Parallelisation of BICG-STAB algorithm for finite element computations

Bedřich Sousedík[1], Jaroslav Novotný[2]

[1] Department of Mathematics,
Faculty of Civil Engineering, Czech Technical University,
Thákurova 7, 166 29 Praha 6
sousedik@mat.fsv.cvut.cz

[2] Institute of Thermomechanics, Czech Academy of Sciences,
Dolejškova 5, 182 00 Praha 8
novotny@bivoj.it.cas.cz

Abstract

We describe parallelization and implementation of BICG-STAB method for systems of unsymmetric linear equations arising in finite element discretization. Global matrix is not constructed, we use element by element approach. All element matrices together with adressing vectors are distributed among processors and read into memory. Then multiplication of element matrices by a global vector is performed simultaneously on different processors. Two codes were developed: using MPI library for distributed memory systems and using OpenMP for shared memory systems. Good scalability is demonstrated on an example with about 30000 unknowns.

Keywords: linear algebra, iterative methods, BICG-STAB, parallelization, shared/distributed memory, OpenMP, MPI

1 Introduction

In many areas of continuum mechanics of solids and fluids (linear and non-linear) solution of large systems of linear equations is required. To achieve reasonable time span for solution of such systems parallelization of corresponding solution procedures is necessary. Matrices constructed using finite element discretization are sparse, so it is convenient to use modifications of solution techniques and procedures.

Parallel hardware and software

We distinguish between shared and distributed memory architectures of computers.

1. **Shared memory** (SMP - symmetrical multiprocessing) - all processors have the same priority in access to shared memory. It is not necessary to change the organisation of data storage. Shared variable programming model is based

on the notion of threads. Different threads can follow different flows of control through the same program, but communicate with each other only via shared data. Programming library: OpenMP.

2. **Distributed memory** - programming is based on the notion of processes (a process is an instance of a running program, together with the program data). The parallelism is achieved by having many processes, which cooperate on the same task. Each process has acces only to its own data and processes communicate each other by sending and receiving messages. Messages have to be programmed explicitely, so serial code has usually to be completely rewritten. Programming library: MPI (Message Passing Interface), PVM (Parallel Virtual Machine).

Parallel libraries

Programming using **OpenMP** represents inserting directives starting with a string !$OMP into source code. If we after linking OpenMP library run the code, the code is after reaching this directive split into several threads running in parallel. Parallel task ends when other directive of the type !$OMP is inserted.

It is always necessary to distinguish, which variables are shared and which need to be private. As threads execute their instructions asynchronously, it is necessary to include synchronization directives to ensure that all instructions occur in the correct order.

When programming using **MPI** every processors reads source code and depending on the parameter MYID (rank of processors from 0 to total number of processors NPROC-1) performs programme instructions. All communication (exchange of data, synchronization) is performed by calling library routines for message passing. The source code is usually written in FORTRAN or C, during compilation are parallel libraries linked to it.

2 BICG-STAB method

Algorithm of BICG-STAB method without preconditioning for solution of a system of linear equations $Ax = b$ can be described as follows:

$x_0 = 0$ . . . initial guess
$r_0 = b - Ax_0$
$\hat{r}_0 = r_0$
$\rho_0 = \alpha = \omega_0 = 1$ . . . variables
$v_0 = p_0 = 0$ . . . vectors
for $i = 1, 2, 3, \ldots$
    $\rho_i = (\hat{r}_0, r_{i-1})$
    $\beta = \frac{\rho_i}{\rho_{i-1}} \frac{\alpha}{\omega_{i-1}}$

$p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$
$v_i = Ap_i$
$\alpha = \frac{\rho_i}{(\hat{r}_0, v_i)}$
$s = r_{i-1} - \alpha v_i$
$t = As$
$\omega_i = \frac{(t,s)}{(t,t)}$
$x_i = x_{i-1} + \alpha p_i + \omega_i s$
if x_i satisfies prescribed accuracy, quit
$r_i = s - \omega_i t$
end

3 Parallelization of BICG-STAB method

Parallelizing of linear algebra algorithms means mainly parallelization of following tasks:

- Linear combination of vectors
- Dot product
- Multiplication of matrix by vector

Further it is necessary in each iteration to evaluate the accuracy of solution by master process. In this case for synchronization purposes it is necessary to send to the other processors a directive to quit the iteration loop. In the following we describe methods of parallelization and implementation of these tasks both for distribuuted and shared memory architectures using MPI and OpenMP libraries, respectively.

Parallelization using MPI

Since MPI is a low level programming tool, it is necessary to programme not only synchronization directives and sending of results, but also distribution of vectors and element matrices among processors. We use following notation:

NPROC	...	number of processors
MYID	...	rank of processor, MYID = 0...NPROC-1
LSOL	...	length of global vector of solution
LSOL_LOC	...	length of local vector on a processor
LSOL_LOCX	...	maximum length of a local vector

Lengths of local vectors can be computed using the following relations:

LSOL_LOCX = (LSOL+NPROC-1) / NPROC
LSOL_LOC = MIN(LSOL-(MYID*LSOL_LOCX), LSOL_LOCX)

a) **parallel linear combination of two vectors**

```
C     distribution of vectors A(LSOL) a B(LSOL) among processors
C     into local vectors A_loc(LSOL_LOCX) B_loc(LSOL_LOCX)
      call MPI_SCATTER(A,LSOL_LOCX,MPI_DOUBLE_PRECISION,A_loc,
     1                   LSOL_LOCX,MPI_DOUBLE_PRECISION,0,
     2                   MPI_COMM_WORLD,IERROR)
C
      call MPI_SCATTER(B,LSOL_LOCX,MPI_DOUBLE_PRECISION,B_loc,
     1                   LSOL_LOCX,MPI_DOUBLE_PRECISION,0,
     2                   MPI_COMM_WORLD,IERROR)
C
C     calculation of linear combination of A and B in local arrays
      DO 1000 i = 1,N_loc
         C_loc(i) = alpha*A_loc(i) + beta*B_loc(i)
 1000 ENDDO
```

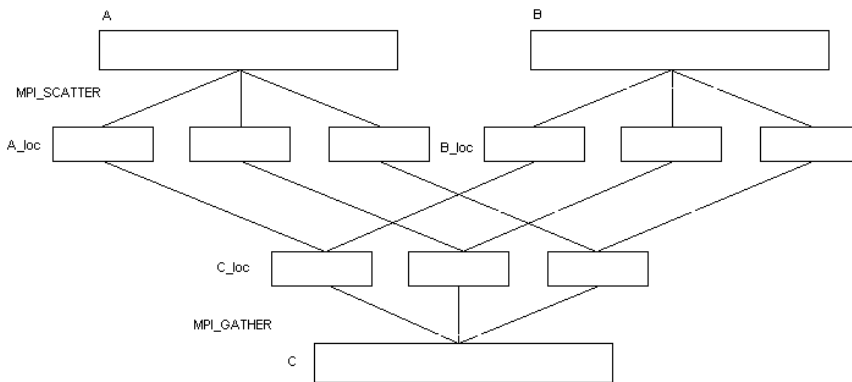

Figure 1: Parallel linear combination of two vectors

b) **parallel dot product,** result received by all processors

```
      call MPI_SCATTER(A,LSOL_LOCX,MPI_DOUBLE_PRECISION,A_loc,
     1                   LSOL_LOCX,MPI_DOUBLE_PRECISION,0,
     2                   MPI_COMM_WORLD,IERROR)
      call MPI_SCATTER(B,LSOL_LOCX,MPI_DOUBLE_PRECISION,B_loc,
     1                   LSOL_LOCX,MPI_DOUBLE_PRECISION,0,
     2                   MPI_COMM_WORLD,IERROR)
C
C     dot products in local arrays
      call scalp(A_loc, B_loc, LSOL_LOC, sc_loc)
C
      call MPI_ALLREDUCE(sc_loc,sc,1,MPI_DOUBLE_PRECISION,
     1                     MPI_SUM,MPI_COMM_WORLD,IERROR)
```

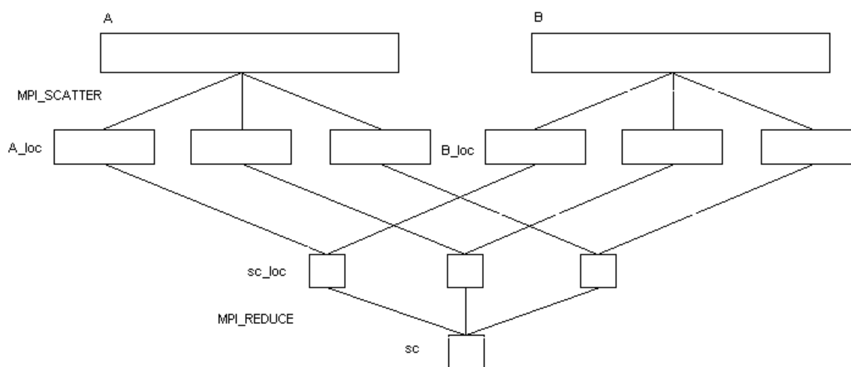

Figure 2: Parallel dot product

c) **Parallelization of multiplication of matrix A by vector u**

We do not construct the global matrix. Before entering the BICG-STAB algorithm the program reads from the disk **all element matrices** and corresponding array of adresses ILVGV - Indeces of Local Variables in Global Vector.

$$A = \sum_{ielem=1}^{NELEM} A_{ielem}$$

We will use the following variables:

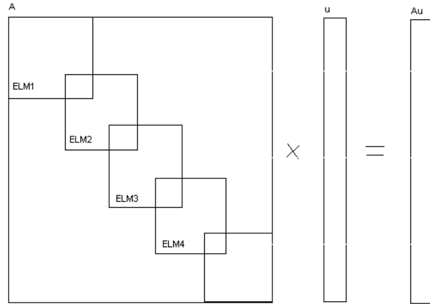

Figure 3: Scheme of matrix-vector multiplication

NEL_LOC ... number of elements distributed to given processor
NEL_LOCX ... maximum number of elements per processor

For computation of number of element matrices on MYID-th processor:

NEL_LOCX = (NELEM + NPROC - 1)/NPROC
NEL_LOC = MIN(NELEM - MYID*NEL_LOCX, NEL_LOCX)

If the global vector u is available on MYID = 0, it is necessary to distribute its copy using broadcast directive to every processor:

```
C     sending of global vector u to all processors
      call MPI_BCAST(u,LSOL,MPI_DOUBLE_PRECISION,0,
     1               MPI_COMM_WORLD,IERROR)
C
      CALL RZERO(Au,LSOL)
C
C
C     multiplication of element matrices by vector u, result in Au_p
      DO 1200 IE_LOC = 1, NEL_LOC
         NEVAB = INEVA_LOC(IE_LOC)
      CALL MULTMV(ILVGV_LOC((IE_LOC-1)*NEVAX+1),NEVAB,
     1         ELM_LOC((IE_LOC-1)*LELMX+1),LELMX,u,Au_p,LSOL)
 1200 ENDDO
C
C     reduction (adding) of vectors Au_p(LSOL) from all processors,
C     resulting global vector is Au(LSOL)
      CALL MPI_REDUCE(Au_p,ELMu,LSOL,MPI_DOUBLE_PRECISION,MPI_SUM,0,
     1                MPI_COMM_WORLD,IERROR)
```

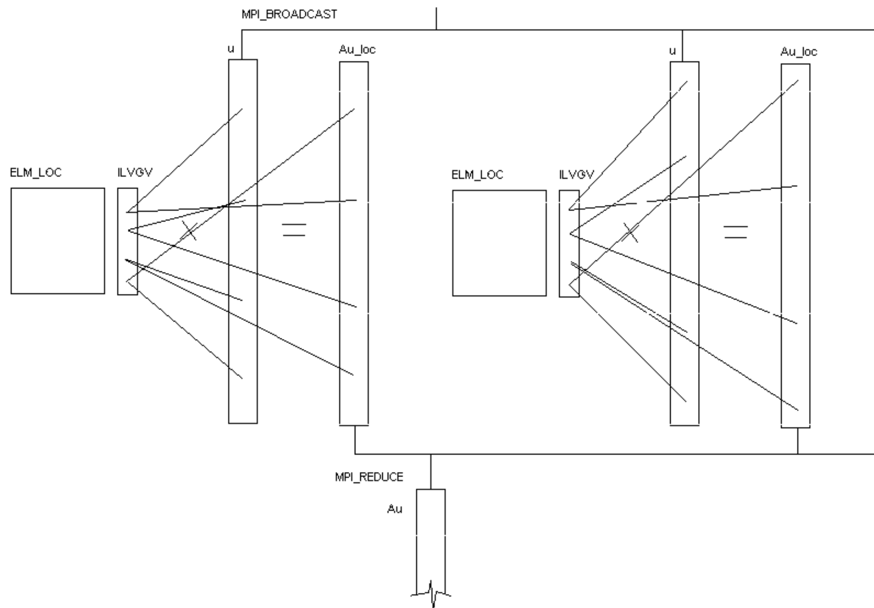

Figure 4: Parallel multiplication of matrix A by vector u

In the case that we have already computed in parallel linear combination of vectors, vector u is available locally distributed, it is necessary to gather this local vectors into the global vector:

```
C     collecting (gathering) of global vector u from local vectors
      call MPI_ALLGATHER(u_loc, LSOL_LOCX, MPI_DOUBLE_PRECISION,
     1        u,LSOL_LOCX,MPI_DOUBLE_PRECISION,
     2           MPI_COMM_WORLD,IERROR )
C
C     inicialization of vector Au_p
      CALL RZERO(Au_p,LSOL)
C
C     multiplication of element matrices by vector u, result in Au_p
      DO 1200 IE_LOC = 1, NEL_LOC
         NEVAB = INEVA_LOC(IE_LOC)
         CALL MULTMV(ILVGV_LOC((IE_LOC-1)*NEVAX+1),NEVAB,
     1         ELM_LOC((IE_LOC-1)*LELMX+1),LELMX,u,Au_p,LSOL)
 1200 ENDDO
C
```

```
C     reduction (adding) of vectors Au_p(LSOL) from all processors,
C     resulting global vector is Au(LSOL)
      CALL MPI_REDUCE(Au_p,Au,LSOL,MPI_DOUBLE_PRECISION,MPI_SUM,0,
     1                MPI_COMM_WORLD,IERROR)
```

Parallelization using OpenMP

a) **parallel linear combination of two vectors**

```
!$OMP PARALLEL DO
      DO 1020 i = 1, n
C
         C(i)  = alpha*A(I) + beta*B(I)
C
 1020 ENDDO
!$OMP END PARALLEL DO
```

b) **parallel dot product,** result received by all processors

```
      sc = 0.
!$OMP PARALLEL DO REDUCTION(+:sc)
      DO 1000 i = 1,n
         sc = sc + A(i)*B(i)
 1000 ENDDO
!$OMP END PARALLEL DO
C
      pscalp = sc
```

c) **Parallelization of multiplication of matrix A by vector u**

```
!$omp parallel do private(NEVAB)
!$omp+shared(ILVGV,INEVA,ELM,LELMX,NELEM,NEVAX,u,LSOL,Au)
      DO 1200 IE = 1, NELEM
C
         NEVAB = INEVA(IE)
C        calling of routine for multiplication of element matrix
C        by a global vector u
         CALL MULTMV(ILVGV((IE-1)*NEVAX+1),NEVAB,
     1               ELM((IE-1)*LELMX+1),LELMX,u,Au,LSOL)
 1200 ENDDO
!$omp end parallel do
```

4 Numerical results

Our implementation of the BICG-STAB method was tested on a quite large system of equations which arises using discretization of linear elasticity problem by finite elements. Prism of dimensions 2 x 1 x 1 m was loaded by volumetric load, material properties were $E = 2.10^{11}$ MPa, $\nu = 0.3$. Finite element mesh consisted of 2000 isoparametric hexahedrons, with 20 nodes (60 unknowns) per element and totally of 9581 nodes. Resulting system of equations represented 28743 unknowns.

For benchmarking we used two parallel servers:

- **DEC ES40 with 4 CPUs Alpha EV6 / 500MHz**
- **Sun Fire E15k with 52 CPUs Ultrasparc III / 900 MHz**

Wall time of frontal solver was 5 min 20 sec on 1 processor on the DEC server. Wall time of the BICG-STAB procedure runs using MPI and OpenMP on different number of processors is given in the following table and graph:
Using MPI library:

No. of CPUs	DEC time	Sun time
1	3 min 22 s	9 min 28 s
2	2 min 06 s	4 min 39 s
4	1 min 25 s	2 min 06 s
8		40.5 s
16		25.8 s
32		17.2 s

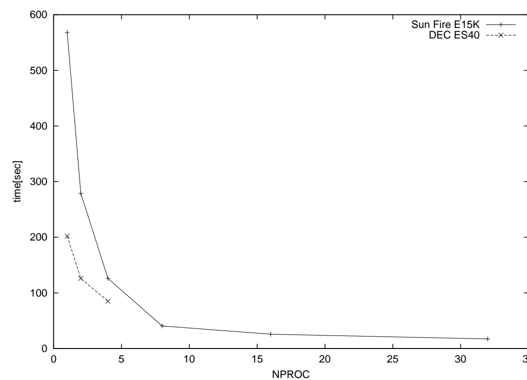

Figure 5: Scalability using MPI implementation

Using OpenMP library:

No. of CPUs	DEC time	Sun time
1		8 min 27 s
2		4 min 15 s
4		1 min 51 s
8		20.0 s
16		8.5 s

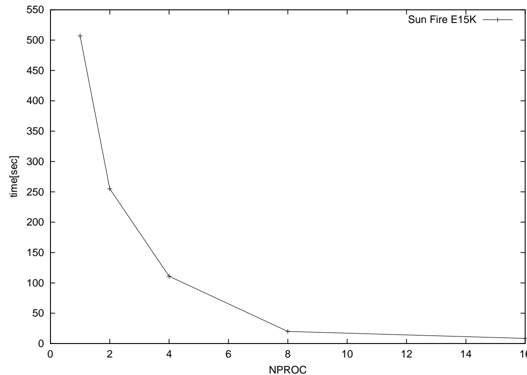

Figure 6: Scalability using OpenMP implementation

To achieve relative error of residual of 1.E-07, 572 iterations of the BICG-STAB algorithm were necessary. Also 120 MB RAM memory and about 70 MB of disk space were needed.

5 Conclusions

We parallelized the BICG-STAB method using both MPI and OpenMP libraries. Numerical results on a test problem from elasticity showed very good scalability. For the improvement of convergence rates implementation of appropriate preconditioning and its parallelization will be necessary.

Acknowledgement: *The authors would like to acknowledge the support of grant GA AV A2120201, research project K1019101, and the European Commission through grant number HPRI-CT-1999-00026 (the TRACS Programme at EPCC Edinburgh).*

References

[1] Pacheco, P.: *Parallel programming with MPI.* Morgan Kaufmann Publishers, San Francisco 1997.

[2] Saad, Y.: *Iterative methods for sparse linear systems.* PWS Publishing company, Boston 1996.

[3] Van Der Vorst, H. A.: *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems.* SIAM J. Sci. Stat. Comput., 1992, Vol. 13, No. 2, pp. 631–644.