Science & Technology Facilities Council

HSL_MA64

PACKAGE SPECIFICATION HSL

1 SUMMARY

For a full matrix that is real symmetric, complex Hermitian, or complex symmetric, this module performs **partial or full factorizations and performs solutions of corresponding sets of equations, paying special attention to the efficient use of cache memory**. In the real and complex Hermitian cases, the matrix need not be positive definite. The module performs symmetric interchanges for stability and uses both 1×1 and 2×2 pivots, assuming that the matrix is **well scaled** in the sense that changes that are small compared to the largest entry can be tolerated. 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.

Eliminations are limited to the leading p rows and columns. Stability considerations may lead to $q \leq p$ eliminations being performed, but there is an option to force all p eliminations to be performed. Using an asterisk to represent taking the transpose or Hermitian transpose of a matrix, the factorization takes the form

$$PAP^* = \begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix} \begin{pmatrix} D & \\ & S_{22} \end{pmatrix} \begin{pmatrix} L_{11}^* & L_{21}^* \\ & I \end{pmatrix}$$

where A has order n, P is a permutation matrix, L_{11} is a unit lower triangular matrix of order q, D is block diagonal of order q, and S_{22} is a matrix of order $n-q$. The permutation matrix P has the form

$$P = \begin{pmatrix} P_1 & \\ & I \end{pmatrix}$$

where P_1 is of order p. Each diagonal block of D has size one or two.

The input format for A is that its lower triangular part is held in lower packed format (that is, packed by columns). This format is also used for S_{22} on return. The matrix $\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix}$ is returned as a sequence of block columns, each of which consists of a triangular matrix packed by columns followed by a rectangular matrix packed by columns.

Subroutines are provided for partial solutions, that is, solving equations of the form

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

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

Reference: J.K. Reid and J.A. Scott (2009). Partial factorization of a dense symmetric indefinite matrix. RAL-TR-2009-015, August 2009.

ATTRIBUTES — Version: 6.3.1 (8 February 2018). **Types:** Real (single, double), Complex (single, double). **Language:** Fortran 95. **Remark:** `HSL_MA54` or `HSL_MP54` should be used if A is known to be positive definite. **Calls:** `_AXPY`, `_COPY`, `_SWAP`, `_GEMV`, `_GEMM`, `_TPSV`. **Parallelism:** May use OpenMP. **Original date:** December 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_MA64_single
```

Double precision version
```
USE HSL_MA64_double
```
Complex version
```
USE HSL_MA64_complex
```
Double complex version
```
USE HSL_MA64_double_complex
```
If it is required to use more than one module at the same time, the derived types (Sections 2.6 and 2.7) must be renamed on all but one of the `USE` statements.

2.2 Package types

We use the term **package type** to mean
`REAL(kind(0.0))` in `HSL_MA64_single`,
`REAL(kind(0.0d0))` in `HSL_MA64_double`,
`COMPLEX(kind(0.0))` in `HSL_MA64_complex`,
`COMPLEX(kind(0.0d0))` in `HSL_MA64_double_complex`.

We also use **real type** to mean
`REAL(kind(0.0))` in `HSL_MA64_single` and `HSL_MA64_complex`,
`REAL(kind(0.0d0))` in `HSL_MA64_double` and `HSL_MA64_double_complex`.

`INTEGER(long)` denotes `INTEGER(kind = selected_int_kind(18))`. All logicals are default logicals.

2.3 Subroutines

The following subroutines are available:

`MA64_factor` partially factorizes a matrix in lower packed format.

`MA64_solveL1` solves $\begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix} x = b$ and `MA64_solveL2` solves $\begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix} X = B$.

`MA64_solveD1` solves $\begin{pmatrix} D & \\ & I \end{pmatrix} x = b$ and `MA64_solveD2` solves $\begin{pmatrix} D & \\ & I \end{pmatrix} X = B$.

`MA64_solveDLT1` solves $\begin{pmatrix} D & \\ & I \end{pmatrix} \begin{pmatrix} L_{11}^* & L_{21}^* \\ & I \end{pmatrix} x = b$ and `MA64_solveDLT2` solves $\begin{pmatrix} D & \\ & I \end{pmatrix} \begin{pmatrix} L_{11}^* & L_{21}^* \\ & I \end{pmatrix} X = B$.

`MA64_solveLT1` solves $\begin{pmatrix} L_{11}^* & L_{21}^* \\ & I \end{pmatrix} x = b$ and `MA64_solveLT2` solves $\begin{pmatrix} L_{11}^* & L_{21}^* \\ & I \end{pmatrix} X = B$.

The derived data type, `MA64_control`, is accessible from the package and holds controlling data (see Section 2.6). The user must declare a scalar of this type and pass it as an actual argument to subroutine `MA64_factor`.
The derived data type, `MA64_info`, is accessible from the package and is used to return information about each procedure call. The user must declare a scalar of this type and pass it as an actual argument to each procedure.

2.4 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`.

2.5 Argument lists and calling sequences

We use square brackets [] to indicate optional arguments, which are at the end of the argument list. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that all optional arguments be called by keyword, not by position**.

2.5.1 Partially factorize a matrix

The real case:

```
call MA64_factor(n,p,nb,nbi,a,la,cntl,q,ll,perm,d,buf,info[,s,n_threads])
```

The complex case:

```
call MA64_factor(n,p,nb,nbi,a,la,cntl,q,ll,perm,d,buf,info,matrix_type[,s,n_threads])
```

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

`p` is a scalar of `INTENT(IN)` and type default `INTEGER`. Eliminations are limited to the first `p` rows and columns. **Restriction:** $0 \leq$ `p` $\leq$ `n`.

`nb` is a scalar of `INTENT(IN)` and type default `INTEGER` that specifies the block size for the block-column format. Section 4.5 contains a discussion of suitable values. **Restriction:** `mod(nb,nbi)` $= 0$.

`nbi` is a scalar of `INTENT(IN)` and type default `INTEGER` that specifies the inner block size. Section 4.5 contains a discussion of suitable values. **Restriction:** `nbi` > 1.

`a` is an array of `INTENT(INOUT)`, package type, and shape `la`. On entry, `a(la+1-ln:la)` holds the matrix in lower packed format, where `ln=(n*(n+1))/2`. If the matrix is real symmetric or complex Hermitian, the imaginary parts of the entries on the diagonal are ignored. On return, `a(1:ll)` holds the matrix $\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix}$ as a sequence of block columns, each of which consists of a triangular matrix packed by columns followed by a rectangular matrix packed by columns. There are `nb` columns in each block column except the last, which may have fewer. On return, `a(la+1-ls:la)` holds the matrix S_{22} in lower packed format, where `ls=((n-q)*(n-q+1))/2`.

`la` is a scalar of `INTENT(IN)` and type `INTEGER(long)` that specifies the size of the array `a`. **Restriction:** `la` $\geq$ `min(n*n,(n*(n+nb+1))/2)`.

`cntl` is a scalar of `INTENT(IN)` and type `MA64_control`. Its purpose is explained in Section 2.6.

`q` is a scalar of `INTENT(OUT)` and type default `INTEGER`. On return, it holds the number of eliminations performed.

`ll` is a scalar of `INTENT(OUT)` and type `INTEGER(long)`. On return, it holds size of the packed matrix $\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix}$, that is, `(q*(n+n-q+1))/2`.

`perm` is an array of shape `(p)`, `INTENT(INOUT)`, and type default `INTEGER`. The input value is ignored if `cntl%twos` is false; otherwise, each sequence of negative values `perm(`$i+1$`)`, `perm(`$i+2$`)`, ..., `perm(`$i+k$`)` must be of even length and is taken to be a recommendation for the 2×2 pivots $(i+1,i+2)$, $(i+3,i+4)$, ..., $(i+k-1,i+k)$. On return, for i = 1, 2, ..., `p`, `perm(`i`)` is set to the index of the row of A that was permuted to row i.

`d` is an array of shape `(2*p)`, `INTENT(OUT)`, and package type. On return, `d(1:2*q)` holds the inverse of D, except that zero diagonal blocks (pivots) are not inverted. Diagonal entries are in `d(1:2*q-1:2)` and entries to the right of the diagonal are in `d(2:2*q-2:2)`. `d(2*q)` is given the value zero.

buf is an array of size (n+n*nb) and package type that is used as workspace.

info is a scalar of INTENT(INOUT) and type MA64_info. Its purpose is explained in Section 2.7.

matrix_type is a scalar of INTENT(IN) and type default INTEGER. It must have the value -4 if the matrix is Hermitian and the value -5 if the matrix is complex symmetric.

s is an optional scalar of INTENT(IN) The presence of s with a value in the range $0 <$ s $<$ p indicates that columns 1:s should be searched last for pivots; if this causes a recommended 2×2 pivot to be split, the recommendation is ignored.

n_threads is an optional scalar of INTENT(INOUT) and type default INTEGER. It has the target attribute. 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 MA64_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 MA64_factor, which means that the number of threads used within MA64_factor can be made to vary during its execution. The argument n_threads is ignored if OpenMP is not enabled.

2.5.2 One partial solution

The real case:

```
call MA64_solveL1  (n,q,nb,b,flag,a,ll)
call MA64_solveD1  (n,q,   b,flag,  d)
call MA64_solveDLT1(n,q,nb,b,flag,a,ll,d)
call MA64_solveLT1 (n,q,nb,b,flag,a,ll)
```

The complex case:

```
call MA64_solveL1  (n,q,nb,b,flag,a,ll,matrix_type)
call MA64_solveD1  (n,q,   b,flag,  d,matrix_type)
call MA64_solveDLT1(n,q,nb,b,flag,a,ll,d,matrix_type)
call MA64_solveLT1 (n,q,nb,b,flag,a,ll,matrix_type)
```

n,q,nb are scalars of INTENT(IN) and type default INTEGER whose values must be unchanged since the call to MA64_factor.

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

flag is a scalar of INTENT(OUT) and type default INTEGER. It has the value zero after a successful return. Nonzero values are explained in Section 2.8.

a is an array of shape ll, INTENT(IN), and package type. It holds the matrices L_{11} and L_{21}, as returned in a(1:ll) by MA64_factor.

ll is a scalar of INTENT(IN) and type INTEGER(long) whose value must be unchanged since the call to MA64_factor.

d is an array of shape (2*q), INTENT(IN), and package type. It holds the inverse of D, as returned by MA64_factor.

matrix_type is a scalar of INTENT(IN) and type default INTEGER. It must have the value -4 if the matrix is Hermitian and the value -5 if the matrix is complex symmetric.

2.5.3 One or more partial solutions

The real case:

```
call MA64_solveL2  (n,q,nb,nrhs,b,ldb,flag,a,ll)
call MA64_solveD2  (n,q,   nrhs,b,ldb,flag,  d)
call MA64_solveDLT2(n,q,nb,nrhs,b,ldb,flag,a,ll,d)
call MA64_solveLT2 (n,q,nb,nrhs,b,ldb,flag,a,ll)
```

The complex case:

```
call MA64_solveL2  (n,q,nb,nrhs,b,ldb,flag,a,ll,matrix_type)
call MA64_solveD2  (n,q,   nrhs,b,ldb,flag,  d,matrix_type)
call MA64_solveDLT2(n,q,nb,nrhs,b,ldb,flag,a,ll,d,matrix_type)
call MA64_solveLT2 (n,q,nb,nrhs,b,ldb,flag,a,ll,matrix_type)
```

`n,q,nb` are scalars of `INTENT(IN)` and type default `INTEGER` whose values must be unchanged since the call to `MA64_factor`.

`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.

`b` is a rank-2 array of `INTENT(INOUT)` and package type. 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` $\geq$ `n`.

`flag` is a scalar of `INTENT(OUT)` and type default `INTEGER`. It has the value zero after a successful return. Nonzero values are explained in Section 2.8.

`a` is an array of shape `ll`, `INTENT(IN)`, and package type. It holds the matrices L_{11} and L_{21}, as returned in `a(1:ll)` by `MA64_factor`.

`ll` is a scalar of `INTENT(IN)` and type `INTEGER(long)` whose value must be unchanged since the call to `MA64_factor`.

`d` is an array of shape `(2*q)`, `INTENT(IN)`, and package type. It holds the inverse of D, as returned by `MA64_factor`.

`matrix_type` is a scalar of `INTENT(IN)` and type default `INTEGER`. It must have the value -4 if the matrix is Hermitian and the value -5 if the matrix is complex symmetric.

2.6 The derived data type for controlling data

The derived data type `MA64_control` is accessible from the package and holds controlling data. The user must declare a scalar of this type and pass it as an actual argument to subroutine `MA64_factor`. Default values for the components are set when the scalar is declared. If other values are wanted, they must be set before calling `MA64_factor`. The components are:

`p_thresh` is a scalar of type default `INTEGER` with default value 32. If `p`$\leq$`p_thresh`, no parallel regions are executed, which has the effect of ignoring the value of `n_threads` and executing on a single thread.

`small` is a scalar of real type. Diagonal entries of D of modulus less than `small` are replaced by zero. A 2×2 pivot will have its off-diagonal entry of modulus greater than `small` or both its diagonal entries of modulus greater than `small`. The default value is 1×10^{-20}.

static is a scalar of real type. If static > 0.0 and p pivots with relative pivot value greater than umin have not been not found, diagonal entries are accepted as pivots after being changed if necessary to have absolute value static (details in Section 4.10). The default value is zero. **Restriction:** Either static is zero or static $\geq$ small.

twos is a scalar of type default LOGICAL. If its value is .true., the signs of perm indicate recommended 2×2 pivots. The default value is .false..

u is a scalar of real type. It holds the initial value of the relative pivot tolerance u, which ensures that the entries of L satisfy the inequality $l_{ij} \leq u^{-1}$, except in those columns for which relaxed and/or static pivoting is active (see Section 4.10). The test for a 1×1 pivot is

$$|a_{kk}| > u \max_{j \neq k} |a_{kj}|,$$

and the test for a 2×2 pivot is

$$\left| \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{j \neq k,l} |a_{kj}| \\ \max_{j \neq k,l} |a_{lj}| \end{pmatrix} < \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}.$$

If u = 0, this is interpreted as a requirement for the pivot to be non-singular. Values of u greater than 0.5 are treated as 0.5 and values less than zero are treated as zero. The default value is 0.1.

umin is a scalar of real type. It holds the minimum value of the relative pivot tolerance. If p stable pivots have not been not found and the candidate pivot with greatest relative pivot value has value $v \geq$ umin, this is accepted as a pivot and u is reset to v. Values of umin greater than u are treated as u and values less than zero are treated as zero. If p=n and both u and umin are greater than 0.5, umin is treated as having the value 0.5. The default value is 1.0.

2.7 The derived data type for informational data

The derived data type MA64_info is accessible from the package and is used to return information from MA64_factor. The user must declare a scalar of this type and pass it as an actual argument to MA64_factor. The components are:

detlog is a scalar of real type. On exit from MA64_factor, it holds the logarithm of the absolute value of the determinant of D or zero if the determinant is zero.

detsign is a scalar of type default INTEGER. On exit from MA64_factor in the real or complex Hermitian case, it holds the sign of the determinant of D or zero if the determinant of D is zero.

detarg is a scalar of package type that is absent in the real case. On exit from MA64_factor in the complex symmetric case, it holds the determinant of D divided by its absolute value or one if the determinant is zero.

flag is a scalar of type default INTEGER. After a successful return, it has the value zero. An unsuccessful call is flagged with one of these negative values:

-1 n < 0.

-2 p < 0.

-3 p $>$ n.

-4 nbi ≤ 1.

-7 la $<$ min(n*n,(n*(n+nb+1))/2).

-10 cntl%static $<$ cntl%small and cntl%static $\neq 0.0$.

-11 perm has an odd number of adjacent negative entries when cntl%twos has the value .true..

-12 mod(nb,nbi) $\neq 0$.

-13 IEEE infinities found in the reduced matrix, probably caused by cntl%small or cntl%u having too small a value.

-14 The value of matrix_type is neither -4 nor -5.

num_neg is a scalar of type default INTEGER. After a successful return from MA64_factor in the real or complex Hermitian case, it holds the number of negative eigenvalues of D.

num_nothresh is a scalar of type default INTEGER. After a successful return from MA64_factor, it holds the number diagonal entries of D that were chosen as 1×1 pivots without satisfying the relative pivot threshold. It will have the value zero if cntl%static is zero.

num_perturbed is a scalar of type default INTEGER. After a successful return from MA64_factor, it holds the number diagonal entries of D that were perturbed to cntl%static or -cntl%static. It will have the value zero if cntl%static is zero.

num_zero is a scalar of type default INTEGER. After a successful return from MA64_factor, it holds the number of zero eigenvalues of D.

num_2x2 is a scalar of type default INTEGER. After a successful return from MA64_factor, it holds the number of 2×2 blocks in D.

usmall is a scalar of real type that is set after a successful return from MA64_factor as follows:

1. if num_perturbed = 0,
   (a) if q = p, usmall is set to the smallest relative pivot value of the chosen pivots;
   (b) q $<$ p,
      i. if the leading block of size p-q of S_{22} is not zero, usmall is set to the largest value of cntl%umin that would have led to a greater value of q;
      ii. if the leading block of size p-q of S_{22} is zero, usmall is set to zero,
2. if num perturbed > 0, usmall is set to -1.

u is a scalar of real type that is set after a successful return from MA64_factor to the final value of the relative pivot tolerance u.

2.8 Error returns from the solve subroutines

If an error occurs in a solve subroutine, flag is set to one of these values:

-1 n < 0.

-4 nb ≤ 1.

-5 nrhs < 0.

-6 ldb $<$ n.

-8 q < 0.

-9 q $>$ n.

-14 The value of matrix_type is neither -4 nor -5.

3 GENERAL INFORMATION

Other routines called directly: The BLAS _AXPY, _COPY, _SWAP, _GEMM, _GEMV, _TRSV, _TRSM.

Input/output: None.

Restrictions: n $\geq$ 0; n $\geq$ p $\geq$ 0; n $\geq$ q $\geq$ 0; nrhs $\geq$ 0; nbi $\geq$ 1; mod(nb,nbi)=0; ldb $\geq$ n; la $\geq$ min(n*n,(n*(n+nb+1))/2); cntl%static $\geq$ cntl%small or cntl%static = 0.0.

Changes from Version 1
The type of arguments la and ll became INTEGER(long) and the type of the integers used internally to address the arrays a and buf became INTEGER(long). These changes allow these arrays to be larger than is possible if they are addressed with default integers of 32 bits.

Changes from Version 2
The inner block size nbi was added so that more operations are performed with BLAS 3, which improves the execution speed. Instead of rearranging the trailing n-p columns, they are updated by making temporary rearrangements of blocks of nb/2 columns.

Changes from Version 3
OpenMP features, the optional argument n_threads and the control component p_thresh were added. Rearrangement of the trailing n-p columns was restored for execution on more than one thread or on a single thread with p>nb.

Changes from Version 4
To separate static pivoting from relaxing the relative pivot threshold, the control component umin was added and the informational component u was added to return the value actually used. The test on the size of la was relaxed.

Changes from Version 5
The complex versions were added. The default values of cntl%small and cntl%umin were changed.

4 METHOD

4.1 The block form

The partial factorization begins by rearranging the lower triangular part of the matrix (or its first p columns if executing on one thread with p$\leq$nb), so that it is held by block columns, with each block having nb columns except the final block which may have fewer. The blocks are held by columns. Each column in a block has the same length. The part above the diagonal is not used.

Once the partial factorization is complete, the matrix is rearranged; columns 1:q are held by block columns, each of which consists of the diagonal block in packed triangular form followed by the off-diagonal part held by columns; columns q+1:n are in packed triangular form.

This form allows the partial solution operations for a single right-hand side to be performed with the BLAS-2 subroutines _gemv and _tpsv and for multiple right-hand sides to be performed with the BLAS-3 subroutine _gemm and repeated calls of _tpsv.

4.2 Factorization in the simple case

We begin by describing the case where the number of threads is one, p>nb, the block size and inner block size are the same, s is absent, the matrix is not found to be singular or near singular, static pivoting is not requested, and no specific 2$\times$2 pivots are requested.

This is the situation at a typical intermediate stage of the factorization:

q holds the number of pivots chosen, their columns are in their final form at the front of a, and the corresponding entries of d are also in their final form.

q1 holds the index of the leading column of the block containing column q+1.

buf is a workarray that holds columns q1:q of the matrix *LD*.

m holds the index of the column being searched for a pivot.

r holds the number of pivot operations that have been applied to columns m+1:p; columns q+1:m of a are fully updated; q1-1 pivot operations have been applied to columns p+1:n.

The largest entry to the left of the diagonal in row m of the reduced matrix is determined, say in column t, and the largest entry below the diagonal in column m is determined. Next, the largest entry to the left or below the diagonal of column t is determined, which permits t,m to be tested for suitability as a 2×2 pivot. If suitable, it is accepted. Otherwise, the diagonal entry in column m is tested for suitability as a 1×1 pivot. There are performance gains from favouring 2×2 pivots in this way.

If no pivot is found in column m and m<p, m is incremented by one, the new column m is updated for pivot operations r+1:q using the BLAS-2 subroutine _gemv and tested. If no pivot is found in column m and m=p, m is reset to q+1, r is reset to q, and the new column m is tested. This continues until q=p or all the remaining columns have failed to provide a pivot.

Once a pivot has been found, the component(s) of *D* are computed, the column(s) of a are copied to the workarray buf and then replaced by the column(s) of *LD*, and q is incremented.

If this completes a block column (q≥q1+nb-1), columns m+1:p are updated for operations r+1:q1+nb-1 using the BLAS-3 subroutine _gemm, r is reset to q1+nb-1, and columns p+1:n are updated for operations q1:q1+nb-1 using the BLAS-3 subroutine _gemm. We allow a 2×2 pivot to span two block columns. In this case, only the first half of the operation will be applied to the remainder of the matrix and the final column of the workarray will need to moved forward, and q and q1 will now have the same value.

Once q=p or all the remaining columns have failed to provide a pivot, columns m+1:p are updated for operations r+1:q using the BLAS-3 subroutine _gemm, and columns p+1:n are updated for operations q1:q using the BLAS-3 subroutine _gemm.

Note that if n and p are large, most of the work is done within _gemm and so the execution will be fast; and almost all the rest is done within _gemv. If p is small almost all the work is done within _gemv.

4.3 OpenMP parallel regions

When executing on more than one thread with p>p_thresh, a parallel region is used for _gemm updating of trailing columns after each block column of the factorization has been calculated unless only one block of columns is to be updated. Each block of columns is further subdivided by rows of size nb so that if l blocks of columns are to be updated, there are $l(l+1)/2$ calls of _gemm, each potentially executed by a separate thread.

4.4 Small values of p

When executing on one thread with p≤nb, only the first p columns are rearranged to block form since only one block update is needed for the remaining columns. The operations are applied in blocks of nb/2 columns; each is rearranged to block form, fully updated using BLAS 3, and rearranged back.

4.5 Choice of the block sizes nb and nbi

The choice of nb and nbi is discussed by Reid and Scott (2009). The value of nb that allows a full matrix of this size 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. If the block size nb is large and the inner block size nbi is the same, the time taken in the BLAS-2 subroutine _gemv may be very significant, sometimes exceeding the time taken in the BLAS-3 subroutine _gemm. If nbi<nb, we can reduce the size of the rectangular block involved in the _gemv update of column m by prior calls of _gemm. Immediately an inner block of L has been found, we apply it to the columns beyond m that are in the same (outer) block.

Reid and Scott experimented using double precision reals on a Dell Precision T5400 with two Intel E5420 quad core processors running at 2.5GHz backed by 8GB of RAM and found that a good choice for one processor was nb=144, nbi=48 and for eight processors was nb=96, nbi=16. If performance is critical, we recommend that different values be tried.

4.6 Factorization with s present

If s is present and in the range 0<s<p, swaps are made between columns j and p+1-j for j=1,min(s,p-s), unless this would involve splitting a recommended 2×2 pivot which may happen if s<p-s. In the case that would cause a split, the swaps are made only for j = 1 to s-1. Following this, the leading s columns (s-1 in the exceptional case) will have been moved to the back of the set of columns 1:p.

4.7 Factorization in the singular case

When column m is searched, if its largest element is found to be less than cntl%small, the diagonal entry is accepted as a zero 1×1 pivot and no corrsponding pivotal operations are applied to the rest of the matrix. To accommodate this, we hold the inverse of D in d, and set the element corresponding to the zero pivot to zero. This results in the column of LD being zero so that no special action is needed in subsequent BLAS-2 and BLAS-3 calls later in the factorization and during the solution. It leads to the correct result when the given set of equations is consistent and avoids the solution having a large norm if the equations are not consistent.

4.8 Requesting particular 2×2 pivots

If m indexes the first half of a requested 2×2 pivot, the largest entry in the column is determined, but it is not accepted as a 1×1 pivot. Processing continues to the next column and now the recommended pivot can be tested for acceptance as a 2×2 pivot. Either it is accepted or the recommendation is cancelled and normal processing is resumed.

4.9 Relaxed threshold pivoting

Relaxed threshold pivoting refers to the case where all requested pivoting was performed (q=p) without static pivoting (next subsection), but some pivots had relative pivot values less than u (and greater than cntl%umin). The smallest relative pivot value is returned in info%usmall.

4.10 Static pivoting

If static pivoting is requested (cntl%static $>$ 0.0), the following procedure is followed whenever no 1×1 or 2×2 candidate pivot satisfies the relative threshold test with $u \geq$ cntl%umin. The 1×1 pivot that is nearest to satisfying the test is chosen and cntl%num_nothresh is incremented by one. If its absolute value is less than cntl%static, the pivot is given the value that has the same sign but absolute value cntl%static and cntl%num_perturbed is incremented by one.

Note that the use of static pivoting usually leads to an inaccurate factorization so that solutions will need to be corrected by iterative refinement or rank updating.

5 EXAMPLE OF USE

The following code reads a matrix in lower packed format and factorizes it. It then reads a right-hand side and solves the corresponding set of equations.

```
program example

  use hsl_ma64_double
  implicit none
  integer, parameter :: wp = kind(1.0d0)
  integer,parameter :: long = selected_int_kind(18) ! Long integer.
  type(ma64_control) cntl
  type(ma64_info) info
  integer :: flag, n, nb, nbi, q
  integer(long) :: la, ll
  real(wp), allocatable :: a(:), b(:), buf(:), d(:)
  integer, allocatable :: perm(:)

! Read the matrix order
  read(*,*) n
  nb = min(n,100)
  nbi = nb
  la = (n*(n+nb+1))/2

! Allocate the arrays
  allocate( a(la), b(n), buf(n+nb*n), d(2*n), perm(n) )

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

! Factorize the matrix
  call ma64_factor( n, n, nb, nbi, a, la, cntl, q, ll, perm, d, buf, info )
  if (info%flag /= 0) call terminate("ma64_factor")

! Read the right-hand side
  read(*,*) b(1:n)

! Solve the equation
  b(1:n) = b(perm(1:n))
  call ma64_solveL1( n, n, nb, b, flag, a, ll )
  if (flag /= 0) call terminate("ma64_solveL1")
  call ma64_solveDLT1( n, n, nb, b, flag, a, ll, d )
  if (flag /= 0) call terminate("ma64_solveDLT1")
  b(perm(1:n)) = b(1:n)
  write(*,'(8f10.3)') b(1:n)
```

```
contains

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

end program example
```

Given the data

```
 3
 0 5 1   5 2   3
 13 21 14
```

which represents the matrix $\begin{pmatrix} 0 & 5 & 1 \\ 5 & 5 & 2 \\ 1 & 2 & 3 \end{pmatrix}$, this produces the output:

```
    1.000     2.000     3.000
```