## 1 SUMMARY

For a matrix that is full, symmetric and positive definite, this module performs **partial factorizations and solutions of corresponding sets of equations, paying special attention to the efficient use of cache memory.** It uses a modification of the code of Andersen, Gunnels, Gustavson, Reid, and Wasniewski (ACM Trans. on Math. Software, **31**, 2005, 201-227). It is suitable for use in a frontal or multifrontal solver, but may also be used for the direct solution of a full set of equations. Optionally, it may be compiled to use OpenMP.

The modification involves limiting the eliminations to the leading $p$ columns. The factorization takes the form

$$A=\begin{pmatrix}A_{11} & A_{21}^T\\ A_{21} & A_{22}\end{pmatrix}=\begin{pmatrix}L_{11} & \\ L_{21} & I\end{pmatrix}\begin{pmatrix}I & \\ & S_{22}\end{pmatrix}\begin{pmatrix}L_{11}^T & L_{21}^T\\ & I\end{pmatrix}$$

where $L_{11}$ is lower triangular and both $A_{11}$ and $L_{11}$ have order $p$. On input, the lower triangular part of $A$ must be stored in lower packed format (that is, packed by columns). The leading part, $\begin{pmatrix}A_{11}\\ A_{21}\end{pmatrix}$, is rearranged to the lower blocked hybrid format of Andersen *et al.* (2005) while the trailing part is left in lower packed format. We will call this format the 'partial blocked hybrid' format. It is also used for the matrices $\begin{pmatrix}L_{11}\\ L_{21}\end{pmatrix}$ and $S_{22}$ of the factorization.

The module has facilities for rearranging a matrix in lower packed format to partial blocked hybrid format and vice-versa.

Subroutines are provided for partial forward and back substitution, that is, solving equations of the form

$$\begin{pmatrix}L_{11} & \\ L_{21} & I\end{pmatrix}X=B \quad \text{and} \quad \begin{pmatrix}L_{11}^T & L_{21}^T\\ & I\end{pmatrix}X=B$$

and the corresponding equations for a single right-hand side $b$ and solution $x$.

There are also subroutines for solving one or more sets of equations after a full factorization ($p=n$) and for extracting the diagonal of $L_{11}$.

**ATTRIBUTES — Version:** 1.4.2. (13 September 2022) **Types:** Real (single, double). **Remark:** The code has been tuned only for 8-byte arithmetic. **Language:** Fortran 95. **Parallelism:** May use OpenMP. **Calls:** `_COPY`, `_DOT`, `_GEMM`, `_GEMV`, `_SYRK`, `_TPSV`, `_TRSM`. **Original date:** July 2007. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

## 2 HOW TO USE THE PACKAGE

### 2.1 Introduction

Access to the package requires a `USE` statement whose simplest form is

Single precision version

```
USE HSL_MA54_single
```

Double precision version

```
USE HSL_MA54_double
```

In `HSL_MA54_single`, all reals are default reals. In `HSL_MA54_double`, all reals are double precision reals.

The following subroutines are available:

`MA54_to_block` rearranges a lower triangular matrix in lower packed format (that is, packed by columns) to partial blocked hybrid format.

`MA54_from_block` rearranges a lower triangular matrix that is in partial blocked hybrid format to lower packed

format.

MA54_factor partially factorizes a matrix in partial blocked hybrid format.

MA54_forward1 and MA54_back1 perform partial forward and back substitution for one set of equations, given the factorization of its matrix in partial blocked hybrid format.

MA54_forward2 and MA54_back2 perform partial forward and back substitution for one or more sets of equations, given the factorization of its matrix in partial blocked hybrid format.

MA54_solve1 solves one set of equations, given the factorization of its matrix in blocked hybrid format.

MA54_solve2 solves one or more sets of equations, given the factorization of its matrix in blocked hybrid format.

MA54_diag extracts the diagonal of $L_{11}$.

#### 2.1.2 OpenMP

OpenMP is used by the package to provide parallelism for shared memory environments. If OpenMP is available, it should be enabled at compilation time by using the correct compiler flag (usually some variant of `-openmp`). The default number of threads may be controlled at runtime by setting the environment variable `OMP_NUM_THREADS`.

### 2.2 Argument lists and calling sequences

#### 2.2.1 Rearrange to or from partial blocked hybrid format

```
call MA54_to_block(n,p,nb,ap,buf,info)
call MA54_from_block(n,p,nb,ap,buf,info)
```

n is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the matrix order. **Restriction:** $n \geq 0$.

p is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the order of $A_{11}$. **Restriction:** $n \geq p \geq 0$.

nb is a scalar of INTENT(IN) and type default INTEGER that specifies the block size for the blocked hybrid format. Section 2.2.8 contains a discussion of suitable values. **Restriction:** $nb \geq 1$.

ap is an array of shape (n*(n+1))/2, INTENT(INOUT), and type REAL. On entry to MA54_to_block, it holds the matrix in lower packed format and on return holds the matrix in partial blocked hybrid format. On entry to MA54_from_block, it holds the matrix in partial blocked hybrid format and on return holds the matrix in lower packed format.

buf is an array of shape (nb*n) and type REAL that is used as workspace.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has one of the values –1, –2, –3, –5; the meanings are given in Section 2.2.9.

#### 2.2.2 Partially factorize a matrix in partial blocked hybrid format

```
call MA54_factor(n,p,nb,ap,buf,info[,n_threads])
```

n,p,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since the call to MA54_to_block.

ap is an array of shape (n*(n+1))/2, INTENT(INOUT), and type REAL. On entry, it holds the matrix in partial blocked hybrid format. On return, it holds the factorized matrix in the same format.

buf is an array of shape (nb*n) and type REAL that is used as workspace.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it is positive or has one of the values –1, –2, –3, –5; the meanings are given in Section 2.2.9.

n_threads is an optional scalar of INTENT(INOUT) and type default INTEGER. If OpenMP is enabled and n_threads is present with a positive value, it determines the number of threads used in each parallel region inside MA54_factor. If OpenMP is enabled and n_threads is absent or present with a negative or zero value, the default number of threads is used. The actual argument may be an OpenMP shared variable that is altered by another thread during the execution of MA54_factor, which means that the number of threads used within MA54_factor can be made to vary during its execution. The argument n_threads is ignored if OpenMP is not enabled.

#### 2.2.3 One partial forward or back substitution

```
call MA54_forward1(n,p,nb,ap,b,info)
call MA54_back1(n,p,nb,ap,b,info)
```

n,p,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since the call to MA54_to_block.

ap is an array of shape (p*(n*2-p+1))/2, INTENT(IN), and type REAL. It holds the leading part of the factorized matrix in blocked hybrid format, as returned by MA54_factor.

b is an array of shape n, INTENT(INOUT) and type REAL. It holds the vector *b* on entry and is overwritten by the vector *x* on return.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has one of the values –1, –2, –3, –5; the meanings are given in Section 2.2.9.

#### 2.2.4 One or more partial forward or back substitutions

```
call MA54_forward2(n,p,nb,nrhs,ap,b,ldb,mb,buf,info)
call MA54_back2(n,p,nb,nrhs,ap,b,ldb,mb,buf,info)
```

n,p,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since the call to MA54_to_block.

nrhs is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the number of right-hand sides. **Restriction:** nrhs ≥ 0.

ap is an array of shape (p*(n*2-p+1))/2, INTENT(IN), and type REAL. It holds the leading part of the factorized matrix in blocked hybrid format, as returned by MA54_factor.

b is a rank-2 array of INTENT(INOUT) and type REAL. Its first extent is ldb and its second extent is at least nrhs. It holds the matrix *B* on entry and is overwritten by the matrix *X* on return.

ldb is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the first extent of the array b. **Restriction:** ldb ≥ n.

mb is a scalar of INTENT(IN) and type default INTEGER that specifies the block size to be used for the columns of the right-hand side matrix. Section 2.2.8 contains a discussion of suitable values. **Restriction:** mb ≥ 1.

buf is an array of shape (p*nb + n*mb) and type REAL that is used as workspace.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has one of the negative values listed in Section 2.2.9.

#### 2.2.5 Solve one set of equations

```
call MA54_solve1(n,nb,ap,b,info)
```

n,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since a call to MA54_to_block with p having the value n.

ap is an array of shape (n*(n+1))/2, INTENT(IN), and type REAL. It holds the factorized matrix in blocked

hybrid format, as returned by MA54_factor with p having the value n.

b is an array of shape n, INTENT(INOUT), and type REAL. It holds the right-hand side $b$ on entry and is overwritten by the solution $x$ on return.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has one of the values –1, –4, –5, –6, –7; the meanings are given in Section 2.2.9.

#### 2.2.6 Solve one or more sets of equations

```
call MA54_solve2(n,nb,nrhs,ap,b,ldb,mb,buf,info)
```

n,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since a call to MA54_to_block with p having the value n.

nrhs is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the number of right-hand sides. **Restriction:** nrhs ≥ 0.

ap is an array of shape (n*(n+1))/2, INTENT(IN), and type REAL. It holds the factorized matrix in blocked hybrid format, as returned by MA54_factor with p having the value n.

b is a rank-2 array of INTENT(INOUT) and type REAL. Its first extent is ldb and its second extent is at least nrhs. It holds the matrix $B$ on entry and is overwritten by the matrix $X$ on return.

ldb is a scalar of INTENT(IN) and type default INTEGER. It must be set by the user to the first extent of the array b. **Restriction:** ldb ≥ n.

mb is a scalar of INTENT(IN) and type default INTEGER that specifies the block size to be used for the columns of the right-hand side matrix. Section 2.2.8 contains a discussion of suitable values. **Restriction:** mb ≥ 1.

buf is an array of shape (p*nb + n*mb) and type REAL that is used as workspace.

info is a scalar of INTENT(OUT) and type default INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has one of the values –1, –4, –5, –6, –7; the meanings are given in Section 2.2.9.

#### 2.2.7 Extract the diagonal of $L_{11}$

```
call MA54_diag(n,p,nb,ap,diag)
```

n,p,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since a call to MA54_to_block.

ap is an array of shape (n*(n+1))/2, INTENT(IN), and type REAL. It holds the factorized matrix in partial blocked hybrid format, as returned by MA54_factor.

diag is a rank-1 array of size p, INTENT(OUT) and type REAL. On return, it holds the diagonal of $L_{11}$.

#### 2.2.8 Choice of the block size nb and mb

The choice of nb was discussed by Andersen *et al.* (2005). The value that allows a full matrix of size nb to fit in the level-1 cache is usually good, but a larger value may give better performance if n is large because of the influence of the level-2 cache. They experimented with the values 40, 72, 100, 200 and found that the best value rose with n but that the following single values were adequate: 40 for the Intel Pentium III; 100 for the IBM Power4 and SGI Origin 200; and 200 for the Alpha EV6, SUN Ultra III, and HP Itanium 2. If performance is critical, we recommend that different values of nb be tried.

For the choice of mb, Andersen *et al.* remark that it is probably best to use a value close to that of nb, but that a larger value may be beneficial provided there is room in level-2 cache for n*mb reals.

#### 2.2.9 Values of the argument info

An unsuccessful call to any of the procedures is flagged with a nonzero value of the argument info. The meanings are as follows:

>0 The leading minor of order info is not positive definite, and the factorization could not be completed.
–1 n < 0.
–2 p < 0.
–3 p > n.
–4 nrhs < 0.
–5 nb < 1.
–6 ldb < n.
–7 mb < 1.

## 3 GENERAL INFORMATION

**Other routines called directly:** The BLAS _COPY, _DOT, _GEMM, _GEMV, _SYRK, _TPSV, _TRSM.

**Input/output:** None.

**Restrictions:** n ≥ 0; n ≥ p ≥ 0; nrhs ≥ 0; nb ≥ 1; ldb ≥ n; mb ≥ 1.

## 4 METHOD

### 4.1 Rearrangement

The rearrangement of MA54_to_block requires each block column of the leading part to be stored by rows instead of columns. This is performed with the help of a work array (buf) of size n*nb. For each block column, the elements of the block are copied to the work array and then copied back into their new positions within ap. Both copies are done efficiently by using the level-1 BLAS _COPY. The first is done column by column, with *j* entries wasted after column *j* of the block. This leaves the rows evenly spaced so that the second copy can be done row by row. We found that this was significantly faster than the code of Andersen *et al.*, which copies the elements of the block column to the work array in their new positions within the block column (too complicated for_COPY) and then copies the whole block column back using a single call of _COPY. In both codes, the time taken by the rearrangement is proportional to the number of elements in the leading part.

The rearrangement of MA54_from_block is performed similarly.

### 4.2 Partial factorization

For the first *p* columns, the Cholesky factorization is performed block column by block column in a left-looking algorithm. The format allows the operations for each previous block column to be performed by one call of the level-3 BLAS _SYRK for the block on the diagonal and one call of the level-3 BLAS _GEMM for each off-diagonal block in the block column. The block on the diagonal is then factorized by a specially written kernel that uses mini-blocks of size 2 to reduce traffic between the memory and the registers. So that the factorized block can be used by the level-3 BLAS _TRSM, the kernel operates in full storage. This requires that the block is copied to a work array of size nb**2. Finally, the off-diagonal part of block column is updated by _TRSM.

A left-looking algorithm is also used for the final *n–p* columns. It is performed block column by block column. To enable the use of level-3 BLAS, a temporary copy in full-storage mode is placed in buf and copied back after modification.

### 4.3 OpenMP parallel regions

When executing on more than one thread, a parallel region is used to apply all the operations on a block column within the first *p* columns. The block operations are as described in Section 4.2 and are performed in the same order

except for

1. the parallelization of the loop that performs the _GEMM updates from a single block column together with the _SYRK update from the next block column (or Cholesky factorization if the next block column is the one being updated), and
2. the parallelization of the loop that performs the_TRSM calls.

To achieve this parallelization, each call of _SYRK and each Cholesky factorization is performed by a single processor and is followed by a barrier. No barrier is placed after the loop that performs the _GEMM updates. For the sake of efficiency, we use schedule(static) for the _GEMM loop. If the barrier had been placed after this loop, there would have always been a load imbalance since the master processor would have executed a share of the loop at least as large as any other processor as well as the _SYRK call.

Similar considerations apply for a block column within the final *n–p* columns, except that now there are no _SYRK updates and no Cholesky factorizations.

### 4.4 Partial forward and back substitution

During forward or back substitution for a single right-hand side, there is a call of the level-2 BLAS _GEMV and a call of the level-2 BLAS _TPSV for each block column. This code is also called by MA54_solve2, MA54_forward2, and MA54_back2 if the number of right-hand sides is less than 4.

If the right-hand side matrix has more than 3 columns, it is divided into blocks of mb columns which are handled in turn. Each is copied into a work array of size n*mb so that it is held contiguously, then forward or back substitution is performed using a call of the level-3 BLAS _GEMM and a call of the level-3 BLAS _TPSV for each block column of the factorized matrix. Finally, the block column of the result is copied back from the work array.

## 5 EXAMPLE OF USE

The following code reads a matrix in lower packed format, converts to partial blocked hybrid format, performs Cholesky factorization, and solves a set of equations.

```
program example

  use hsl_ma54_double
  implicit none
  integer, parameter :: wp = kind(1.0d0)
  integer :: info, n, nb
  real(wp), allocatable :: ap(:), b(:), buf(:)

! Read the matrix order
  read(*,*) n
  nb = min(n,100)

! Allocate the arrays
  allocate(ap(n*(n+1))/2, b(n), buf(nb*n))

! Read the lower-triangular matrix in the lower packed format
  read(*,*) ap(1:n*(n+1))/2

! Transform the matrix to partial blocked hybrid format
  call ma54_to_block( n, n, nb, ap, buf, info )
  if (info /= 0) call terminate("ma54_to_block")

! Factorize the matrix
  call ma54_factor( n, n, nb, ap, buf, info )
  if (info /= 0) call terminate("ma54_factor")
```

```
  ! Read the right-hand side and solve the equation
    read(*,*) b(1:n)
    call ma54_solve1( n, nb, ap, b, info )
    if (info /= 0) call terminate("ma54_solve1")
    write(*,'(8f10.3)') b(1:n)

  ! Deallocate the arrays
    deallocate(ap, b, buf)

  contains

    subroutine terminate(name)
      character(*) name
      write(*,*) "Stopping after failure in ",name," with info = ", info
      deallocate(ap, b, buf)
      stop
    end subroutine terminate

  end program example
```

Given the data

```
  3
  5 1 1   5 1   5
  7 7 7
```

this produces the output:

```
       1.000     1.000     1.000
```