HSL_MP48

1 SUMMARY

The module HSL_MP48 **solves sets of $n \times n$ unsymmetric linear systems of equations $\mathbf{Ax} = \mathbf{b}$, in parallel using Gaussian elimination. The matrix A must have been preordered to singly-bordered block-diagonal form**

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

MPI is used for message passing.

A partial LU decomposition is performed on each of the submatrices $(\mathbf{A}_{ll}\ \mathbf{C}_l)$ separately. Once all possible eliminations have been performed, for each submatrix there remains a Schur complement matrix $\mathbf{F}_l$. The variables that remain are called interface variables and the interface matrix $\mathbf{F}$ is formed by summing the matrices $\mathbf{F}_l$. Gaussian elimination is used to factorize $\mathbf{F}$, using the HSL sparse direct solver MA48. Block forward elimination and back substitution completes the solution.

The user's matrix data may optionally be held in unformatted sequential files. In addition, L and U factors for the submatrices may optionally be written to sequential files. This reduces main memory requirements when the number N of submatrices is greater than the number of processes used.

The HSL package HSL_MC66 (included with this package) may be used for preordering the matrix to singly-bordered block-diagonal form.

ATTRIBUTES — **Version:** 2.1.1. (16th September 2024) **Types:** Real (single, double). **Calls:** HSL_MP01, KB08, MA48, MA52, MC46 and the BLAS routines I_AMAX, _AXPY, _SCAL, _SWAP, _GEMV, _TPSV, _GEMM, _TRSM. _TRSV. **Remark:** HSL_MC66 may be used for preordering. **Language:** Fortran 95 + TR 15581 (allocatable components). **Original date:** March 2003. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

The module HSL_MP48 has six separate phases:

(1) Initialize

(2) Preliminary analyse

(3) Analyse

(4) Factorize

(5) Solve

(6) Finalize

Prior to calling the initialize phase, the user must choose the number of processes and must initialize MPI by calling MPI_INIT on each process. The user must also define an MPI communicator for the package. The communicator defines the set of processes to be used by MP48. The processes have rank 0, 1, 2,... The **host** is the process with rank zero. The host performs the initial checking of the data, distributes data to the remaining processes, collects computed data from the processes, factorizes and solves the interface problem, and generally overseas the computation. With the other processes, the host (optionally) participates in generating the partial LU decompositions of the submatrices.

Each phase must then be called in order by each process in the communicator. The user may either do this by calling each phase from his or her calling program or can use the option of making a single call that initiates calls to each phase in turn. 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 MP48 communicator). The calling sequence is illustrated by the simple example given in Section 5.

The initialize phase gives default values to all control parameters. The preliminary analyse phase computes lists of border columns (interface variables) and assigns each submatrix to a processor. The analyse phase then prepares the data for factorization, choosing pivot sequences for each submatrix. During the factorize phase, the matrix factors are generated. The user may then solve for one or more right-hand sides by repeatedly calling the solve phase. The finalize phase deallocates all arrays that have been allocated by the package. 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.

Access to the module requires a `USE` statement and the user must declare a structure `data` of type `MP48_DATA` defined by the module. `HSL_MP48` has a single user-callable subroutine `MP48A/AD` with a single parameter `data` of type `MP48_DATA`. If the user wishes to run more than one instance of the module at the same time, a separate parameter of type `MP48_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 `MP48A/AD` must be used.

Single precision version

```
        USE HSL_MP48_SINGLE
        ...
        INTEGER ERCODE
        TYPE (MP48_DATA) ::  data
        ...
        CALL MPI_INIT(ERCODE)
        ...
  ! Set components of data
        ...
        CALL MP48A (data)
        ...
        CALL MPI_FINALIZE(ERCODE)
```

Double precision version

```
        USE HSL_MP48_DOUBLE
        ...
        INTEGER ERCODE
        TYPE (MP48_DATA) ::  data
        ...
        CALL MPI_INIT(ERCODE)
        ...
  ! Set components of data
        ...
        CALL MP48AD (data)
        ...
        CALL MPI_FINALIZE(ERCODE)
```

In `HSL_MP48_SINGLE`, with the exception of wall clock timings, all reals are default reals. Wall clock timings (see

Section) are measured using MPI_WTIME and are double precision reals. In HSL_MP48_DOUBLE, all reals are double precision reals. In both modules, all integers are default integers and all characters are default characters.

2.2 Input and output components

Prior to the first call to MP48A/AD for the current instance, the user must initialize the following component of data:

data%COMM is an INTEGER scalar that must hold an MPI communicator. data%COMM must be initialized prior to the first call to MP48A/AD to define the set of processes to be used by MP48. Before each phase is called, the user must have completed all 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 MP48A/AD, a job parameter is needed. This parameter determines the phase of the package to be performed.

data%JOB is an INTEGER scalar that must be initialized on **all** processes within the communicator before **each** call to MP48A/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 checks the user's integer data (see Section 2.2.1), generates lists of border columns, renumbers the variables in each submatrix and, optionally, allocates submatrices to processes (the user may choose how to allocate the submatrices; see data%ICNTL(10) in Section 2.3).

JOB = 3. For each submatrix ($\mathbf{A}_{ll}$ $\mathbf{C}_l$), permutations $\mathbf{P}_l$ and $\mathbf{Q}_l$ suitable for the partial *LU* decomposition are computed. $\mathbf{P}_l$ and $\mathbf{Q}_l$ are chosen to preserve sparsity and control numerical stability. Packed storage is used until a density threshold is reached, from which point full storage is used (see data%CNTL(1) in section 2.3). There is an option for dropping small entries from the factorization (see data%CNTL(3) and data%CNTL(4) in Section 2.3). A call with JOB = 3 must be preceded by a call with JOB = 2.

JOB = 4 uses the information from the call with JOB = 3 to generate the partial *L* and *U* factors for the submatrices and then uses MA48 to form the *L* and *U* factors for the interface problem. The matrices $\mathbf{P}_l$ are altered if necessary for numerical stability, but the matrices $\mathbf{Q}_l$ are not altered. An option exists for subsequent calls for matrices with the same sparsity pattern (see Section 2.2.4, parameter data%FACT_JOB). A call with JOB = 4 must be preceded by a call with JOB = 3.

JOB = 5 uses the factors produced by a call with JOB = 4 to solve systems of the form $\mathbf{Ax}=\mathbf{b}$. A call with JOB = 5 must be preceded by a call with JOB = 4 but several calls with JOB = 5 may follow a single call with JOB=4.

JOB = 6 deallocates all arrays that have been allocated by the module and, optionally, deletes all sequential files that have been used to hold the matrix factors. A call with JOB = 6 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.

In addition, the following values of data%JOB may be used to combine one or more of the above calls.

JOB = 23 performs JOB = 2 followed by JOB = 3.

JOB = 24 performs JOB = 2 followed by JOB = 3 and JOB = 4 (combines analyse and factorize calls).

JOB = 25 performs JOB = 2, followed by JOB = 3, JOB = 4, and JOB = 5 (combines analyse, factorize, and solve calls).

2.2.1 Input components for `data%JOB` = 2 (and `data%JOB` = 23, 24, 25)

Prior to a call with `data%JOB` = 2 (or `data%JOB` = 23, 24, 25), the following components **must** be set by the user on the host:

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

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

`data%NEQSB` is an `INTEGER` rank-1 allocatable array that must 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 an `INTEGER` rank-1 allocatable array 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 the row 2, and so on. Within a row, the variable indices may be in any order. If variable indices less than 1 or greater than `data%NEQ` are found, the computation terminates with an error. Duplicate indices within a row are permitted if `data%ICNTL(9)` = 0 (their values will be summed during the `data%JOB` = 3 call). This component is altered.

`data%EQPTR` is an `INTEGER` rank-1 allocatable array that must be allocated wit 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. There must be no null rows. This component is not altered on the host.

Additionally, the following component must be allocated and set by the user if the control component `data%ICNTL(10)` is not equal to 0 or 1.

`data%INV_LIST` is an `INTEGER` rank-1 allocatable array that must be allocate 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).

2.2.2 Output components for `data%JOB` = 2 (and `data%JOB` = 23, 24, 25)

The following components are allocated on each process.

`data%INV_LIST` is an `INTEGER` rank-1 allocatable array 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 (and `data%JOB` = 23, 24, 25)

`data%NBLOCK`, `data%NEQ`, `data%NEQSB`, `data%EQVAR`, `data%EQPTR`, and `data%INV_LIST` must be passed unchanged since the call with `data%JOB` = 2 and are not altered by a call with `data%JOB` = 3 unless duplicate entries are found. If duplicate entries are found and `data%ICNTL(9)` = 0, they are summed and `data%EQVAR` and `data%EQPTR` are altered. The remaining components of `data` that must be set (on the host only) depend on the value of the control component `data%ICNTL(7)`.

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

`data%VALNAM` is a `CHARACTER*128` rank-1 allocatable array that must be allocated with size at least `data%NBLOCK`. On entry, `data%VALNAM(L)` must contain the name of the file holding the entries of the rows of **A** belonging to submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`). These names must all be different. `data%VALNAM` is not accessed if `data%ICNTL(7)` = `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` that must be allocated and must contain the entries of 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` 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.

2.2.4 Input components for `data%JOB` = `4` (and `data%JOB` = `24` and `25`)

`data%NBLOCK`, `data%NEQ`, `data%NEQSB`, `data%EQVAR`, `data%EQPTR`, and `data%INV_LIST` must be passed unchanged since the call with `data%JOB` = `3` and are not altered. In addition, the following component must be set on the host:

`data%FACT_JOB` is an `INTEGER` scalar that must be set by the user to one of the following values:

1 The values of the entries of the matrix **A** must be unchanged since the call to `MP48A/AD` with `data%JOB` = `3` and no other calls to `MP48A/AD` may have been made since the `data%JOB` = `3` call.

2 The values of the entries of the matrix **A** may have changed since the call to `MP48A/AD` with `data%JOB` = `3`. This may be the first call `MP48A/AD` since the `data%JOB` = `3` call, or it may follow other calls to `MP48A/AD` with `data%JOB` = `4`. Numerical pivoting is performed. This may be the first call since the call to `MP48A/AD` with `data%JOB` = `3`, or it may follow other calls with `data%JOB` = `4` and `data%FACT_JOB` = `1` or `2`.

3 Fast call for the case where the values of the entries of the matrix **A** may have changed since a previous call to `MP48A/AD` with `data%JOB` = `4` and `data%FACT_JOB` = `1` or `2`. No numerical pivoting is performed, which may be numerically unstable if the matrix entries are markedly different from those of the earlier call. This call is not available when any entries have been dropped (see `data%CNTL(3)` and `data%CNTL(4)` in Section 2.3).

This component is not altered. If `data%JOB` = `24` or `25`, `data%FACT_JOB` is not accessed (a value equal to `1` is used). **Restriction:** `data%FACT_JOB` = `1`, `2`, or `3`.

The following component must be allocated and set by the user on the host if `data%ICNTL(11)` < `0`.

`data%FILES` is a `CHARACTER*128` rank-2 allocatable array that must be allocated with size `2` by `data%NBLOCK+1`. On entry, `data%FILES(J, L)`, `J` = `1`, `2`, must hold the names of the sequential files for the real and integer factor data for submatrix `L` (`L` = `1`, `2`,..., `data%NBLOCK`) and for the interface problem when `L` = `data%NBLOCK+1`. These names must all be different.

2.2.5 Input components for `data%JOB` = `5` (and `data%JOB` = `25`)

Prior to a call with `data%JOB` = `5` (and `data%JOB` = `25`) the following components must be set by the user on the host:

`data%B` is a `REAL` rank-1 allocatable array that must be allocated with size at least `data%NEQ`. On entry, `data%B(I)` must be set by the user to the I-th component of the right-hand side of the equations being solved (I = 1, 2,..., data%NEQ). This component is not altered.

The following component must be allocated by the user on the host if data%ICNTL(13) ≠ 0.

data%X is a REAL rank-1 allocatable array that must be allocated with size at least data%NEQ. data%X need not be set on entry. This component is altered.

2.2.6 Output components for data%JOB = 5 (and data%JOB = 25)

On a call with data%JOB = 5 (and data%JOB = 25) the following component is used to hold the solution vector, on the host only.

data%X is a REAL rank-1 allocatable array of size data%NEQ. On exit, on the host data%X(I) holds the I-th component of the solution (I = 1, 2,..., data%NEQ).

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 (on each call with data%JOB > 1, the host broadcasts the control parameters to the remaining processes).

data%ICNTL is a rank-1 INTEGER array of size 20.

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 used to control the full-matrix processing. It has default value 32. Possible values are:

≤0 Level 1 BLAS used.

1 Level 2 BLAS used.

≥2 Level 3 BLAS used by MP48A/AD with data%JOB = 4 (factorize phase), with block column size ICNTL(5). Level 2 BLAS used by MP48A/AD with data%JOB = 5 (solve phase).

ICNTL(6) is used when a real or an integer array that holds data for the factors is too small. The array is reallocated with its size increased by at least the factor ICNTL(6). The default value is 2.

ICNTL(7) is used to control how the user wishes to supply the matrix **A** on a call with data%JOB = 3. The default value is 1. The options are:

1 The submatrices ($\mathbf{A}_{ll}$ $\mathbf{C}_l$) are held in unformatted sequential files. The data required by the process with rank IPROC must be readable by that process (IPROC = 0, 1, ..., data%NPROC-1). 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 sequential file holding the values of the entries in the rows of the submatrices that can be read by process `IPROC`. Each process only requires storage for one submatrix at a time. The entries of the submatrices must be written to the sequential files in the same order as they are held in `data%EQVAR` on the `data%JOB = 2` call. The name of the file for submatrix `L` must be given in `data%VALNAM(L)` (`L = 1, 2, ..., data%NBLOCK`). The values of the entries are read during a call with `data%JOB = 3` and reread during the call with `data%JOB = 4`.

`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. The values of the entries are read during a call with `data%JOB = 3` and stored for the subsequent factorization.

`3` The user must supply the submatrix data in memory on the host using `data%VALUES`. This option avoids reading from sequential files although it does involve more data movement between processes. The values of the entries are read during a call with `data%JOB = 3` and stored for the subsequent factorization.

`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 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 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) = 1, 2, 3, 4,` or `5`.

`ICNTL(8)` has default value `3`. If `ICNTL(8)` has a positive value, each pivot search in `MP48A/AD` is limited to a maximum of `ICNTL(8)` columns. If `ICNTL(8)` is set to `0`, a full Markowitz search is used to find the best pivot. This requires extra workspace and is usually only a little slower, but can occasionally be very slow. It may result in reduced fill-in.

`ICNTL(9)` controls the action if duplicate entries within a row are found. If `ICNTL(9)=0` (the default) and duplicates are found, the values of duplicates are summed. Otherwise, if duplicates are found, the program terminates with the error flag `-3`.

`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 preliminary analyse phase (`data%JOB = 2`) and the host is involved in the submatrix factorizations. If `ICNTL(10)=1`, the choice is again made automatically but (assuming the number of processes is at least 2) the host is not involved in the submatrix factorizations. This option can be useful in a distributed memory environment for large problems and/or problems with a relatively large or dense interface. For all other values of `ICNTL(10)`, the user must choose a process for each submatrix using `data%INV_LIST`.

`ICNTL(11)` controls whether or not sequential files are used to hold factor data that is generated during the computation. If `ICNTL(11)=0` (the default), files are **not** used and the data is held in main memory. Otherwise, sequential files are used (if one process is to factorize more than one submatrix then using files reduces storage requirements but the extra I/O involved can increase the overall computational time). If `ICNTL(11)>0`, the code automatically names the files and they are written to the current directory. The files for the real and integer factor data for submatrix 1 are called fact.0001, integ.0001, for submatrix 2 they are fact.0002, integ.0002, and so on. The files for the real and integer factor data for the interface problem are fact_interf, integ_interf. If `ICNTL(11)<0`, the user must supply names for the files in `data%FILES` (see Section 2.2.4). If the user wishes to run a second instance of the module before the final call for the first instance (`data%JOB = 6`) and wants to use files for the data, `data%ICNTL(11)` **must** be set to a negative value and file names provided by the user in `data%FILES`.

`ICNTL(12)` controls whether the user wants the sequential 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 `MP48A/AD` (`data%JOB` = `6`), the sequential files are deleted. Otherwise, the files are disconnected but not deleted. `ICNTL(12)` is not used if `ICNTL(11)` is equal to `0`.

`ICNTL(13)` controls whether the user allocates the solution vector `data%X`. If `ICNTL(13)` = `0` (the default), `data%X` is allocated by the code on a `dataJOB` = `5` (or `data%JOB` = `25`) call. Otherwise, `data%X` must be allocated by the user on the host prior to a `dataJOB` = `5` (or `data%JOB` = `25`)call.

`ICNTL(14)` to `ICNTL(20)` are not currently used but are set to zero by the call with `data%JOB` = `1`.

`data%CNTL` is a `REAL` rank-1 array of size `10`.

`CNTL(1)` has default value 0.5. During each submatrix factorization, `MP48A/AD` switches to full matrix processing if the ratio of number of entries in the reduced matrix to the number that it would have as a full matrix is `CNTL(1)` or more. A value greater than 1.0 is treated as 1.0 and a value less than 0.0 is treated as 0.0.

`CNTL(2)` has default value 0.01 and determines the balance in `MP48A/AD` between pivoting for sparsity and for stability, values near zero emphasising sparsity and values near one emphasizing stability. A value greater than 1.0 is treated as 1.0 and a value less than 0.0 is treated as 0.0.

`CNTL(3)` has default value zero. If it is set to a positive value, `MP48A/AD` will drop from the factors any entry whose modulus is less than `CNTL(3)`. The factorization will then require less storage but will be inaccurate. A value less than 0.0 is treated as 0.0.

`CNTL(4)` has default value zero. If `MP48A/AD` finds a column of the reduced submatrix with entries all of modulus less than or equal to `CNTL(4)`, all such entries are dropped from the factorization (and contribute to the count in `data%DROP`). Every pivot is also required to have absolute value greater than `CNTL(4)`. A value less than 0.0 is treated as 0.0.

`CNTL(5)` to `CNTL(10)` are not currently used but are set to zero by the call with `data%JOB` = `1`.

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 each call to `MP48A/AD`.

`data%ERROR` is an `INTEGER` scalar 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 an `INTEGER` scalar that is set to the number of processes used by `MP48`. `data%NPROC` is the number of processes associated with the communicator `data%COMM` and is set by a call within `MP48A/AD` to `MPI_COMM_SIZE`.

`data%RANK` is an `INTEGER` scalar 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` **(and** `data%JOB = 23, 24, 25`**).**

`data%BORDER` is an `INTEGER` scalar that holds the total number of columns in the border. If there are any null columns, they are included in the border.

`data%IOSTAT` is an `INTEGER` scalar that holds the Fortran `IOSTAT` parameter.

`data%STAT` is an `INTEGER` scalar that holds the Fortran `STAT` parameter.

`data%TIMEA` is a `REAL` scalar that holds the elapsed wall clock time (in seconds) for the host to generate the list of border columns. The time is measured using `MPI_WTIME`.

`data%NGUARD` is an `INTEGER` rank-1 allocatable array of size `data%NBLOCK`. `data%NGUARD(L)` holds the number of columns belonging to the border for submatrix `L` (`L = 1`, 2,..., `data%NBLOCK`).

Information available on exit from a call with `data%JOB = 3` **(and** `data%JOB = 23, 24, 25`**).**

`data%IDUP` is an `INTEGER` scalar that holds the number of duplicate entries that have ben found in **A**.

`data%IOSTAT` is an `INTEGER` scalar that holds Fortran `IOSTAT` parameter.

`data%OPSA` is a `REAL` rank-1 allocatable array of size `data%NBLOCK`. `data%OPSA(L)` holds the predicted number of floating-point operations needed to factorize submatrix `L` (`L = 1`, 2,..., `data%NBLOCK`).

`data%STAT` is an `INTEGER` scalar that holds Fortran `STAT` parameter.

`data%TIMEA` is a `REAL` rank-1 allocatable array of size `data%NBLOCK`. `data%TIMEA(L)` holds the elapsed wall clock time (in seconds) for the analyze phase for submatrix `L` (`L = 1`, 2,..., `data%NBLOCK`). The time is measured using `MPI_WTIME`.

Information available on exit from a call with `data%JOB = 4` **(and** `data%JOB = 24, 25`**).**

`data%NINTER` is an `INTEGER` scalar that holds the order of the interface matrix (this will be at least `data%BORDER` and may be larger if the problem is singular).

`data%INTER_COL` is an `INTEGER` rank-1 array of length `data%NINTER` that holds the indices of the columns belonging to the interface matrix.

`data%NE_INTER` is an `INTEGER` scalar that holds the number of entries in the interface matrix.

`data%EST_RANK` is an `INTEGER` scalar that holds the estimated rank of **A**.

`data%DROP` is an `INTEGER` scalar that holds the total number of entries dropped from the data structure.

`data%FLOPS` is a `REAL` scalar that holds the total number of floating-point operations performed in factorizing the matrix.

`data%IOSTAT` is an `INTEGER` scalar that holds Fortran `IOSTAT` parameter.

`data%NZ` is a `REAL` scalar that holds the total number of entries in the factors.

`data%OPS_MA48` is a `REAL` scalar that holds the total number of floating-point operations performed in factorizing the interface matrix.

`data%STAT` is an `INTEGER` scalar that holds Fortran `STAT` parameter.

`data%STORINT` is a `REAL` scalar that holds the total integer storage for the factors in `INTEGER` words.

`data%TIME_MA48A` is a `REAL` scalar that holds the elapsed wall clock time (in seconds) for the analyze phase for the interface problem. The time is measured using `MPI_WTIME`.

`data%TIME_MA48F` is a `REAL` scalar that holds the elapsed wall clock time (in seconds) for the factorize phase for the interface problem. The time is measured using `MPI_WTIME`.

`data%OPS` is a `REAL` rank-1 allocatable array of size `data%NBLOCK`. `data%OPS(L)` holds the number of floating-point operations performed in factorizing submatrix `L` (`L = 1`, 2,..., `data%NBLOCK`).

data%STORAGE is a REAL rank-2 allocatable array of size 2 by data%NBLOCK. data%STORAGE(1,L) and data%STORAGE(2,L) hold, respectively, the real and integer storage for the partial factorization of submatrix L (L = 1, 2,..., data%NBLOCK).

data%TIMEF is a REAL rank-1 allocatable array of size data%NBLOCK. data%TIMEF(L) holds the elapsed wall clock time (in seconds) for the factorize phase for submatrix L (L = 1, 2,..., data%NBLOCK). The time is measured using MPI_WTIME.

Information available on exit from a call with data%JOB = 5 **(and** data%JOB = 25**).**

data%IOSTAT is an INTEGER scalar that holds Fortran IOSTAT parameter.

data%STAT is an INTEGER scalar that holds Fortran STAT parameter.

data%TIMES is a REAL rank-1 allocatable array of size data%NBLOCK. data%TIMES(L) holds the elapsed wall clock time (in seconds) for the solve phase for submatrix L (L = 1, 2,..., data%NBLOCK). The time is measured using MPI_WTIME.

data%TIME_MA48F is a REAL scalar that holds the elapsed wall clock time (in seconds) for the solve phase for the interface problem. The time is measured using MPI_WTIME.

2.5 Error diagnostics

On successful completion, a call to MP48 will exit with 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 (and data%JOB = 23, 24, 25)

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

–2 Either $\texttt{data\%NBLOCK} \le 1$ or $\texttt{data\%NBLOCK} > \texttt{data\%NEQ}$.

–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 lie out of range (less than 1 or greater than data%NEQ).

–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 (null row).

–6 data%ICNTL(7) out of range (ie. $\texttt{data\%ICNTL(7)} \neq 1$, 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 $\texttt{data\%NEQSB(L)} < 1$ for one or more of the submatrices L ($1 \le \texttt{L} \le \texttt{data\%NBLOCK}$).

–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) not equal to 0 or 1).

–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.

–19 The call follows a call with data%JOB = 6.

2.5.3 Error and warning diagnostics for data%JOB = 3 (and data%JOB = 23, 24, 25)

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

–3 Duplicate entries have been found (`data%ICNTL(9)` ≠ `0`). The number of duplicate entries is `data%IDUP`.

–10 If `data%ICNTL(7) = 1` or `2`, either `data%VALNAM` is not allocated or is allocated with 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`.

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

–14 Error in Fortran `INQUIRE` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–17 Error in Fortran `OPEN` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–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 = 6`.

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

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

+1 Duplicate entries have been found (`data%ICNTL(9) = 0`). The values of such entries are summed. The number of duplicate entries is `data%IDUP`.

2.5.4 Error diagnostics for `data%JOB = 4` (and `data%JOB = 24, 25`)

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

–10 If `data%ICNTL(7) = 1` or `2`, either `data%VALNAM` is not allocated or is allocated with 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`. If the number of submatrices exceeds the number of processes and the user is not using sequential files (`data%ICNTL(11) = 0`), it may be possible to avoid this error by rerunning with `data%ICNTL(11)` ≠ `0`. Alternatively, the user may try running with `data%ICNTL(10) = 1`.

–12 `data%FACT_JOB` has an invalid value.

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

–14 Error in Fortran `INQUIRE` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–15 On a call with `data%FACT_JOB = 3`, either the matrix entries are unsuitable for the pivot sequence chosen on the preceding `data%FACT_JOB = 1` or `2` call or the order and/or the number of nonzero entries in the interface matrix are not the same as on the earlier call.

–17 Error in Fortran `OPEN` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–19 An error was returned on a previous call or the call was not preceded by a call with `data%JOB = 3`, or follows a call with `data%JOB = 6`.

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

–21 Interface matrix is structurally rank deficient.

–25 `data%FILES` is either not allocated or is allocated but with incorrect size (`ICNTL(11)` < `0`).

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

+2 The matrix is singular.

2.5.4 Error diagnostics for `data%JOB` = 5 (and `data%JOB` = 25)

A negative value for `data%ERROR` is associated with a fatal error. Error messages are output by the host 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.

–14 Error in Fortran `INQUIRE` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–17 Error in Fortran `OPEN` statement. The `IOSTAT` parameter is returned in `data%IOSTAT`.

–19 An error was returned on a previous call or the call was not preceded by a call with `data%JOB` = 4, or follows a call with `data%JOB` = 6.

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

–22 `data%B` is either not allocated or is allocated but with incorrect size.

–23 `data%X` is either not allocated or is allocated but with incorrect size (`data%ICNTL(13)` nonzero only).

2.5.5 Error diagnostics for `data%JOB` = 6

A negative value for `data%ERROR` is associated with a fatal error. Error messages are output by host 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.

Other routines called directly: The HSL routines `HSL_MP01`, `KB08`, `MA48`, `MA52`, `MC46`. The BLAS routines `ISAMAX/IDAMAX`, `SAXPY/DAXPY`, `SSCAL/DSCAL`, `SSWAP/DSWAP`, `SGEMV/DGEMV`, `STPSV/DTPSV`, `SGEMM/DGEMM`, `STRSM/DTRSM`. `STRSV/DTRSV`. In addition, MPI routines are called.

Workspace: Workspace is allocated by the code on each process as required. The amount of workspace needed is dependent upon how the matrix data is stored (see `data%ICNTL(7)` in Section 2.3), on whether or not the generated factors are written to sequential files, and on the assignment of submatrices to processes.

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%NEQSB(:)` > 0,
0 ≤ `data%INV_LIST(:)` < `data%NPROC` (`data%ICNTL(10)` ≠ 0 or 1),
`data%ICNTL(7)` = 1, 2, 3, 4, or 5,
`data%FACT_JOB` = 1, 2 or 3.

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

`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 border columns. The submatrices are shared between the processes. By default, each process is assigned an equal (or nearly equal) number of submatrices.

`data%JOB` = `3`

Data for each submatrix is sent from the host to its assigned process `IPROC`. Process `IPROC` performs the analyse phase for each of its assigned submatrices using an internal subroutine that is a modified version of the HSL routine `MA50A/AD`. The columns in the border are supplied last.

`data%JOB` = `4`

After checking the input data, process `IPROC` performs the factorize phase for each of its assigned submatrices using an internal subroutine that is a modified version of the HSL routine `MA50B/BD`. If `data%FACT_JOB = 1`, the values of the entries of the matrix must be unchanged since the call with `data%JOB = 3`. If `data%FACT_JOB = 2`, the sparsity pattern of the matrix must be unchanged but the values may be different; numerical pivoting is incorporated for stability. If `data%FACT_JOB = 3`, at least one other call with `data%JOB = 4` and `data%FACT_JOB = 1` or `2` must already have been made. In this case, the values of the entries of the matrix may have changed but the pivot sequence is not modified and so this option could be numerically unstable.

Once all possible eliminations have been done, the Schur complement matrices that remain for each submatrix are sent to the host. The host assembles the interface matrix as a sparse matrix and uses the HSL routines `MA48A/AD` and `MA48B/BD` to perform analyse and factorize for the interface problem.

`data%JOB` = `5`

For each of its assigned submatrices, process `IPROC` performs forward elimination using an internal subroutine that is a modified version of `MA50C/CD`. The partial solution vectors are sent to the host. Forward elimination and back substitution for the interface problem is performed by the host using `MA48C/CD`. The solution for the interface problem is sent to each process, and each process calls the modified version of `MA50C/CD` to perform the back substitution on its submatrices. The final solution is assembled on the host.

`data%JOB` = `6`

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

References

Duff, I. S. and Scott, J. A. (2002). A parallel direct solver for large sparse highly unsymmetric linear systems. Rutherford Appleton Laboratory Report RAL-TR-2002-033 (available from www.numerical.rl.ac.uk/reports/reports.html). Also published in ACM Trans. Mathematical Software, **30** (2004), 95-117.

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 SPEC48_DOUBLE
! Program to illustrate use of MP48.

      USE HSL_MP48_DOUBLE
      IMPLICIT NONE

      TYPE (MP48_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 MP48AD(data)
! Reset control parameters (if required)
! Read all values on host
      data%ICNTL(7) = 3
      IF (data%RANK.EQ.0) THEN
         OPEN (UNIT=50,FILE='hsl_mp48ds.data')
         READ (50,*) data%NEQ,data%NBLOCK,data%NE
! 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),STAT=ST )

! Read matrix 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)
! Also read right hand side
         READ (50,*) data%B(1:data%NEQ)
      END IF
      CALL MPI_BARRIER(data%COMM,ERCODE)

! Call MP48A/AD (combine analyse/factorize/solve)
      data%JOB = 25
      CALL MP48AD(data)
      IF (data%RANK.EQ.0) THEN
        IF (data%ERROR.LT.0) THEN
          WRITE (6,*) ' Unexpected error return'
        ELSE
           WRITE (6,'(//A/,6ES11.3)') &
         ' The solution is: ',data%X(1:data%NEQ)
        END IF
      END IF

      data%JOB = 6
      CALL MP48AD(data)
      CALL MPI_FINALIZE(ERCODE)
      STOP

      END PROGRAM SPEC48_DOUBLE
```

The input data needed is:

```
   6   2  15
   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:
  1.000E+00  2.000E+00  1.000E+00  1.000E+00  1.000E+00  1.000E+00
```