BFSAI-IC OpenMP Implementation: User's guide

Carlo Janna, Massimiliano Ferronato, Nicola Castelletto
Dept. Mathematical Methods and Models for Scientific Applications
University of Padova, Padova, Italy

E-mail: {janna,ferronat,castel}@dmsa.unipd.it

January, 2011

Contents

1 Theoretical background

The objective of the BFSAI-IC OpenMP Implementation code is to solve a symmetric positive definite (SPD) linear system:

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, with the aid of the Preconditioned Conjugate Gradient (PCG) algorithm. The BFSAI-IC preconditioner:

$$M^{-1} = W^T W = F^T J_L^{-T} J_L^{-1} F \tag{2}$$

consists of:

- a unitary lower triangular matrix $F \in \mathcal{W}_{\mathcal{S}_{BL}}$ such that $\|D - FL\|_F$ is minimum for an arbitrary block diagonal matrix D, with L the exact lower Cholesky factor of A and $\mathcal{W}_{\mathcal{S}_{BL}}$ the set of matrices with non-zero pattern $\mathcal{S}_{BL}$;
- a lower triangular matrix J_L containing the lower Incomplete Cholesky factors of the n_b diagonal blocks of FAF^T.

The non-zero pattern $\mathcal{S}_{BL}$ is diagonal for the n_b diagonal blocks and uses the lower non-zero pattern of A^k for the off-block diagonal part. The density of the BFSAI-IC preconditioner is controlled by a set of user-specified parameters:

1. ρ_B: maximum allowable number of non-zeroes for each row of any diagonal block of FAF^T in excess of the corresponding row of A;
2. δ: tolerance below which an entry in the i-th row of F is dropped relative to the 2-norm of the same row;
3. ρ_L: maximum allowable number of non-zeroes for each row of the lower triangular factor of any diagonal block of FAF^T in excess of the corresponding row of FAF^T;
4. τ_L: tolerance below which an entry in the i-th row of J_L is dropped relative to the 2-norm of the same row.

The details of the computation of F and J_L along with the overall theoretical background are provided in the paper:

Carlo Janna, Massimiliano Ferronato and Giuseppe Gambolati. A Block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems. SIAM J. Sci. Comput., 32:2468-2484, 2010.

2 The BFSAI-IC OpenMP Implementation code

BFSAI-IC OpenMP Implementation is coded in Fortran90 using integer, logical, character and double precision real variables. The main program source file is `test_bfsai.f90`. The matrices A and F are stored in CSR format, while each diagonal block of J_L is stored in MSSR format. The code structure is the following:

1. link to the Input/Output units (subroutine `openio`);
2. reading and storage of matrix A (subroutine `mk_CSRMAT` and `readmat`);
3. reading and storage of the right-hand side vector $\mathbf{b}$;
4. reading of the Input parameters;
5. construction of the parallel data structure to handle A, F and J_L with n_p processors (subroutine `mk_OMPDTSTR`);
6. estimate of the maximum number of non-zeroes stored in F and J_L;
7. allocation of the BFSAI-IC data structure (subroutine `mk_BFSAI`);
8. BFSAI-IC computation (subroutine `compute_BFSAI`);
9. allocation of the PCG data structure (subroutine `mk_BFSAI_CG`);
10. solution of the linear system $A\mathbf{x} = \mathbf{b}$ (subroutine `PCG_solv`);
11. printing of the Output results.

The BFSAI-IC OpenMP Implementation package is designed to be easily linked to other user's codes. The following classes are defined:

- `class_OMPDTSTR`: variables for the parallel handling of matrices and vectors;
- `class_CSRMAT`: variables for the CSR matrix storage;
- `class_BFSAI`: variables for the BFSAI-IC preconditioner;
- `class_BFSAI_CG`: variables for the PCG algorithm.

Dynamic allocation and deallocation of variables is allowed for using a constructor and destructor subroutine, respectively. Each class contains also a member subroutine characterized by the prefix `errchk` providing detailed information on the encountered errors.

`mk_OMPDTSTR` is the constructor for the `class_OMPDTSTR` variables requiring the following exchange parameters:

- `nequ` (integer): matrix size;
- `nproc` (integer): number of processors used in the code execution;
- `nbloc` (integer): number of diagonal blocks;
- `DTSTR_var` (OMPDTSTR): parallel data structure for matrix and vector handling;
- `info` (integer): error code;
- `blksbdv` (integer array, optional): if present, a user-specified block subdivision, e.g. arising from a graph partitioning; if not present, the block subdivision is automatic.

`dlt_OMPDTSTR` is the destructor for the `class_OMPDTSTR` variables requiring the following exchange parameters:

- `DTSTR_var` (OMPDTSTR): parallel data structure to be deallocated;
- `info` (integer): error code.

`mk_CSRMAT` is the constructor for the `class_CSRMAT` variables requiring the following exchange parameters:

- `nn` (integer): matrix size;
- `nt` (integer): number of non-zero matrix entries;
- `mat` (CSRMAT): matrix in CSR format;
- `info` (integer): error code.

`dlt_CSRMAT` is the destructor for the `class_CSRMAT` variables requiring the following exchange parameters:

- `mat` (CSRMAT): matrix to be deallocated;
- `info` (integer): error code.

`mk_BFSAI` is the constructor for the `class_BFSAI` variables requiring the following exchange parameters:

- `DEBUG` (logical): if `.true.` prints out F and J_L;
- `SPD_OPT` (logical): if `.true.` enforces positive definiteness of J_L;
- `nn` (integer): size of A;
- `nproc` (integer): number of processors used in the code execution;
- `nbloc` (integer): number of diagonal blocks;
- `nnz_F` (integer): maximum number of non-zero entries in F;
- `nnz_JL` (integer): maximum number of non-zero entries in J_L;
- `kappa` (integer): power of A used for the selection of the non-zero pattern of F;
- `rho_B` (integer): parameter ρ_B controlling the fill-in of the diagonal blocks of FAF^T;
- `rho_L` (integer): parameter ρ_L controlling the fill-in of the diagonal blocks of J_L;
- `delta` (double precision real): parameter δ controlling the dropping of the smallest F entries;
- `tau_L` (double precision real): parameter τ_L controlling the dropping of the smallest J_L entries;

- `prec` (BFSAI): BFSAI-IC preconditioner;
- `info` (integer): error code.

`dlt_BFSAI` is the destructor for the `class_BFSAI` variables requiring the following exchange parameters:

- `mat` (BFSAI): BFSAI-IC preconditioner to be deallocated;
- `info` (integer): error code.

`mk_BFSAI_CG` is the constructor for the `class_BFSAI_CG` variables requiring the following exchange parameters:

- `nequ` (integer): matrix size;
- `iout` (integer): output unit for the PCG convergence profile;
- `itmax` (integer): maximum number of PCG iterations;
- `isol` (integer): if $\neq 0$ set the initial solution to $M^{-1}\mathbf{b}$, otherwise uses the array already stored in the solution vector;
- `tol_CG` (double precision real): exit tolerance on the relative residual $\|\mathbf{b} - A\mathbf{x}\|_2/\|\mathbf{b}\|_2$;
- `PCG_var` (BFSAI_CG): PCG parameters and work arrays;
- `info` (integer): error code.

`dlt_BFSAI_CG` is the destructor for the `class_BFSAI` variables requiring the following exchange parameters:

- `PCG_var` (BFSAI_CG): PCG parameters and work arrays to be deallocated;
- `info` (integer): error code.

The subroutine `compute_BFSAI` computes the BFSAI-IC preconditioner using the following exchange variables:

- `cpt_PATT` (logical): if `.true.` the non-zero pattern of F is that of A^k; if `.false.` uses the pattern saved in `prec_BFSAI%iat_F` and `prec_BFSAI%ja_F`;
- `parDTSTR_mat_A` (OMPDTSTR): parallel data structure for matrix and vector handling;
- `mat_A` (CSRMAT): matrix A in CSR format;
- `prec_BFSAI` (BFSAI): BFSAI-IC preconditioner;
- `info` (integer): error code;
- `vinfo` (integer array): detailed error information.

The subroutine `PCG_solv` solves a linear system with a BFSAI-IC PCG algorithm using the following exchange variables:

- `mat_A` (CSRMAT): system matrix in CSR format;
- `parDTSTR_mat_A` (OMPDTSTR): parallel data structure for matrix and vector handling;
- `prec_BFSAI` (BFSAI): BFSAI-IC preconditioner;
- `PCG` (BFSAI_CG): PCG parameters and work arrays;
- `rhs` (double precision real array): right-hand side vector;
- `sol` (double precision real array): solution vector;
- `info` (integer): error code.

The subroutine `prt_BFSAI` prints out F, F^T and J_L.

2.1 Input data

The Input data must be provided in ASCII text files. The program requires four Input units to be set:

- `test_bfsai.fnames`: contains the names of the Input/Output units linked to the code;
- a `mat_A` file containing the matrix A;
- a `vec_b` file containing the right-hand side vector $\mathbf{b}$;
- a `parm` file containing the user-specified parameters needed by BFSAI-IC and the PCG solver.

The names of the `mat_A`, `vec_b` and `parm` files are user-specified.

The Input file `test_bfsai.fnames` lists the names of the following units:

- the `mat_A` file (Input unit);
- the `vec_b` file (Input unit);
- the `parm` file (Input unit);
- a `vec_x` file (Output unit) with the system solution $\mathbf{x}$;
- a `log` file (Output unit) with the execution and error flags.

Each file name must be written within apexes and is case sensitive in Unix-like operating systems.

The `mat_A` file needs a header with the variables:

- `nn` (integer): size n of A (> 0),

- `nt` (integer): number of non-zeroes n_t of A ($\geq n$),

separated by a comma or a blank space. The non-zeroes of A follow in coordinate format, i.e. row index i, column index j, non-zero entry $[A]_{ij}$.

The `vec_b` file contains the n components of $\mathbf{b}$ separated by a comma or a blank space.

The `parm` file lists the values of the following variables:

- `np` (integer): number of processors n_p used in the code execution ($1 \leq n_p \leq N_p$, with N_p the maximum number of available processors),
- `nb` (integer): number of diagonal blocks n_b ($\geq n_p$);
- `kappa` (integer): power of A used for the selection of the non-zero pattern of F (≥ 1),
- `rho_b` (integer): parameter ρ_B controlling the fill-in of the diagonal blocks of FAF^T (≥ 0),
- `rho_l` (integer): parameter ρ_L controlling the fill-in of the diagonal blocks of J_L (≥ 0),
- `nnz_F` (integer): maximum number of non-zero entries allowed for in F ($\geq n$, suggested value $n_t \cdot 4^{(k-1)}$),
- `nnz_JL` (integer): maximum number of non-zero entries allowed for in J_L ($\geq n_t + 2n_b$, suggested value $n_t + 2n_b + n(\rho_B + \rho_L)$),
- `delta` (double precision real): parameter δ controlling the dropping of the smallest F entries (≥ 0.00),
- `tau_l` (double precision real): parameter τ_L controlling the dropping of the smallest J_L entries (≥ 0.00),
- `iout` (integer): output unit for the PCG convergence profile (if ≤ 0 no print),
- `itmax` (integer): maximum number of PCG iterations (> 0),
- `tol_CG` (double precision real): exit tolerance on the relative residual $\|\mathbf{b} - A\mathbf{x}\|_2 / \|\mathbf{b}\|_2$ (> 0).

The parameters above must be listed one each line.

2.2 Output results

The Output results consist of two files and a summary printed out on the screen. The Output files are:

- a `vec_x` file containing the converged solution $\mathbf{x}$,
- a `log` file containing the possible execution and/or error flags.

The names of the `vec_x` and `log` file are user-specified and included in the Input file `test_bfsai.fnames`.

The `vec_x` file lists the components of the converged solution $\mathbf{x}$.

The `log` file provides all the flags (warnings and/or errors) reported during the code execution.

The summary printed out on the screen at the end of the execution includes:

- `info` (integer): error code (if = 0 no errors encountered, else see the `log` file for details),
- `n_iter` (integer): number of PCG iterations to converge,
- `bnorm` (double precision real): 2-norm of the right-hand side $\mathbf{b}$,
- `resini` (double precision real): initial PCG relative residual,
- `resiter` (double precision real): final PCG relative residual,
- `resreal` (double precision real): real PCG relative residual,
- `T_prec` (real): execution time in seconds for the BFSAI-IC computation,
- `T_sol` (real): execution time in seconds for the PCG to converge,
- `T_tot` (real): total execution time in seconds.

3 Numerical examples

A numerical test is included in the available package using a matrix A with $n = 63$ and $n_t = 347$ (file `mat_63`) and a unitary right-hand side $\mathbf{b}$ (file `rhs`). The results are obtained on a machine equipped with an Intel(R) Core(TM) i7 CPU 920 at 2.67GHz with 4 computing cores (HT disabled), 128 kbyte of L1-Cache, 1Mbyte of L2-Cache, 8 Mbyte of L3-Cache, and 6 Gbyte of core memory.

The `parm_1` parameter file:

```
1         !  nproc
1         !  nbloc
2         !  kappa
0         !  rho_B
0         !  rho_L
1000      !  nzmax_F
1000      !  nzmax_JL
0.00d0    !  delta
0.0d0     !  tau_L

101       !  iout
990       !  itmax
1.0d-10   !  tol_CG
```

provides the following terminal log and PCG convergence profile:

```
Reading matrix and rhs
End reading

Computation of the A^kappa pattern

Computation of F

Computation of JL

Preconditioner computed

info = 0
System solved

-------------------------------------------------

Info:  0

# iter:  17

bnorm:  0.7937E+01
resini:  0.9815E+00
resiter:  0.7204E-10
resreal:  0.7204E-10

T_prec:  0.00
T_sol:  0.00
T_tot:  0.00

mat_F (nnz/density):  63 0.182
mat_JL (nnz/density):  174 0.501
Prec.  (nnz/density):  237 0.683


    iter        resiter
       1  0.2240872E+01
       2  0.1057432E+01
       3  0.3351808E+00
       4  0.7910917E-01
       5  0.2074677E-01
       6  0.2292885E-02
       7  0.7514988E-03
       8  0.8680529E-03
       9  0.2054852E-03
      10  0.8945910E-04
      11  0.1705649E-04
      12  0.2435383E-05
```

```
    13   0.3911664E-06
    14   0.4839887E-07
    15   0.7707951E-08
    16   0.4178030E-09
    17   0.7204189E-10
```

The `parm_2` and `parm_3` file prescribe $n_p = n_b = 2$ and 4, respectively, obtaining the following terminal logs:

```
Reading matrix and rhs
End reading

Computation of the A^kappa pattern

Computation of F

Computation of JL

Preconditioner computed

info = 0
System solved

-------------------------------------------------

Info:  0

# iter:  18

bnorm:  0.7937E+01
resini:  0.1002E+01
resiter:  0.3483E-10
resreal:  0.3483E-10

T_prec:  0.00
T_sol:  0.00
T_tot:  0.00

mat_F (nnz/density):  78 0.225
mat_JL (nnz/density):  170 0.490
Prec.  (nnz/density):  248 0.715
```

and:

```
Reading matrix and rhs
End reading

Computation of the A^kappa pattern
```

```
Computation of F

Computation of JL

Preconditioner computed

info = 0
System solved

-------------------------------------------------

Info:  0

# iter:  18

bnorm:  0.7937E+01
resini:  0.9541E+00
resiter:  0.3086E-10
resreal:  0.3087E-10

T_prec:  0.04
T_sol:  0.12
T_tot:  0.16

mat_F (nnz/density):  105 0.303
mat_JL (nnz/density):  165 0.476
Prec.  (nnz/density):  270 0.778
```

4 Copyright

BFSAI-IC OpenMP Implementation is freely available for scientific (non-commercial) use. It was written by Carlo Janna with contributions from his co-authors Massimiliano Ferronato and Nicola Castelletto.

1. BFSAI-IC OpenMP Implementation can be used only for the purpose of internal research excluding any commercial use of BFSAI-IC OpenMP Implementation as such or as a part of a software product. Users who want to integrate (parts of) BFSAI-IC OpenMP Implementation into commercial products need to have a license agreement.

2. BFSAI-IC OpenMP Implementation is provided on an "as is" basis and for the purpose described in paragraph 1 only. In no circumstances can neither the authors nor their institutions be held liable for any deficiency, fault or other mishappening with regard to the use or performance of BFSAI-IC OpenMP Implementation.

3. All scientific publications, for which BFSAI-IC OpenMP Implementation has been used, shall mention usage of BFSAI-IC OpenMP Implementation, and shall refer to the following publication:

Carlo Janna, Massimiliano Ferronato and Giuseppe Gambolati. A Block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems. SIAM J. Sci. Comput., 32:2468-2484, 2010.

Concerning the citation of the software package itself (current version is 1.0) we recommend to refer to it in the following way:

Carlo Janna, Massimiliano Ferronato and Nicola Castelletto. BFSAI-IC OpenMP Implementation. Available online at http://www.dmsa.unipd.it/~ferronat/software.html. Release V1.0, January 2011.

Please, document possible bugs and failures of the BFSAI-IC OpenMP Implementation code to one of the authors.