HSL_MP43

1 SUMMARY

The module `HSL_MP43` **uses the multiple front method to solve sets of unsymmetric linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ (or $\mathbf{A}\mathbf{X} = \mathbf{B}$) where $\mathbf{A}$ has been preordered to singly-bordered block-diagonal form**

$$\begin{pmatrix} \mathbf{A}_{11} & & & & \mathbf{C}_1 \\ & \mathbf{A}_{22} & & & \mathbf{C}_2 \\ & & \cdots & & \cdot \\ & & & \mathbf{A}_{NN} & \mathbf{C}_N \end{pmatrix}.$$

The HSL routines `MA42` and `MA52` are used with MPI for message passing.

In the multiple front method, a partial frontal decomposition is performed on each of the submatrices $(\mathbf{A}_{ll}\ \mathbf{C}_l)$ separately. Thus, on each submatrix, L and U factors are computed. Once all possible eliminations have performed, for each submatrix there remains a frontal matrix $\mathbf{F}_l$. The variables that remain in the front are called interface variables and the interface matrix $\mathbf{F}$ is formed by summing the matrices $\mathbf{F}_l$. The interface matrix $\mathbf{F}$ is also factorized using the frontal method. Block back-substitution completes the solution.

The matrix data and/or the matrix factors may optionally be held in direct-access files.

ATTRIBUTES — **Version:** 2.1.0. (20 April 2023) **Types:** Real (single, double). **Remark:** If **A** is a sum of finite elements, use `HSL_MP42` or `HSL_MP62`. **Calls:** `HSL_MP01`, `KB08`, `MA42`, `MA52`, `MC62`. **Language:** Fortran 95 + TR 15581 (allocatable components). **Original date:** September 2000. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

The module `HSL_MP43` has five separate phases:

(1) Initialize

(2) Analyse

(3) Factorize

(4) Solve

(5) Finalize

Prior to calling the first phase, the user must choose the number of processes and must initialize MPI by calling `MPI_INIT` on each process. The processes have rank `0`, `1`, `2`,... The **host** is the process with rank zero. The user must also define an MPI communicator for the package. The communicator defines the set of processes to be used by `MP43`. Each phase must then be called in order by each process (although the solve phase is optional). Before calling each phase, the user must have completed all other tasks that he or she was performing with the defined communicator (and with any other communicator that overlaps the `MP43` communicator). During the factorize phase, the matrix factors are generated. If right-hand sides are passed to the factorize phase, the solution is returned to the user at the end of the factorize phase. The user may solve for further right-hand sides, by calling the solve phase. The finalize phase deallocates all arrays that have been allocated by the package and, optionally, deletes all direct-access files used to hold the matrix factors. The user may factorize more than one matrix at the same time by running more than one instance of the package; an instance of the package is terminated by calling the finalize phase. After the finalize phase and once the user has completed any other calls to MPI routines he or she wishes to make, the user should call `MPI_FINALIZE` to terminate the use of MPI. This is illustrated in the simple example code given in Section 5.

Access to the module requires a USE statement and the user must declare a structure data of type MP43_DATA defined by the module. HSL_MP43 has a single user-callable subroutine MP43A/AD with a single parameter data of type MP43_DATA. If the user wishes to run more than one instance of the module at once, a separate parameter of type MP43_DATA is needed for each instance.

The parameter data has many components. In the following sections we describe the components of interest to the user. In Section 2.2, we describe the components that must be set by the user and that contain the solution vector, in Section 2.3 we describe the components that control the action, and in Section 2.4 we describe the components that hold information of potential interest to the user.

The following pseudocode illustrates how MP43A/AD must be used.

Single precision version

```
USE HSL_MP43_SINGLE
...
INTEGER ERCODE
TYPE (MP43_DATA) data
...
CALL MPI_INIT(ERCODE)
...
CALL MP43A (data)
...
CALL MPI_FINALIZE(ERCODE)
```

Double precision version

```
USE HSL_MP43_DOUBLE
...
INTEGER ERCODE
TYPE (MP43_DATA) data
...
CALL MPI_INIT(ERCODE)
...
CALL MP43AD (data)
...
CALL MPI_FINALIZE(ERCODE)
```

2.2 Input and output components

Prior to the first call to MP43A/AD for the current instance, the user must initialize the following component of data:

data%COMM is a scalar of type default INTEGER that must hold an MPI communicator. data%COMM must be initialized prior to the first call to MP43A/AD to define the set of processes to be used by MP43. Before each phase is called, the user must have completed all other tasks he or she was performing that involved data%COMM (or involved any other communicator that overlaps data%COMM). data%COMM is not altered. Note that the code may be run using a single process.

For each call to MP43A/AD, a job parameter is needed. This parameter determines which phase of the package is to be performed.

data%JOB is a scalar of type default INTEGER that must be initialized on **all** processes before **each** call to MP43A/AD. It must be given the same value on all processes. It is not altered.

JOB = 1 initializes an instance of the module. A call with JOB = 1 must be made before any other calls to the module. On exit, the components of data that control the action contain default values. If the user wishes to use values other than the defaults, the corresponding components of `data` should be reset after the call with `JOB` = `1`. Full details of the control components of `data` are given in Section 2.3.

`JOB` = `2` uses variable lists to generate lists of interface variables and, optionally, to reorder the rows within each submatrix and allocate submatrices to processes.

`JOB` = `3` uses the information from the call with `JOB` = `2` to generate the submatrix *L* and *U* factors, form the *L* and *U* factors for the interface problem and, optionally, to solve the equations $\mathbf{AX}=\mathbf{B}$. A call with `JOB` = `3` must be preceded by a call with `JOB` = `2`.

`JOB` = `23` performs `JOB` = `2` then `JOB` = `3`.

`JOB` = `4` uses the factors produced by a call with `JOB` = `3` to rapidly solve further systems of the form $\mathbf{AX}=\mathbf{B}$. A call with `JOB` = `4` must be preceded by a call with `JOB` = `3` (or `23`) but several calls with `JOB` = `4` may follow a single call with `JOB` = `3` (or `23`).

`JOB` = `5` deallocates all arrays that have been allocated by the module and, optionally, deletes all direct-access files that have been used to hold the matrix factors. A call with `JOB` = `5` should be made after all other calls for the current instance are complete. Note that components of `data` that are allocated by the user are **not** deallocated.

2.2.1 Input components for `data%JOB` = `2` or `23`

Prior to a call with `data%JOB` = `2` or `23`, the following components **must** be set by the user on the host (note that if `data%JOB` = `23`, the input components described in Section 2.2.3 must also be set):

`data%NBLOCK` is a scalar of type default `INTEGER` that must be set to the number *N* of submatrices. This component is not altered. **Restriction:** `data%NBLOCK` > `1`.

`data%NEQ` is a scalar of type default `INTEGER` that must be set to the order of **A**. This component is not altered. **Restriction:** `data%NEQ` ≥ `data%NBLOCK`.

`data%NEQSB` is a rank-1 allocatable array of type default `INTEGER` that mus be allocated by the user with size `data%NBLOCK`. On entry, `data%NEQSB(L)` must hold the number of rows in the `L`-th submatrix (`L` = `1`, 2,..., `data%NBLOCK`). This component is not altered. **Restriction:** `data%NEQSB(:)` > `0`.

`data%EQVAR` is a rank-1 allocatable array of type default `INTEGER` that must be allocated by the user and set to contain lists of the variable indices in each of the rows of **A**. The variable indices for row 1 must precede those for row 2, and so on. Within a row, the variable indices may be in any order. If duplicate indices are detected in a row, or if variable indices less than `1` are found, the computation terminates with an error. This component is not altered.

`data%EQPTR` is a rank-1 allocatable array of type default `INTEGER` that must be allocated with size at least `data%NEQ+1`. `data%EQPTR(I)` must contain the position in `data%EQVAR` of the first entry in the `I`-th row of **A** (`I` = `1`, 2,..., `data%NEQ`), and `data%EQPTR(data%NEQ+1)` must be set to the position after the last entry in the last row. This component is not altered.

Additionally, the following component must be allocated and set if the control component `data%ICNTL(10)` (see Section 2.3) is nonzero.

`data%INV_LIST` is a rank-1 allocatable array of type default `INTEGER` that must be allocated with size at least `data%NBLOCK`. On entry, `data%INV_LIST(L)` must hold the rank of the process that is to factorize submatrix `L` (`L` = `1`, 2,..., `data%NBLOCK`). This component is not altered. **Restriction:** `0` ≤ `data%INV_LIST(:)` < `data%NPROC-1` (where `data%NPROC` is the number of processes, see Section 2.4.1).

The following component must be allocated and set if `data%ICNTL(9)` = `1`.

`data%WT` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version) that must be allocated with size `3` by `data%NBLOCK`. It must be set so that `data%WT(1`:3,`L)` holds the weights to be used by the row ordering code `MC62` on submatrix `L` (`L` = `1`, 2,..., `data%NBLOCK`).

The following component must be allocated and set if `data%ICNTL(9)` $>$ `1`.

`data%NORDER` is a rank-2 allocatable array of type default `INTEGER` that must be allocated with size $\max_{1 \le L \le data\%NBLOCK}$ `data%NEQSB(L)+1` by `data%NBLOCK`. It must hold the order in which the rows are passed to the frontal matrix. Within submatrix `L`, the rows are locally labelled `1`, 2,..., `data%NEQSB(L)`. On submatrix `L`, the rows will be passed to the frontal matrix during the factorize phase in the order `data%NORDER(1, L)`, `data%NORDER(2, L)`,..., `data%NORDER(data%NEQSB(L), L)`. Note that if the user has already used `MP43` to factorize a matrix with a given sparsity pattern and wishes to factorize another matrix with the same pattern, significant savings may be made by setting `data%ICNTL(9)` to a value greater than `1` and leaving `data%NORDER` unchanged since the factorization of the original matrix.

2.2.2 Output components for `data%JOB` = `2` or `23`

The following components are allocated on each process.

`data%INV_LIST` is a rank-1 allocatable array of type default `INTEGER` of size `data%NBLOCK`. On exit, `data%INV_LIST(L)` holds the rank of the process that is to factorize submatrix `L` (`L` = `1`, 2,..., `data%NBLOCK`).

`data%ENTRIES` is a rank-1 allocatable array of type default `INTEGER` of size `data%NBLOCK`. On exit, `data%ENTRIES(L)` holds the number of nonzero entries in submatrix `L` (`L` = `1`, 2,..., `data%NBLOCK`).

`data%ICOUNT` is a rank-1 allocatable array of type default `INTEGER` of dimension `0:data%NPROC-1`. On exit, `data%ICOUNT(IPROC)` holds the number of submatrices assigned to process `IPROC` (`IPROC` = `0`, `1`,..., `data%NPROC-1`).

`data%IBLOCK` is a rank-1 allocatable array of type default `INTEGER`. On exit, on process `IPROC`, `data%IBLOCK(J)` holds the index of the `J` th submatrix that is to be factorized by the process (`J` = `1`, 2,..., `data%ICOUNT(IPROC)`).

2.2.3 Input components for `data%JOB` = `3` or `23`

Prior to a call with `data%JOB` = `3` or `23` the following additional components must be set on the host (note that if `data%JOB` = `23`, the input components described in Section 2.2.1 must also be set):

`data%NRHS` is a scalar of type default `INTEGER` that must be set to the number of right-hand sides. This component is not altered. **Restriction:** `data%NRHS` $\ge$ `0`.

`data%B` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version). If `data%NRHS` $>$ `0`, `data%B` must be allocated with size `data%NEQ` by `data%NRHS`. On entry, `data%B(I, J)` must be set by the user to the `I`-th component of the `J`-th right-hand side of the equations being solved (`I` = `1`, 2,..., `data%NEQ`, `J` = `1`, 2,..., `data%NRHS`). This component is not altered.

The remaining components of `data` that must be set depend on the settings for the control components.

The following component must be set on the host if `data%ICNTL(7)` $=$ `0` (the default).

`data%MFREQ` is a scalar of type default `INTEGER` that must be at least as large as the maximum number of entries per row (see `data%MAX_NDF` in Section 2.4.2). `data%MFREQ` determines the record length given in the `RECL=` specifier of the `OPEN` statements for the direct-access files for the matrix data (see `data%ICNTL(7)` in Section 2.3). This component is not altered.

The following component must be allocated and set on the host if `data%ICNTL(7)` $=$ `0` (the default), `1`, or `2`.

`data%VALNAM` is a rank-1 allocatable array of type default `CHARACTER*128` that must be allocated with size at least `data%NBLOCK`. On entry, `data%VALNAM(L)` must contain the name of the file holding the rows of **A** belonging to submatrix `L` (`L` = `1`, 2,..., `data%NBLOCK`). `data%VALNAM` is not accessed if `data%ICNTL(7)` $\ge$ `3`. This component is not altered.

The following component must be allocated and set on the host only if `data%ICNTL(7)` $=$ `3` and on each process if `data%ICNTL(7)` $=$ `4`.

data%VALUES is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) that must be allocated and must contain the rows of **A**. The entries must be in the order given by data%EQVAR. This component is not altered.

The following component must be allocated and set on each process IPROC if data%ICNTL(7) = 5.

data%RVAL is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) that must be allocated and must contain the rows of the submatrices assigned to process IPROC. The entries for submatrix data%IBLOCK(1) must precede those for data%IBLOCK(2), and so on, and within each submatrix, the entries must be in the order given by data%EQVAR. The size of data%RVAL on process IPROC must be at least $\sum$data%ENTRIES(L) where the summation is over the submatrices L assigned to IPROC. This component is not altered.

The following component must be allocated and set if data%ICNTL(13) is nonzero.

data%LENBUF is a rank-2 allocatable array of type default INTEGER that must be allocated with size 3 by data%NBLOCK+1. It is used to hold the sizes (in words) of the buffers (workspace arrays) used by the frontal solver MA42 for the matrix factors. On entry, data%LENBUF(J, L), J = 1, 2, 3, must hold, respectively, the buffer size for the *U* factor (including the corresponding right-hand sides), the *L* factor, and the row and column indices for the factors on submatrix L (L = 1, 2,..., data%NBLOCK). data%LENBUF(J, data%NBLOCK+1), J = 1, 2, 3, must hold the corresponding buffer sizes for the interface problem, which is solved on the host. data%LENBUF(2, :) is set to zero if the user has reset data%ICNTL(11) to a nonzero value (see Section 2.3). Note that the workspace required by MP43 is dependent on the size of the buffers and for efficiency the buffers should be chosen as large as space permits. If direct-access files are not being used (data%ICNTL(14) = 0), the buffers must be large enough to hold the matrix factors. On exit, data%LENBUF holds the buffer sizes used by MP43. These differ from the input values if the user-supplied values are too large to enable the required workspace to be allocated, or if direct-access files are not being used and one or more of the buffer sizes supplied by the user is too small (see warning +4). **Restriction:** data%LENBUF(:,:) $\geq$ 1.

The following component must be allocated and set if data%ICNTL(14) < 0.

data%FILES1 is a rank-2 allocatable array of type default CHARACTER*128 that must be allocated with size 3 by data%NBLOCK+1. On entry, data%FILES1(J, L), J = 1, 2, 3, must hold the names of the direct-access files for the *U* factor, the *L* factor, and the row and column indices for the factors on submatrix L (L = 1, 2,..., data%NBLOCK) and for the interface problem when L = data%NBLOCK+1.

The following component must be allocated and set if data%ICNTL(15) < 0.

data%FILES2 is a rank-2 allocatable array of type default CHARACTER*128 that must be allocated with size 2 by data%NBLOCK. On entry, data%FILES2(1, L)) must hold the name of the sequential file for holding the data that remains in the frontal matrix once the factorization on submatrix L is complete (L = 1, 2,..., data%NBLOCK). If data%NRHS > 0, data%FILES2(2, L) must hold the name of the sequential file for the corresponding right-hand sides. data%FILES2(2, L) is not accessed if data%NRHS = 0.

The following component must be allocated and set if data%CNTL(2) < 0.0.

data%ORDER62 is a rank-1 allocatable array of type default INTEGER that must be allocated with size at least the number of rows in the interface problem. It must hold the order in which the rows of the interface problem are assembled. Within the interface matrix the rows are locally labelled 1, 2,..., data%NINTER, with the rows from submatrix 1 first, followed by those from submatrix 2, and so on. Note that if the user has already used MP43 to factorize a matrix with a given sparsity pattern and wishes to factorize another matrix with the same pattern, significant savings may be made by setting data%CNTL(2) less than zero and leaving data%ORDER62 unchanged since the factorization of the original matrix.

2.2.4 Output components for `data%JOB = 3` or `23`

The following component is allocated if `data%NRHS` > `0` and is used to hold the solution vector.

`data%X` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version) of size `data%NEQ` by `data%NRHS`. On exit, if `ICNTL(17)` = `0`, on the host `data%X(I, J)` holds the solution for the `I`-th component of the `J`-th right-hand side (`I` = `1`, `2`,..., `data%NEQ`, `J` = `1`, `2`,..., `data%NRHS`). If `ICNTL(17)` ≠ `0`, if the solution for `I`-th component has been computed by the process with rank `IPROC`, then on process `IPROC`, `data%X(I, J)` holds the solution for the `I`-th component of the `J`-th right-hand side and is set to zero otherwise (`IPROC` = `0`, `1`,..., `data%NPROC`).

2.2.5 Input components for `data%JOB = 4`

Prior to a call with `data%JOB` = `4` the following components must be set on the host:

`data%NRHS` is a scalar of type default `INTEGER` that must be set to the number of right-hand sides. This component is not altered. **Restriction:** `data%NRHS` ≥ `1`.

`data%B` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version) that must be allocated with size `data%NEQ` by `data%NRHS`. On entry, `data%B(I, J)` must be set by the user to the `I`-th component of the `J`-th right-hand side of the equations being solved (`J` = `1`, `2`,..., `data%NRHS`). This component is altered.

2.2.6 Output components for `data%JOB = 4`

The following component is allocated on a call with `data%JOB` = `4` and is used to hold the solution vector, on the host only.

`data%X` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version) of size `data%NEQ` by `data%NRHS`. On exit, if `ICNTL(17)` = `0`, on the host `data%X(I, J)` holds the solution for the `I`-th component of the `J`-th right-hand side (`I` = `1`, `2`,..., `data%NEQ`, `J` = `1`, `2`,..., `data%NRHS`). If `ICNTL(17)` ≠ `0`, if the solution for `I`-th component has been computed by the process with rank `IPROC`, then on process `IPROC`, `data%X(I, J)` holds the solution for the `I`-th component of the `J`-th right-hand side and is set to zero otherwise (`IPROC` = `0`, `1`,..., `data%NPROC`).

2.3 Control components

On exit from the initial call (`data%JOB` = `1`), the control components of `data` are set to default values. If the user wishes to use values other than the defaults, the corresponding components of `data` should be reset on the host process after the initial call.

`data%ICNTL` is a rank-1 array of type default `INTEGER` and size `30`.

`ICNTL(1)` is the unit number for error messages and has the default value `6`. Printing of error messages is suppressed if `ICNTL(1)` < `0`.

`ICNTL(2)` is the unit number for warning messages and has the default value `6`. Printing of warning messages is suppressed if `ICNTL(2)` < `0`.

`ICNTL(3)` is the unit number for diagnostic printing and has the default value `6`. Printing is suppressed if `ICNTL(3)` < `0`.

`ICNTL(4)` is used to control printing of diagnostic messages (all printing is on the host only). It has default value `1`. Possible values are:

≤`0` No printing.

`1` Error and warning messages only.

`2` As `1`, plus some additional diagnostic printing.

3 As 2, but timings of parts of the code (elapsed wall clock times in seconds) are also printed.

`ICNTL(5)` is not currently used. It is given the default value `0`.

`ICNTL(6)` controls whether zeros in the front are exploited. Zeros within the front are ignored if `ICNTL(6)=0`. The default value is `1`.

`ICNTL(7)` is used to control how the user wishes to supply the matrix **A** on a call with `data%JOB` = `3` or `23`. There are 4 options:

- `0` The submatrices ($\mathbf{A}_{ll}\ \mathbf{C}_l$) are held in direct-access files. The data required by the process with rank `IPROC` must be readable by that process (`IPROC` = `0`, `1`, ..., `data%NPROC-1`). For each submatrix, the data is read in row-by-row as required by the corresponding process. This minimises storage requirements and data movement between processes. After a call with `data%JOB = 2`, `data%INV_LIST(L)` holds the rank of the process that is to factorize submatrix `L`. For each submatrix `L` to be factorized by process `IPROC`, there must be an unformatted direct-access file holding the values of the entries in the rows of the submatrices that can be read by process `IPROC`. The record length given in the `RECL=` specifier of the `OPEN` statements for the direct-access files holding the submatrices must be that required for a `REAL` (or double precision `REAL` in the `double` version) array of size `data%MFREQ`. The entries of the submatrices must be written to the direct-access files in the same order as they are held in `data%EQVAR`. The name of the file for submatrix `L` must be given in `data%VALNAM(L)` (`L` = `1`, `2`, ..., `data%NBLOCK`).
- `1` As `0`, except the data are held in unformatted sequential files. In this case, all the data for a submatrix is read in by the process to which it is assigned at once. This requires more memory.
- `2` As `1`, except the host must be able to read all the files. For each submatrix, the data is read by the host and then passed to the process assigned to that submatrix before the factorization begins. This requires the host to have more memory and each process must be able to store the data for all the submatrices assigned to it at once.
- `3` The user must supply the submatrix data in memory on the host using `data%VALUES`. This option avoids reading from direct-access or sequential files although it does involve more data movement between processes than `ICNTL(7)=0` or `1`.
- `4` The user must supply all the submatrix data in memory on each process using `data%VALUES`. Supplying the data in this way avoids reading from direct-access or sequential files as well as data movement between processes. This option is suitable for shared memory machines.
- `5` On each process, the user must supply the data for each of the submatrices assigned to it in memory using `data%RVAL`. Supplying the data in this way avoids reading from direct-access or sequential files as well as data movement between processes; the amount of data required to be held on each process is less than for `ICNTL(7)=4` (assuming more than one process is used).

Restriction: `ICNTL(7)` = `0`, `1`, `2`, `3`, `4`, or `5`.

`ICNTL(8)` has the default value `0`. If the matrix is found to be singular during the decomposition and `ICNTL(8)` is equal to `0`, an error flag is set and the computation terminates (see error return `-12` in Section 2.5). If `ICNTL(8)` is nonzero, a warning is given and the computation continues.

`ICNTL(9)` controls the order in which the rows in the submatrices are passed to the frontal solver. If `ICNTL(9)=0` (the default), during the analyse phase (`data%JOB = 2`), the order is generated automatically using `MC62` with default setting for the weights. If `ICNTL(9)=1`, the row order is again generated using `MC62` but the weights must be supplied by the user in `data%WT`. If `ICNTL(9)>1`, the user must specify the assembly order using the array `data%NORDER`. If `ICNTL(9)<0`, within each block the first row is assembled first, followed by the second row, and so on.

`ICNTL(10)` is used to control whether the user wishes to decide which process is to factorize which submatrix. If

ICNTL(10) = 0 (the default), this choice is made automatically during the analyse phase (data%JOB = 2). Otherwise, the user must choose a process for each submatrix using data%INV_LIST.

ICNTL(11) controls whether the user wishes to call MP43 with JOB = 4 to solve for further right-hand sides. If ICNTL(11) = 0 (the default), both the *L* and *U* factors are stored and the solve phase may be called. If ICNTL(11) is nonzero, the *L* factor is not stored. This reduces storage requirements but MP43 may not be called with JOB = 4.

ICNTL(12) controls whether the user wants the direct-access files used to hold the matrix factors to be deleted at the end of the computation. If ICNTL(12) = 0 (the default), when the final call is made to MP43 (data%JOB = 5), the direct-access files are deleted. Otherwise, the direct-access files are disconnected but not deleted. ICNTL(12) is not used if ICNTL(14) is equal to 0.

ICNTL(13) controls the size of the buffers (work arrays) associated with the direct-access files used to hold the matrix factors. If ICNTL(13) = 0 (the default), the buffer sizes are chosen by the code. If ICNTL(13) is nonzero, the user must set buffer sizes in data%LENBUF.

ICNTL(14) controls whether or not direct-access files are used to hold the matrix factors. If ICNTL(14) = 0 (the default), direct-access files are **not** used and the factors are held in main memory. Otherwise, unformatted direct-access files are used. If ICNTL(14) > 0, the code automatically names the files and they are written to the current directory. The files for the factors on submatrix 1 are called ufact.0001, lfact.0001, integ.0001, on submatrix 2 they are ufact.0002, lfact.0002, integ.0002, and so on. The files for the factors for the interface problem are ufact_interf, lfact_interf, integ_interf. If ICNTL(14) < 0, the user must supply names for the files in data%FILES1. If the user wishes to run a second instance of the module before the final call for the first instance (data%JOB = 5) and wants to use files for the factors, data%ICNTL(14) **must** be set to a negative value and file names provided by the user in data%FILES1.

ICNTL(15) controls whether or not sequential files are used to hold the data that remains in the frontal matrix and corresponding right-hand side once the factorization on the submatrix is complete. If ICNTL(15) = 0 (the default), files are **not** used. Otherwise, unformatted sequential files are used (using files reduces storage requirements but the extra I/O involved can increase the overall computational time). If ICNTL(15) > 0, the code automatically names the files and they are written to the current directory. The remaining data for submatrix 1 is written to files fvar.0001 and (if data%NRHS > 0) frhs.0001, for submatrix 2 the files are fvar.0002 and frhs.0002, and so on. If ICNTL(15) < 0, the user must supply names for the files for the factors in data%FILES2. If the user wishes to run a second instance of the module before the final call for the first instance (call with JOB = 5) and wants to use files to hold the remaining frontal data, data%ICNTL(15) **must** be set to a negative value and file names provided by the user in data%FILES2.

ICNTL(16) has the default value 8. ICNTL(16) is the minimum number of variables that are eliminated at each stage of the factorization. Increasing ICNTL(16) in general increases the number of floating-point operations and real storage requirements but allows greater advantage to be taken of Level 3 BLAS and reduces integer storage.

ICNTL(17) controls whether or not the solution vector is assembled on the host. If ICNTL(17) = 0 (the default), the solution is assembled in data%X on the host. Otherwise, on exit, each process holds the part of the solution vector that it computed.

ICNTL(18) controls whether or not the BLAS are used during the solve phase JOB = 4. If ICNTL(18) = 0 (the default) and data%NRHS = 1, BLAS are not used. Otherwise, BLAS are used.

ICNTL(19) to ICNTL(30) are not currently used but are set to zero.

data%CNTL is a rank-1 array of type default REAL (or double precision REAL in the double version) and size 10.

CNTL(1) has the default value zero. The matrix is declared singular if, during the factorization, the entry of largest absolute value in any column is less than or equal to CNTL(1).

CNTL(2) controls whether or not the rows of the interface problem are reordered using MC62. If the density of the interface problem (that is, the the number of entries in the interface matrix divided by the product of the number of interface rows and columns) is less than CNTL(2), or if CNTL(2) is equal to zero, MC62 is used (with default settings). If CNTL(2) is greater than 0.5, MC62 is not used and the interface rows from block 1 are numbered first, followed by those from block 2, and so on. If CNTL(2) < 0, the row order for the interface problem must be supplied by the user in data%ORDER62. The default value is CNTL(2) = 0.05. Note that if there are a large proportion of nonzero entries in the interface matrix, using MC62 may add a significant overhead and is unlikely to reduce the frontsizes for the interface problem. However, if the interface problem is very sparse, MC62 can result in significant svaings.

CNTL(3) to CNTL(10) are not currently used but are set to zero.

2.4 Information components

The components of data described in this section are used to hold information that may be of interest to the user. Some of the information is available on each process and some only on the host.

2.4.1 Information on each process

The following information is significant on each process. The information is available after a call to MP43 with data%JOB = 1, 2, 3, 23, 4, 5.

data%ERROR is a variable of type default INTEGER that is used as an error and a warning flag. A nonzero value indicates an error has been detected or a warning has been issued (see Section 2.5). If an error is detected, the information contained in the other components of data described in this section may be incomplete.

data%NPROC is a variable of type default INTEGER that is set to the number of processes used by MP43. data%NPROC is the number of processes associated with the communicator data%COMM and is set by a call within MP43 to MPI_COMM_SIZE.

data%RANK is a variable of type default INTEGER that holds the rank of the process in the global communicator data%COMM. The host is defined to be the process with data%RANK = 0.

2.4.2 Information available on the host

The following information is available **only** on the host. If an error is detected (see Section 2.5), the information may be incomplete.

Information available on exit from a call with data%JOB = 2 or 23.

data%IOSTAT is a variable of type default INTEGER that holds Fortran IOSTAT parameter.

data%MAX_NDF is a variable of type default INTEGER that holds the maximum number of entries in a row of **A**.

data%STAT is a variable of type default INTEGER that holds Fortran STAT parameter.

data%NGUARD is a rank-1 allocatable array of type default INTEGER of size data%NBLOCK. data%NGUARD(L) holds the number of interface variables for submatrix L (L = 1, 2,..., data%NBLOCK).

data%FLAG_52 is a rank-1 allocatable array of type default INTEGER of size data%NBLOCK. data%FLAG_52(L) holds the MA52A/AD error flag for submatrix L (L = 1, 2,..., data%NBLOCK).

data%NDF is a rank-1 allocatable array of type default INTEGER of size data%NBLOCK. data%NDF(L) holds the number of variables in submatrix L (L = 1, 2,..., data%NBLOCK).

data%NFRONT is a rank-2 allocatable array of type default INTEGER of size 2 by data%NBLOCK. data%NFRONT(1:2, L) hold the maximum row and column frontsizes for the frontal elimination on submatrix L (L = 1, 2,..., data%NBLOCK).

data%AVFRT is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) of size data%NBLOCK. data%AVFRT(L) holds the average frontsize for the frontal elimination on submatrix L (L = 1, 2,..., data%NBLOCK).

data%OPS is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) of size data%NBLOCK. data%OPS(L) holds an estimate of the number of floating-point operations (flops) in the innermost loops for the frontal elimination on submatrix L (L = 1, 2,..., data%NBLOCK).

data%FLSIZE is a rank-2 allocatable array of type default REAL of size 2 by data%NBLOCK. Provided the matrix is non-singular, data%FLSIZE(J, L), J = 1, 2, 3 hold, respectively, the upper bounds in storage needed for the real data for the *U* and *L* factors and the integer data for the factors on submatrix L (L = 1, 2,..., data%NBLOCK). Note that, if data%NRHS right-hand sides are to be solved on the subsequent call with data%JOB = 3 then, since the right-hand side vectors are stored with the *U* factor, the storage needed for the *U* factor (including right-hand sides) is data%FLSIZE(1,L)+ data%NRHS*(data%NDF(L) – data%NGUARD(L))).

data%NORDER is a rank-2 allocatable array of type default INTEGER of size $\max_{1 \le L \le NBLOCK}$ data%NEQSB(L)+1 by data%NBLOCK. Within submatrix L, the rows are locally labelled 1, 2,..., data%NEQSB(L). Within submatrix L, the rows will be passed to the frontal matrix during the factorize phase in the order data%NORDER(1, L), data%NORDER(2, L),..., data%NORDER(data%NEQSB(L), L).

data%WT is a rank-2 allocatable array of type default REAL (or double precision REAL in the double version) of size 3 by data%NBLOCK. data%WT(1:3,L) holds the weights used by the row ordering code MC62 on submatrix L (L = 1, 2,..., data%NBLOCK).

Information available on exit from a call with data%JOB = 3 or 23.

data%IOSTAT is a variable of type default INTEGER that holds Fortran IOSTAT parameter.

data%NINTER is a variable of type default INTEGER that holds the number of rows in the interface problem.

data%NINTER COL is a variable of type default INTEGER that holds the number of columns in the interface problem. This may differ from data%NINTER if the problem is singular.

data%NONZERO is a variable of type default INTEGER that holds the number of entries in the interface matrix.

data%STAT is a variable of type default INTEGER that holds Fortran STAT parameter.

data%STATIC is a variable of type default INTEGER that holds the total number of static condensations performed.

data%SINGULAR is a variable of type default LOGICAL that is set to .TRUE. if the matrix is found to be singular.

data%FLOPS is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of floating-point operations (flops) in the innermost loops of the factorization (this is the total for all the submatrices and the interface).

data%NZL is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of entries in the *L* factors.

data%NZU is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of entries in the *U* factors.

data%NZURHS is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of entries in the *U* factors, plus the right-hand sides (if data%NRHS = 0, data%NZU = data%NZURHS).

data%STORINT is a variable of type default REAL (or double precision REAL in the double version) that holds the total storage for the row and column indices in INTEGER words.

data%ORDER62 is a rank-1 allocatable array of type default INTEGER of size data%NINTER. Within the interface matrix, the rows are locally labelled 1, 2,..., data%NINTER (with the rows from the first submatrix numbered first, followed by the second submatrix, and so on) The order in which the interface rows are passed to the frontal solver is data%ORDER62(1), data%ORDER62(2),..., data%ORDER62(data%NINTER).

`data%NLEFT` is a rank-2 allocatable array of type default `INTEGER` of size `2` by `data%NBLOCK`. `data%NLEFT(1:2, L)` holds the number of rows and columns left in the front at the end of the elimination process for submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`).

`data%INFO_42` is a rank-2 allocatable array of type default `INTEGER` of size `23` by `data%NBLOCK+1`. `data%INFO_42(:, L)` holds the `MA42B/BD` integer information array for submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`) and for `L` = `data%NBLOCK+1` it holds the `MA42B/BD` integer information array for the interface problem. Note that on the submatrices the information in `data%INFO_42` is incomplete since the factorization on the submatrix is a partial factorization (interface variables not eliminated).

`data%RINFO_42` is a rank-2 allocatable array of type default `REAL` (or double precision `REAL` in the `double` version) of size `2` by `data%NBLOCK+1`. `data%RINFO_42(:, L)` holds the `MA42B/BD` real information array for submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`) and for `L` = `data%NBLOCK+1`, it holds the `MA42B/BD` real information array for the interface problem. Note that on the submatrices the information in `data%RINFO_42` is incomplete since the factorization on the submatrix is a partial factorization (interface variables not eliminated).

`data%INFO_62` is a rank-2 array of type default `INTEGER` of size `10` by `data%NBLOCK`. If `ICNTL(9)` ≥ `0`, `data%INFO_62(:, L)` holds the `MC62A/AD` integer information array for the submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`). If `MC62` is used to order the interface problem (see `CNTL(2)`), `data%INFO_62(:, NBLOCK+1)` holds the `MC62A/AD` integer information array for the interface problem and is set to zero otherwise.

`data%RINFO_62` is a rank-2 array of type default `REAL` (or double precision `REAL` in the `double` version) of size `30` by `data%NBLOCK`. If `ICNTL(9)` ≥ `0`, `data%RINFO_62(:, L)` holds the `MC62A/AD` real information array for the submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`). If `MC62` is used to order the interface problem (see `CNTL(2)`), `data%RINFO_62(:, NBLOCK+1)` holds the `MC62A/AD` real information array for the interface problem and is set to zero otherwise.

Information available on exit from a call with `data%JOB` = `4`.

`data%IOSTAT` is a variable of type default `INTEGER` that holds Fortran `IOSTAT` parameter.

`data%STAT` is a variable of type default `INTEGER` that holds Fortran `STAT` parameter.

2.5 Error diagnostics

On successful completion, a call to `MP43` will exit with the component `data%ERROR` set to `0`. Other values for `data%ERROR` and the reasons for them are given below.

2.5.1 Error diagnostics for `data%JOB` = `1`

–1 MPI has not been initialized by the user. Immediate return. An error message is printed on the default output unit.

2.5.2 Error and warning diagnostics for `data%JOB` = `2` or `23`

A negative value for `data%ERROR` is associated with a fatal error. Error messages are output on unit `data%ICNTL(1)`. Possible negative values are:

–2 Either `data%NBLOCK` ≤ `1` or `data%NBLOCK` > `data%NEQ`.

–4 Either `data%EQVAR` is not allocated or has been allocated with size less than `data%EQPTR(data%NEQ+1)-1`. This error is also returned if one or more variable indices in `data%EQVAR` is out of range (ie. is less than 1) or there are duplicate entries in a row. `data%IOUT` and `data%IDUP` hold, respectively, the number of out of range and duplicate entries.

–5 Error detected in `data%EQPTR`. Either `data%EQPTR` has not been allocated or has been allocated with size less than `data%NEQ+1`, or the entries of `data%EQPTR` are not monotonic increasing.

–6 `data%ICNTL(7)` out of range (ie. `data%ICNTL(7)` ≠ `0`, `1`, `2`, `3`, `4`, or `5`).

–7 Either the array data%NEQSB has not been allocated or has been allocated with size less than data%NBLOCK, or data%NEQSB(L) < 1 for one or more of the submatrices L (1 ≤ L ≤ data%NBLOCK).

–8 One or more of the submatrices have no interface variables.

–9 Either the array data%INV_LIST has not been allocated or has been allocated with size less than data%NBLOCK, or an entry in data%INV_LIST is out of range (data%ICNTL(10) nonzero).

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT. If the user is not using direct-access files (data%ICNTL(14) = 0), it may be possible to avoid this error by rerunning with data%ICNTL(14) ≠ 0, or if data%ICNTL(13) is nonzero, by reducing the buffer sizes held in data%LENBUF.

–13 data%JOB does not have the same value on all processes or has an invalid value.

–18 The call follows a call with data%JOB = 5.

–24 Either the array data%NORDER has not been allocated or has been allocated with an incorrect size (ICNTL(9) > 1). This error is also returned if the supplied row order is found not to be a permutation for one or more of the submatrices.

–27 Either the array data%WT has not been allocated or has been allocated with an incorrect size (ICNTL(9) = 1).

Warning messages are associated with positive values of data%ERROR. Warning messages are output on data%ICNTL(2). Possible warnings are:

+1 MC62A/AD has not reduced the average the frontsize for one or more of the submatrices.

2.5.3 Error diagnostics for data%JOB = 3 or 23

A negative value for data%ERROR is associated with a fatal error. Error messages are output on unit data%ICNTL(1). Possible values are:

–10 If data%ICNTL(7) = 0, 1, or 2, either data%VALNAM is not allocated or is allocated with the size less than data%NBLOCK. If data%ICNTL(7) = 3 or 4, either data%VALUES is not allocated or is allocated with size less than data%EQPTR(data%NEQ+1)-1. If data%ICNTL(7) = 5, either data%RVAL is not allocated or is allocated with incorrect size.

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT.

–12 Singularity detected in the matrix during the factorization with the control component data%ICNTL(8) equal to zero.

–13 data%JOB does not have the same value on all processes or has an invalid value.

–14 Error in Fortran INQUIRE statement. data%IOSTAT holds the IOSTAT parameter.

–15 Error when writing to a direct-access file.

–16 Error when reading from a direct-access file.

–17 Error in Fortran OPEN statement.

–19 An error was returned on a previous call or the call follows a call with data%JOB = 1 (no data%JOB = 2 call) or follows a call with data%JOB = 5.

–20 Failed to find a unit to which a file could be connected.

–21 Either data%LENBUF has not been allocated or has been allocated with incorrect size, or data%LENBUF(1,L) ≤ 0, or data%LENBUF(3,L) ≤ 0, or data%ICNTL(11) = 0 and data%LENBUF(2,L) ≤ 0 (1 ≤ L ≤ data%NBLOCK+1) (data%ICNTL(13) nonzero).

–22 data%NRHS < 0. This error is also returned if data%NRHS > 0 but data%B is either not allocated or is allocated but with incorrect size.

–23 data%ICNTL(7) = 0 and data%MFREQ is less than the maximum number of entries in a row. data%MAX_NDF holds the maximum number of entries in a row.

–25 data%FILES1 is either not allocated or is allocated but with incorrect size (ICNTL(14) < 0).

–26 data%FILES2 is either not allocated or is allocated but with incorrect size (ICNTL(15) < 0).

–28 Either the array data%ORDER62 has not been allocated or has been allocated with an incorrect size (CNTL(2) < 0.0). This error is also returned if the supplied row order for the interface matrix is found not to be a permutation.

Warning messages are associated with positive values of data%ERROR. Warning messages are output on data%ICNTL(2). Possible warnings are:

+2 The matrix **A** has been found to be singular and the control component data%ICNTL(8) was nonzero (see Section 2.4).

+4 One or more of the user-supplied buffer sizes in data%LENBUF have been altered (data%ICNTL(13) nonzero).

+8 MC62A/AD has either not been used on the interface problem or has not reduced the average the frontsize for the interface problem.

+3,+5,+6,+7,+9,+10,+11,+12,+13,+14,+15 Combination of the above warnings.

2.5.4 Error diagnostics for data%JOB = 4

A negative value for data%ERROR is associated with a fatal error. Error messages are output on unit data%ICNTL(1). Possible values are:

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT.

–13 data%JOB does not have the same value on all processes or has an invalid value.

–16 Error when reading from a direct-access file.

–17 Error in Fortran OPEN statement.

–19 An error was returned on a previous call or the call was not preceded by a call with data%JOB = 3 or 23, or follows a call with data%JOB = 3 and data%ICNTL(11) nonzero, or follows a call with data%JOB = 5.

–22 data%NRHS < 1. This error is also returned if data%NRHS ≥ 1 but data%B is either not allocated or is allocated but with incorrect size.

2.5.5 Error diagnostics for data%JOB = 5

A negative value for data%ERROR is associated with a fatal error. Error messages are output on unit data%ICNTL(1). Possible values are:

–13 data%JOB does not have the same value on all processes or has an invalid value.

3 GENERAL INFORMATION

3.1 Summary of information.

Use of common: Common blocks are not used.

Other routines called directly: The HSL routines HSL_MP01, MA42I/ID, MA42A/AD, MA42B/BD, MA42J/JD, MA52A/AD, MA52K/KD, MC62I/ID, MC62A/AD, KB08A/AD. Subroutines private to the module are MP43B/BD, MP43C/CD, MP43D/DD, MP43F/FD, MP43G/GD, MP43H/HD, MP43K/KD, MP43L/LD, MP43M/MD, MP43N/ND, and MP43P/PD. In addition, MPI routines are called.

Workspace: Workspace is allocated by the code as required. The amount of workspace needed is dependent upon how the matrix data are stored, on `data%LENBUF`, and on the order of **A**.

Input/output: The output units for the error and warning messages are `data%ICNTL(1)` and `data%ICNTL(2)` (see Section 2.3). The output unit for diagnostic printing is `data%ICNTL(3)`.

Restrictions:
`data%NBLOCK` > `1`,
`data%NEQ` ≥ `data%NBLOCK`,
`data%NRHS` ≥ `0` (`data%JOB` = `3` or `23`),
`data%NRHS` ≥ `1` (`data%JOB` = `4`),
`data%NEQSB(:)` > `0`,
`0` ≤ `data%INV_LIST(:)` < `data%NPROC` (`data%ICNTL(10)` nonzero),
`data%LENBUF(:,:)` ≥ `1` (`data%ICNTL(13)` nonzero).

Portability: Fortran 95 + TR 15581 (allocatable components) with MPI for message passing.

Changes between Version 1.0.0 and Version 2.0.0:
The addition of `HSL_MP01` to HSL has allowed the source form to be changed to free format and means that the user of no longer needs an `INCLUDE` line for the MPI constants. All the pointer array components have been changed to allocatable components, which should be more efficient and avoids any danger of memory leakage.

4 METHOD

`HSL_MP43` implements a multiple front algorithm. Details of the algorithm are given in the reports Duff and Scott (1994) and Scott (1999, 2001) (reports available from www.numerical.rl.ac.uk/reports/reports.html).

`data%JOB` = `1`

The control components are given default values.

`data%JOB` = `2`

The input data is first checked for errors. The control components and scalar input components are then broadcast from the host to all processes. The host process calls `MA52A/AD` to generate lists of interface variables. The submatrices are shared between the processes so that each process has (approximately) the same number of submatrices to reorder. The reordering is done by the processes using `MC62`. Unless `data%ICNTL(10)` has been reset to a nonzero value, the host uses data from `MC62` to reassign each submatrix to a process. This division of the submatrices between the processes aims to balance the floating-point operations.

`data%JOB` = `3`

Data for each submatrix is sent from the host to its assigned process `IPROC`. Process `IPROC` generates unit numbers for the direct-access files that will hold the matrix factors, calls the analyse and factorize phases of `MA42`, and uses `MA52K/KD` to preserve the partial factorization.

Data from `MA52K/KD` is sent to the host. The host uses `MA42` to solve the interface problem. If `data%NRHS` > `0`, the solution for the interface problem is sent to each process, and on each process `MP43F/FD` is called to perform the back substitution for its assigned submatrices.

`data%JOB` = `4`

The host first performs error checks and broadcasts the number of right-hand sides to each process. For each of its assigned submatrices, process `IPROC` calls `MP43G/GD` to perform forward substitution. The partial solution vectors are sent to the host. Forward and back substitution for the interface problem is performed by the host using `MP43D/DD`. The solution for the interface problem is sent to each process, and on each process `MP43F/FD` is called to perform the back substitution on its submatrices.

`data%JOB` = 5

Arrays allocated by the code are deallocated, the direct-access files used to hold the matrix factors are closed and (optionally) are deleted.

MPI is used throughout for message passing.

References

Duff, I. S. and Scott, J. A. (1994). The use of multiple fronts in Gaussian elimination. Rutherford Appleton Laboratory Report RAL-94-040.

Scott, J. A. (1999). The design of a parallel frontal solver. Rutherford Appleton Laboratory Report RAL-TR-99-075.

Scott, J. A. (2001). The design of a portable parallel frontal solver for chemical process engineering problems. Rutherford Appleton Laboratory Report RAL-TR-2001-011.

5 EXAMPLE OF USE

We wish to factorize the matrix **A** given by

$$\mathbf{A} = \begin{pmatrix} 1. & 2. & & & 1. & \\ & 1. & & & & -1. \\ 2. & & & & -2. & \\ & & 1. & 1. & & \\ & & & 3. & -1. & \\ & & 2. & -1. & 1. & 1. \end{pmatrix},$$

and solve $\mathbf{Ax} = \mathbf{b}$ for the right-hand side

$$\mathbf{b} = \begin{pmatrix} 6. \\ 1. \\ 0. \\ 2. \\ 2. \\ 3. \end{pmatrix}.$$

The following program may be used to solve this problem.

```
      PROGRAM SPEC43
! Program to illustrate use of MP43.

      USE HSL_MP43_DOUBLE
      IMPLICIT NONE

      TYPE (MP43_DATA) data
      INTEGER ERCODE,ST

      CALL MPI_INIT(ERCODE)
! Define a communicator for the package
      data%COMM = MPI_COMM_WORLD
! Initialize package
      data%JOB = 1
      CALL MP43AD(data)
! Reset control parameters (if required)
! Read all values on host
      data%ICNTL(7) = 3
! Suppress diagnostic printing
      data%ICNTL(4) = 0
! Allow variables to be eliminated one at a time.
      data%ICNTL(16) = 1
      IF (data%RANK.EQ.0) THEN
         OPEN (UNIT=50,FILE='mp43ads.data')
         READ (50,*) data%NEQ,data%NBLOCK,data%NE,data%NRHS
! Allocate arrays for matrix data
         ALLOCATE(data%NEQSB(1:data%NBLOCK),STAT=ST )
         ALLOCATE(data%EQPTR(1:data%NEQ+1),STAT=ST )
         ALLOCATE(data%EQVAR(1:data%NE),STAT=ST )
         ALLOCATE(data%VALUES(1:data%NE),STAT=ST )
         ALLOCATE(data%B(1:data%NEQ,1:data%NRHS),STAT=ST )

! Read data on host.
         READ (50,*) data%NEQSB(1:data%NBLOCK)
         READ (50,*) data%EQPTR(1:data%NEQ+1)
         READ (50,*) data%EQVAR(1:data%NE)
         READ (50,*) data%VALUES(1:data%NE)
         READ (50,*) data%B(1:data%NEQ,1:data%NRHS)
      END IF

! Call MP43 for analyse/factorize/solve
      data%JOB = 23
      CALL MP43AD(data)
      IF (data%ERROR.LT.0) THEN
         WRITE (6,*) ' Unexpected error return'
         GO TO 950
      END IF
      IF (data%RANK.EQ.0) WRITE (6,'(//A,6G10.4)')
     &  ' The solution is: ',data%X(1:data%NEQ,1)

  950 data%JOB = 5
      CALL MP43AD(data)
      CALL MPI_FINALIZE(ERCODE)
      STOP

      END PROGRAM SPEC43
```

The input data needed is:

```
   6   2  15   1
   3   3
   1   4   6    8  10  12   16
   1   2   5    6   2   1    5   3   4   4    5    4   3   5   6
 1.0 2.0 1.0 -1.0 1.0 2.0 -2.0 1.0 1.0 3.0 -1.0 -1.0 2.0 1.0 1.0
 6.0 1.0 0.0  2.0 2.0 3.0
```

Assuming the code is run on two processes, this produces the following output:

```

 The solution is:  1.000     2.000     1.000     1.000     1.000     1.000
```