Liviu Octavian Mafteiu-Scai /International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 1, January -February 2013, pp.1898-1907

Average Bandwidth Relevance in Parallel Solving Systems of Linear Equations

Liviu Octavian Mafteiu-Scai
Computer Science Department, West University of Timisoara, Timisoara, Romania

ABSTRACT:
This paper presents some experimental results obtained on a parallel computer IBM Blue Gene /P that shows the average bandwidth reduction [11] relevance in the serial and parallel cases of gaussian elimination and conjugate gradient. New measures for the effectiveness of parallelization have been introduced in order to measure the effects of average bandwidth reduction. The main conclusion is that the average bandwidth reduction in sparse systems of linear equations improves the performance of these methods, a fact that recommend using this indicator in preconditioning processes, especially when the solving is done using a parallel computer.

KEYWORDS: average bandwidth, linear system of equations, parallel methods, gaussian elimination, conjugate gradient, sparsematrices

1. THEORETICAL CONSIDERATIONS

The systems of linear equations appear in almost every branch of science and engineering. The engineering areas in where sparse and large linear systems of equation arise frequently include the chemical engineering processes, design and computer analysis of circuits, power system networks and many others. The search for efficient solutions is being driven by the need to solve huge systems - millions of unknowns- on parallel computers. The interest in parallel solving systems of equations, especially those very large and sparse, has been very high, there are hundreds of papers that deal with this subject. As solving methods there are *direct* methods and *iterative* methods. Gaussian elimination and conjugate gradient are two popular examples in this respect. In gaussian elimination the solution is exact and it is obtained in finitely many operations. The conjugate gradient method generates sequences of approximations that converge in the limit to the solution. For each of them there are many variants developed and the literature is very rich in describing these methods, especially in the case of serial implementations. Below, are listed suscint, by some particularities of parallel implementations of these two methods, case where the matrix system is partitioned per rows. It is considered a *nxn* system, the case when the size of system *n* is divisible with number of partitions *p* and the partitions are equals in terms of number of rows, ie *k=n/p* rows in each partition, ie a partition p_i of size k will include consecutive rows / equations between $\left(\frac{(i-1)\cdot n}{k}+1\right)$ and $\left(\frac{(i-1)\cdot n}{k}+k+1\right)$.

Parallel gaussian elimination (GE)

The main operations performed there are: local pivot determination for each partition in part, global pivot detemination, pivot row exchange, pivot row distribution, computing the elimination factors, computing the matrix elements. Because the values of the unknowns depend on each other and are computes one after another, the computation of the solutions in the backward substitution is inherently serial. In gaussian elimination is an issue with load balancing because some processors are idle since all their work is done.

Paralel conjugate gradient (CG)

The CG method is very good for large and sparse linear systems because it has the property of uniform convergence, but only if the associated matrix is symmetric and positive definite. The parallelism in CG algorithm derives from parallel matrix-vector product. Other operations can be performed in parallel as long as there is no dependency between them, such as for example, updating the residual vector and the vector solution. But these latter operations can not be performed before performing the matrix-vector product and the matrix-vector product in a new iteration can not be performed until the residual vector is updated. So, there are two moments in which processors must synchronize before they can be continue the work. It is desirable that between these two points of synchronization, the processors do not have periods of inactivity, which is an ideal case. In practice, the efficiency of the computation follows a minimization of this waiting/idle time for synchronization.

It has been observed that a particular preparation of system before application a numerical method for solving, leads to an improvement of the process and the solution. This preparation was called *preconditioning*. In time, many preconditioning methods have been proposed, designed to improve the process of solving a system of equations. There are many studies on the influence of preconditioning to parallel solving the systems of linear equations [1, 2, 3].

Reducing the bandwidth of associated matrix is one of these preconditioning methods and for this there are a lot of methods, the most popular being presented in works such as [4, 5, 6, 7 and 8]. In paper

[9] it is presented a study of parallel iterative methods Newton, conjugate gradient and Chebyshev, including the influence of bandwidth reduction in terms of convergence of these methods.

In paper [10] it was proposed a new measure for sparse matrices called *average bandwidth* (mbw). In [11] algorithms and comparative studies related to this new indicator was made. Paper [12] proposes methods that allow for a pair of matrix lines/columns, without performing interchange, qualitative and quantitative measurement of opportunity for interchange in terms of bandwidth and average bandwidth reduction.

According to [11], the *average bandwidth* is deffined by relation:

$$mbw = \frac{1}{m}\sum_{aij\neq 0}|i-j|, i = \overline{1n},\ j = \overline{1n} \quad (1)$$

where *m* is the number of non-zero elements and *n* is the size of the matrix *A*.

The reasons, specified in [11], for using average bandwidth (*mbw*) instead bandwidth (*bw*) in preconditioning before parallel solving system of equations are:

- *mbw* reduction leading to a more uniform distribution of non-zero elements around the main diagonal and along the main diagonal;
- *mbw* is more sensitive than the *bw* to the presence around the main diagonal of the so-called "holes", that are compact regions of zero values;
- *mbw* is less sensitive to the presence of some isolated non-zero elements far from the main diagonal. In case of a matrix which minimizes *mbw* will have most non-zero elements very close to the main diagonal and very few non-zero elements away from it. This is an advantage according to the paper [13], as to be seen in Figure 1 c). For the same matrix 1a) two algorithms CutHill-McKee for 1b) were used and the one proposed in [10] for 1c), the first to reduce the bandwidth *bw* and the second to reduce the average bandwidth *mbw*.

Paper [15] describes a set of indicators to measure the effectiveness of parallel processes. From that work, two simple but relevant indicators were chosen: *RelativeSpeedup* (*Sp*) and *RelativeEfficiency* (*Ep*) described by relations (2) and (3).

$$S_p = \frac{T_1}{T_p} \quad (2)$$

$$E_p = \frac{S_p}{p} = \frac{T_1}{p \cdot T_p} \quad (3)$$

where p is the number of processors, T_1 is the execution time achieved by a sequential algorithm and T_p is the execution time obtained with a parallel algorithm with p processors.

When $Sp = p$ we have a *linear speedup*. Sometimes in practice, there is an interesting situation, known as *super linear speedup* when $Sp > p$. One possible cause is the *cache effect*, resulted from memories hierarchy of parallel computers [16]. In our experiments such situations have been encountered, some of which are contained in tables 1 and 3.

Note: In our experiments, because we were especially interested in the effects of *mbw* reduction, in relations (2) and (3) we consider T_1 as execution time before *mbw* reduction, serial case.

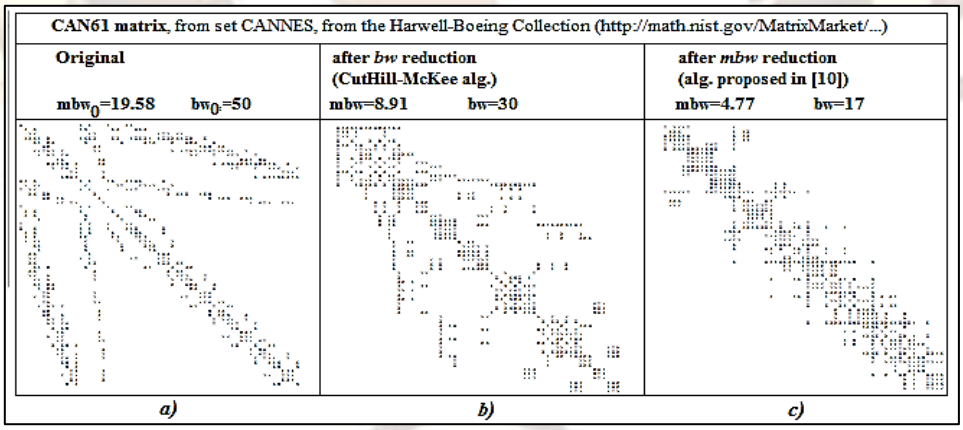

Figure 1. A matrix after *bw*reduction and after *mbw* reduction

Gain Time (*GT*) measure is introduced, which is, in percents, the difference between the execution time before *mbw* reducing (T_b) and the execution time after *mbw* reducing (T_a), related to the execution time before *mbw* reducing (T_b), for the same partitioning:

$$GT = \frac{(T_b - T_a)}{T_b} * 100 \quad (4)$$

positive values showing a more efficiency in terms of execution time after *mbw* reducing.

It introduced the measure *Increase of efficiency* (*IE*), which is, in percent, difference between the relative efficiency after *mbw* reducing (Ep_a) and relative efficiency before *mbw* reducing (Ep_b) reported to the relative efficiency before *mbw* reduction (Ep_b), for the same partitioning:

$$IE = \frac{Ep_a - Ep_b}{Ep_b} * 100 \quad (5)$$

It was noted by *A* the average number of iterations required for convergence and with A_p the average

number of iterations required for convergence per processor. In the experiments performed, *A* was the average value obtained for 50 systems with same size but with different nonzero elements distribution and different sparsity. So, $A_p = A/p$ where p is the number of processors/partitions. It will be seen from experiments that between *Ap* and execution time there is a directly proportional relation, so that A_p can be a measure of the parallelization efficiency.

Using only the value of the average number of iterations per processor A_p, can allow an estimation of the efficiency by neglecting the hardware factor (communication time between processors, cache effect, etc.), whose influence is difficult to calculate. Based on these considerations, it is proposed a new measure of efficiency based on the average number of iterations per processor, called *Efficiency in Convergence* (*EC*). It shows for two situations S_a (after) and S_b (before), how it has decreased/increased, in percent, the average number of iterations per processor in situation S_b from the situation S_a. In this study (the relevance of average bandwidth reduction in parallel case) the situation S_a is represented by A_{p_a} and situation S_b *by* A_{p_b} where indices "*a*" and "*b*" refers to "after *mbw* reduction" and "before *mbw* reduction". So, the computing relation is:

$$EC = \frac{A_{p_a} - A_{p_b}}{A_{p_a}} \cdot 100 \qquad (6)$$

The next section will show that *mbw* reducing leads to efficient parallel computing process.

2. THE EXPERIMENTS

The experiments were performed on IBM Blue Gene /P supercomputer. In experiments it has been implemented the gaussian elimination without pivoting and the preconditioned conjugate gradient with block Jacobi preconditioning.

To implement these serial and parallel numerical methods, it was used IBM XL C compiler version 9.0 under Linux. For parallel implementation of the GE and CG methods it was used MPI (Message-Passing Programming), a standard in parallel computing which enables overlapping communications and computations among processors and avoid unnecessary messages in point to point synchronization.

For parallel partitioning, the system of equations was divided equally between processors using the divisors of the system size.

Matrices/systems chosen for the experiments were generated randomly, positive definite, symmetric and sparse, with a sparsity degree between 5 and 20%. The size of these matrices were between 10 and 1000, the weight representing a size of 200. In the experiments there were generated and used matrices with uniform and nonuniform distribution of nonzero elements. In terms of reducing the average bandwidth *mbw*, values obtained were between 10 and 70%. For each instance/partitioning were used 50 systems with the same size but different as sparsity and distribution of nonzero elements. Sizes were varied with ratio 10, from 10 to 1000.

In case of CG, for arithmetic precision required by the convergence, epsilon values chosen were 10^{-10}, 10^{-30}, 10^{-100} and 10^{-200} and the initial vector that was used $X_0=\{0, 0,\ldots,0\}$.

2.1 Serial case

Gaussian elimination. It has been experimentally observed that in general, in GE, the average bandwidth reduction and/or bandwidth reduction did not significantly affect the computation in terms of its efficiency. Only in the cases where $mbw << n$ or $bw << n$ there was an increase in process efficiency because the complexity decreases from $O(n^3)$ to $O(nb^2)$, as is mentioned in [14] regarding *bw*.

Conjugate gradient. Experiments represented in Figure 2 show a general sensitivity to the *mbw* value, especially at larger sizes than 100 and a greater accuracy computation.

2.2 Parallel case

2.2.1 Gaussian elimination. According to our experiments it resulted that in approximately 60% of cases, the *mbw* reduction , leads to an increase in efficiency, but without major differences in terms of execution time (order of magnitude). It was observed that increasing the number of processors involved in computation, first leading to a decrease in execution time, reaching a minimum value followed by an increase in execution time with increasing number of processor, as can be seen in Figure 3. An example is shown below.

Example 1: size of system: 500x500, 1486 non-zero values, uniform distribution; before *mbw* reduction: mbw_0=110.33 bw_0=499; after *mbw* reduction: *mbw*=12.44 *bw*=245.

In experiments there were encountered situations (some partitioning) when the gaussian elimination failed. Possible causes include: rounding errors, numerical instability or main diagonal of the associated matrix contains zeros or very small values

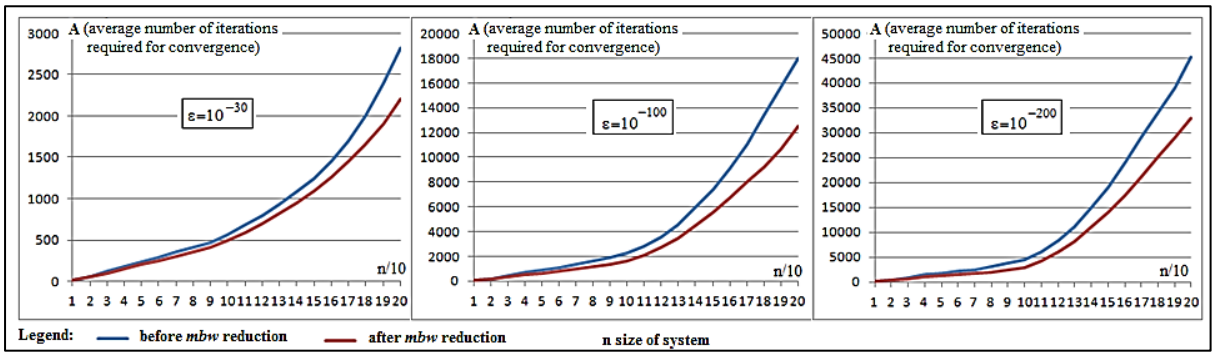

Figure 2 Conjugate gradient-serial case

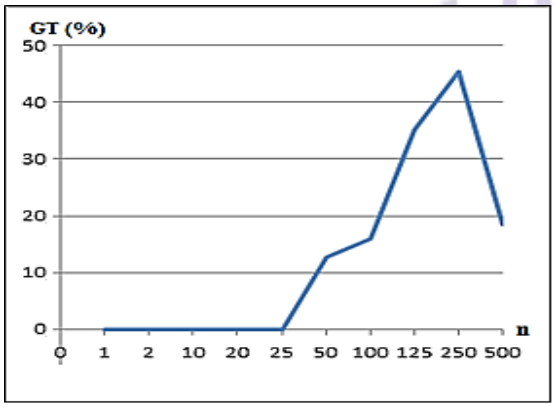

Figure 3 Gain Time: Example 1

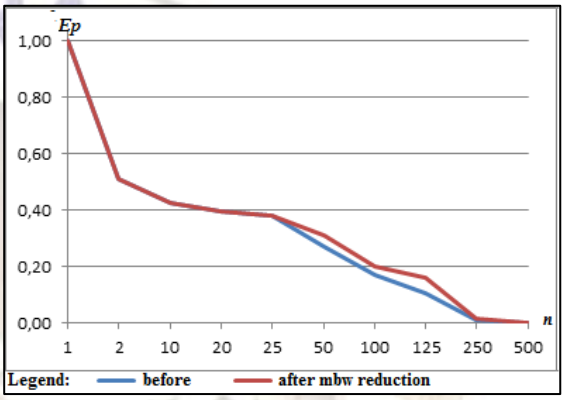

Figure 5 *Relative Efficiency:* Example 1

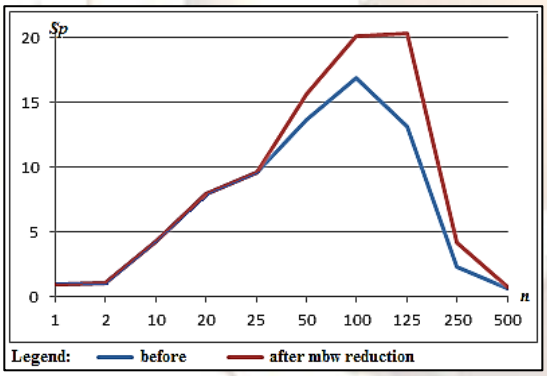

Figure 4 *Relative speedup*: Example 1

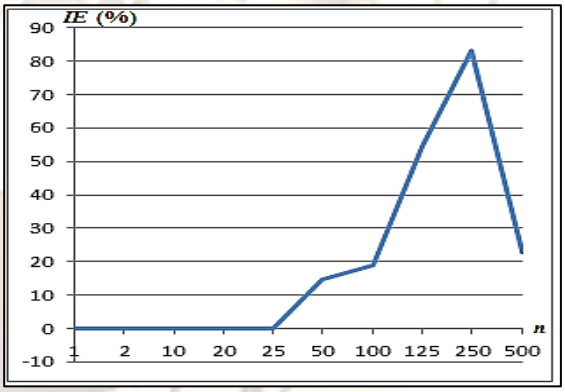

Figure 6 *Increase of Efficiency:*Example 1

Proce-ssors	Runtime (s)		*GT (%)*	*Sp*		*Ep*		*IE (%)*
	before*mbw*	after		before	After	before	after	
1	3,609060	3,609085	-0,000693	1,000000	0,999993	1,000000	0,999993	-0,000693
2	3,547747	3,547770	-0,000648	1,017282	1,017276	0,508641	0,508638	-0,000648
10	0,842680	0,842976	-0,035126	4,282836	4,281332	0,428284	0,428133	-0,035114
20	0,453526	0,454285	-0,167355	7,957780	7,944484	0,397889	0,397224	-0,167076
25	0,376788	0,376625	0,043260	9,578490	9,582635	0,383140	0,383305	0,043279
50	0,264829	0,231279	12,668552	13,627888	15,604789	0,272558	0,312096	14,506289
100	0,213262	0,179467	15,846705	16,923127	20,109881	0,169231	0,201099	18,830760
125	0,273594	0,177232	35,220911	13,191276	20,363478	0,105530	0,162908	54,370803
250	1,598770	0,873000	45,395523	2,257398	4,134089	0,009030	0,016536	83,135166
500	6,175630	5,036316	18,448547	0,584404	0,716607	0,001169	0,001433	22,621972
Average	*1,735589*	*1,143524*	*12,741968*	*7,042048*	*8,475456*	*0,327547*	*0,341137*	*19,330474*

Table 1 Example 1: experimental results

2.2.2 Conjugate gradient

Figure 7 shows the global effect of reducing *mbw* in the case of parallel conjugate gradient, especially with the increasing the size of systems and with increased computing accuracy.

We mention that in Figure 7 are represented the average values obtained for the different systems of equations (sparsity, distribution, etc.) before and after *mbw* reduction

Below are presented some examples (2, 3, 4, 5 and 6) of different situations, which shows the effects of *mbw* reducing in parallel solving systems of linear equations using conjugate gradient method.

Example 2: size of system: 1000x1000, 36846 non-zero values, uniform distribution; before *mbw* reduction: mbw_0=413 bw_0=999; after *mbw* reduction: *mbw*=211 *bw*=911.

In Table 2, Figure 8 and 9 it is shown the correlation between execution time and average number of iterations per processor *Ap*, which justifies the use of the last as an indicator of performance measurement.

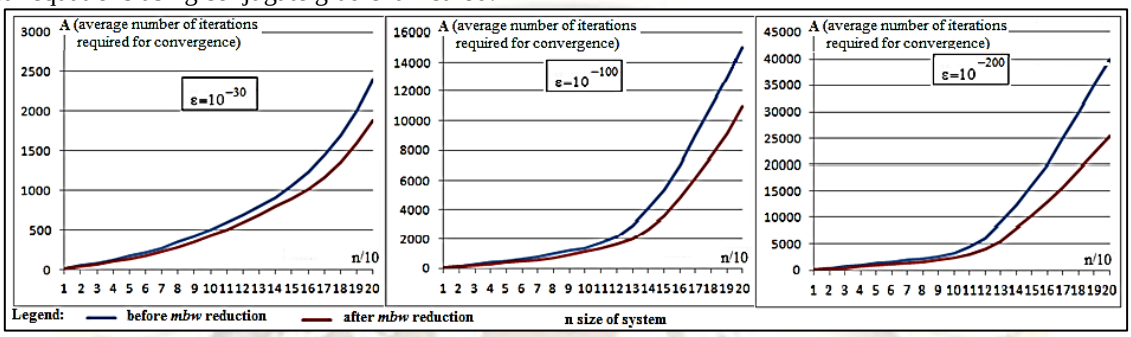

Figure 7 Conjugate gradient-parallel case

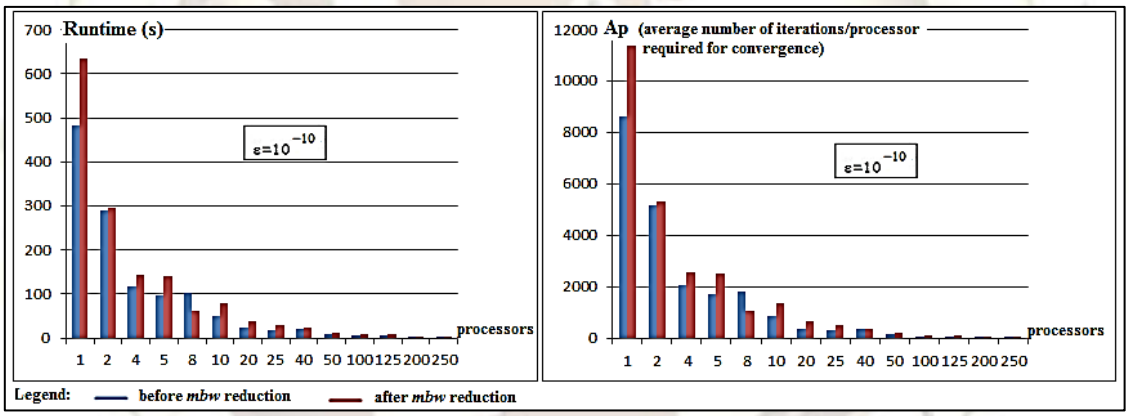

Figure 8 The correlation between Ap and Runtime, from Example 2 (e-10)

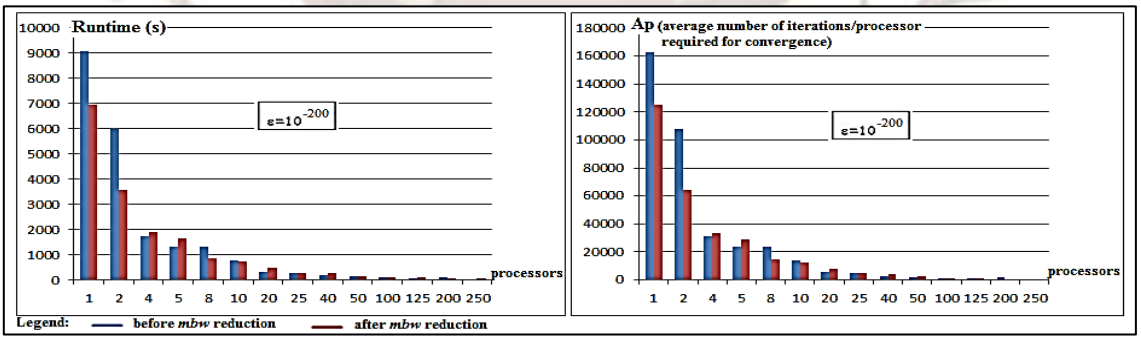

Figure 9 The correlation between Ap and Runtime, from Example 2 (e-200)

Accuracy	e-10							e-200						
	Before *mbw*			After *mbw* reduction			GT (%)	Before *mbw* reduction			After *mbw* reduction			GT (%)
Processors	A	Ap	Runtime (s)	A	Ap	Runtime (s)		A	Ap	Runtime (s)	A	Ap	Runtime (s)	
	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8646	864	482	11412	1141	636	-32	162681	162681	9065	124993	12499	6965	**23**
2	1036	518	289	10646	5323	297	-3	215659	107830	6009	128283	64142	3575	**41**
4	8236	205	116	10300	2575	145	-25	123443	30861	1738	135138	33785	1903	-9
5	8664	173	98	12602	2520	142	-45	118771	23754	1333	146061	29212	1639	-23
8	1463	183	103	8637	1080	61	**41**	189500	23688	1336	119041	14880	839	**37**
10	8673	867	49	13761	1376	78	-59	137918	13792	776	127943	12794	720	**7**
20	7859	393	23	13578	679	39	-73	112891	5645	324	166420	8321	477	-47
25	8417	337	19	12548	502	29	-49	121422	4857	279	117369	4695	269	**3**
40	1457	364	22	15337	383	23	-4	121392	3035	178	177855	4446	261	-46
50	8741	175	10	10427	209	12	-19	113060	2261	134	122097	2442	145	-8
100	7945	79	5	13201	132	9	-64	152264	1523	96	142257	1423	90	**7**
125	1190	95	6	16159	129	8	-35	113625	909	59	170478	1364	88	-50
200	1031	52	4	9216	46	3	**10**	327724	1639	117	139755	699	50	**57**
250	1273	51	4	8773	35	3	**30**	119635	479	36	155445	622	46	-30
Av.	***1012***	***156***	***88***	***11900***	***1886***	***106***	***-21***	***152142***	***27354***	***1534***	***140938***	***21701***	***1219***	***21***

Table 2 Example 2: experimental results

Processors	e-10					e-200				
	Before *mbw* reduction		After *mbw* reduction		IE	Before *mbw* reduction		After *mbw* reduction		IE
	Sp	Ep	Sp	Ep		Sp	Ep	Sp	Ep	
1	1,00	1,00	0,76	0,76	-24,23	1,00	1,00	1,30	1,30	30,15
2	1,66	0,83	1,62	0,81	-2,69	1,51	0,75	2,54	1,27	**68,11**
4	4,15	1,04	3,32	0,83	-20,03	5,22	1,30	4,76	1,19	-8,65
5	4,94	0,99	3,40	0,68	-31,26	6,80	1,36	5,53	1,11	-18,68
8	4,66	0,58	7,89	0,99	**69,46**	6,78	0,85	10,80	1,35	**59,19**
10	9,84	0,98	6,21	0,62	-36,95	11,68	1,17	12,59	1,26	7,80
20	21,37	1,07	12,38	0,62	-42,04	28,00	1,40	19,00	0,95	-32,16
25	24,79	0,99	16,65	0,67	-32,84	32,51	1,30	33,64	1,35	3,45
40	22,15	0,55	21,31	0,53	-3,80	50,83	1,27	34,74	0,87	-31,66
50	46,00	0,92	38,66	0,77	-15,95	67,50	1,35	62,53	1,25	-7,36
100	93,24	0,93	56,68	0,57	-39,21	93,98	0,94	100,71	1,01	7,16
125	76,82	0,61	56,91	0,46	-25,92	154,67	1,24	103,16	0,83	-33,30
200	126,42	0,63	140,79	0,70	11,37	77,77	0,39	182,14	0,91	**134,20**
250	121,96	0,49	173,85	0,70	42,55	253,83	1,02	195,61	0,78	-22,94
Average	*39,93*	*0,83*	*38,60*	*0,69*	***-10,82***	*56,58*	*1,10*	*54,93*	*1,10*	***11,09***

Table 3 Example 2: experimental results

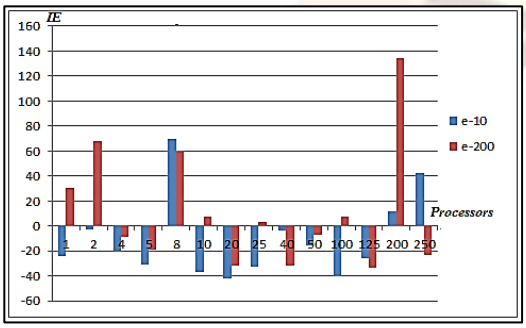

Figure 10: IE from Example 2-experimental results

Example 3:size of system: 100x100, 1182 non-zero values, sparsity=11,82%, uniform distribution; before *mbw* reduction: mbw_0=33.61 bw_0=99; after *mbw* reduction: *mbw*=9.97 *bw*=60.

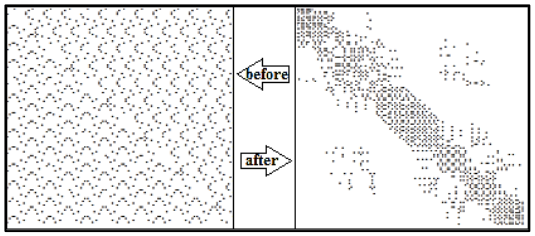

Figure 11: Matrix form from Example 3 before and after*mbw* reduction

In Figure 12, according to equation (6), we see that reducing the average bandwidth *mbw* has the best effect in the case of 10 partitions, in terms of number of iterations per processor necessary for convergence.

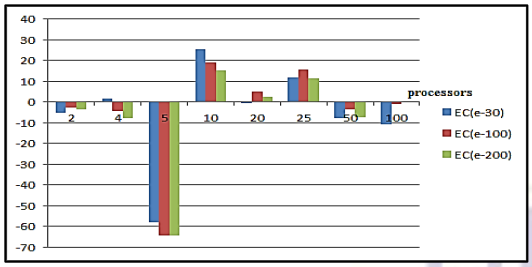

Figure 12:Efficiency inconvergence-Example 3

Example 4:size of system: 100x100, 924 non-zero values, sparsity=9.24%, nonuniform distribution; before *mbw* reduction: mbw_0=38.73 bw_0=99; after *mbw* reduction: *mbw*=20.05 *bw*=80.

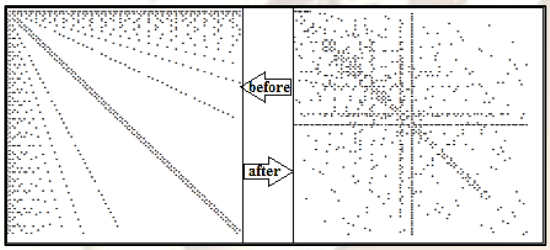

Figure 13: Matrix form from Example 4 before and after*mbw* reduction ($\varepsilon=10^{-30}$)

This is an example in which all partitioning are favorable to average bandwidth reduction, as it can be seen in figure 14.

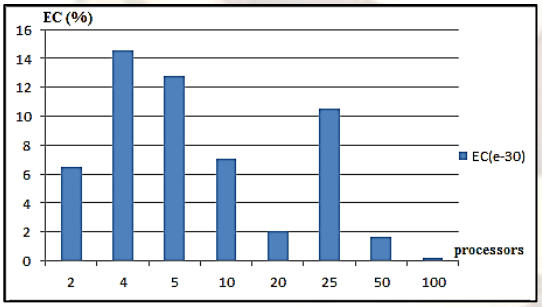

Figure 14:Efficiency in convergence-Example 4

Example 5:size of system: 100x100, 1898 non-zero values, sparsity=18.98%, nonuniform distribution; before *mbw* reduction: mbw_0=36.38bw_0=97; after *mbw* reduction: *mbw*=21.63*bw*=97.

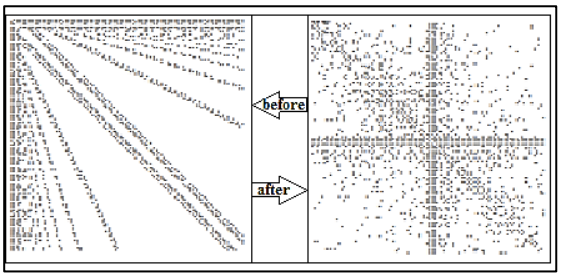

Figure 15 Matrix form from Example 5 before and after*mbw* reduction ($\varepsilon=10^{-30}$)

In some cases, some partitioning leads to a drastic decrease inefficiency after *mbw* reducing, but in general, there are other favorable situations in terms of convergence, as can be seen in figures 16 and 12.

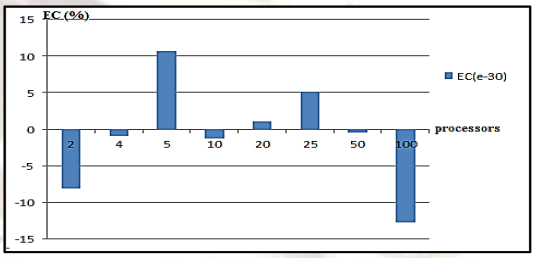

Figure 16 Efficiency inconvergence-Example 5

Example 6:size of system: 200x200, 4814 non-zero values, sparsity=12.03%, uniform distribution; before *mbw* reduction: mbw_0=66.96 bw_0=196; after *mbw* reduction: *mbw*=28.34 *bw*=196.

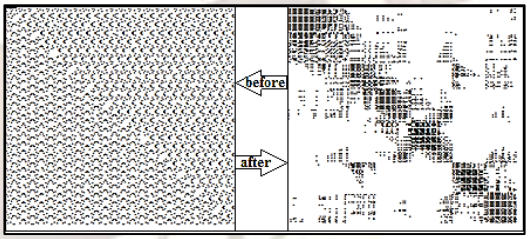

Figure 17 Matrix form from Example 6 before and after*mbw* reduction

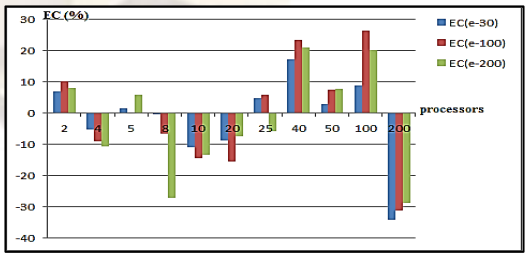

Figure 18 Efficiency in convergence-Example 6

2.3Comparing the *mbw* reducing effects at GE and CG

In order to see which method is the most strongly influenced by the reduction of *mbw* there were performed a series of comparative experiments. In

examples 7, 8 and 9 are presented experimental results for the gaussian elimination and conjugate gradient in solving same linear systems of equations. Computation accuracy imposed for convergence in the case of conjugate gradient was 10^{-10} and 10^{-200}. The results presented in the tables are obtained after the *mbw* reduction.

Example 7: size of system: 100x100, 2380 non-zero values, uniform distribution; before *mbw* reduction: mbw_0=36.97 bw_0=99; after *mbw* reduction: *mbw*=21.84 *bw*=91.

Runtime (s) / Processors	Gaussianeliminat ion	Conjugate Gradient	
		e-10	e-200
1	0,055312	0,0856	0,71408
2	0,055265	0,0483	0,35996
4	0,042756	0,0280	0,22991
5	0,040240	0,0217	0,17075
10	0,034234	0,0129	0,09611
20	0,032118	0,0091	0,06396
25	0,032066	0,0084	0,06069
50	0,035373	0,0074	0,05209
100	0,051450	0,0073	0,04848
Average	*0,042090*	*0,025462*	*0,199563*

Table 4 Comparative experimental results

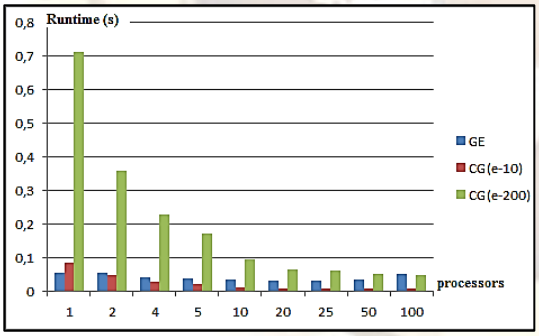

Figure 19

Example 8:size of system: 100x100, 508 non-zero values, uniform distribution; before *mbw* reduction: mbw_0=26.81 bw_0=95; after *mbw* reduction: *mbw*=5.18 *bw*=55.

Runtime (s) / Processors	Gaussianeliminat ion	Conjugate Gradient	
		e-10	e-200
1	0,053286	0,0689	0,5235
2	0,053243	0,0394	0,2876
4	0,040771	0,0230	0,1668
5	0,038213	0,0171	0,1254
10	0,032215	0,0108	0,0802
20	0,030087	0,0076	0,0564
25	0,030005	0,0070	0,0456
50	0,061563	0,0063	0,0397
100	0,111814	0,0062	0,0400
Average	*0,050133*	*0,020763*	*0,151748*

Table 5 Comparative experimental results

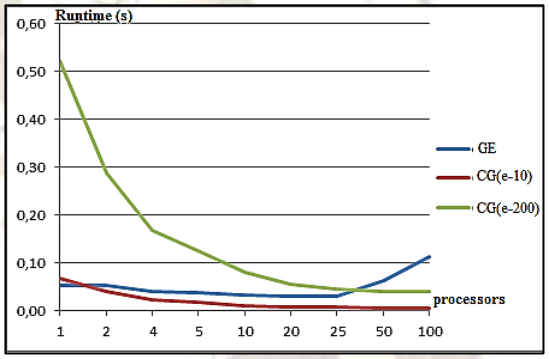

Figure 20 Experimental results: CG vs. GE

Example 9 (same system as in Example 6)

Runtime(s) / Processors	Gaussia nelimina tion	Conjugate gradient	
		e-10	e-200
1	3,60908	2,6408	19,46738
2	3,54777	1,3360	10,05824
10	0,84297	0,2855	2,115549
20	0,45428	0,1582	1,132339
25	0,37662	0,1312	0,885119
50	0,23127	0,0844	0,51753
100	0,17946	0,0681	0,299843
125	0,17723	0,0615	0,256805
250	0,87353	0,0575	0,187639
500	5,03631	0,0481	0,139639
Average	*1,532857*	*0,487175*	*3,506010*

Table 6 Comparative experimental results: CG vs. GE

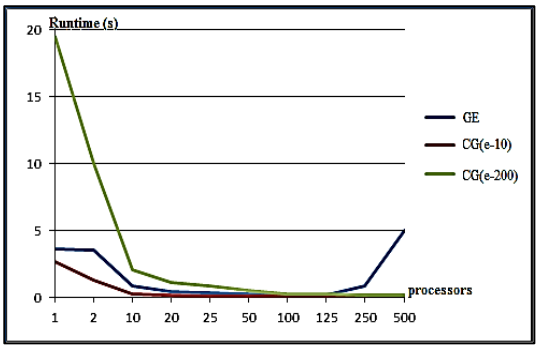

Figure 21 Experimental results: CG vs. GE

The execution time is significantly lower in the case of conjugate gradient, especially if using more processors for processing.
Increasing the number of processors in the case of conjugate gradient leads to the possibility of performing calculations with high accuracy (e-200) and low execution time, comparable to those obtained by the gaussian elimination method or that obtained with conjugate gradient using low accuracy calculations (e-10).

3. FINAL CONCLUSIONS AND FUTURE WORK

The proposed indicator in [11], average bandwidth *mbw*, was validated by experimental results presented in this paper and recommended its use in the preconditioning of large systems of linear equations, especially when the solving is done using a parallel computer.
Using the proposed indicator *average number of iterations per processor* A_p, is a good choice because it allows an estimation of the efficiency by neglecting the hardware factor, in case of parallel iterative methods. Also, the proposed indicator *Efficiency in Convergence* (*EC*), based on A_p, shows in comparative studies for parallel iterative methods, in an intuitive way, the obtained progress/regression.
The proposed indicators *Gain Time* (*GT*), and *Increase of efficiency* (*IE*), in comparative studies show clear by and intuitive by the obtained progress/regression.
Extending the study to the case of nonlinear systems of equations are required to be made, encouraging results are also expected in conditions as shown in [9], under certain conditions nonlinear parallel case can be reduced to linear one.
The influence of *mbw* reducing in the case of unequal and overlapping partitions (equal&overlap, unequal&non-overlapping, and unequal&overlapping) will be a future study.

4. REFERENCES

[1] Michele Benzi, *„Preconditioning Techniques for Large Linear Systems: A Survey”*, Journal of Computational Physics, Vol. 182, Issue 2, 1 November 2002, Pages 418–477, Elsevier 2002
[2] Yousef Saad, *„Iterative Methods for Sparse Linear Systems”*, second edition, ISBN-13: 978-0-898715-34-7 SIAM 2003
[3] O. Axelsson, *„A survey of preconditioned iterative methods for linear systems of algebraic equations”*, Springer, BIT Numerical Mathematics 1985, Volume 25, Issue 1, pp 165-187 , 1985
[4] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*. In Proc. of ACM, pages 157-172, New York, NY, USA, 1969. ACM.
[5] N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer. *An algorithm for reducing the bandwidth and profle of a sparse matrix*. SIAM Journal on Numerical Analysis, 13:236-250, 1976.
[6] R. Marti, M. Laguna, F. Glover, and V. Campos. *Reducing the bandwidth of a sparse matrix with tabu search*. European Journal of Operational Research, 135:450-459, 2001.
[7] R. Marti, V. Campos, and E. Pinana, *A branch and bound algorithm for the matrix, bandwidth minimization*, European Journal of Operational Research, 186:513-528, 2008.
[8] A. Caprara and J. Salazar-Gonzalez, *Laying out sparse graphs with provably minimum Bandwidth*, INFORMS J. on Computing, 17:356-373, July 2005
[9] S. Maruster, V. Negru, L.O. Mafteiu-Scai*,” Experimental study on parallel methods for solving systems of equations*”, Proc. of 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 2012 (will appear in IEEE-CPS Feb.2013)
[10] L.O. Mafteiu-Scai, ”*Bandwidth reduction on sparse matrix”*, In West University of Timisoara Annals, Mathematics and Computer Science series, volume XLVIII fasc. 3, Timisoara, Romania, pp 163-173, ISSN:1841-3293, 2010.
[11] L. O. Mafteiu-Scai, V. Negru, D. Zaharie, O. Aritoni, *„Average Bandwidth Reduction in Sparse Matrices Using Hybrid Heuristics”*, in Proc. KEPT 2011, selected papers, (extended version) ed. M. Frentiu et. all, Cluj-Napoca, 2011, pp 379-389, Presa Universitara Clujeana, ISSN 2067-1180, 2011
[12]L. O. Mafteiu-Scai, *„Interchange Opportunity In Average Bandwidth Reduction In Sparse Matrix*, In West University of Timisoara Annals, Mathematics and Computer Science series , volume L fasc. 2, Timisoara, Romania, pp 01-14, ISSN:1841-3293, Versita Publishing DOI: 10.2478/v10324-012-0016-1, 2012.
[13] P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland, *„Parallel numerical linear algebra, chapter A comparison of parallel solvers for diagonally dominant and general narrowbanded*

linear systems", pages 35-56. Nova Science Publishers, Inc., Commack, NY, USA, 2001
[14] S. S. Skiena, *"The algorithm design manual"*, second ed., Springer, ISBN 978-1-84800-069-8, 2012
[15] S. Sahni, V. Thanvantri, "*Paralel computing: performance metrics and models*", Computer & Information Sciences Dep., Univ. of Florida, Rep. USA Army Research Office, Grant DAA H04-95-1-0111, 1995
[16] D. P. Helmbold, C. E. McDowell, *"Modeling speedup(n) greater than n" IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 250–256, 1990.