Iterative solvers based on Krylov subspace methods.

1 Methods description

1.1 Methods to implement

The system of linear equations:

$$Ax = b,$$

where $A \in R^{N*N}$, $x, b \in R^N$.

Krylov subspace method is a projection method on Krylov subspaces:

$$K_m(A, v) = spanv, Av, A^2v, ..., A^{m-1}v$$

The idea of projection method is to extract an approximate solution from this subspace. In general method consists of two parts:

1. Construction of the ortogonal basis of the subspace
2. Finding a vector from the subspace wich provides minimization of residual in some norm or satisfy another conditions.

1.1.1 Conjugate Gradient Method (CG) [3]

This method requires A to be symmetric and positive defenite.
The CG constructs the ith iterate $x^{(i)}$ as an element of $x^{(0)} + spanr^{(0)}, ..., A^{i-1}r^{(0)}$ so that $(x^{(i)} - x)^T A(x^{(i)} - x)$ is minimized, where x is the axact solution of $Ax = b$. For symmetric A an ortogonal basis for the Krylov subspace $spanv, Av, A^2v, ..., A^{m-1}v$ can be constructed with only three-term recurrences, such a recurrence also suffices for generating the residuals. So there is no need to store a lot of vectors. Residual vector $r^{(i)}$ in this iterative process is made ortogonal to all the previous residual vectors.

- Application:
  For symmetric and positive defenite matrix A.
- Computations per iteration:
  One matrix-vector product, three vector updates, and two inner products.
- Storage:
  4 additional vectors of size N.

1.1.2 BiConjugate Gradient Method (BICG) [3]

The CG method cannot be applied for nonsymmetric systems because the residual vectors cannot be made ortogonal with short recurrences. The BICG replaces ortogonal sequence of residuals by two mutually ortogonal sequences, at the price of no longer providing minimization. The update relations in BICG are the same as in CG but based also on A^T.

- Application:
  For nonsymmetric matrix A. (The method can fail because there are divisions on some inner products which can become zero and cause a breakdown. But still there are some tricks to avoid the breakdowns.)
- Computations per iteration:
  The same as in CG but twice more. And a transpose of matrix A is required.
- Storage:
  8 additional vectors of size N. (and may be A^T depending on implementation)

1.1.3 BiConjugate Gradient Method (BICGSTAB) [3]

The BICGSTAB method was developed to solve nonsymmetric systems while avoiding irregular convergence patterns. BICGSTAB computes $r^{(i)} = Q_i(A)P_i(A)r^{(0)}$, where Q_i is as ith degree polynomial describing a steepest descent update. This method often provides a speed of covergence twice faster than BICG.

- Application:
  For nonsymmetric matrix A. (The method can fail because there are divisions on some inner products which can become zero and cause a breakdown. But there are some tricks to avoid breakdowns.)
- Computations per iteration:
  Two matrix-vector products, four inner products, for vector updates.
- Storage:
  8 additional vectors of size N.

1.1.4 Generalized Minimal Residual (GMRES) [3]

This method is applicable for nonsymmetric systems and has no breakdown situations. It constructs a sequence of ortogonal basis vectors of Krylov subspace, but in the absence of symmetry this can't be done with short recurrence, so all previously computed vectors are stored. That is why restarted method is used. Only the first M vectors are to compute at each iteration. After computing this vectors a problem of minimization is being solved. The problem leads to the system of equations with Hessenberg matrix MxM. The Householder transformations are used to transform this matrix into upper-triangle form and then the system is solved with back substitution. After constructing ith iterate $x^{(i)}$ method restarts.

- Application:
  For nonsymmetric matrix A. Method is free of breakdown situations.
- Computations per iteration:
  At least M+1 matrix-vector products, $M^2/2$ inner products and vector updates. (And there are some more operations with objects of size M and MxM)
- Storage:
  More than M+1 vectors of size N, a number of vectors of size M and Hessebberg matrix MxM.

1.2 Implementation issues

1.2.1 Choise of sparse matrix realization

This methods were done as a template functions providing replacement of matrix class without changing a code. Two kind of sparse matrix were tested:

- Manel's SNDM class: Matrix is represented as a vector of sparse vectors wich are based on a map. It was designed for nonstructured mesh and was expected to have lower performance.
- Common representation as a set of vectors Ap,Aw,Ae,As,An. For details see [1]

Time of one iteration using these methods was measured for each of this two kind of matrices. Problem description is of no value in this case.
Methods: CG, BICG, BICGSTAB, Restarted GMRES with restart m=50.

Results of the tests are following:

Table 1: Mesh 100x100

Method	sndm	common	acceleration, times
CG	0.0216	0.00227	9.5
BI-CG	0.041	0.0045	9.1
BI-CGSTAB	0.042	0.0047	8.9
GMRES	1.31	0.34	3.9

Table 2: Mesh 200x200

Method	sndm	common	acceleration, times
CG	0.084	0.011	7.6
BI-CG	0.163	0.019	8.6
BI-CGSTAB	0.167	0.023	7.3
GMRES	6.6	2.6	2.5

So the use of SNDM matrix can cause up to 10 times decrease of performance. In all following computations the common matrix representation is used.

1.2.2 Additional containers and functions

The methods CG, BICG, BICGSTAB don't require additional containers. This methods were done as template functions like this:

template < class m, class v, class p >

where m - matrix class, v - vector class, p - preconditioner class.

One additional vector operation was created: expr(v X, v Y, double a, double b)
This operation computes $X = aX + bY$.

Containers and operations for GMRES:

- Hessenberg matrix
  This matrix is represented as a set of vectors with numbers i=1,..,N, where ith vector has size i+1; This way provides an economy of storage because zero elements are not stored.
- Householder rotation
  This class is used to represent rotation matrix. Only one integer value and two double values are stored. A sequence of Householder transformations is stored in array of this rotation matrices.
  For this class only one additional operation was used:
  v & mult(HouseRot &J, v& h) - applies rotation to the vector h.
  This operation requres only 4 multiplications of double.
  (Note: all this additional operations are not tempate functions yet.)

2 Performance tests with Jacobi preconditioner.

2.1 Diffusion problem

2.1.1 Problem description

The system of linear equations:

$$Ax = b,$$

where $A \in R^{N*N}$, $x, b \in R^N$.
In this case A is a common pentadiagonal matrix given in terms of vectors Ap, Aw, Ae, As, An.
Eech of this vectors is given in a matrix form. Let $N = n * n$. All this vectors can be represented as $Ap_k = a^p_{i,j}$, $Aw_k = a^p_{i,j}$, $Ae_k = a^p_{i,j}$, $As_k = a^p_{i,j}$, $An_k = a^p_{i,j}$, where $k = i + n(j-1)$, $i = 1..n$, $j = 1..n$.
In this notation matrix A is following:

$$\begin{cases} a^p_{i,j} = 1\ ,\ a^e_{i,j} = -1,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = 1,\ j = 1..n \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = -1,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = n,\ j = 1..n \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = -1,\ a^s_{i,j} = 0,\ i = 1..n,\ j = 1 \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = -1,\ i = 1..n,\ j = n \\ a^p_{i,j} = 8\ ,\ a^e_{i,j} = -1,\ a^w_{i,j} = -1,\ a^n_{i,j} = -1,\ a^s_{i,j} = -1,\ i = n/2,\ j = n/2 \\ \textit{In all other poins}: \\ a^p_{i,j} = 4\ ,\ a^e_{i,j} = -1,\ a^w_{i,j} = -1,\ a^n_{i,j} = -1,\ a^s_{i,j} = -1. \\ b_i = 0,\ i = 1..N \end{cases}$$

This matrix is not symmetric so conjugate gradient method is inapplicable. The matrix can be easily made symmetric by following transformations:
New matrix A_{sym} of size (n-2)x(n-2) is made from A by throwing out boundary points. (It is possible because of "free" boundary conditions.) It is done like showed in this picture.

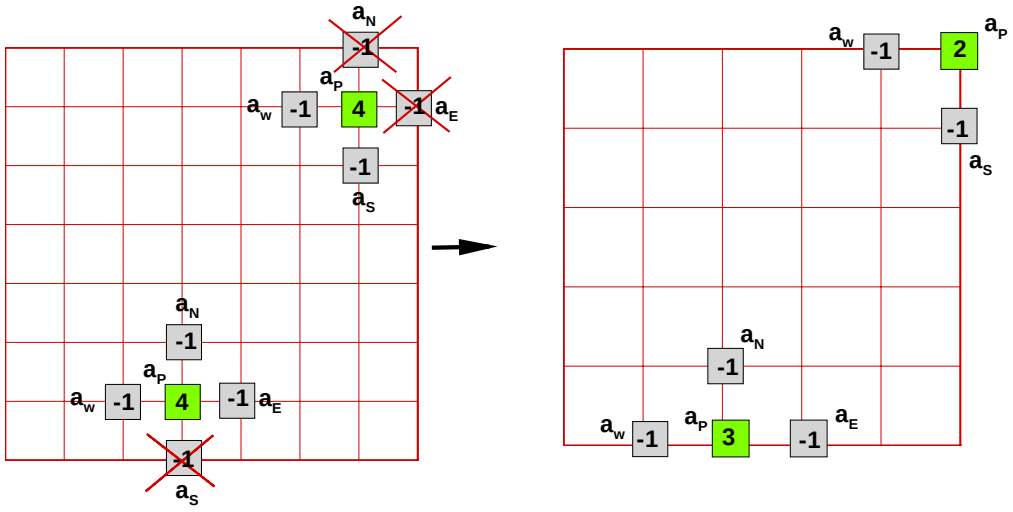


Figure 1: Throwing out boundary poins.

This symmetric matrix is used instead of previous nonsymmetric in all the following computations.

Initial guess: $x^{(0)}_{i+n(j-1)} = cos(2\pi x_i) + cos(2\pi y_j)$, $x_i = (i-1)/(n-1)$, $y_j = (j-1)/(n-1)$, $i = 1..n$, $j = 1..n$.
Condition of finishing iterative process is:

$$\frac{||Ax^{(0)} - b||}{||Ax^{(i)} - b||} < 10^{-6},$$

where $x^{(0)}$ is the initial guess and $x^{(i)}$ is the ith iterate.

2.1.2 Results

In all the tables below "N of cv" means number of control volumes, "Time" means CPU time, "N it." means number of iterations. Here in the table are the results of the methods with Jacobi preconditioner and without a preconditioner.

Table 3: Conjugate gradient methods

	CG		BICG		BICGSTAB	
N of cv	Time	N It.	Time	N It.	Time	N It.
100 (10x10)	0.001	47	0.002	47	0.001	34
100 (10x10) Precond.	0.001	42	0.01	42	0.001	32
1024 (32x32)	0.016	137	0.03	137	0.026	106
1024 (32x32) Precond.	0.018	135	0.035	135	0.028	103
2500 (50x50)	0.058	190	0.124	190	0.104	161
2500 (50x50) Precond.	0.069	199	0.143	199	0.111	150
4096 (64x64)	0.221	241	0.42	241	0.383	200
4096 (64x64) Precond.	0.222	239	0.45	239	0.399	202
10000 (100x100)	0.68	372	1.35	372	1.2	302
10000 (100x100) Precond.	0.72	369	1.42	369	1.2	298
14400 (120x120)	1.22	444	2.46	444	2.16	365
14400 (120x120) Precond.	1.31	442	2.56	442	2.26	367
22500 (150x150)	3.54	545	6.39	545	5.9	434
22500 (150x150) Precond.	3.73	549	6.88	549	6.1	434
35721 (189x189)	5.98	683	10.2	683	11.03	573
35721 (189x189) Precond.	6.20	691	10.8	691	10.1	547
51076 (226x226)	11.9	799	21.39	799	20.7	638
51076 (226x226) Precond.	12.9	814	23.2	814	22.3	670
101124 (318x318)	27.45	1106	46.96	1106	44.19	797
101124 (318x318) Precond.	28.8	1123	49.6	1123	43.06	755
149769 (387x387)	48.74	1353	83.9	1353	100.9	1262
149769 (387x387) Precond.	51.47	1364	88.9	1364	101.1	1235
201601 (449x449)	76.14	1557	130.6	1557	137.8	1230
201601 (449x449) Precond.	79.8	1569	137.1	1569	162.0	1443
251001 (501x501)	105.8	1732	181.8	1732	181.9	1402
251001 (501x501) Precond.	109.6	1733	189.1	1733	186.8	1349
301401 (549x549)	138.7	1893	238.5	1893	202.6	1299
301401 (549x549) Precond.	142.8	1886	245.9	1886	261.8	1596

It looks like Jacobi preconditioner gives no effect in this case.

GMRES was tested using 7 values of restart length: 35, 55, 100, 170, 240,300,380. Restart length means the number of basis vectors of Krylov subspace to construct at one iteration. Here in the tables are the results of the GMRES(m) with Jacobi preconditioner and without a preconditioner. Minimal CPU time is marked bold in following table.

Table 4: GMRES(m) CPU time

N of cv	m=35	m=55	m=100	m=170	m=240	m=300	m=380
100 (10x10)	**0.002**	0.003	0.007	0.018	0.034	0.052	0.057
100 (10x10 Precond.	**0.003**	0.003	0.006	0.017	0.034	0.053	0.060
1024 (32x32)	0.157	**0.122**	0.199	0.28	0.546	0.83	1.325
1024 (32x32) Precond.	0.165	**0.123**	0.199	0.28	0.545	0.835	1.323
2500 (50x50)	1.662	0.849	**0.756**	1.366	1.33	2.04	3.22
2500 (50x50 Precond.)	1.666	0.848	**0.758**	1.366	1.329	2.042	3.221
4096 (64x64)	3.716	3.638	3.025	2.307	**2.224**	3.399	5.351
4096 (64x64) Precond.	3.715	3.633	3.024	2.317	**2.223**	3.399	5.353
10000 (100x100)	27.49	17.855	11.399	**9.315**	12.022	18.442	14.668
10000 (100x100) Precond.	27.458	17.868	11.389	**9.307**	12.019	18.534	14.66
14400 (120x120)	55.2	49.46	43.334	27.348	**26.609**	27.066	42.875
14400 (120x120) Precond.	54.29	56.197	43.438	27.084	**26.117**	26.811	42.315
22500 (150x150)	201.081	110.72	168.637	**95.136**	104.409	119.812	128.11
22500 (150x150) Precond.	205.7	108.335	166.4	**95.38**	105.98	126.495	130.65
35721 (189x189)	552.719	523.286	482.545	369.47	452.244	351.247	**333.96**
35721 (189x189) Precond.	552.846	522.105	482.811	368.916	451.548	350.18	**334.7**
51076 (226x226)	1104.39	918.204	747.821	757.623	**450.39**	497.173	473.815
51076 (226x226 Precond.	1102.35	916.606	747.14	758.041	**448.865**	496.787	473.794
101124 (318x318)	3652.39	3418.73	3334.07	3062.97	2419.83	**1775.78**	2512.88
101124 (318x318) Precond.	3663.94	3415.34	3322.43	3061.63	2419.44	**1781.89**	2202.18

Table 5: GMRES(m) number of iterations (equal for both kind of preconditioner)

N of cv	m=35	m=55	m=100	m=170	m=240	m=300	m=380
100 (10x10)	2	1	1	1	1	1	1
1024 (32x32)	14	4	2	1	1	1	1
2500 (50x50)	43	10	3	2	1	1	1
4096 (64x64)	51	24	7	2	1	1	1
10000 (100x100)	153	46	10	3	2	2	1
14400 (120x120)	206	86	26	6	3	2	2
22500 (150x150)	273	106	34	7	4	3	2
35721 (189x189)	448	191	58	16	10	5	3
51076 (226x226)	601	228	62	23	7	5	35
101124 (318x318)	1041	440	141	47	19	9	8

In this case Jacobi preconditioner also gives no effect.

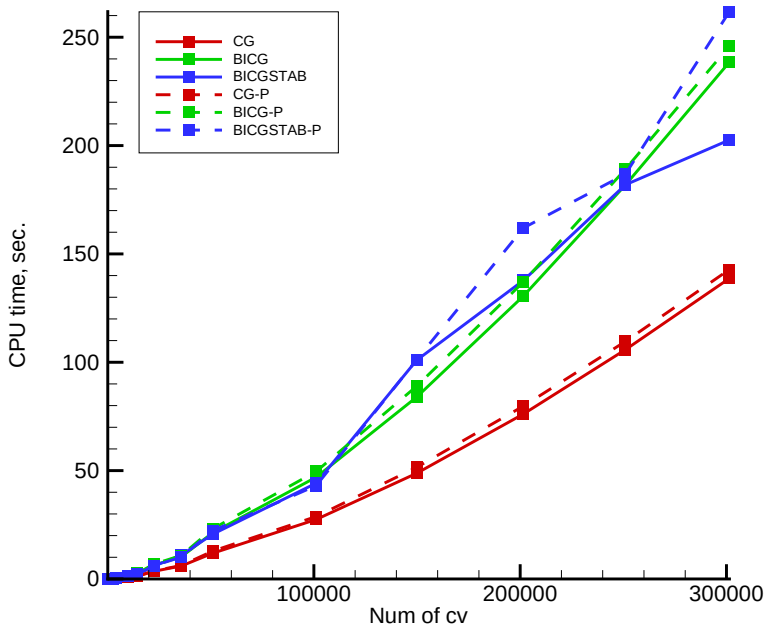


Figure 2: CG methods. (Methods with preconditioner are marked -P)

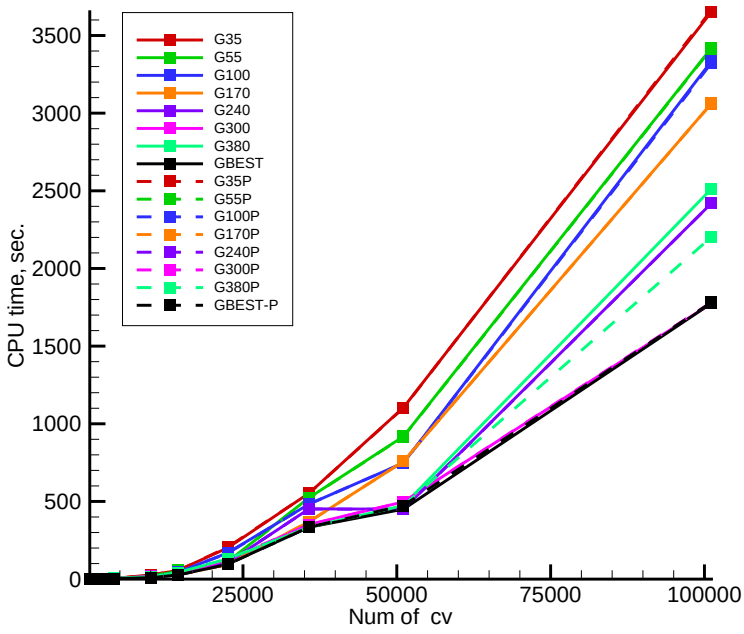


Figure 3: GMRES(m). (Methods with preconditioner are marked P)

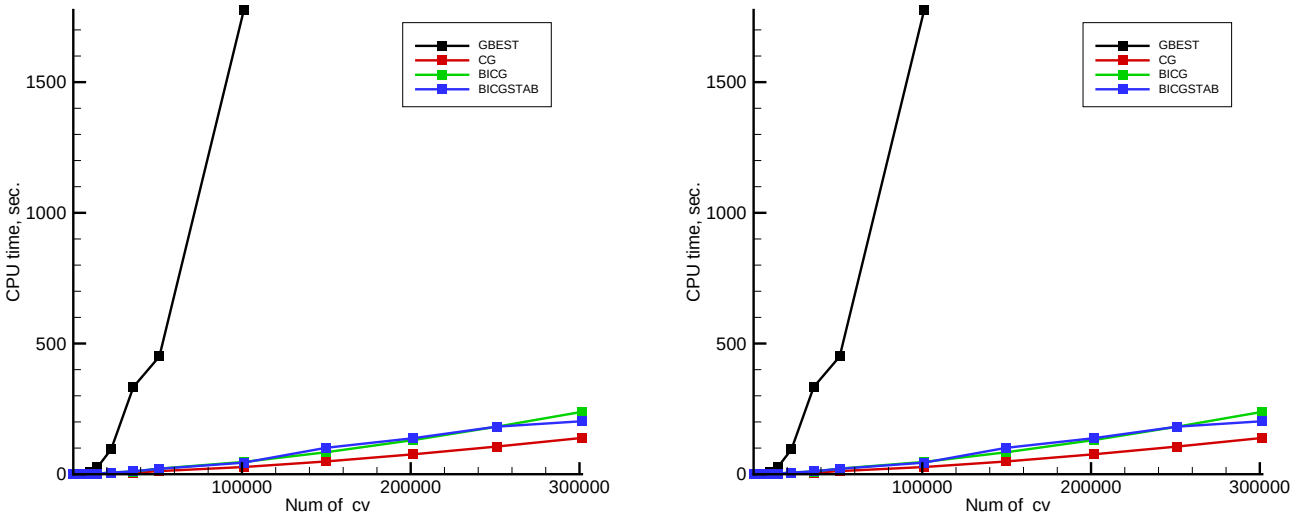


Figure 4: Comparison of CG methods and GMRES(BEST) (no preconditioner). Right graph is in logscale.

2.1.3 Comparison with multigrid

Here are the CPU time measurments of multigrid solver applied to the same test problem. (See [4]) In the table below CPU time of the multigrid solver based on Gauss-Seidel plus TDMA solver is compared with CG, BICG, BICGSTAB and GMRES that uses the best of these restart parameters (35,55,100,170,240,300,380) for each mesh size. No precontitioner used.

Table 6: CPU time of the multigrid solver compared with Krylov subspace methods

N of cv	GSTDMA-ACM	CG	BICG	BICGSTAB	GMRES(best)
100 (10x10)	0.005	0.001	0.002	0.001	0.002
1024 (32x32)	0.046	0.016	0.03	0.026	0.122
2500 (50x50)	0.239	0.058	0.124	0.104	0.756
4096 (64x64)	0.351	0.221	0.42	0.383	2.224
10000 (100x100)	1.724	0.68	1.35	1.2	9.315
14400 (120x120)	1.723	1.22	2.46	2.16	26.6
22500 (150x150)	4.292	3.54	6.39	5.9	95.1
35721 (189x189)	6.066	5.98	10.2	11.03	334
51076 (226x226)	9.749	11.9	21.39	20.7	450
101124 (318x318)	19.684	27.45	46.96	44.19	449
149769 (387x387)	31.875	48.74	83.9	100.9	1776
201601 (449x449)	51.006	76.14	130.6	137.8	
251001 (501x501)	71.918	105.8	181.8	181.9	
301401 (549x549)	87.764	138.7	238.5	202.6	

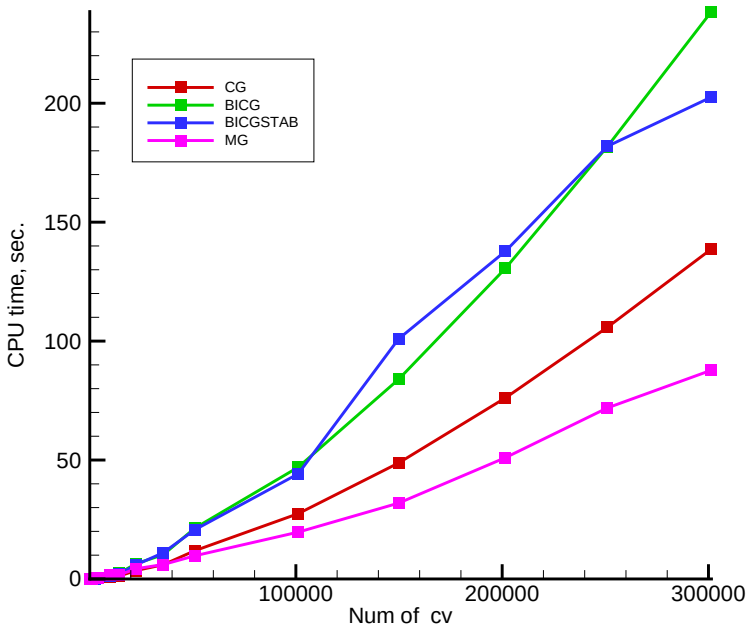


Figure 5: Comparison of multigrid and CG-methods.

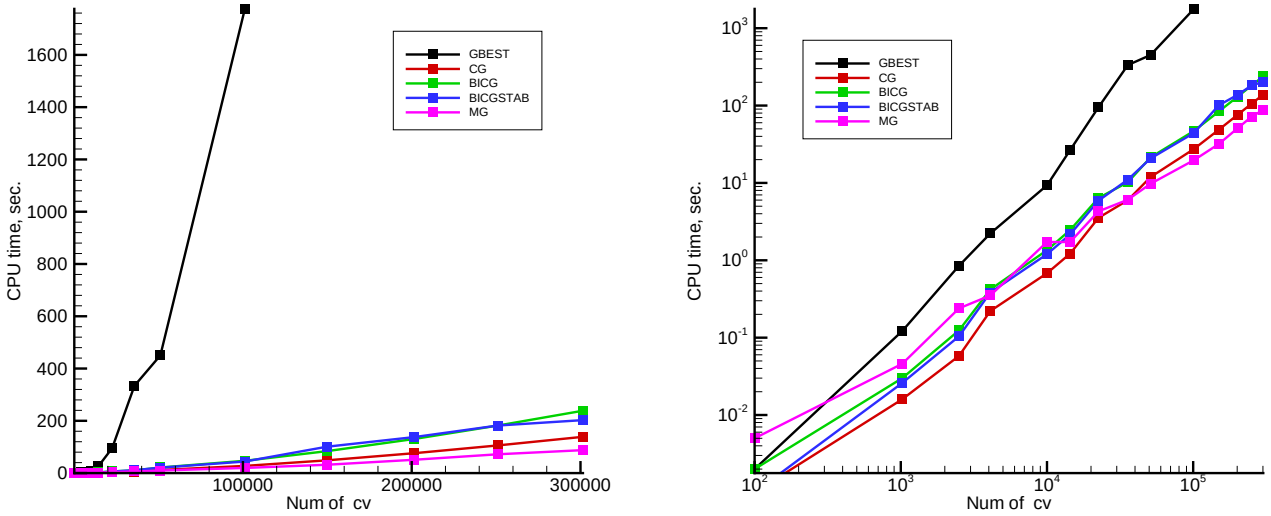


Figure 6: Comparison of multigrid and krylov subspace methods. Right graph is in log scale.

2.2 Convection problem (solenoidal velocity field)

2.2.1 Problem description

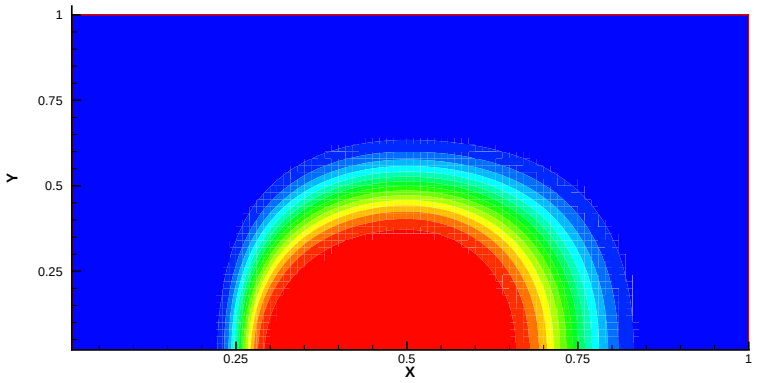


Figure 7: Convection in solenoidal velocity field

For details about the problem see [6]

The system of linear equations:

$$Ax = b,$$

where $A \in R^{N*N},\ x, b \in R^N$.
In this case A is a common pentadiagonal matrix given in terms of vectors Ap, Aw, Ae, As, An.
Eech of this vectors is given in a matrix form. Let $N = n * n$. All this vectors can be represented as $Ap_k = a^p_{i,j},\ \ Aw_k = a^p_{i,j},\ \ Ae_k = a^p_{i,j},\ \ As_k = a^p_{i,j},\ \ An_k = a^p_{i,j},\ \ b_k = b_{i,j}$, where $k = i + n(j-1),\ i = 1..n,\ j = 1..n$.
In this notation matrix A is following:

$$\begin{cases} a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = 1,\ j = 1..n \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = n,\ j = 1..n \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = 1..n/2,\ j = 1 \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = -1,\ a^s_{i,j} = 0,\ i = n/2+1..n,\ j = 1 \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = 1..n,\ j = n \\ a^p_{i,j} = 1\ ,\ a^e_{i,j} = 0,\ a^w_{i,j} = 0,\ a^n_{i,j} = 0,\ a^s_{i,j} = 0,\ i = n/2,\ j = n/2 \\ In\ all\ other\ poins: \\ a^e_{i,j} = D_e * S(P_e) + max(0, -F_e) \\ a^w_{i,j} = D_w * S(P_w) + max(0, -F_w) \\ a^s_{i,j} = D_s * S(P_s) + max(0, -F_s) \\ a^n_{i,j} = D_n * S(P_n) + max(0, -F_n) \\ a^p_{i,j} = -(a^e_{i,j} + a^w_{i,j} + a^s_{i,j} a^n_{i,j}); \end{cases}$$

where
$F_e = u * dy$
$F_w = u * dy$
$F_n = v * dx$
$F_s = v * dx$

$D_e = dy/dx$
$D_w = dy/dx$
$D_n = dx/dy$
$D_s = dx/dy$

$P_e = F_e/D_e$
$P_w = F_w/D_w$

$P_n = F_n/D_n$
$P_s = F_s/D_s$

$x = (i-1) * dx$
$y = (j-1) * dy$

$dx = 2.0/(Nx-1)$
$dy = 1.0/(Ny-1)$

$u = Pe * y * (1-(x-1)*(x-1))$
$v = -Pe * (x-1) * (1-y*y)$

$Pe = 10000$ - Peclet number

$S(P) = 1$ for any P (Upwind scheme)

Vector b is following:

$$\begin{cases} b_{i,j} = 1 + tanh(10(2((i-1)dx-1)+1)),\ i = 1..n/2,\ j = 1 \\ b_{i,j} = 0,\ i = n/2..n,\ j = 1 \\ b_{i,j} = 1 - tanh(10),\ i = 1..n,\ j = n \\ b_{i,j} = 1 - tanh(10),\ i = 1,\ j = 1..n \\ b_{i,j} = 1 - tanh(10),\ i = n,\ j = 1..n \\ In\ all\ other\ poins: \\ b_{i,j} = 0; \end{cases}$$

Initial guess: $x^{(0)}_{i+n(j-1)} = cos(2\pi x_i) + cos(2\pi y_j)$, $x_i = (i-1)/(n-1)$, $y_j = (j-1)/(n-1)$, $i = 1..n$, $j = 1..n$.
Condition of finishing iterative process is:

$$\frac{||Ax^{(0)} - b||}{||Ax^{(i)} - b||} < 10^{-6},$$

where $x^{(0)}$ is the initial guess and $x^{(i)}$ is the ith iterate.

2.2.2 Results using no preconditioner

In all the tables below "N of cv" means number of control volumes, "Time" means CPU time, "N it." means number of iterations.
Conjugate gradient methods failed to converge on on mesh 150x75 and bigger.

Table 7: Conjugate gradient methods without a preconditioner

	BICG		BICGSTAB	
N of cv	Time	N It.	Time	N It.
200 (20x10)	0.009	84	0.008	70
800 (40x20)	0.056	177	0.059	235
1800 (60x30)	0.202	284	0.314	579
3200 (80x40)	0.698	445	1.184	988
5000 (100x50)	1.734	634	2.838	1452
11250 (150x75)	—	—	—	—
20000 (200x100)	—	—	—	—
45000 (300x150)	—	—	—	—
80000 (400x200)	—	—	—	—

GMRES was tested using 5 values of restart length: 50, 100, 200, 300, 400. Restart length means the number of basis vectors of Krylov subspace to construct at one iteration. GMRES with restart parameters 50 and 100 failed to converge on mesh 150x75 and bigger. Minimal CPU time is marked bold in following table.

Table 8: GMRES(m) without preconditioner

	GMRES(50)		GMRES(100)		GMRES(200)		GMRES(300)		GMRES(400)	
N of cv	Time	N It.	Time	N It.	Time	N It.	Time	N It.	Time	N It.
200 (20x10)	**0.008**	2	0.014	1	0.051	1	0.145	1	0.295	1
800 (40x20)	0.28	16	**0.15**	2	0.3	1	0.65	1	1.1	1
1800 (60x30)	0.8	16	1.3	7	**0.7**	1	1.5	1	2.6	1
3200 (80x40)	**2.0**	22	2.9	9	2.6	2	2.8	1	4.5	1
5000 (100x50)	6.1	39	7.4	14	5.8	3	**4.3**	1	8.2	1
11250 (150x75)	—	—	**21.8**	17	47.6	10	42	4	36.5	2
20000 (200x100)	—	—	**85**	21	175.3	12	284.4	8	232	4
45000 (300x150)	—	—	**354**	34	680	16	1140	13	1608	10
80000 (400x200)	—	—	**992**	49	1859	26	2862	18	3920	14

2.2.3 Results using Jacobi preconditioner (diagonal scaling)

Here acceleration means relation between CPU time without preconditioner and the CPU time when using Jacobi preconditioner.

Table 9: Conjugate gradient methods with Jacobi preconditioner

	BICG		BICGSTAB	
N of cv	Time (acceleration)	N It.	Time (acceleration)	N It.
200 (20x10)	0.005 (1.8)	36	0.004 (2.0)	33
800 (40x20)	0.03 (1.86)	86	0.022 (2.68)	78
1800 (60x30)	0.103 (1.96)	136	0.075 (4.19)	125
3200 (80x40)	0.306 (2.28)	180	0.223 (5.31)	176
5000 (100x50)	1.41 (1.2)	473	0.501 (5.66)	234
11250 (150x75)	13.5	1706	1.97	380
20000 (200x100)	49.1	2510	7.76	510
45000 (300x150)	134.3	3166	26.47	745
80000 (400x200)	218	3847	41.4	931
125000 (500x250)	> 1000	> 10000	79.5	1133

GMRES was tested using 5 values of restart length: 3, 4, 5, 10, 15. Minimal CPU time is marked bold in following table.

Table 10: GMRES(m) with Jacobi preconditioner

	GMRES(3)		GMRES(4)		GMRES(5)		GMRES(10)		GMRES(15)	
N of cv	Time	N It.	Time	N It.	Time	N It.	Time	N It.	Time	N It.
200 (20x10)	**0.004**	21	0.004	17	0.004	15	0.006	9	0.005	6
800 (40x20)	**0.021**	38	0.022	31	0.022	26	0.027	16	0.032	12
1800 (60x30)	**0.067**	55	0.077	45	0.069	36	0.084	22	0.122	18
1800 (80x40)	0.18	71	**0.17**	57	0.18	48	0.24	28	0.32	21
3200 (100x50)	**0.39**	89	0.39	69	0.45	58	0.55	33	0.63	24
5000 (150x75)	**1.39**	127	1.39	100	1.44	84	1.68	47	2.0	33
20000 (200x100)	5.65	166	**5.33**	130	5.8	108	6.21	60	8.46	43
45000 (300x150)	**18.5**	243	18.6	188	19.1	155	23.5	85	28.4	61
80000 (400x200)	**31.8**	316	32.7	247	34.3	202	44.1	110	55.6	78
125000 (500x250)	**59.6**	389	61.7	303	64.4	250	85.2	135	106	96

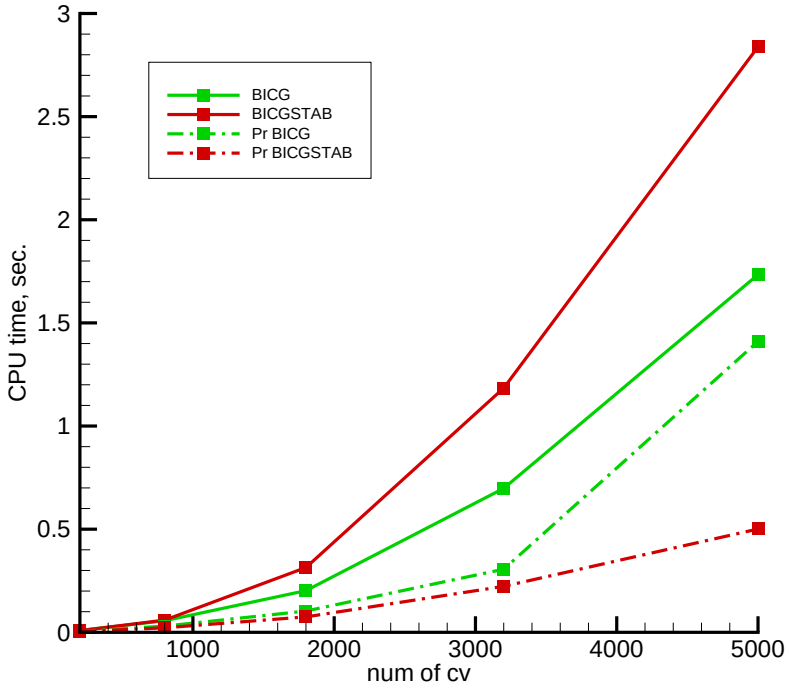


Figure 8: Comparison of BICG and BICGSTAB when using Jacobi preconditioner and without preconditioner. Note that maximum cv number is only 5000 due to fail of not preconditioned methods.

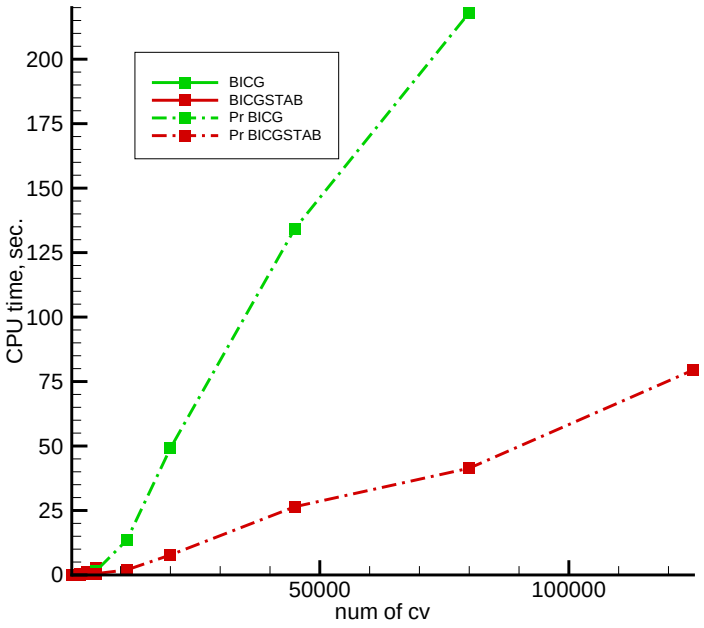


Figure 9: Comparison of BICG and BICGSTAB when using Jacobi preconditioner.

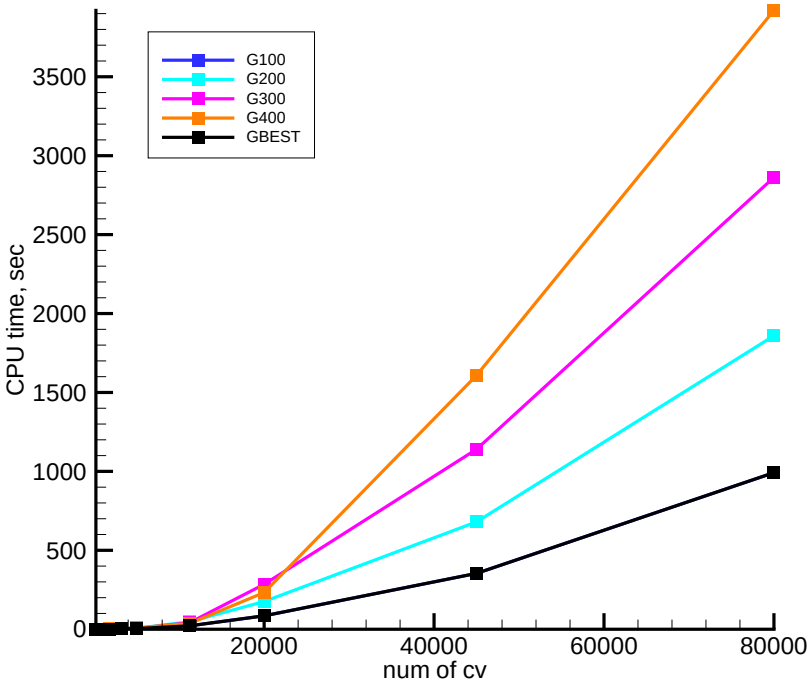


Figure 10: Comparison of GMRES(m) without preconditioner for different m

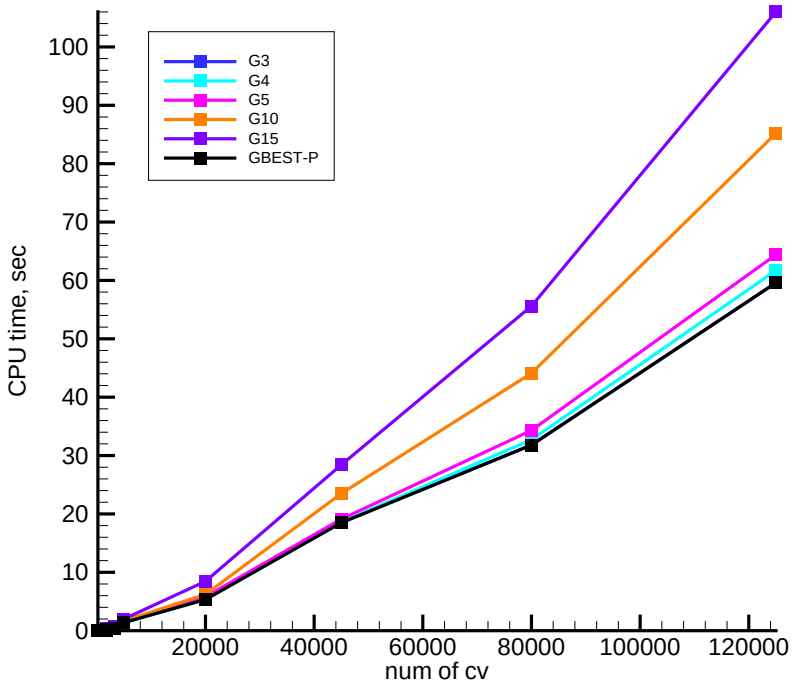


Figure 11: Comparison of GMRES(m) with Jacobi preconditioner for different m

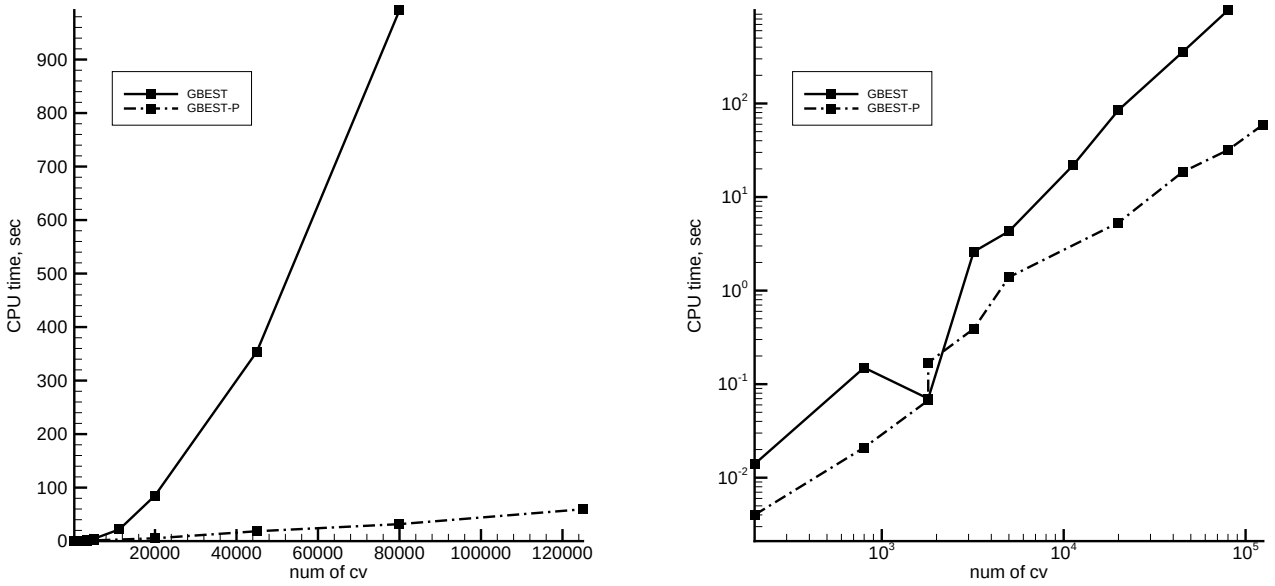


Figure 12: Comparison of GMRES(best m of 3,4,5,10,15) when using Jacobi preconditioner (G BEST-P) and GMRES(best m of 50,100,200,300,400) without preconditioner (G BEST). (Common scale - left, log scale - right)

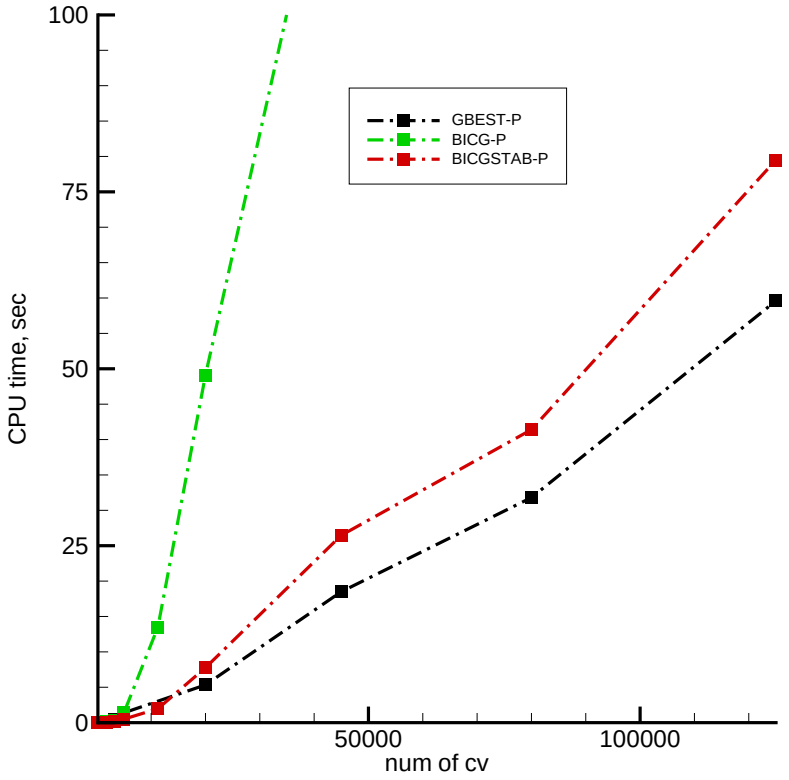


Figure 13: Comparison of GMRES(best m) and conjugate gradient methods when using Jacobi preconditioner.

2.3 Comparison with multigrid

Here are the CPU time measurments of multigrid solver applied to the same test problem. (See [4]) In the table below CPU time of the multigrid solver based on Gauss-Seidel plus TDMA solver is compared with BICGSTAB and GMRES that uses the best of these five restart parameters (3,4,5,10,15) for each mesh size.

Table 11: CPU time of the multigrid solver compared with Krylov subspace methods

N of cv	GSTDMA-ACM	BICGSTAB	GMRES(best)
200 (20x10)	0.001	0.004	0.004
800 (40x20)	0.004	0.022	0.021
1800 (80x40)	0.012	0.223	0.17
3200 (100x50)	0.023	0.501	0.39
5000 (150x75)	0.082	1.97	1.39
20000 (200x100)	0.280	7.76	5.33
45000 (300x150)	0.847	26.47	18.5
80000 (400x200)	1.460	41.4	31.8
125000 (500x250)	1.479	79.5	59.6

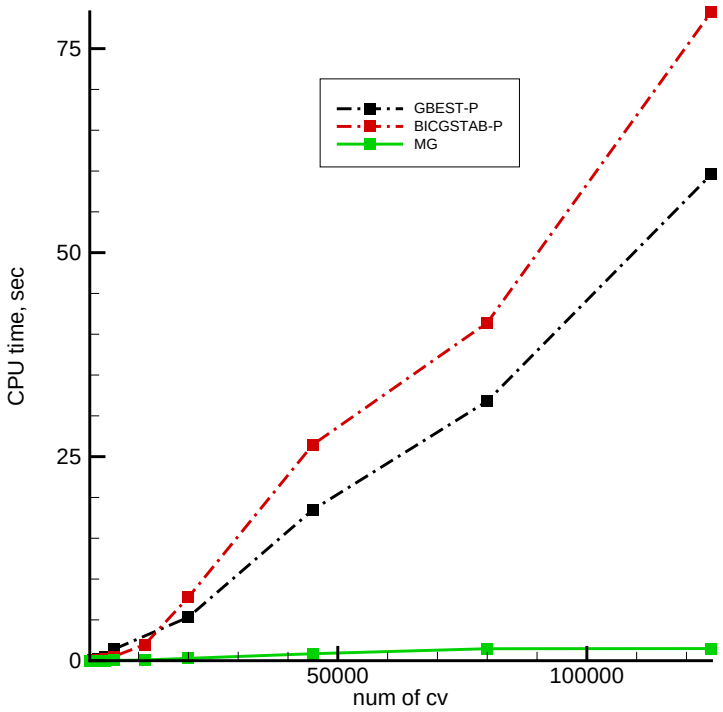


Figure 14: Comparison of GMRES(best m) and conjugate gradient methods when using Jacobi preconditioner with multigrid

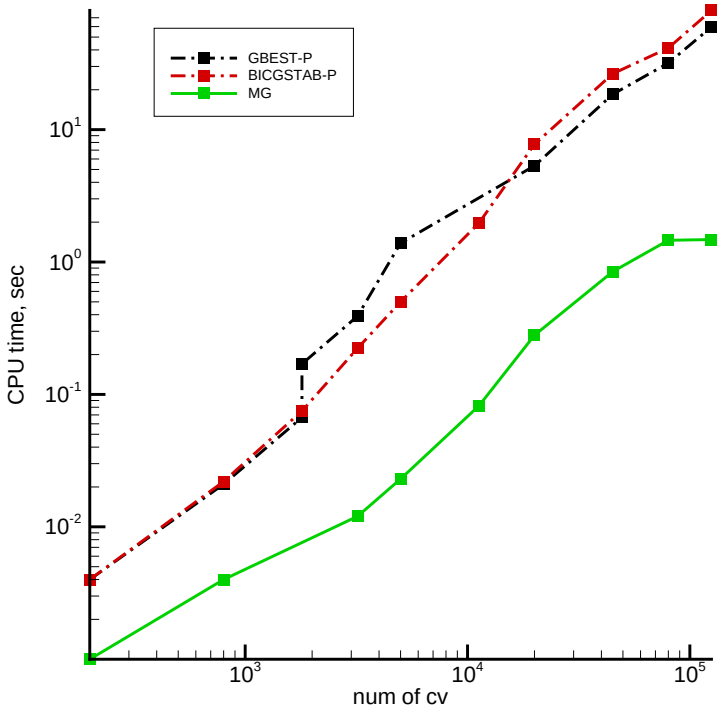


Figure 15: Representation in a log scale

3 Parallel realization

3.1 MPI interface class and additional functions

Special class MPIAssist was designed to make parallel implementation easier. At the present it has been adopted for 2D structured mesh only. Here is a description of functionality of this class. The processor on wich the function is being called will be denoted as current processor, and "Me" is the number of the current processor. Current subdomain means the subdomain that belongs to the current processor. Global number means the number of node in whole domain. Local number means the number of node in current subdomain. List of member functions is given below.

Initialization functions:

- MPIassist(int argc, char **argv, int sx, int sy)
  Contsructor initialises MPI, calls decomposition function and then sets MPI buffer size for asynchronous data transfer.
- void Decompose()
  This decomposition function uses N_x - number of nodes on x axis in whole domain, N_y -number of nodes on y axis in whole domain , NP - numper of processors to determine NP_x and NP_y - numbers of processors on x and y axises. Thus $NP_x * NP_y = NP$. Then this function determines following values using the number of current processor:
  - MeX - number of current processor on x axise
  - MeY - number of current processor on y axise
  - MyNx - size of current subdomain on x axise
  - MyNy - size of current subdomain on y axise
  - Ibeg - global number of first node of current subdomain on x axise
  - Jbeg - global number of first node of current subdomain on y axise

  If the function fails (no possible decomposition for given domain size and number of processors) program stops. Details about domain decompositon are in the next subsection.

Functions to access class data members:

- inline int Me() - returns Me
- inline int MeX() - returns MeX
- inline int MeY() - returns MeY
- inline int NP()- returns NP
- inline int NPX() - returns NP_x
- inline int NPY() - returns NP_y
- inline int MyNx() - returns MyNx
- inline int MyNy() - returns MyNy
- inline int GlobalNx() - returns N_x
- inline int GlobalNy() - returns N_y
- inline int Ibeg() - returns the global number of first node of current subdomain on x axise
- inline int Jbeg()- returns the global number of first node of current subdomain on y axise
- inline int Iend()- returns the global number of last node of current subdomain on x axise
- inline int Jend()- returns the global number of last node of current subdomain on y axise

Function to process MPI calls:

- inline void mpi_CheckErr(int r,char *txt)
  Function checks r - return value of MPI function. If r is not a value of successful completion the function prints an error message with name of failed MPI function (txt parameter).

- inline int Finalize() - calls MPI_Finalize().

Functions to update halo:
(Details about data transmission are in the next subsection.)

- void UpdateHalo(dv & X, int hs)
  Function updates halo of vector X. hs is the width of the halo. This functions is to be used for nonoverlapped data transfer. It allocates buffers and completes data transfer. (overlapped means buffered data transfer is overlapped by computaions)

- void SetBuf(int hs, MPIBuffers &BUF)
  Preallocates buffer BUF for overlapped data transfer.

- void UpdateHaloSend(dv & X, int hs, MPI_Status *sts , MPI_Request *rr ,MPIBuffers&BUF)
  Function starts asynchronous buffered data transfer to update halo of vector X. hs is the width of the halo. This functions is to be used for overlapped data transfer. Buffer BUF must be allocated by previous function before using this function. This function is used together with following function.

- void UpdateHaloRecv(dv & X, int hs, MPI_Status *sts , MPI_Request *rr ,MPIBuffers &BUF)
  Function waits while all the data transfer started by previous function is finished. Then it updates the halo of vector x.

- void DeleteBuf(MPIBuffers &BUF)
  Deletes buffer allocated by SetBuf.

Then there are additional non-member functions:

- long LocalGlobal(long i, int Nx, int Ny, int mNx, int mNy, int IBEG, int JBEG, int hs)
  Returns the global number of node that corresponds to i - the number of element in a vector. If vector has no halo then i is equal to local number of the node.

- long GlobalLocal(long i, int Nx, int Ny, int mNx, int mNy, int IBEG, int JBEG, int hs) Returns the number of element in a vector that corresponds to i - global number of node . If vector has no halo function returns local number of the node.

- dv & pmult(MPIassist &MyMPI, Acoef &A, dv& X, dv& Q)
  Parallel multiplication of the common pentadiagonal matrix A on vector X. Q is a resulting vector. This function uses nonoverlapped data transfer.

- dv & pmult2(MPIassist &MyMPI, Acoef &A, dv& X, dv& Q)
  The same as previous but uses overlapped data transfer.

- double pdot(MPIassist &MyMPI, dv &a,dv &b)
  Parallel inner vector product for vectors of equal size.

- double pdot(MPIassist &MyMPI, dv &a,dv &b, int hs)
  Parallel inner vector product for vectors of different size. It is used when one of the vectors has a halo and another has not. hs is a halo width.

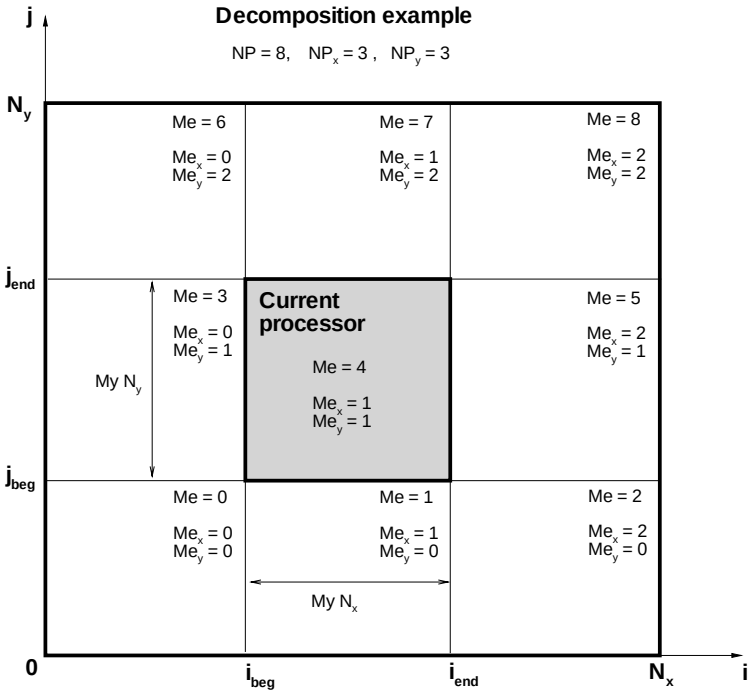
- dv & eq(MPIassist &MyMPI, dv &a,dv &b, int hs)
  This function is used instead of operation a=b when one of the vectors has a halo and another has not. hs is a halo width.

- PWriteResult(MPIassist &MyMPI, dv & X, char *fn)
  Writes data of vector X that correspond to the current subdomain data. It also writes all the additional information to allow junction program to build a single file for whole domain. fn - filename. X must not have a halo. (see subsection 4.6 for details)

3.2 Domain decomposition and data transmission

Decomposition on both axises is used. All subdomains are of equal size in this case. The following labelling is used:

- N_x - number of current processor on x axise
- N_y - number of current processor on x axise
- NP - number of current processor on x axise
- NP_x - number of current processor on x axise
- NP_y - number of current processor on x axise
- MeX - number of current processor on x axise
- MeY - number of current processor on y axise
- MyNx - size of current subdomain on x axise
- MyNy - size of current subdomain on y axise
- Ibeg - global number of first node of current subdomain on x axise
- Jbeg - global number of first node of current subdomain on y axise
- Iend- global number of last node of current subdomain on x axise
- Jend- global number of last node of current subdomain on y axise

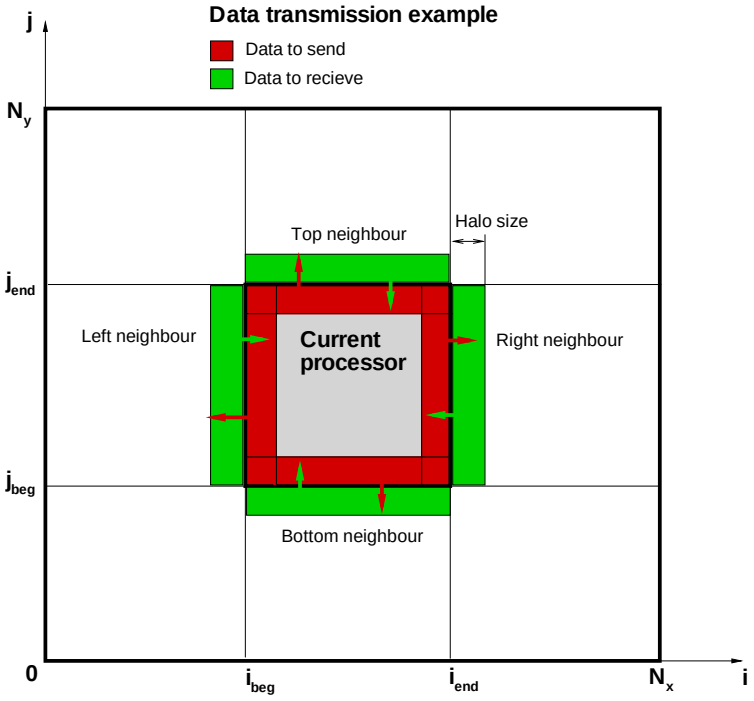


First of all decomposition exists not for all the combinations of domain's size and number of processors. Program is terminated if there is no way to decompose.

Decomposition function finds an optimal NP_x and NP_y for given N_x, N_y and NP. Optimal means minimization of size of data to transfer. The size is proportional to length of borders of subdomain. Square shape of subdomain gives minimal border length. So the problem is to find such a decomposition that subdomains have most "square" shape. It means $|\frac{NP_x}{NP_y} - 1|$ is minimal.

The asynchronous buffered comunication functions MPI_Isend and MPI_Irecv are used for point-to-point data transfer. All send and recieve functions start at the same time when process communicates with its neighbours. Then MPI_Waitall waits until all the transmissions have finished. If overlaping of data transfer by computations is used then MPI_Waitall is placed after all computations that don't require data from halo nodes.

All data to send is placed into send buffers. There are four send buffers - one for each direction. Buffers for left and right directions have size hs*MyNy, buffers for top and bottob directions have size hs*MyNx, where hs is a halo width.



The MPI_Allgather function is used for broadcast communications wich are required for inner products. It seems not the best solution to use this function so it is used temporarily.

3.3 Parallel versions of Krylov subspace methods functions

The only things needed to make sequential functions parallel are the replacement of sequential operations by their parallel versions and correction of vector sizes.

First there are two kind of vectors: vectors with halo and vectors without it. Vector must has a halo only if it is involved in matrix vector product. Thus there are changes in sizes of some vectors. Then calls of matrix vector product function are replaced by calls of parallel function.

Then all inner products are replaced by parallel versions. There are two functions for parallel inner product: first for vectors without halo and second is used when one vector has halo nodes and another has not.

Then vector update operations are modified where vectors with and without halo are involved. Function dv & eq(MPIassist &MyMPI, dv &a,dv &b, int hs) described above is used for conversion of one vector type to another.

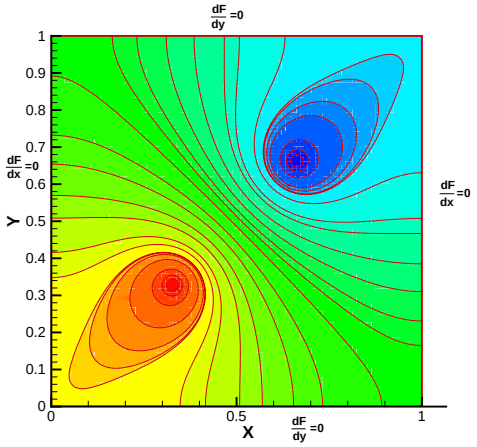
Preconditioner is set to identity matrix in these parallel versions. The preconditioner must be parallelized separately.

3.4 Validaion test for parallel versions of Krylov subspace methods

Test problem description:
Poisson equation:

$$\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} = b$$

Area: $\Omega = (0, L_x) * (0, Ly), \quad L_x = 1, \; L_y = 1$
Boundary conditions: $\dfrac{\partial F}{\partial n} = 0$ everywhere on the boundary
Sources:
$\int\limits_{\Omega} b = 0 \quad b(L_x/3, L_y/3) = 1, \; b(2L_x/3, 2L_y/3) = -1$
Mesh: Uniform, Nx=Ny.



Results of test problem solved by parallel solvers were compared for equality with results of sequential solvers. Residual norms on each iteration and total number of iterations were also compared for equality. All these results obtained using parallel versions with different number of processors were equal to results of sequential versions.

But there was a mystery, that is still not solved. In some case at the last iteration the difference about 10^{-16} between residual norms was found.

3.5 Choice of ortogonalization algorithm for GMRES

There are two versions of Arnoldi algorithm for building ortogonal basis of the Krylov subspace K_m. For details see [4].

- Arnoldi algorithm

  1. Choose a vector v_1 of norm 1
  2. For j=1,2,...,m Do:
  3. Compute $h_{ij} = (Av_j, v_i)$ for i=1,2,...,j
  4. Compute $w_j = Av_j - \sum_{i=1}^{j} h_{ij} v_i$
  5. $h_{j+1,j} = ||w_j||_2$. If $h_{j+1,j} = 0$ then Stop
  6. $v_{j+1} = w_j / h_{j+1,j}$
  7. EndDo

- Arnoldi-Modified Gram-Schmidt

  1. Choose a vector v_1 of norm 1
  2. For j=1,2,...,m Do:
  3. Compute $w_j = Av_j$
  4. For i=1,2,...,j Do:
  5. $h_{ij} = (w_j, v_i)$
  6. $w_j = w_j - h_{ij} v_i$
  7. EndDo
  8. $h_{j+1,j} = ||w_j||_2$. If $h_{j+1,j} = 0$ then Stop
  9. $v_{j+1} = w_j / h_{j+1,j}$
  10. EndDo

This two algorithms are mathematically equivalent but in presence of round-off the second formulation is much more reliable. But in parallel implementation the second version requires j broadcast communications to compute j inner products for each j=1,...,m. So it requires 0.5(m+1)m broadcast communications to build the basis because inner products are coupled with vector updates (lines 5,6 of 2nd). First version requires only m broadcast communications. It is because inner products and update of vector w_j are separated (lines 3,4 of 1st) so all data about j inner products can be transmitted with only one communication.

Both versions were tested in a sequential mode. Results seems to be equal. But there is a mysterious decrease of performance up to 10% when using first one although the number of computational operations seems to be equal. Information about the tests of these algorithms on a parallel computer see in the next section.

3.6 File output

Here is a description of file output function and junction utility.

File output function:

int PWriteResult(MPIassist &MyMPI, dv & X, char *fn)

X - vector without halo, fn - filename. The name of output file will be fn + number of current processor. Output is in binary format. This function first writes a structure with decomposition information:

```
typedef struct DataInfo{
    int me;       current processor
    int mnp;      number of processors
    int mXnp; int mYnp;int mZnp;    number of processors on X, Y and Z axises
    int meX; int meY; int meZ;   number of current processor on X,Y and Z axises
    int Nx; int Ny; int Nz;   size of whole domain on X,Y and Z axises
    int mNx; int mNy; int mNz;  size of subdomain on X,Y and Z axises
    int IBEG; int IEND;   Position of subdomain in global numeration.
    int JBEG; int JEND;
    int KBEG; int KEND;
};
```

Then it writes the data of vector X.

Junction utility:

This program combines files of all subdomains into one file in techplot format. Files must be placed together into one directory. Program call is "combine [filename]". Filename means part of name before number of processor, for example to combine files DATA001, DATA002, DATA003, DATA004, DATA005, command is "combine DATA". If no filename specified then default name "test" is used.

4 Performance tests of Krylov subspace parallel solvers.

4.1 Hardware description

Each node has: 900MHz CPU, 256Kb chache memory, 512Mb RAM.
Network is based on a 100Mbit Fast Ethernet switch.

4.2 Test of matrix-vector product performance. Overlaping of computations and communications

Matrix vector product (MVP) is a primary operation in these methods. Here are the tests of two MVP implementations. The tests show acceleration of each MVP function when using different number of processors.

Common pentadiagonal matrix representation (set of vectors A_P, A_W, A_E, A_S, A_N) was used in these tests. These two kinds of parallel MVP implementations were tested:

- MVP function without overlapping buffered data transmission by computations.
  This function has following algorithm:

  1. Halo update:
     - Copying data to transmit into send buffers
     - Initialization of send and recv buffered transmission functions
     - Waiting this mpi calls to complete
     - Copying data from recieve buffer into halo nodes
  2. Calculation over all nodes of subdomain

- MVP function with overlapping (MVPO)
  This function has following algorithm:

  1. Halo update initialization:
     - Copying data to transmit into send buffers
     - Initialization of send and recv buffered transmission functions
  2. Calculation over all inner nodes of subdomain wich has no halo neighbours
  3. Halo update completion:
     - Waiting this mpi calls to complete
     - Copying data from recieve buffer into halo nodes
  4. Calculation over boundary nodes of subdomain that need data from halo nodes

MVPO was expected to give good increase in acceleration when the number of processors is close to the number that gives maximal acceleration without overlapping. It was also expected to rise the number of processors that gives maximal acceleration. Tests were done using 1,2,4,9,16 and 25 processors and no considerable increase of acceleration was obtained. Probably the number of processors was too low. Another possible reason is the hardware that may be incapable of doing network transactions simultaneously with computations.

In the table below there are a CPU time and acceleration of MVP with no overlapping when using different number of processors. Filed Time in the table is the CPU time of 1000 matrix vector products. Field x is the acceleration (times). Acceleration using N processors means relation between CPU time on one processor and CPU time on N processors.

Table 12: MVP with no overlapping

Num. of proc:	1		2		4		9		16		25	
N of cv	Time	x	Time	x	Time	x	Time	x	Time	x	Time	x
57600	18.8	1	10	1.88	5.79	3.17	2.92	6.47	1.27	14.84	0.9	20.8
230400	74.1	1	39.4	1.88	20.6	3.59	9.26	8	5.69	12.55	3.88	18.36
921600	291.6	1	157.6	1.85	80.3	3.63	37	7.89	21.1	13.82	13.3	21.92
1440000	468	1	254	1.84	194.2	2.41	55.5	8.43	45.5	10.28	21.1	22.18
3686400	1260	1	688	1.83	397	3.17	140.5	8.97	122.6	10.28	52.8	23.86

In the diagram each curve correspond to each mesh size.
Meshes:
240x240 (57600 cv), 480x480 (230400 cv), 960x960 (921600 cv), 1200x1200 (1440000 cv), 1920x1920 (3686400 cv).
Black line is the maximum possible acceleration (number of CPU).

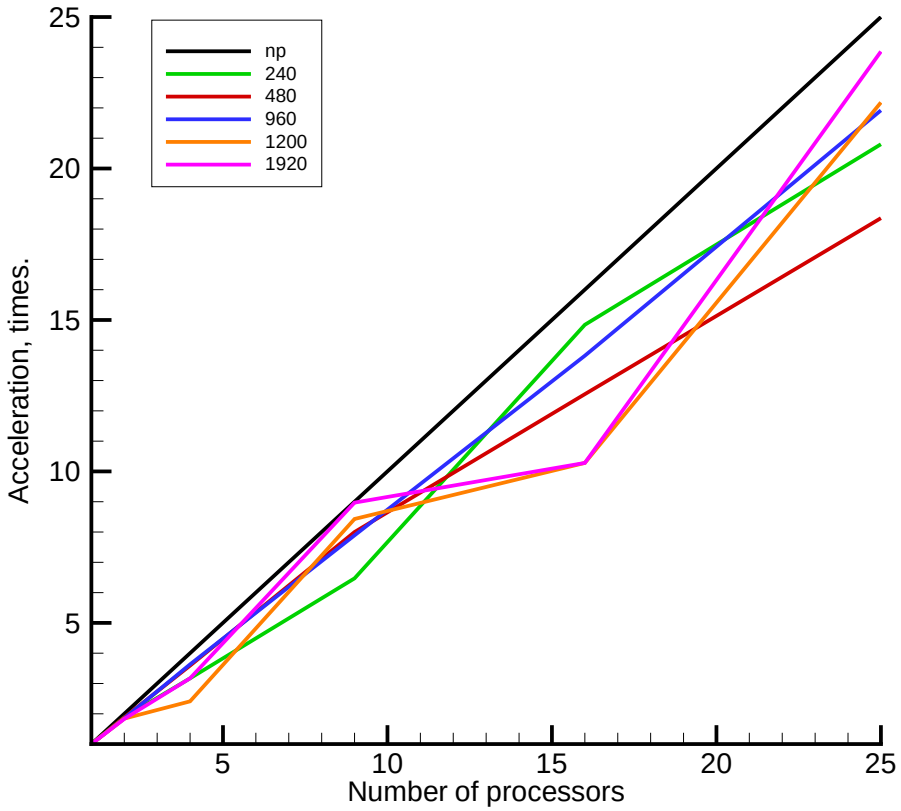


Figure 16: MVP with no overlapping

Here is a comparison of MVP and MVPO for each mesh size. In the following table there is an increase of performance in repcents when using MVPO instead of MVP.

Table 13: MVP with overlapping: % of increase of performance

Num. of proc:	2	4	9	16	25
Num of cv	%	%	%	%	%
57600	-12	2	8	13	14
230400	-3	-2	0	3	5
921600	-1	-1	0	-2	-1
1440000	2	1	-2	0	-1
3686400	3	2	0	3	0

In the diagram below resuls using MVPO are marked with “o” in legend and the curves that correspond to MVPO are dashed.

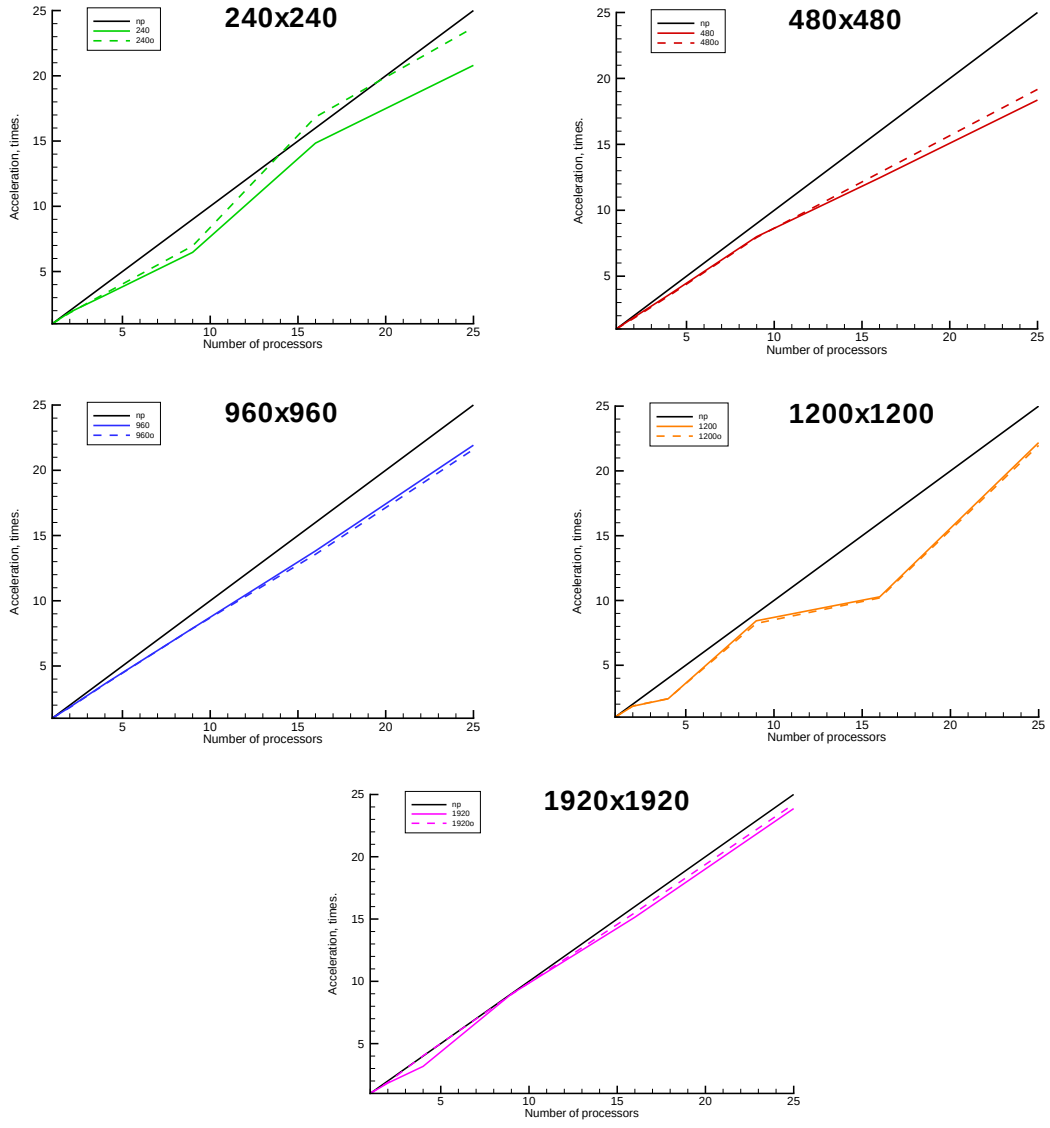


Figure 17: MVP

In this case overlapping of computations and communications gives no substantial increase of performance.

5 Test of CG-methods performance

The description of problem used in these tests is of no value because only the time of fixed number of iterations was measured. The acceleration using different mesh sizes and different number of processors was obtained:

Table 14: CG-methods acceleration: Mesh 57600 cv (240x240)

N of CPU	CG	BI-CG	BI-CGSTAB
2	1.91	1.92	1.92
4	3.7	3.55	3.68
9	6.93	7.16	6.85
16	8.95	10.58	8.74
25	14.35	11.8	9.24

Table 15: CG-methods acceleration: Mesh 230400 cv (480x480)

N of CPU	CG	BI-CG	BI-CGSTAB
2	1.78	1.86	1.99
4	3.57	3.6	3.48
9	7.55	7.65	7.48
16	12.33	12.54	12.24
25	16.1	17.19	16.49

Table 16: CG-methods acceleration: Mesh 921600 cv (960x960)

N of CPU	CG	BI-CG	BI-CGSTAB
2	1.95	1.97	1.99
4	3.75	3.78	3.74
9	7.91	8.38	8.12
16	13.66	14.44	13.92
25	20.5	21.54	21.26

Table 17: CG-methods acceleration: Mesh 1440000 cv (1200x1200)

N of CPU	CG	BI-CG	BI-CGSTAB
2	2.06	1.95	2
4	3.36	2.72	2.85
9	8.23	8.03	8.18
16	13.07	11.72	11.82
25	22.46	22.01	21.39

The results of the tests show the growth of efficiency of parallelization with number of cv. (Efficiency of parallelization means relation between achieved acceleration and number of processors.)

This diagram is convenient for comparison of CG-methods behaviour with growth of CPU number for each mesh size.

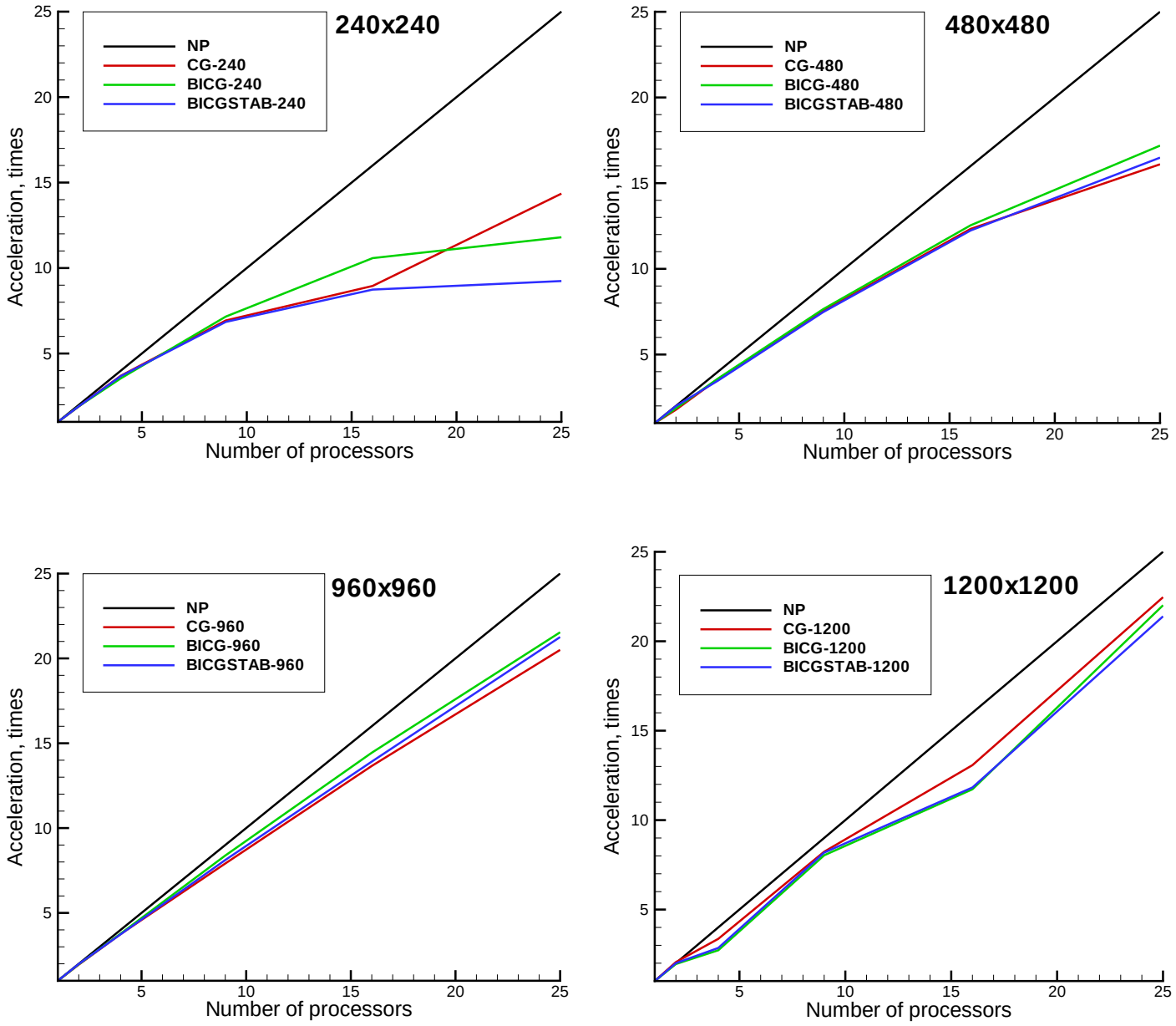


Figure 18: CG-methods acceleration: methods comparison

This diagram is convenient to see changes in acceleration when using different mesh sizes for each method.

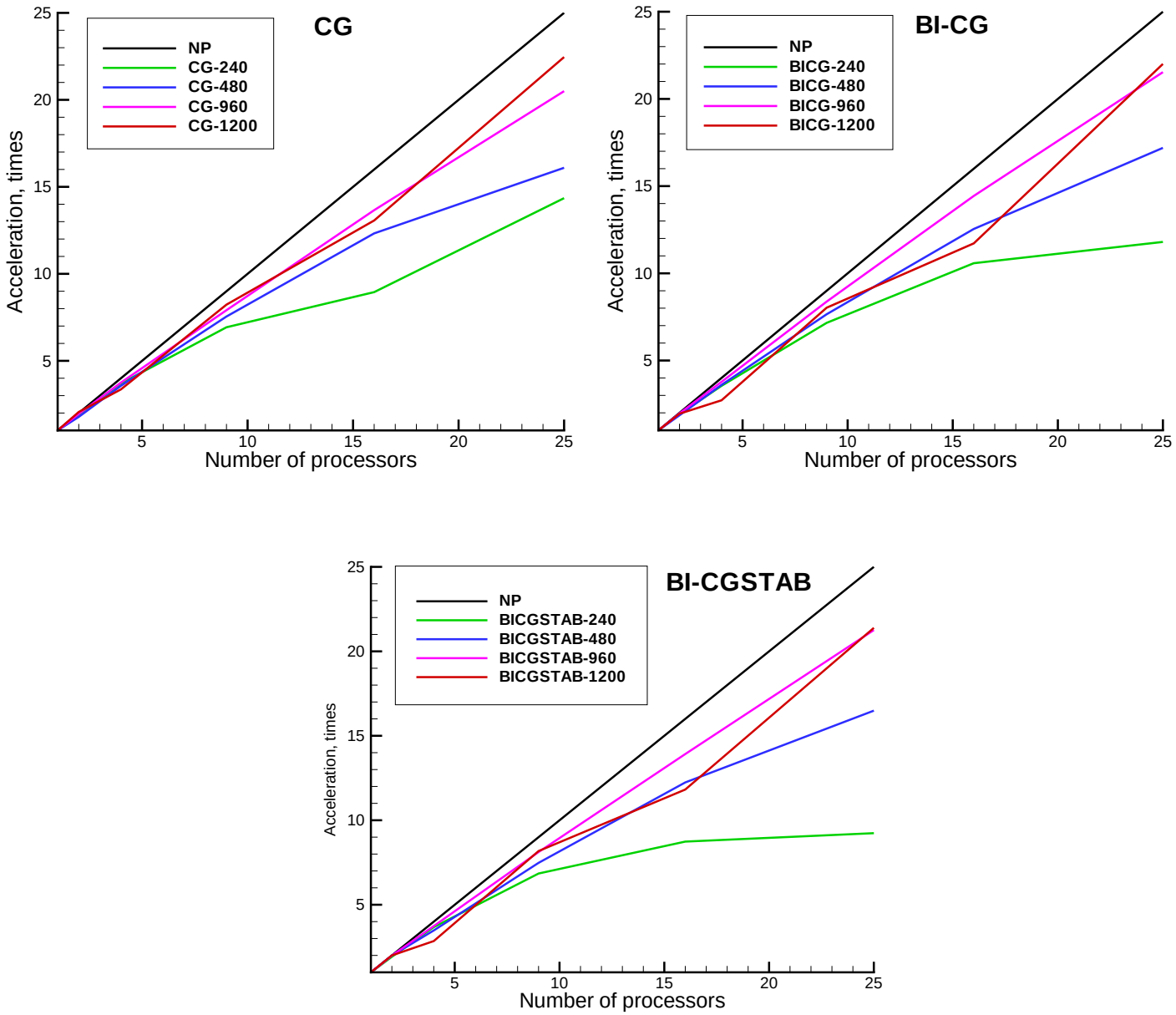


Figure 19: CG-methods acceleration: mesh size comparison

6 Test of GMRES performance

The description of problem used in these tests is of no value because only the time of fixed number of iterations was measured. Two versions of GMRES [4] were tested. The only difference between these methods is in the orthogonalization algorithm.

6.1 Orthogonalization algorithm for GMRES

There are two versions of Arnoldi algorithm for building ortogonal basis of the Krylov subspace K_m.

- Arnoldi algorithm [4]

  1. Choose a vector v_1 of norm 1
  2. For j=1,2,...,m Do:
  3. Compute $h_{ij} = (Av_j, v_i)$ for i=1,2,...,j
  4. Compute $w_j = Av_j - \sum_{i=1}^{j} h_{ij} v_i$
  5. $h_{j+1,j} = ||w_j||_2$. If $h_{j+1,j} = 0$ then Stop
  6. $v_{j+1} = w_j / h_{j+1,j}$
  7. EndDo

- Arnoldi-Modified Gram-Schmidt [4]

  1. Choose a vector v_1 of norm 1
  2. For j=1,2,...,m Do:
  3. Compute $w_j = Av_j$
  4. For i=1,2,...,j Do:
  5. $h_{ij} = (w_j, v_i)$
  6. $w_j = w_j - h_{ij} v_i$
  7. EndDo
  8. $h_{j+1,j} = ||w_j||_2$. If $h_{j+1,j} = 0$ then Stop
  9. $v_{j+1} = w_j / h_{j+1,j}$
  10. EndDo

This two algorithms are mathematically equivalent but in presence of round-off the second formulation is much more reliable. But in parallel implementation the second version requires j broadcast communications to compute j inner products for each j=1,...,m. So it requires 0.5(m+1)m broadcast communications to build the basis because inner products are coupled with vector updates (lines 5,6 of 2nd). First version requires only m broadcast communications. It is because inner products and update of vector w_j are separated (lines 3,4 of 1st) so all data about j inner products can be transmitted with only one communication. GMRES with Arnoldi-Modified Gram-Schmidt algorithm is the basic method and it will be denoted as GMRES in text below. GMRES with Arnoldi algorithm is the method to prove it's advantage in parallel computaions and it will be denoted as GMRES2

Results of the performance tests are in the table below. Field x means acceleration - relation between CPU time on 1 CPU and time on N CPU. Number of CPU N also shows maximal possible acceleration. Note: acceleration of GMRES2 is the relation between CPU time of GMRES (not GMRES2!) on 1 CPU and CPU time of GMRES2 on N CPU. It is because GMRES obviously preferable on a single processor.

Table 18: GMRES acceleration: Mesh 57600 cv (240x240)

	GMRES(5)		GMRES2(5)		GMRES(20)		GMRES2(20)		GMRES(50)		GMRES2(50)	
Num of CPU	Time of 100 iter.	x	Time of 100 iter.	x	Time of 50 iter.	x	Time of 50 iter.	x	Time of 10 iter.	x	Time of 10 iter.	x
1	52.88	1	55.64	0.95	240.8	1	248	0.97	259.2	1	273	0.95
2	27.3	1.94	28.15	1.88	124.0	1.94	126.9	1.89	134.9	1.93	137.6	1.88
4	13.3	3.98	13.45	3.93	57.7	4.17	57.0	4.22	62.1	4.17	61.3	4.23
9	6.5	8.13	6.4	8.26	27.0	8.92	25.5	9.44	28.1	9.22	26.6	9.74
16	4.8	11.02	4.3	12.3	20.5	11.7	16.1	15.0	21.9	11.8	15.8	16.4
25	4.42	11.96	3.63	14.6	20.9	11.5	12.4	19.4	22.4	11.6	12.9	20.1

Table 19: GMRES acceleration: Mesh 230400 cv (480x480)

	GMRES(5)		GMRES2(5)		GMRES(20)		GMRES2(20)		GMRES(50)		GMRES2(50)	
Num of CPU	Time of 100 iter.	x	Time of 100 iter.	x	Time of 20 iter.	x	Time of 20 iter.	x	Time of 5 iter.	x	Time of 5 iter.	x
1	215	1	230	0.93	391	1	410	0.95	535	1	558	0.96
2	110.8	1.94	114.8	1.87	197.6	1.98	207	1.89	267.3	2	279	1.92
4	57.2	3.76	59.6	3.61	102.4	3.82	106	3.69	138.2	3.87	143	3.74
9	26.0	8.27	26.1	8.24	47.3	8.27	47.4	8.25	64.5	8.3	63.8	8.39
16	15.14	14.2	14.7	14.6	26.2	14.9	24.6	15.9	35.9	14.9	31.7	16.9
25	10.7	20.1	10.12	21.2	18.3	21.37	15.3	25.6	24.8	21.6	19.9	26.9

Table 20: GMRES acceleration: Mesh 921600 cv (960x960)

	GMRES(5)		GMRES2(5)		GMRES(20)		GMRES2(20)		GMRES(50)		GMRES2(50)	
Num of CPU	Time of 25 iter.	x	Time of 25 iter.	x	Time of 5 iter.	x	Time of 5 iter.	x	Time of 2 iter.	x	Time of 2 iter.	x
1	224	1	233	0.96	390	1	409	0.95	856	1	903	0.95
2	109.4	2.05	114.5	1.96	196	1.99	206	1.89	425	2.01	446	1.92
4	56.9	3.94	59.7	3.75	101	3.86	106	3.68	218	3.93	229	3.74
9	25.7	8.72	26.7	8.39	45.8	8.52	47.6	8.19	98.9	8.65	102.4	8.36
16	15.4	14.5	15.3	14.6	26.5	14.7	27.1	14.4	57	15.0	57.8	14.8
25	10.1	22.2	10.0	22.4	17.7	22.0	17.5	22.3	38.3	22.3	37.3	22.9

In some cases achieved acceleration exceeds maximal possible. Probably this is due to cache memory effect. Advantage of GMRES2 over GMRES as it was expected is getting higher when restart lenght grows and getting lower when meshs size grows. GMRES2 is almost twice faster than GMRES on mesh 240x240 with restart lenght 50. But these methods are almost equal on mesh 960x960 with restart lenght 50 and lower.

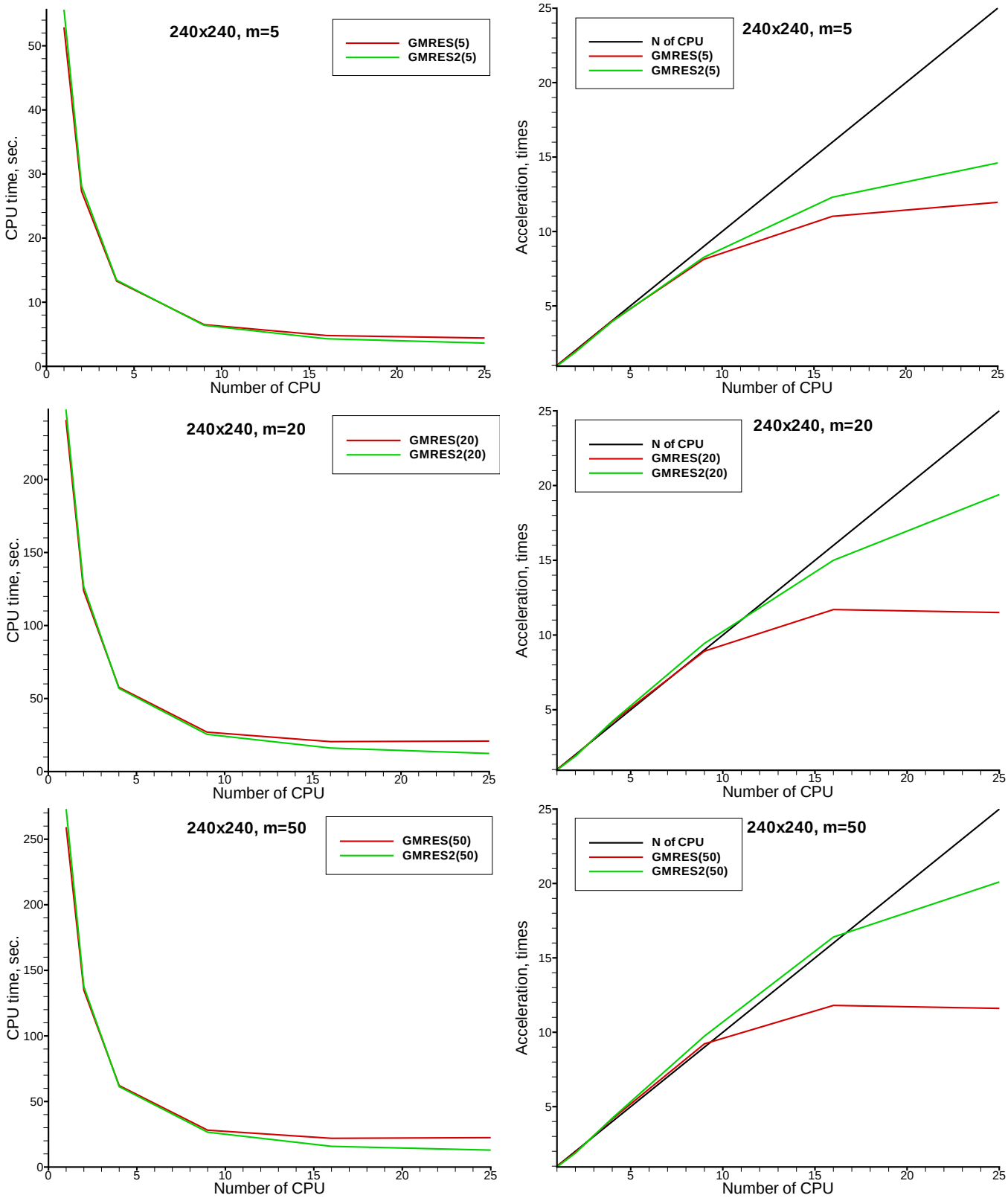


Figure 20: GMRES, mesh 240x240. CPU time of fixed number of iterations (left) and acceleration (right)

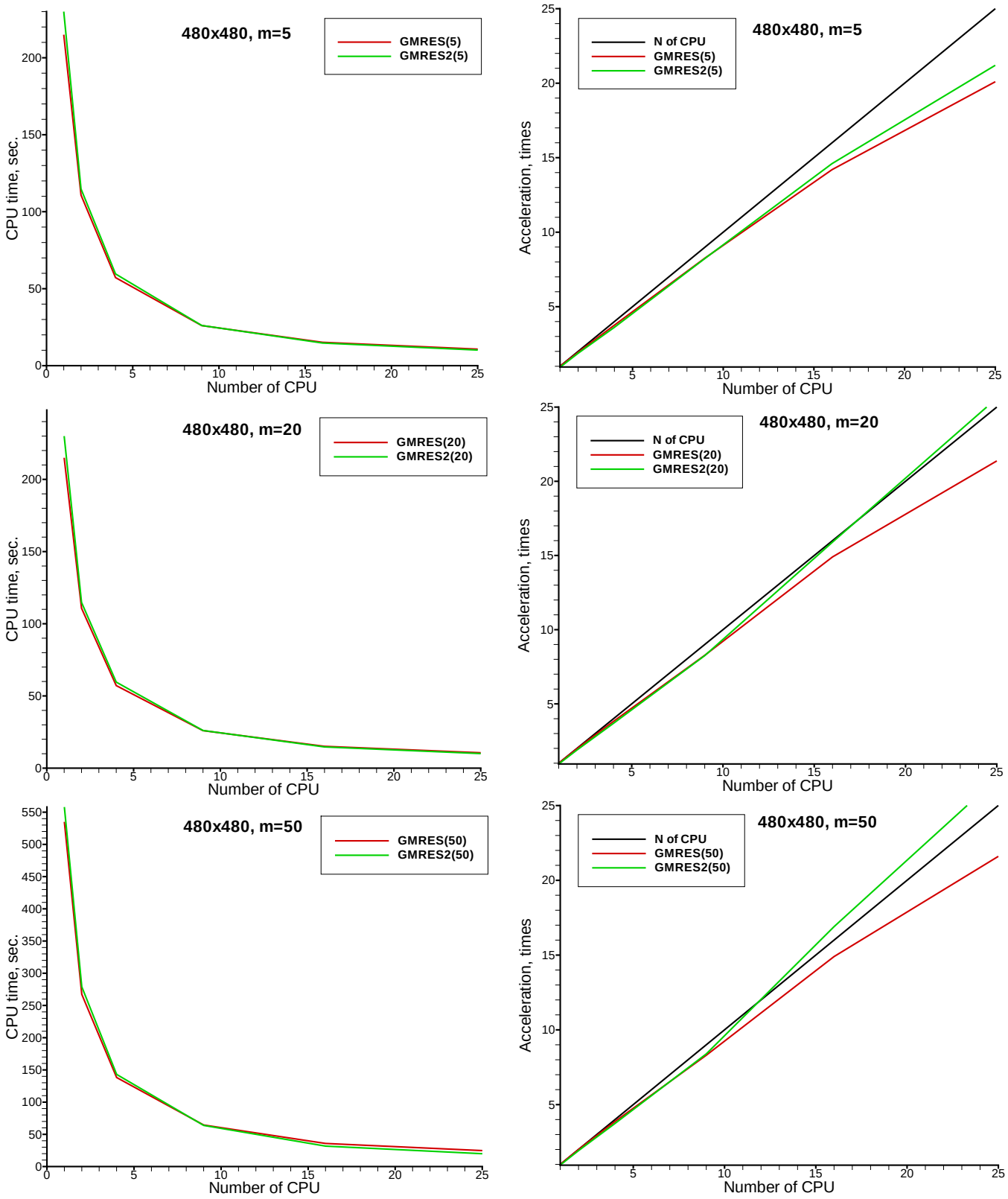


Figure 21: GMRES, mesh 480x480. CPU time of fixed number of iterations (left) and acceleration (right)

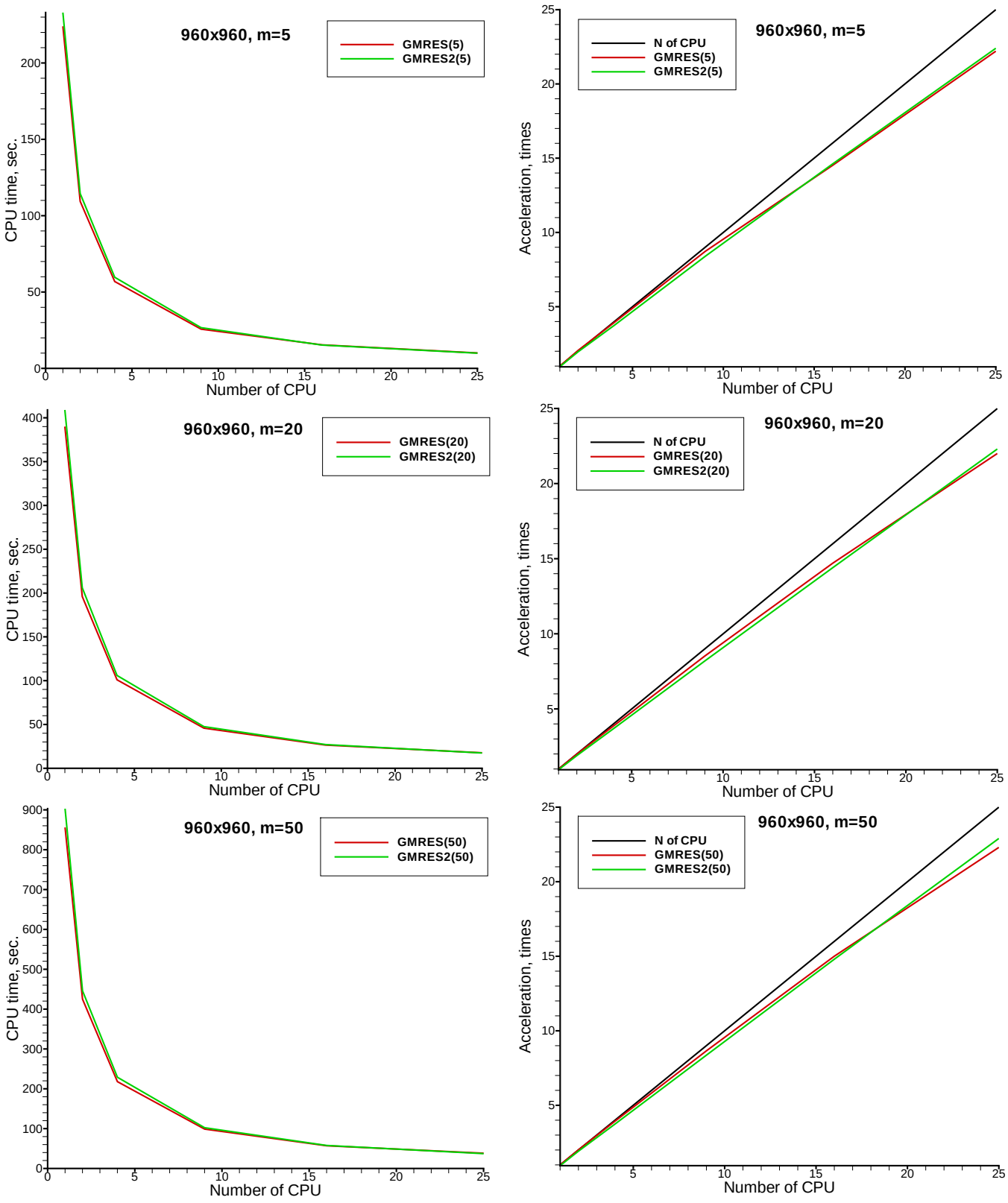


Figure 22: GMRES, mesh 960x960. CPU time of fixed number of iterations (left) and acceleration (right)

7 Summary table of acceleration results

This table shows acceleration (Field x in the table) and efficiency of parallelization (Field % in the table) of Krylov subspace solvers for different mesh size when using 25 CPU. Efficiency of parallelization means relation between achieved acceleration and maximum possible acceleration wich is equal to number of CPU. It is easy to compare abilities of acceleration of these solvers with this table. Two implementations of matrix vector product with 5-diagonal matrix are also included in this table as it is primary parallel operation in this solvers.

Table 21: Summary table of acceleration achieved using 25 CPU

Mesh size:	240x240 (57600)		480x480 (230400)		960x960 (921600)	
Method	x	%	x	%	x	%
CG	14.35	57	16.1	64	20.5	82
BI-CG	11.8	47	17.9	72	21.54	86
BI-CGSTAB	9.24	39	16.49	66	21.26	85
GMRES(5)	11.96	48	20.1	80	22.2	89
GMRES2(5)	14.6	58	21.2	84	22.4	90
GMRES(20)	11.5	46	21.37	85	22.0	89
GMRES2(20)	19.4	77	25.6	102	22.3	89
GMRES(50)	11.6	46	21.6	86	22.3	89
GMRES2(50)	20.1	80	26.9	108	22.9	92
MVP	20.8	83	18.36	73	21.92	88
MVPO	23.7	95	19.17	77	21.6	86

In some cases efficiency of parallelization exceeded 100%. Probably this happened due to cache memory effect.

References

[1] Suhas V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Corporation, USA, 1980.

[2] H K Versteeg, W Malalasekera, *An introduction to computational fluid dynamics, The finite volume method*, Longman Scientific & Technical, USA, 1995.

[3] Richard Barret, Michael Berry, Tony Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Ronald Pozo, Charles Romine, and Henk van der Vorst, *Templates for the solution of linear sistems: Building blocks for iterative methods.*

[4] Yousef Saad, *Iterative Methods for Sparse Linear Systems*, second edition, January 2000.

[5] Yousef Saad and Martin H. Schultz, *GMRES: Generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. stat. comput.

[6] Numerical Heat Transfer, vol 5:439-461, 1982

[7] Vladimir Bobkov, *Report on multigrid*, 2004.