ISSN 2176-5480

22nd International Congress of Mechanical Engineering (COBEM 2013)
November 3-7, 2013, Ribeirão Preto, SP, Brazil
Copyright © 2013 by ABCM

SOLVING LARGE SPARSE LINEAR SYSTEMS RESULTING FROM FLUID MECHANICS FLOW EQUATIONS ON MULTICORE PROCESSORS

João Batista Aparecido
Natallie Zilio de Souza
João Batista Campos Silva
Faculdade de Engenharia de Ilha Solteira – UNESP, Depto. Engenharia Mecânica. Av. Brasil Centro, 56, CEP 15385-000, Ilha Solteira - SP.
jbaparecido@dem.feis.unesp.br
natalliezilio@yahoo.com.br
jbcampos@dem.feis.unesp.br

Abstract. Mathematical modelling of fluid flow and heat transfer processes leads to systems of second order linear or non linear partial differential equations. For solving such systems of partial differential equations through the use of numerical methods such as finite elements is necessary to do the discretization process that transforms the original systems of equations, defined over a continuum domain, into a linear or non linear algebraic system of equations. Due to the characteristics of such methods for the partial differential equations domain as well for the equations themselves, generally the algebraic system that appears has the coefficient matrix with a very high sparsity. In this work we present new implementations for parallel processing of routines capable to solve large linear sparse systems with positive definite coefficient matrix, exploiting and preserving the initial sparsity. When split techniques such as the Characteristic Based Split (CBS) are employed, a Poisson equation for the pressure field is obtained. For this kind of equation, conjugate gradient methods are appropriated. In this work, it is used the conjugate gradient method for solution of large sparse linear systems running on multicore processors.

Keywords: Iterative Solution, Sparse System, Conjugate Gradient, Parallel Processing, Multicore.

1. INTRODUCTION

In the solution of fluid flows and heat transfer problems by numerical methods such as finite element method (FEM) and/or finite difference method (FDM), systems of partial differential equations are transformed to large and highly sparse systems of algebraic equations. More than ninety nine percent of the elements in the matrices are nulls, so some structured data is almost mandatory to prevent storage of zeros and to reduce cost of numerical solutions of the linear systems resulting.

There are several iterative methods that can be used to solve linear systems, for example, Gauss-Seidel, Jacobi, SOR, SSOR and conjugate gradient methods. Campos-Silva & Aparecido (2003) presented results of data structure in the context of the Gauss-Seidel and SOR methods. Aparecido et al. (2011) presented data structure and algorithms to solve large sparse linear systems using conjugate gradient and preconditioned conjugate gradient methods. In both works were used serial processing. Aparecido et al. (2012) analyzed the use of the conjugate gradient method (CG) to solve large and sparse linear systems but running in parallel under the paradigm of shared memory. Particularly, to achieve that we used OpenMP – Open Multi-Processing (Chapman et al., 2008). In the present work we extend further the problem and algorithm presented by Aparecido et al. (2012) by increasing the problem size and extending the OpenMP parallel constructs.

A set of linear systems similar to those that originate in three dimensional applications of FEM and/or FDM are solved by a CG algorithm running under OpenMP and the results of performance, speedup and efficiency of the method are presented and discussed. It has been considered linear problems with up to one hundred million of unknowns.

2. SOLUTION OF LINEAR SYSTEMS AND METHODS FOR MINIMIZATION

This paper is an extension to shared memory parallel processing from our previous paper on this subject (Aparecido et al., 2011). Here we shortened a little bit some mathematical details that can be found there.

Consider the linear system

$$\mathbf{Ax^*} = \mathbf{b} \tag{1}$$

where $\mathbf{A} \in \Re^{n \times n}$ is the coefficient matrix and $\mathbf{b} \in \Re^{n}$ the independent vector. Both are supposedly known. $\mathbf{x^*}$ is the vector of unknowns or solution vector to be determined.

The above equation can be restructured as

$$\mathbf{r}(\mathbf{x}) \equiv \mathbf{b} - \mathbf{A}\mathbf{x} \tag{2}$$

in which the solution vector $\mathbf{x}^*$ was replaced by a generic vector $\mathbf{x}$, so there will be a residual vector $\mathbf{r}(\mathbf{x})$. Of course, when $\mathbf{x} = \mathbf{x}^*$ the residue will be zero, $\mathbf{r}(\mathbf{x}) = \mathbf{0}$. Applying the rules of linearity, it is clear that the residue $\mathbf{r}(\mathbf{x})$ is linear with $\mathbf{x}$, and thus infinitely differentiable, so $\mathbf{r}(\mathbf{x}) \in C^\infty(\Re^n)$.

Additionally, we can define a quadratic functional as follows

$$\mathrm{F}(\mathbf{x}) \equiv \tfrac{1}{2}\mathbf{x}^\mathrm{T}\mathbf{A}\mathbf{x} - \mathbf{x}^\mathrm{T}\mathbf{b}, \tag{3}$$

whose gradient and Hessian are

$$\mathbf{g}(\mathbf{x}) \equiv \frac{\partial \mathrm{F}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{A}\mathbf{x} - \mathbf{b} = -\mathbf{r}(\mathbf{x}), \tag{4}$$

$$\mathbf{G}(\mathbf{x}) \equiv \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{A}. \tag{5}$$

Having obtained the gradient and Hessian of the function $\mathrm{F}(\mathbf{x})$, one can use some method of minimization to obtain the position $\mathbf{x}^*$ and the value of $\mathrm{F}(\mathbf{x}^*)$ at the minimum point. Note that at the minimum point $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, $\mathbf{r}(\mathbf{x}) = \mathbf{0}$ and $\mathbf{A}\mathbf{x}^* - \mathbf{b} = \mathbf{0}$. Therefore, calculation of the minimum point of the functional $\mathbf{F}(\mathbf{x})$, Eq. (3), corresponds to the solution of the linear system, Eq. (1).

The Hessian matrix (**G**) in the case is the same matrix of coefficients, **A**, and it must be positive definite, to attend the second necessary condition for the occurrence of a strong minimum in a given position, and should also be symmetrical, since

$$\mathrm{G}_{ij}(\mathbf{x}) = \frac{\partial^2 \mathrm{F}(\mathbf{x})}{\partial \mathrm{x}_i \partial \mathrm{x}_j} = \frac{\partial^2 \mathrm{F}(\mathbf{x})}{\partial \mathrm{x}_j \partial \mathrm{x}_i} = \mathrm{G}_{ji}(\mathbf{x}). \tag{6}$$

3. METHOD OF CONJUGATE GRADIENTS (CG)

A classical method of minimization that is most used in the solution of large sparse linear systems is the Conjugate Gradient Method. Different definitions of the functional $\mathrm{F}(\mathbf{x})$, Eq. (3), will lead to different variants of the method (Barret et al., 1994). In Aparecido et al. (2011) we give some details about those methods and its mathematical foundations. There are vast literature about conjugate gradient methods and solution of large sparse linear systems by using conjugate gradient methods, we can cite Golub and van Loan(1996), Golub and Meurant(1983), and Faber and Manteufel(1984).

Doing so Aparecido et al. (2012) obtained an algorithm for the serial processing implementation of the Conjugate Gradient Method.

Algorithm 1 – Method of Conjugate Gradients (CG)
$\mathbf{x}_0$ = initial vector
ε = stop parameter (positive and sufficiently small)
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\rho_0 = \mathbf{r}_0^\mathrm{T}\mathbf{r}_0$
k = 0
while $\|\mathbf{r}_k\| \geq \varepsilon$
  k = k + 1
  if k = 1
    $\mathbf{p}_1 = \mathbf{r}_0$
  else
    $\beta_k = \frac{\rho_{k-1}}{\rho_{k-2}}$
    $\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}$
  end if
  $\mathbf{w}_k = \mathbf{A}\mathbf{p}_k$

ISSN 2176-5480

$$\alpha_k = \frac{\rho_{k-1}}{\mathbf{p}_k^T \mathbf{w}_k}$$

$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_k$$

$$\rho_k = \mathbf{r}_k^T \mathbf{r}_k$$

end while

When Hestenes & Stiefel(1952) created the Conjugate Gradient Method, they presented formulation very similar to that shown above to the solution of linear systems. This methodology is suitable for solving large sparse linear systems, since do not perform computations “inside” the matrix **A** and thus avoids the phenomenon of fill-in common in direct methods. In the above algorithm the coefficient matrix **A** is used in only one matrix-vector product. Initially this method was designed as a direct method since, in exact arithmetic, the algorithm converges to the exact solution. However, in computer arithmetic, with round-off error, this finite termination in n steps is not guaranteed. On the other hand, for large linear systems a set of n iterations represent a high computational cost. Thus, for large linear systems, the conjugate gradient method is used with termination based on maximum number of iterations, usually much less than n, and in the value of the norm of the residue. The idea of considering the conjugate gradient method as iterative method was developed by Reid (1971). The use of iterative conjugate gradient method is useful; however the rate of convergence is critical to its success (van der Sluis and van der Vorst, 1992).

4. METHOD OF CONJUGATE GRADIENTS (CG) FOR PARALLEL PROCESSING WITH SHARED MEMORY

Also Aparecido et al. (2012) parallelized the Algorithm 1 that became Algorithm 2 aimed to run using OpenMP. For that case it was parallelized just the most computer intensive part of the algorithm that is the product matrix-vector $\mathbf{Ap_k}$.

Algorithm 2 – Method of Conjugate Gradients (CG) – Parallel OpenMP (Aparecido et al., 2012)

$\mathbf{x_o}$ = initial vector

ε = stop parameter (positive and sufficiently small)

$\mathbf{r_o} = \mathbf{b} - \mathbf{Ax_o}$

$$\rho_o = \mathbf{r}_o^T \mathbf{r}_o$$

k = 0

while $\|\mathbf{r_k}\| \geq \varepsilon$||

k = k + 1

if k = 1

$\mathbf{p_1} = \mathbf{r_0}$

else

$$\beta_k = \frac{\rho_{k-1}}{\rho_{k-2}}$$

$$\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}$$

end if

!Here starts parallel section

!$OMP PARALLEL DO SHARED(*list-of-shared*) PRIVATE(*list-of-private*) SCHEDULE(STATIC) &

!$OMP NUM_THREADS(intNumThreads) DEFAULT(NONE)

$$\mathbf{w}_k = \mathbf{Ap}_k$$

!$OMP END PARALLEL DO

!Here finishes parallel section

$$\alpha_k = \frac{\rho_{k-1}}{\mathbf{p}_k^T \mathbf{w}_k}$$

$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_k$$

$$\rho_k = \mathbf{r}_k^T \mathbf{r}_k$$

end while

In the present work we extend further the parallelization presented by Aparecido et al. (2012). To do that we moved upwards the placing of PARALLEL SECTION beginning and moved downwards its end. Besides we introduced some parallel DO constructs to do some linear combination of vectors and computing some scalar products. Additionally was

necessary to introduce the SINGLE construct to avoid data corruption when different threads write to a given memory position. The resulting Algorithm 3 is as follows:

Algorithm 3 – Method of Conjugate Gradients (CG) – Parallel OpenMP
$\mathbf{x}_o$ = initial vector
ε = stop parameter (positive and sufficiently small)
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\rho_o = \mathbf{r}_o^T \mathbf{r}_o$
k = 0
while $\|\mathbf{r}_k\| \geq \varepsilon\|$

$$\beta_k = \frac{\rho_{k-1}}{\rho_{k-2}}$$

!$OMP PARALLEL SHARED(*list-of-shared*) NUM_THREADS(intNumThreads) DEFAULT(NONE)

k = k + 1
if k = 1
!$OMP DO PRIVATE(*list-of-private*) SCHEDULE(STATIC)
$\mathbf{p}_1 = \mathbf{r}_0$
!$OMP END DO
else
!$OMP DO PRIVATE(*list-of-private*) SCHEDULE(STATIC)
$\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}$
!$OMP END DO
end if
!$OMP DO PRIVATE(*list-of-private*) SCHEDULE(STATIC)
$\mathbf{w}_k = \mathbf{A}\mathbf{p}_k$
!$OMP END DO

!$OMP DO PRIVATE(*list-of-private*) SCHEDULE(STATIC) REDUCTION(*operator:variable*)
$\gamma_k = \mathbf{p}_k^T \mathbf{w}_k$
!$OMP END DO

!$OMP SINGLE

$$\alpha_k = \frac{\rho_{k-1}}{\gamma_k}$$

!$OMP END SINGLE

!$OMP DO PRIVATE(*list-of-private*) SCHEDULE(STATIC) REDUCTION(*operator:variable*)
$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$
$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{w}_k$
$\rho_k = \mathbf{r}_k^T \mathbf{r}_k$
!$OMP END DO

!$OMP END PARALLEL
end while

Note that OpenMP is no verbose allowing with few lines of code to produce big computational effects. OpenMP have some constructs that are used do parallelize code (Chapman et al., 2008). OpenMP has a webpage http://openmp.org were information can be obtained.

In the pseudo code above the main constructs are:

- PARRALLEL – starts the parallel section creating some threads;
- DO – starts the parallelization of the outermost loop inside the parallel construct;
- SHARED(*list-of-shared*) – declares which computational entities (the *list-of-shared*) are shared among all threads;
- PRIVATE(*list-of-private*) – declares which computational entities (the *list-of-private*) are not shared among all

ISSN 2176-5480

threads having its own memory allocation for each thread;

- SCHEDULE(STATIC) – indicates how the loop workload will be distributed among all threads. The clause STATIC means that the workload will be distributed equally;
- NUM_THREADS(intNumThreads) – declares how many working threads are intended to be used. The aimed quantity of threads is equal to intNumThreads and must be less or equal to the maximum quantity of available threads for a given processor.
- DEFAULT(NONE) – Means that no one default are allowed in that construct.
- SINGLE – The code block inside a SINGLE construct will be computed by just one thread, the first that arrives. At the final of the construct there are implicit BARRIER.

5. SOLUTION OF AN ELLIPTICAL EQUATION

In many solutions of fluid flows and heat transfer equations by some numerical method, split techniques or fractional schemes are used in order to explore the parabolical, hyperbolical or elliptical nature of those equations. For example, in using the Galerkin finite element method stabilized by the Characteristic Based Split scheme (Lewis, Nithiarasu and Seetharamu, 2004), the velocity field is obtained explicitly and the pressure field is obtained implicitly by solving a Poisson equation that is an elliptical equation in its mathematical nature. That equation has the form:

$$\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}+\frac{\partial^2 p}{\partial z^2}=\frac{1}{\Delta t}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right). \tag{7}$$

where p is the pressure field, (u,v,w) is an intermediate velocity field that does not satisfy the mass conservation and Δt is the time step. In a 3D flow, the discretized equation results in a very large system of linear equations like that defined in Eq. (1). And to obtain results in relatively short times, parallel processing can be employed.

6. DEFINITION OF AN EXAMPLE CASE

Here we use as example the same problem presented by Aparecido et al. (2012). Be the linear system **Ax** = **b**, where

$$\mathbf{A}\equiv[\mathrm{A}_{i,j}],\ \mathbf{b}\equiv[\mathrm{b}_i]\ \text{where}\ \mathrm{A}_{i,j}\equiv\begin{cases}-1, & \mathit{if}\ \ i=j+\mathrm{int}(n^{2/3})\\ -1, & \mathit{if}\ \ i=j+\mathrm{int}(n^{1/3})\\ -1, & \mathit{if}\ \ i=j+1\\ +6, & \mathit{if}\ \ i=j\\ -1, & \mathit{if}\ \ j=i+1\\ -1, & \mathit{if}\ \ j=i+\mathrm{int}(n^{1/3})\\ -1, & \mathit{if}\ \ j=i+\mathrm{int}(n^{2/3})\\ 0, & \mathit{otherwise}\end{cases}\quad \text{and}\ \mathrm{b}_i=1/i;\quad i,j=1,2,...,n. \tag{8}$$

This is a positive definite and symmetric matrix inspired by heptadiagonal matrices that appear in the discretization, using a variety of methods, of the steady or unsteady Poisson partial differential equation, when defined over three-dimensional domains. The storage methodology used in this project is generic and can absorb various settings of matrices. The solution of similar type of linear systems, using some stationary methods and data structures like CRS - Compressed Row Storage was treated in Campos-Silva & Aparecido (2003). In Aparecido et al. (2011) was defined a data structure so called SCRS – Symmetric Compressed Row Storage aimed on saving some memory storage by exploiting the matrix symmetry. Also in this work we used SCRS.

7. RESULTS AND DISCUSION

Results presented in this paper were run in a desktop personal computer with a quad-core Intel i7 processor providing 8 threads. This type of computer uses three kinds of caches: L_1, L_2 and L_3. So, the flow of data and

ISSN 2176-5480

instructions from RAM to cores, back and forth, is complex and if the algorithm does not exploit adequately the characteristics of processor then computations would be inefficient.

We solved the proposed sparse linear systems from 10 million until 100 million of unknowns. We developed two main codes: the first one running serial; the second running in parallel, OpenMP, with from 2 to 4 threads. We avoided using from 5 to 8 threads because Aparecido et al. (2012) showed that for that quantity of active threads the quad-core processor does not perform well running that algorithm.

The parallel code running with 2 to 4 threads were developed using Algorithm 3 shown previously in this paper.

The memory footprint for Algorithms 1, 2 and 3 were taken to be approximately equal to ($60n+12nnz$) bytes. Where n is the order of the matrix **A** and nnz is the number of non zero elements in it. Results for memory footprint are shown in Figure 1. One can see that memory footprint is linear with matrix **A** order. This characteristic can be used to define a sparse matrix: that its memory footprint be linear with its order.

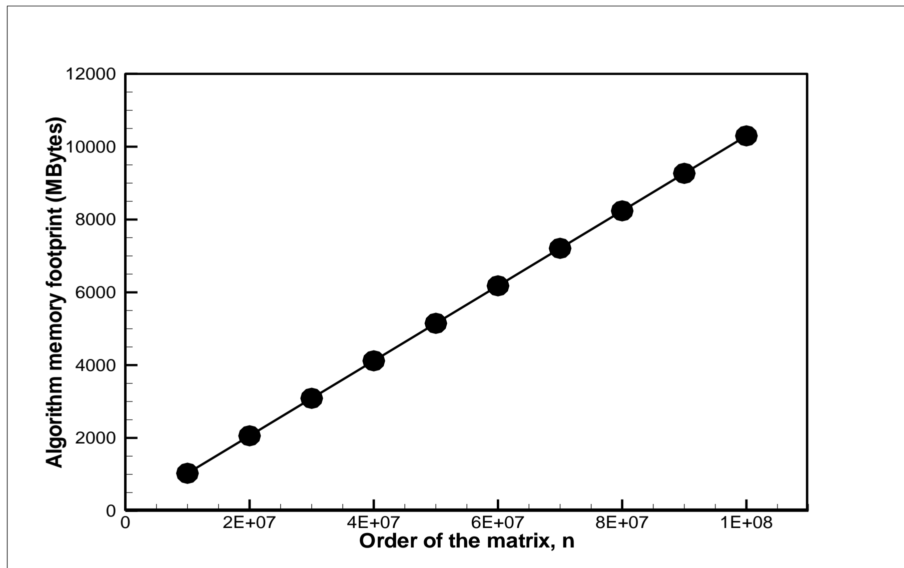

Figure 1 – Memory footprint for running the Conjugate Gradient Algorithms 1, 2 and 3.

In Figure 2 we show the number of iterations to achieve convergence under the stopping criteria $\|\mathbf{r}\| \le \varepsilon = 10^{-14}$. Generally, as expected when the order of the matrix **A** grows the number of iterations to achieve convergence also increases. It is notable that for 100 million of unknowns the algorithms spent just about 3500 iterations to converge.

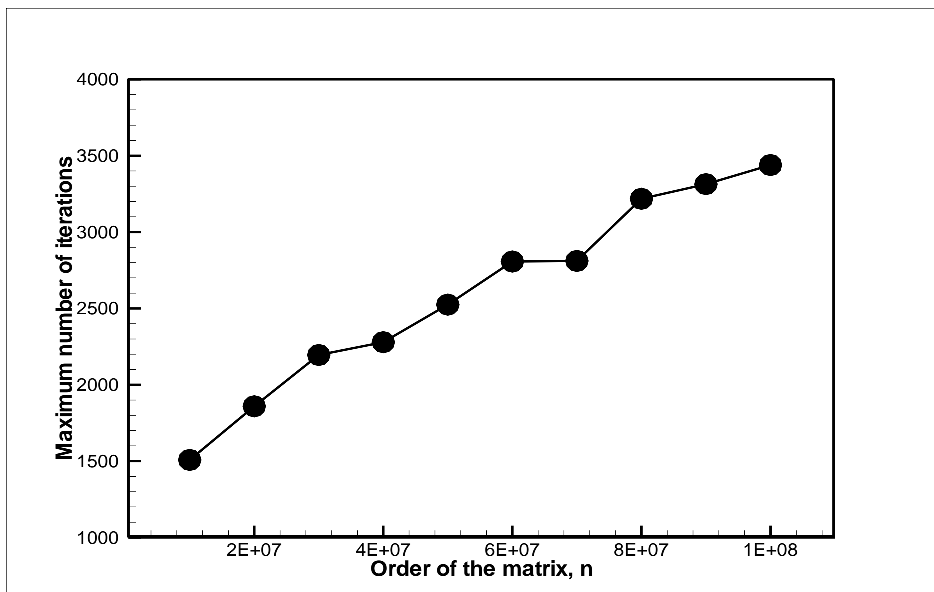

Figure 2 – Number of iterations to achieve convergence under the criteria $\|\mathbf{r}\| \le \varepsilon = 10^{-14}$

ISSN 2176-5480

In Figure 3 we can see that the best performance is reached when using 3 threads and the worst performance is

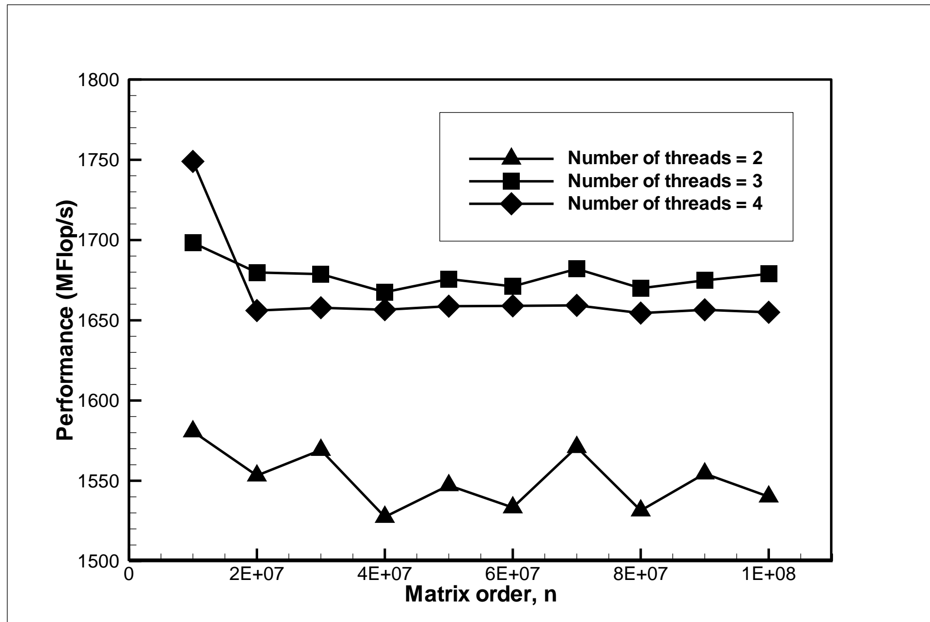

Figure 3 – Performance as function of matrix order and parameterized by the number of threads.

obtained for 2 threads. Performance for 4 threads is similar to that one obtained for 3 threads. Relatively, to the size of the problem, n, performance remains almost constant.

Similarly, to Figure 3 the best results obtained for speedup, Figure 4, were obtained using 3 threads and the worst were obtained using 2 threads. Relatively, to the size of the problem, n, the speedup grows slightly as the problem size grows. All curves generally presents a wavy behavior that is caused by data and instructions flow, back and forth, from RAM to cores, passing through caches (Wiggers et al., 2007).

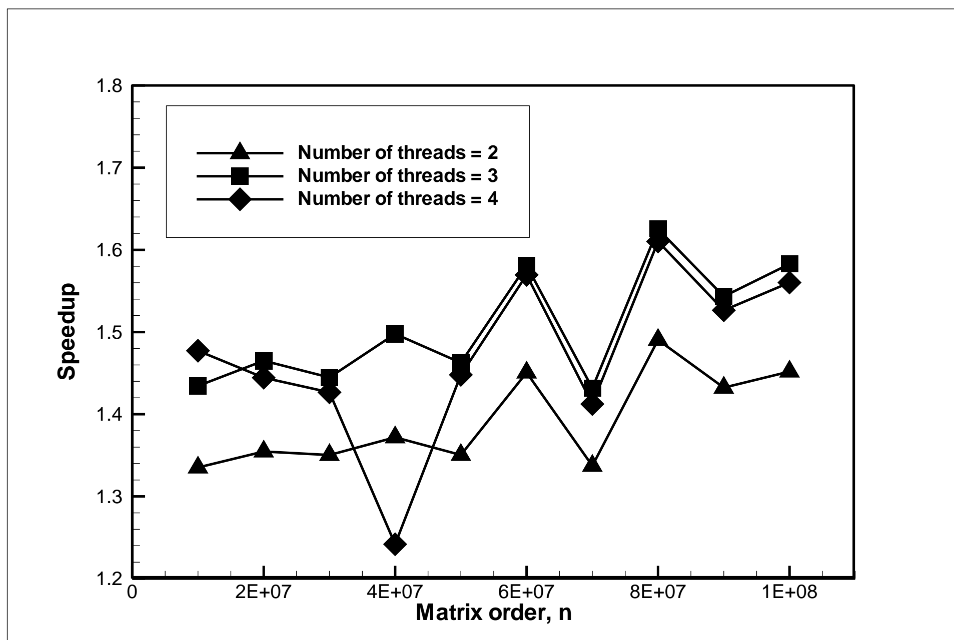

Figure 4 – Speedup as function of matrix order and parameterized by the number of threads.

The best efficiency obtained in the parallel multithreading is achieved using 2 threads, about 70%, as seen in Figure 5. Best performance and speedup happen for 3 threads but efficiency happens for 2 threads. The efficiency for 3 and 4 threads are about 50% and 38%, respectively.

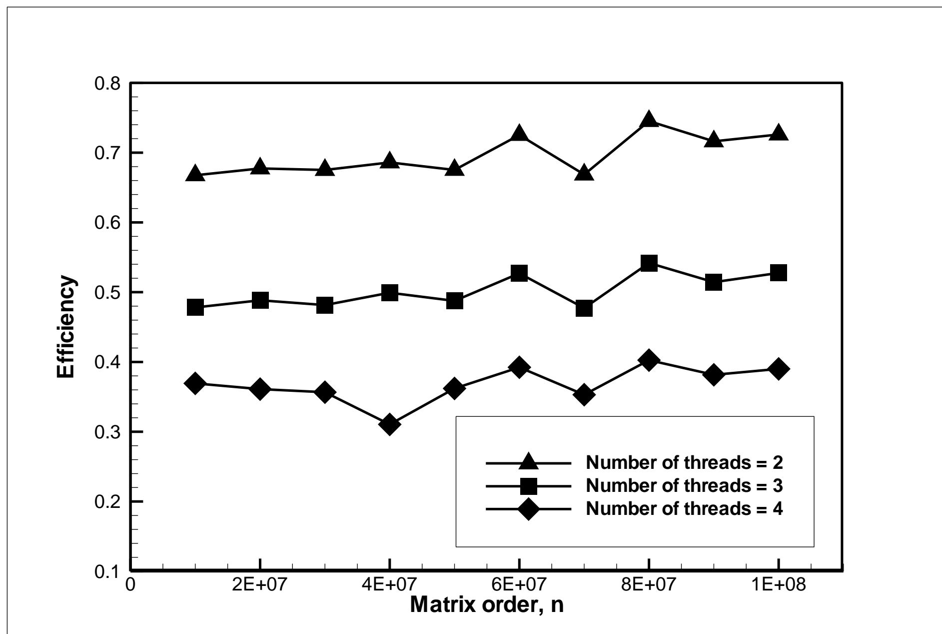

Figure 5 – Efficiency as function of matrix order and parameterized by the number of threads.

In Figure 6 are shown the results obtained using Algorithms 3. Note that the norm of the residual goes down until reaching the stopping criteria that was $\|\mathbf{r}\| \leq \varepsilon = 10^{-14}$. For 100 million of unknowns were necessary about 3200 iterations to reach convergence. The wall time spent to solve the problem with 10 million unknowns using 2 threads spent about 238s and to solve with 100 million unknowns, 2 threads, spent about 5099s. We speculate that in the same computer one can reach until about 150 million of unknowns, running in about 8500s.

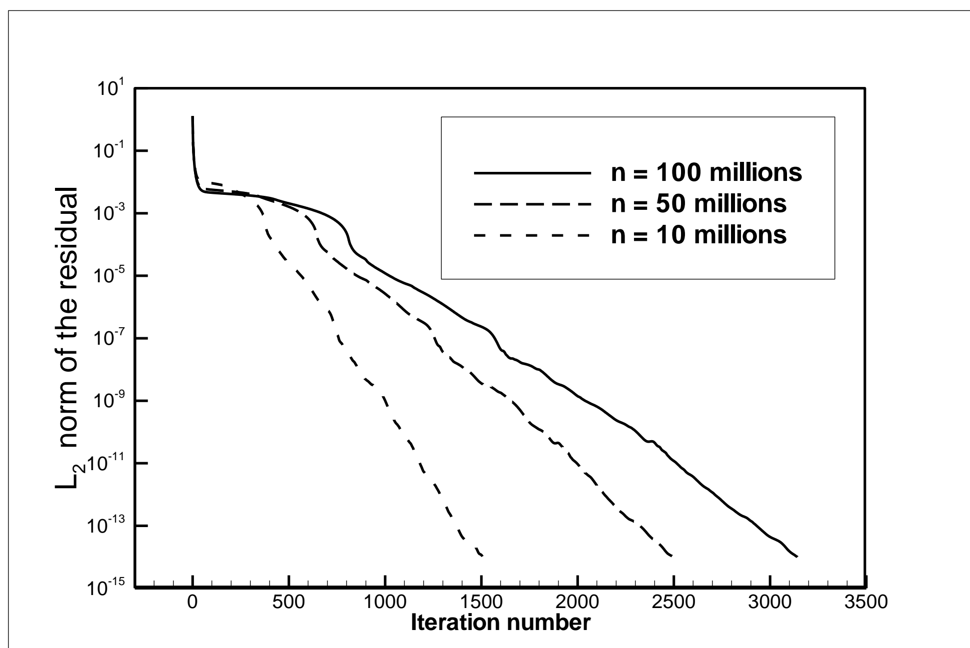

Figure 6 – L_2 norm of the residual as function of the number of iterations.

8. CONCLUSION

From the results and reasoning displayed before we can conclude that:

- OpenMP is a non verbose application programming interface that allows with few lines of code to produce code parallelization. Naturally, the code that goes inside an OpenMP construct must be in according with

parallel processing theory and with OpenMP standards;

- Parallelized Conjugate Gradient Algorithm 3 presented here worked well and was able to solve sparse linear systems with up to 100 million of unknowns in about 5100 seconds;
- The best performance and speedup were obtained using 3 threads;
- The best efficiency was obtained using 2 threads.
- Maybe, Algorithm 3 could be improved through a cautions profiling and rearranging some parts of it;
- Algorithm 3 and/or the i7 processor does not scales so well and thus efficiency goes down quickly as the number of threads increases.

9. ACKNOWLEDGEMENTS

The authors would like to thank **Fundação de Amparo a Pesquisa do Estado de São Paulo - FAPESP** for supporting our projects in the last two decades.

10. REFERENCES

Aparecido, J. B., Souza, N. Z. and Campos-Silva, J. B., 2012. “Conjugate Gradient Method for Solving Large Sparse Linear Systems on Multi-Core Processors”. In: *10th World Congress on Computational Mechanics*, São Paulo. 10th World Congress on Computational Mechanics, Belo Horizonte, MG: ABMEC, 2012. v. 1.

Aparecido, J. B., Souza, N. Z. and Campos-Silva, J. B., 2011. “Data Structure and the Pre-Conditioned Conjugate Gradient Method for Solving Large Sparse Linear Systems with Symmetric Positive Definite Matrix”. *Proceedings of CILAMCE-2011*, Ouro Preto.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J. M., Dongarra, J., Eikhout, V., Pozo, R. , Romine, C. and Van Der Vorst, H., 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, *SIAM*, Philadelphia, PA.

Campos-Silva, J. B. and Aparecido, J. B., 2003. “Data structure and stationary iterative solution methods for large sparse linear systems”. *Proceedings of the 17th International Congress of Mechanical Engineering – COBEM 2003*, Paper 0036, November 10-14, São Paulo, SP.

Chapman, B., Jost, G. and van der Pas, R., 2008. *Using OpenMP*, The MIT Press, Cambridge, 2008.

Faber, V. and Manteuffel, T., 1984. “Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method”, *SIAM J. Numer. Anal.*, 21, pp. 315-339, 1984.

Golub, G. H. and Meurant, G., 1983. “Résolution Numérique des Grandes Systèmes Linéaires”, *Collection de la Direction des Etudes et Recherches de l´Electricité de France*, vol. 49, Eyolles, Paris.

Golub, G. H. and van Loan, C. F., 1996. *Matrix Computations*, Third Edition, The John Hopkins University Press.

Hestenes, M. R. and Stiefel, E., 1952. “Methods of conjugates gradients for solving linear systems”, *J. Res. Nat. Bur. Standards*, 49, pp. 409-436.

Lewis, R. W., Nithiarasu, P., Seetharamu, K. N., 2004. *Fundamentals of the finite element method for heat and fluid flow*. John Wiley, 335 p.

Reid, J., 1971. “On the method of conjugate gradients for the solution of large sparse systems of linear equations”, in *Large Sparse Sets of Linear Equations*, Ed. J. Reid, Academic Press, London, pp. 231-254.

Van Der Sluis, A. and Van Der Vorst, H., 1992. “The Rate of Convergence of Conjugate Gradients”, *Numer. Math.*, 48, pp. 543-560.

Wiggers, W.A., Bakker, V., Kokkeler, A. B. J. and Smit, G. J. M., 2007. “Implementing the conjugate gradient algorithm on multi-core systems”, *In: International Symposium on System-on-Chip*, SoC, Tampere, Finland.

11. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.