HSL_MC66

1 SUMMARY

To **order an unsymmetric matrix A into singly bordered blocked diagonal (SBBD) form**. Given the sparsity pattern of a matrix, this routine generates a row and column ordering that can be used to reorder the matrix into the following SBBD form

$$\begin{pmatrix} A_{11} & & & & S_1 \\ & A_{22} & & & S_2 \\ & & \ddots & & \vdots \\ & & & A_{KK} & S_K \end{pmatrix}.$$

Here K is the user-defined number of blocks. The aim is to minimize the size of the border in the above matrix, also known as the *net-cut*, and to achieve *load balance* by ensuring that the rectangular matrices A_{ii} are of similar sizes.

The MC66 algorithm uses a multilevel approach combined with a Kernighan-Lin type refinement algorithm. Full details are discussed in Hu, Maguire and Blake, *Computers and Chemical Engrg.* **21** (2000), pp.1631-1647.

MC66 may be used to preorder an unsymmetric matrix for use with the sparse matrix solver HSL_MP43.

ATTRIBUTES — Version: 2.2.1 (20 September 2022). **Types:** Real (single, double). **Uses:** HSL_FA14 (Fortran 95 version), HSL_MC65, HSL_ZD11. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** January 2001. **Origin:** Y.F. Hu, Daresbury Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement to use the modules of HSL_MC66.

Single precision version

```
USE HSL_MC66_SINGLE
```

Double precision version

```
USE HSL_MC66_DOUBLE
```

If it is required to use both modules at the same time, the derived type MC66_CONTROL must be renamed in one of the USE statements.

2.2 The derived data types

2.2.1 The derived data type for the control of the subroutine

The derived data type MC66_CONTROL is used to control the subroutine. Components of this derived type are initialised to their default values as part of the type declaration, and the user does not need to set them unless values other than the default are required. The following components are employed:

LP is an INTEGER scalar used as the output stream for error messages. If it is negative, these messages will be suppressed. The default value is 6.

WP is an INTEGER scalar used as the output stream for warning messages. If it is negative, these messages are suppressed. The default value is 6.

MP is an INTEGER scalar used as the output stream for diagnostic messages. If it is negative, these messages are suppressed. The default value is 6.

PRINT_LEVEL is an INTEGER scalar indicating the level of diagnostic printing desired. If PRINT_LEVEL < 0, no diagnostic messages will be printed, if PRINT_LEVEL = 0, basic diagnostic messages will be printed, and if PRINT_LEVEL = 1, more detailed diagnostic messages will be printed. The default value is -1. Values greater than 1 are replaced by 1.

The remaining control parameters are only likely to be of interest to the more experienced user.

MAX_IMBALANCE is a REAL (double precision REAL for HSL_MC66_DOUBLE) scalar. MC66 employs a recursive bisection procedure and MAX_IMBALANCE indicates the targeted maximum imbalance for each bisection. Given a bisection that partitions an $m \times n$ matrix into a SBBD form with two blocks having row dimensions m_1 and m_2, the imbalance of the bisection is defined to be $|m_1 - m/2|/(m/2)$ (or equivalently $|m_2 - m/2|/(m/2)$, thus for example, MAX_IMBALANCE = 0.01 indicates a target imbalance of 1% or under during each bisection. Values less than 0 are replaced by 0; values greater than 1 are replaced by 1. The default value is 0.01. There is a trade off between MAX_IMBALANCE and the net-cut of the final ordering. A larger MAX_IMBALANCE tends to give a smaller net-cut but a poorer load balance.

MGLEVEL is an INTEGER scalar that holds the maximum number of levels in the multilevel hierarchy. The default value is HUGE(0). Smaller values may require less CPU time but may produce orderings of a poorer quality. Values less than 1 are replaced by 1.

COARSEN_SCHEME is an INTEGER scalar that determines the coarsening scheme used in the multilevel algorithm. If COARSEN_SCHEME = 1, a heavy edge collapsing scheme is used. If COARSEN_SCHEME = 2, a more aggressive coarsening scheme is used, which is less CPU intensive and less memory demanding, but tends to give orderings of slightly poorer quality. The default value is 1. A value less than 1 is replaced by 1 and a value greater than 2 is replaced by 2.

COARSEST_SIZE is an INTEGER scalar that holds the problem size in the multilevel hierarchy after which there is no further coarsening. The default value is 100. Values less than 2 are replaced by 2. The quality of the final ordering is not very sensitive to this parameter.

NUM_COARSEST_KL is an INTEGER scalar that holds the number of runs of the Kernighan-Lin refinement algorithm on the coarsest level. The best of these NUM_COARSEST_KL runs gives the ordering on the coarsest level. The default value is 4. Values less than 1 are replaced by 1.

GRID_RDC_FAC is a REAL (double precision REAL for HSL_MC66_DOUBLE) scalar that determines the minimum reduction in problem size that must be achieved between one level and the next during the multilevel coarsening. The default value is 0.75, which means that the problem size in the next level must be no more than 75% of that in the current level. We recommend GRID_RDC_FAC lies between 0.5 and 1.

KL_AGGRESSIVE is a REAL (double precision REAL for HSL_MC66_DOUBLE) scalar that indicates whether aggressive Kernighan-Lin refinement is used (see Section 5 for further details). Aggressive Kernighan-Lin refinement is used if KL_AGGRESSIVE > 0; otherwise, moderate Kernighan-Lin refinement is used. Aggressive Kernighan-Lin refinement can be very CPU intensive, but often gives orderings of smaller net-cut. The default value is -1.

3 THE ARGUMENT LISTS

We use square brackets `[ ]` to indicate `OPTIONAL` arguments.

3.1 To generate the ordering for a matrix

```
CALL MC66(M,N,NZ,IRN,JCN,NBLOCKS,CONTROL,SEED,ROW_ORDER,&
   INFO[,ROWPTR,COLUMN_ORDER,COLPTR,NETCUT,ROWDIFF,KBLOCKS])
```

`M` is an `INTEGER` scalar of `INTENT (IN)`. On entry, it must be set by the user to hold the number of rows of A. **Restriction:** `M` $\geq$ `1`.

`N` is an `INTEGER` scalar of `INTENT (IN)`. On entry, it must be set by the user to hold the number of columns of A. **Restriction:** `N` $\geq$ `1`.

`NZ` is an `INTEGER` scalar of `INTENT (IN)`. On entry it must be set by the user to hold the number of entries in A. **Restriction:** `NZ` $\geq$ `0`.

`IRN` is an `INTEGER` array of rank one of size `NZ` with `INTENT (IN)`. On entry, it must be set by the user to hold the row indices of the matrix A.

`JCN` is an `INTEGER` array of rank one of size `NZ` with `INTENT (IN)`. On entry, it must be set by the user to hold the column indices of the matrix A.

`NBLOCKS` is an `INTEGER` scalar of `INTENT (IN)`. On entry, it must be set by the user to hold the required number of blocks in the SBBD form that the sparse matrix is to be reordered into. **Restriction:** 1 $\leq$ `NBLOCKS` $\leq$ `min(M,N)`.

`CONTROL` is a scalar of type `MC66_CONTROL` with `INTENT (INOUT)`. Its components control the ordering algorithm, as explained in Section 2.2.1. The user does not need to set its components unless values other than the default are required; if the user has reset any to a value ouside its permitted range, the value will be changed (see Section 2.2.1).

`SEED` is a scalar of type `FA14_SEED` with `INTENT (INOUT)`. It is used to hold the seed for the random numbers generated by `HSL_FA14` for the calculation. It should not be altered by the user.

`ROW_ORDER` is an `INTEGER` array of rank one of size `M` with `INTENT (OUT)`. On exit, `ROW_ORDER(I)` is the original row index of the `I`-th row of the reordered matrix in SBBD form.

`INFO` is an `INTEGER` scalar of `INTENT (OUT)` that is used as an error/warning flag. Negative values indicate an error and positive values a warning. On exit, possible values are:

- `INFO = 0` if the subroutine completed successfully.
- `INFO = -1` if memory allocation failed.
- `INFO = -2` if memory deallocation failed.
- `INFO = -3` if `NBLOCKS` ≤ 0 or `NBLOCKS` $>$ min(`M,N`).
- `INFO = -4` if `M` ≤ 0.
- `INFO = -5` if `N` ≤ 0.
- `INFO = -6` if `NZ` ≤ 0.
- `INFO = -7` if the number of vertices in A^TA exceeds the number addressable by default integer.

- INFO = 1 if the number KBLOCKS of blocks in the SBBD form is less than the requested number of blocks NBLOCKS.
- INFO = 2 if one or more entries in IRN lies outside the range [1,M]. These entries are ignored.
- INFO = 3 if one or more entries in JCN lies outside the range [1,N]. These entries are ignored.
- INFO = 4 if duplicated entries were found.
- INFO = 5 if one or more empty columns were found. Such empty columns are placed at the last diagonal block of the reordered matrix in SBBD form.

Note that INFO = 1 overwrites other positive values of INFO.

ROWPTR is an OPTIONAL INTEGER array of rank one of size NBLOCKS+1 with INTENT (OUT). On exit, ROWPTR(I) is the starting row index for the I-th diagonal block in the reordered SBBD matrix (I = 1, ..., NBLOCKS). ROWPTR(NBLOCKS+1) = M.

COLUMN_ORDER is an OPTIONAL INTEGER array of rank one of size N with INTENT (OUT). On exit, COLUMN_ORDER(I) is the original column index of the I-th column of the reordered matrix in SBBD form.

COLPTR is an OPTIONAL INTEGER array of rank one of size NBLOCKS+1 with INTENT (OUT). On exit, COLPTR(I) gives the starting column index for the I-th diagonal block in the reordered SBBD matrix (I = 1, ..., NBLOCKS), and COLPTR(NBLOCKS+1) holds the starting column index of the border of the reordered matrix in SBBD form.

NETCUT is an OPTIONAL INTEGER scalar of INTENT (OUT). On exit, it holds the net-cut (the number of columns in the border of the SBBD form).

ROWDIFF is an OPTIONAL REAL (double precision REAL for HSL_MC66_DOUBLE) scalar of INTENT (OUT). On exit, it holds the difference between the maximum block row dimension and the average block row dimension (= M/NBLOCKS), divided by the average block row dimension, expressed as a percentage. Thus ROWDIFF = 10 denotes a 10% imbalance.

KBLOCKS is an OPTIONAL INTEGER scalar of INTENT (OUT). On exit, it holds the number *K* of blocks in the SBBD form. In general, KBLOCKS = NBLOCKS but KBLOCKS may be smaller than NBLOCKS (see INFO = 1).

3.2 Printing error/warning messages

Subroutine MC66_PRINT_MESSAGE can be used after return from MC66 to print an error or warning message related to a particular error flag INFO.

```
CALL MC66_PRINT_MESSAGE(INFO[,LP,WP,CONTEXT])
```

INFO is an INTEGER scalar of INTENT (IN). On entry, it must be set by the user to hold the error flag generated when calling subroutine MC66.

LP is an OPTIONAL INTEGER scalar of INTENT (IN). If present, on entry, it must be set by the user to hold the unit number for the error message. If this number is negative, printing is suppressed. If LP is not present, the message will be printed on unit 6.

WP is an OPTIONAL INTEGER scalar of INTENT (IN). If present, on entry, it must be set by the user to hold the unit number for the warning message. If this number is negative, printing is suppressed. If WP is not present, the message will be printed on unit 6.

CONTEXT is an OPTIONAL CHARACTER (LEN=*) scalar with INTENT (IN). If present, on entry, it must be set by the user to provide the context under which the error/warning occurs. It is printed ahead of the error or warning message.

4 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: `HSL_FA14` (Fortran 95 version), `HSL_MC65`, `HSL_ZD11`.

Input/output: Error, warning and diagnostic messages only. Error messages on unit `CONTROL%LP` and warning and diagnostic messages on units `CONTROL%WP` and `CONTROL%MP`, respectively. These have default value `6`, and printing of these messages is suppressed if the relevant unit number is set negative.

Restrictions: $1 \le$ `NBLOCKS` $\le$ `min(M,N)`, `N` $\ge$ `1`, `M` $\ge$ `1` and `NZ` $\ge$ `0`.

Changes from Version 1.0.0: `HSL_ZD11` used instead of `HSL_ZD01`.

5 METHOD

The `HSL_MC66` subroutine implements a multilevel recursive bisection algorithm, with the aim of minimizing the *net-cut* of the final SBBD matrix for a given number of blocks, while maintaining *load balance*. The multilevel approach generates, from the matrix to be ordered, a series of problems of smaller and smaller sizes. The smallest problem is then bisected into SBBD form with two blocks using several runs of the Kernighan-Lin algorithm, starting with a random initial ordering. The best ordering obtained from these runs are prolongated to larger problems and further refined using the Kernighan-Lin algorithm until a bisection of the original matrix is derived. This multilevel process is recursively applied until the desired number of blocks is achieved.

The Kernighan-Lin algorithm starts with an initial row bisection of a matrix, and attempts to swap rows to reduce the net-cut (the number of columns that have row entries in both parts of the bisection). Swaps that increase the net-cut (such a swap is called an uphill move) may be allowed in the hope that they may result in large reductions in the net-cut in future swaps. In the aggressive version of the algorithm (`KL_AGGRESSIVE > 0`), there is no limit to the number of uphill moves. In the moderate version of the algorithm (`KL_AGGRESSIVE` $\le$ `0`), no more than `100` uphill moves are allowed.

A fuller account of the complete algorithm is given in Hu, Maguire and Blake, *Computers and Chemical Engrg.* **21** (2000), pp.1631-1647.

6 EXAMPLE OF USE

We illustrate the use of `MC66` by reordering the following 4×4 matrix.

$$\begin{pmatrix} X & X & 0 & 0 \\ X & 0 & X & X \\ X & X & 0 & 0 \\ X & 0 & X & X \end{pmatrix}$$

For reference, after reordering, the matrix becomes

$$\begin{pmatrix} X & 0 & 0 & X \\ X & 0 & 0 & X \\ 0 & X & X & X \\ 0 & X & X & X \end{pmatrix}$$

Program

```
program example
  use HSL_mc66_double

  INTEGER, PARAMETER :: myreal = KIND( 1.0D+0 )
  INTEGER, PARAMETER :: myint = KIND( 1)

  ! mc66 controller:
  type (mc66_control) :: control

  ! random number seed
  type (fa14_seed) :: seed

  ! nz: number of nonzeros
  ! n: number of rows.
  ! m: number of columns
  ! irn: row indices of the matrix
  ! jcn: column indices of the matrix
  integer (kind=myint) :: nz
  integer (kind=myint) :: n,m
  integer (kind=myint), allocatable :: irn(:)
  integer (kind=myint), allocatable :: jcn(:)

  ! rowptr: rowptr[1:nblocks] is the starting row index for
  !         each diagonal block. rowptr[nblocks+1]=m
  ! colptr: colptr[1:nblocks] is the starting column index
  !         for each diagonal block. colptr[nblocks+1]
  !         is the starting column index of the border.
  integer (kind = myint), pointer, dimension (:) :: rowptr
  integer (kind = myint), pointer, dimension (:) :: colptr

  ! column order: column_order(i) is the original column index
  !   of the i-column of the reordered matrix
  ! row ordering: row_order(i) is the original row index
  !   of the i-row of the reordered matrix
  ! row_position: row_position(i) is the new row index
  !   of the original row i.
  ! column_position: column_position(i) is the new column index
  !   of the original column i.
  integer (kind = myint), pointer, dimension (:) :: row_order,&
       column_order,row_position,column_position

  ! nblocks: number of blocks
  ! netcut: column size of size of border
  ! info: info tag
  ! kblocks: the actual number of blocks in the SBBD form
  ! rowdiff: row dimension imbalance in percentage term
  integer (kind = myint) :: nblocks
  integer (kind = myint) :: netcut
  integer (kind = myint) :: info
  integer (kind = myint) :: kblocks
  real (kind = myreal) :: rowdiff

  integer (kind = myint) :: i

  nz = 10
  m = 4; n = 4; nblocks = 2
  allocate(irn(nz),jcn(nz))
  irn = (/4,4,4,2,2,2,1,1,3,3/)
  jcn = (/4,3,1,4,3,1,2,1,2,1/)

  write(*,"(a)") "the original matrix is "
  do i = 1, nz
     write(*,"(i4,i4)") irn(i),jcn(i)
  end do
  allocate(row_order(m),rowptr(nblocks+1), &
       column_order(n), colptr(nblocks+1))

  write(*,"(a)") " "
  write(*,"(a)") "generating the ordering ===== "
  call mc66(m,n,nz,irn,jcn,nblocks,control,seed, &
       row_order,info,rowptr,column_order,colptr,&
       netcut,rowdiff,kblocks)
  if (info /= 0) then
     call mc66_print_message(info)
     if (info < 0) stop "mc66 failed"
  end if

  write(*,"(a,I10)")       "netcut =                  ",netcut
  write(*,"(a,f10.2,'%')") "row dimension imbalance = ", rowdiff

  ! dump block sizes
  do i = 1, nblocks
     write(*,"('block ',i4,' of dimension  ',i10,' X ',i10)") &
          i,rowptr(i+1)-rowptr(i),colptr(i+1)-colptr(i)
  end do

  ! reorder the original matrix
  allocate(column_position(n), row_position(m))
  do i = 1, n
     column_position(column_order(i)) = i
  end do
  do i = 1, m
     row_position(row_order(i)) = i
  end do
  irn = row_position(irn)
  jcn = column_position(jcn)
  write(*,"(a)")  " "
  write(*,"(a)") "the reordered matrix is "
  do i = 1, nz
     write(*,"(i4,i4)") irn(i),jcn(i)
  end do

  deallocate(row_order,rowptr,column_order,colptr)
  deallocate(column_position,row_position)
  deallocate(irn,jcn)

end program example
```

This produces the following output

```
the original matrix is
   4   4
   4   3
   4   1
   2   4
   2   3
   2   1
   1   2
   1   1
   3   2
   3   1

generating the ordering =====
netcut =                           1
row dimension imbalance =       0.00%
block    1 of dimension           2 X          1
block    2 of dimension           2 X          2

the reordered matrix is
```

```
    4    3
    4    2
    4    4
    3    3
    3    2
    3    4
    1    1
    1    4
    2    1
    2    4
```