by N positions from the main diagonal;
A_{3D}	has a 7-diagonal structure, in which all the non-zero coefficients lie on: the main diagonal (6), the two neighboring diagonals (-1), two diagonals (-1) removed by N positions from the main diagonal and two other diagonals (-1) removed by N×N positions from the main diagonal.

Our final choice of data structure for matrices was primarily dictated by goals taken among of them first of all we wanted to operate on a very large sets of data. In result, in spite of the true sparsity of matrices A we store them in a two different formats

1. dense - which is standard practice for storing full matrices, and
2. diagonal [3] - which is a most preferred for the diagonal matrices resulting from finite difference discretization technique.

In this way we can easily operate on the diagonal scheme which may allow the computational problem to be kept in main memory, and dense array scheme which can not to be kept in main memory. For this, in fact we created two versions of our codes for each Poisson problem, namely: Pois1D(_DNS, _DIA), Pois2D(_DNS, _DIA) and Pois3D(_DNS, _DIA). The needed resources for storage of the matrix coefficients have been summarized in Tab.2.

Table 2. Size of coefficient matrices for different formats of storage.

	Dense format		**Diagonal format**			
		Matrix size	Diagonals		Matrix size	Problem size
1D	$N \times N$	261121	3	$N \times 3$	1533	511
2D	$N^2 \times N^2$	68184176641	5	$N^2 \times 5$	1305605	262144
3D	$N^3 \times N^3$	887503681	7	$N^3 \times 7$	14338681	2048383

Next, in this section we present preliminary results of possible performance of basic algebra operations attainable on the tested machines. The summary of technical specification of the used machines is given in Tab. 3.

Table 3. Machines used in tests.

Name	**PC**	**Mordor2 cluster**
CPU	Intel Core 2 Duo E6600	Intel Xeon X5355
OS	Ubuntu 7.10	CentOS 4.6
C compiler	gcc ver. 4.1.3	gcc ver. 3.4.6
C flags	no optimization	no optimization
BLAS[/*]	in line[/**]	in line[/**]

/*	Basic Linear Algebra Subprograms (BLAS) - proposed by Lawson et al. [4], at present optimized and implemented in a form of kernel routines of different specialized linear algebra packages.
/**	The results presented in the paper were obtained by running the simple loops as an in-line code.

The point of the first experiments was to get basic information about instruments for timing codes running on Linux platforms and especially to find attainable timer resolution. Because there are several Linux timers as: *clock*, *getrusage*, *clock_gettime*, *gettimeofday* in the first experiment we performed a several tests in order to chose the one with the best resolution and which invoking give the most stable results. For this we looked for the elapsed time in calculation one of the simplest algebraic operation (vector updating isolated to the one element)

$$x = x + \alpha \cdot y, \tag{12}$$

where x, y and α are known numbers (in double precision).

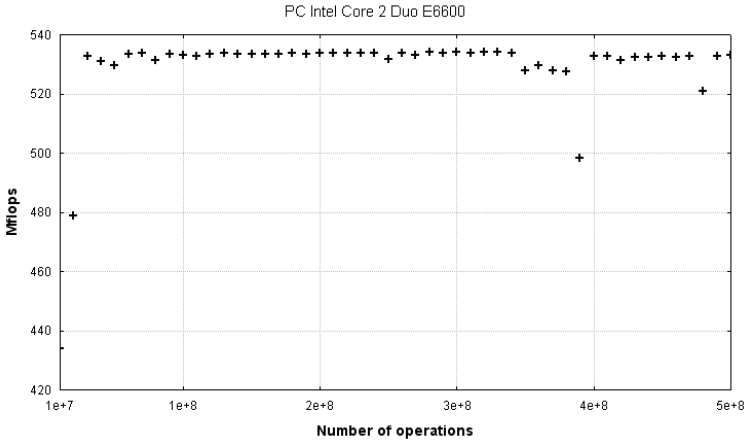


Fig. 4. Performance chart for simple updating - PC Intel Core 2 Duo E6600.

Because the timers have a different resolution (declared μs, 1 ms, $1/100^{th}$, 1s) therefore the timing has been obtained by execute calculation many times (repetitions). As a result of this preliminary set of experiments we decided to use for timing the C *gettimeofday* function which provides accuracy to approximately 1μs. Using the function we estimate performance of the used machines by repetitions of (12)

from 10^7 to the $5 \cdot 10^7$ times. The result of these calculations are given in Fig. 4 and Fig. 5.

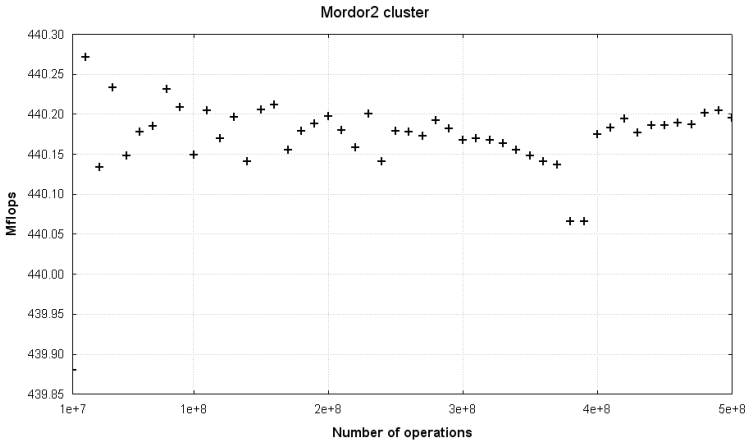


Fig. 5. Performance chart for simple updating - Mordor2 cluster Intel Xeon X5355.

4. Iterative solver performance

Among of many solution techniques for symmetric systems the conjugate gradient (CG) method is an ideal iterative solver. The CG algorithm can be also treated as a generic for all family of Krylov subspace methods. The CG algorithm in your classical form which we will called next as CGHS, is well known and his pseudo-code is given by Algorithm 1 [3].
As we can see in a sequence of CGHS algorithm there are three basic linear algebra operations:

1. `_dot` - scalar (inner or dot) product. There are two `_dot`'s in the algorithm what gives 4·n operations.

$$s = x \cdot y = (x, y) = \sum_{i=1}^{n} x_i \cdot y_i,$$

2. `_axpy` - vector update (where underscore mark in the operation name is used according to known from BLAS terminology prefix convention). There are three `_axpy`'s in the algorithm what gives 6·n operations.

$$y = y + \alpha \cdot x \ ,$$

where x and y are vectors of length n, and α is a number.

3. `matvec` - matrix and vector multiplication. There are one `matvec` in the algorithm what gives $2 \cdot n^2$ operations.

$$y = Ax ,$$

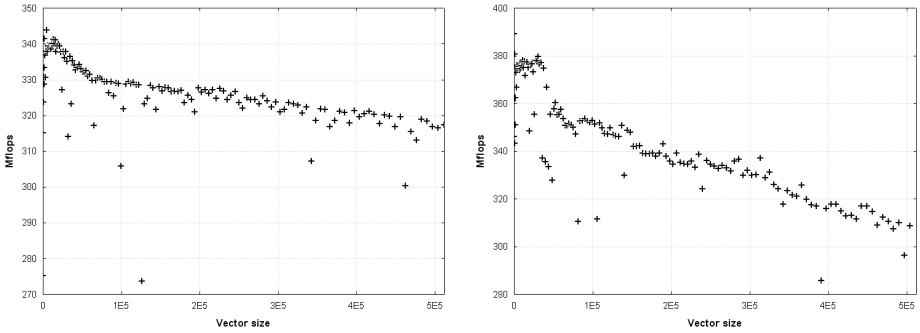
where A in our case is two-dimensional array which is $A_{n \times n}$ in dense format and $A_{n \times d}$ - banded, (incorporated diagonal format storage).

Table 4. Number of memory references and floating point operations for vectors of length n [5].

	Read	Write	Flops	Flops/mem access
_dot	$2 \cdot n$	1	$2 \cdot n$	1
_axpy	$2 \cdot n$	n	$2 \cdot n$	$\frac{2}{3}$
matvec	$n^2 + n$	n	$2 \cdot n^2$	2

Table 5. System parameters.

System parameters	PC Intel Core 2 Duo E6600	Mordor2 cluster
Clock rate	2.4 GHz	2.66 GHz
Bus speed	1066 MHz	1333 MHz
L1 cache	64 KB	128 KB
L2 cache	4 MB	8 MB

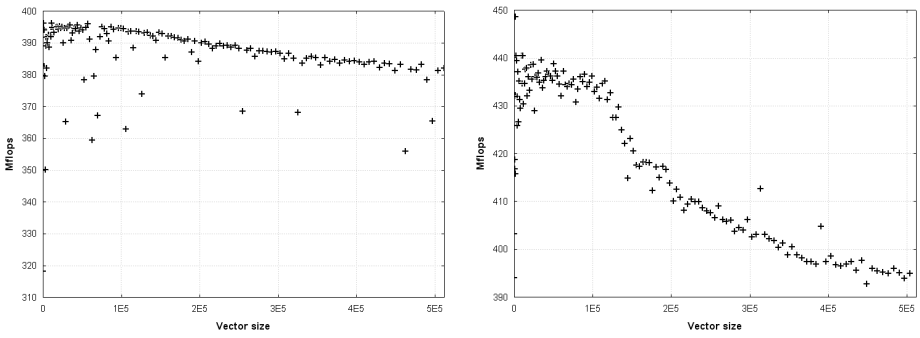


(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 6. Daxpy

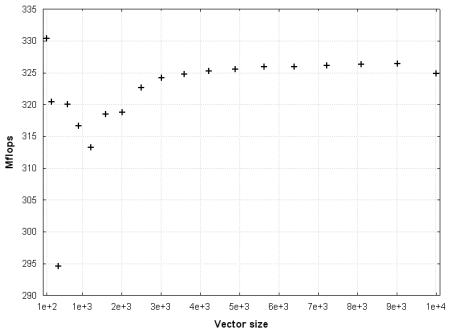
We timed the solver in two stages. First stage refers to the execution time which accounts only for the solver basic algebraic operation (`_axpy`, `_dot` and `matvec` for different format of matrices storage). The results are given in Figures 6÷9. All results in the Figures clearly show that exist some kind of mismatch between CPU and memory performance. This mismatch has an unfavorable influence on computer performance because as a general CPU's outperforms memory systems. The system parameters of the platform/machines we use in the paper are characterized in Tab. 5. The Mflops rates reflect also the ratios of memory access of the two machines. The high rates are for vectors that can be held in the on-chip cache. The low rates with very long vectors are related to the memory bandwith. The matvec has about the same performance as daxpy if the matrix can be held in cache. Otherwise, it is considerably reduced.

Next, in the second stage we measured the overall performance of the CGHS routine according to the scheme given in Algorithm 1. The timing covers entire iteration process which takes into account the time spent by the program/code for executing all commands and instructions.

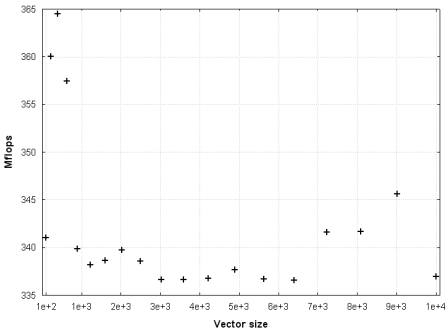


(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 7. Ddot

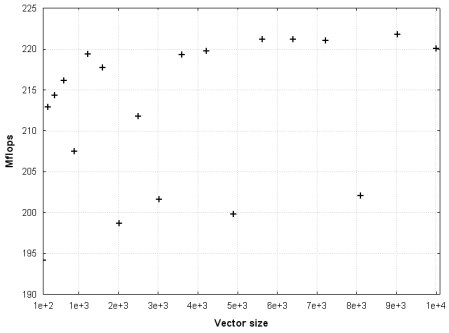


(a) PC Intel Core 2 Duo E6600

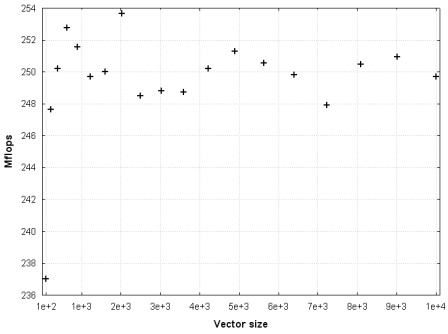


(b) Mordor2 cluster Intel Xeon X5355

Fig. 8. Matvec DNS



(a) PC Intel Core 2 Duo E6600



(b) Mordor2 cluster Intel Xeon X5355

Fig. 9. Matvec DIA

5. Summary

In the paper, we described the implementation of the iterative, conjugate gradient based solver for the Poisson equation. The special attention has been paid on estimate of performance of the most time consuming parts of the code.

Summarizing our achievements we compare our performance results to theoretical peak performance of the two, various platforms used in the experiments. As a peak performance we take counted number of floating point additions and multiplications which can be completed in a period of machine cycle time. So, during the one cycle we can estimate theoretical peak performance as a

$$\frac{1\text{operation}}{1\text{cycle}} \cdot \text{cycletime} = \text{peak performance}\,[\text{Mflop/s}].$$

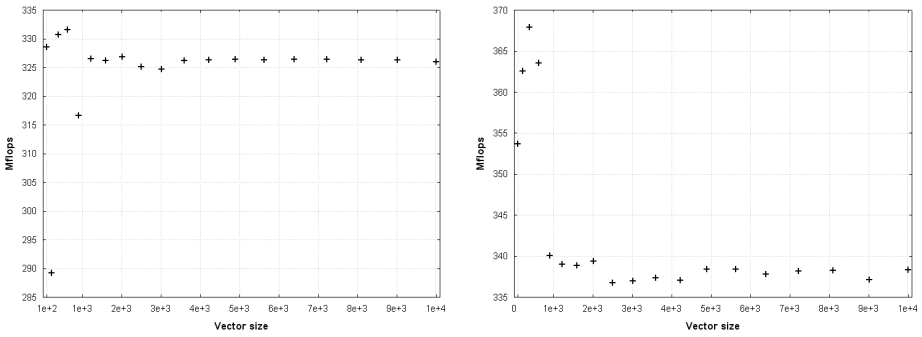
Table 6 shows comparison of the used computers peak performance and attained, average performance of the CGHS solver. It is clear, that at the stage we can say

Table 6. Summary of the CGHS efficiency results.

Machine	**Peak performance** [Mflop/s]	**CGHS performance** [Mflop/s]	**CGHS efficiency** [%]
PC Intel Core 2 Duo E6600	2400	285 ÷ 335	12 ÷ 14
Mordor2 cluster	2600	335 ÷ 370	13 ÷ 14

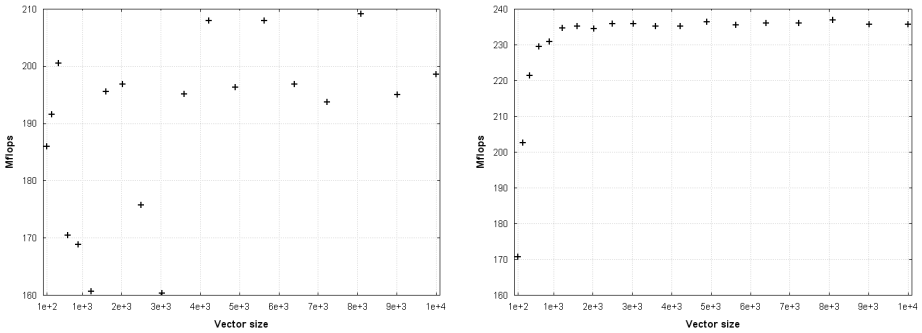
only that we posses unoptimized, correct sequential codes for solution the Poisson equation (Pois1D, Pois2D, Pois3D). At the further stages of the project we would like to do the following extensions of the current implementation, by

- adopting or preparing own instrumentation necessary to the baseline performance measurement (including possible implementation of a free available profilers),
- performing compiler optimization with the special emphasis on free available gcc compilers,
- linking optimized libraries of the BLAS kernels and asses the possible improvements in attainable performance,
- modification of source solver code taking into account possible identification of the performance bottlenecks.



(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 10. Overall performance of the CGHS routine - dense matrix of coefficients



(a) PC Intel Core 2 Duo E6600 (b) Mordor2 cluster Intel Xeon X5355

Fig. 11. Overall performance of the CGHS routine - diagonal matrix of coefficients

Algorithm 1 The standard (unpreconditioned) CGHS algorithm

$k = 0$
$x_0 = 0$
if $x_0 \neq 0$ **then**
  $r_0 = Ax_0 - b$
else
  $r_0 = b$
end if
$\rho_0 = \|r_0\|^2$
while $\sqrt{\rho_k} > \varepsilon \cdot \|r_0\|$ **do**
  if $k = 0$ **then**
    $p_1 = r_0$
  else
    $p_{k+1} = r_k + (\rho_{k-1}/\rho_k) \cdot p_k$ `// _axpy`
  end if
  $k = k + 1$
  $w_k = A \cdot p_k$ `// matvec`
  $\alpha_k = \rho_{k-1}/p_k^T \cdot w_k$ `// _dot`
  $x_k = x_{k-1} + \alpha_k \cdot p_k$ `// _axpy`
  $r_k = r_{k-1} - \alpha_k \cdot w_k$ `// _axpy`
  $\rho_k = \|r_k\|^2$ `// _dot`
end while
$x = x_k$

References

[1] Dongarra J., Luszczek P., Petitet A.: The LINPACK Benchmark: Past, Present, and Future. Concurrency and Computation: Practice and Experience, Vol. 15, No. 9, 2003, pp. 803-820.

[2] Gościk J., Gościk J.: Numerical efficiency of iterative solvers for the Poisson equation using computer cluster. Zeszyty Naukowe Politechniki Białostockiej, Seria: Informatyka, Vol. 3, 2008.

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

[4] Lawson C.L, Hanson R.J., Kincaid D.R., Krogh F.T.: Basic linear algebra subprograms for Fortran usage. ACM Transactions on Mathematical Software, Vol. 5, No. 3, September 1979, pp. 308-323.

[5] Arbenz P., Peterson W.: Introduction to Parallel Computing - A Practical Guide with examples in C. Oxford University Press, 2004, Series: Oxford Texts in Applied and Engineering Mathematics No. 9, Oxford 2004.

EFEKTYWNOŚĆ NUMERYCZNA ALGORYTMU GRADIENTÓW SPRZĘŻONYCH - IMPLEMENTACJA SEKWENCJNA

Streszczenie: Przedstawiono wyniki realizacji drugiego etapu projektu mającego na celu opracowanie i wdrożenie algorytmów rozwiązywania wielkich układów równań liniowych generowanych w procesie aproksymacji eliptycznych równań różniczkowych o pochodnych cząstkowych (PDE) metodą różnic skończonych. W szczególności skoncentrowano się na implementacji wersji sekwencyjnej najbardziej reprezentatywnej metody iteracyjnej zdefiniowanej w przestrzeni Kryłowa (metody gradientów sprzężonych). W pierwszej części pracy opisano szczegóły implementacji schematu iteracyjnego rozwiązywania dyskretnej postaci równania Poissona, uogólniając sformułowanie również do zagadnień przestrzennie trójwymiarowych. W drugiej części pracy skoncentrowano się przedstawieniu czasu wykorzystania procesora podczas wykonywania najbardziej czasochłonnych operacji algebry liniowej na macierzach rzadkich. Oceny poprawności formalnej jak też i wydajności obliczeniowej stworzonego kodu sekwencyjnego dokonano poprzez rozwiązanie trzech zagadnień testowych z wykorzystaniem dwóch komputerów o różnej konfiguracji sprzętowej.

Słowa kluczowe: Metody iteracyjne, Metoda różnic skończonych, Równanie Poissona, Wydajność kodu sekwencyjnego