RANDUTV-SM

1.0

Generated by Doxygen 1.8.17

Chapter 1

Introduction

`randUTV SM-DM` is a software package that accompanies the submission *Efficient algorithms for computing a rank-revealing UTV factorization on parallel computing architectures* and includes an algorithm-by-blocks implementation for shared-memory architectures (`SM` from now on) and a distributed memory implementation (`DM` from now on) to compute a *full* factorization of a given matrix that provides low-rank approximations with near-optimal error.

This guide complements the submitted paper and includes basic setup and execution steps for the package. The provided software package includes all the algorithms described in the paper, namely:

- Directory `basic_randutv_sm` contains the *algorithm-by-blocks* for shared-memory architectures that leverages the `libflame` SuperMatrix infrastructure for the implementation.
- Directory `basic_randutv_dm` contains the distributed-memory implementation based on ScaLAPACK.

1.1 Basic installation steps for SM

1.1.1 Prerequisites

The package has been tested with the Intel 2018 compiler suite, although any other modern compiler would suffice.

The package requires an installation of the `libflame` library with support for the SuperMatrix runtime task scheduler with OpenMP multithread support enabled.

To install the library, follow the next steps:

```
## Clone the library from the official Git repository.
https://github.com/figual/libflame.git
## Configure the library for appropriate SuperMatrix and OpenMP support.
## Choose the desired library installation location.
./configure --prefix=/opt/libflame --enable-max-arg-list-hack \\
--enable-supermatrix --enable-multithreading=openmp
## Compile and install the library.
make && make install
```

1.1.2 Package compilation

To compile the package, just tune the appropriate `libflame` installation location in the Makefile (variable LIBF↩ LAME) and execute `make`.

1.1.3 Testing

A simple testing driver (basic_test.c) is provided to test correctness and performance. The driver tests a small case (`m=n=8, nb=3`), although larger matrices and different block sizes can be easily tested by modifying the code. The macro `PRINT_DATA` can be activated to show input and output data prior and after factorization.

```
# Compile software
make
# Run basic test
./basic_test.x
```

1.1.4 Sample output for the basic test

The following is a sample output of the basic driver provided in the package for `m=n=8, nb=3`. The output includes the contents of the original matrix, and matrices U, T, V after factorization, together with residual computations and report.

```
 Ai = [
5.658107e-01 2.533418e-02 2.689885e-01 2.234422e-01 1.470419e-01 9.781572e-01 3.210570e-01 4.331079e-01
6.109299e-01 3.162596e-01 9.233823e-02 2.144171e-01 6.236489e-02 4.471605e-01 8.889882e-01 1.360038e-01
5.057681e-01 6.127619e-02 8.809632e-01 1.198589e-02 7.724467e-02 3.548297e-01 2.567804e-01 9.622619e-01
1.796469e-01 8.365902e-02 5.625732e-01 8.683906e-01 9.637282e-01 9.549481e-01 9.845709e-01 7.648950e-01
8.166863e-01 9.767909e-01 6.635139e-01 3.317871e-01 2.458365e-01 4.252188e-01 8.704291e-01 6.721158e-01
1.834717e-01 9.780583e-01 9.814409e-01 5.361120e-01 6.618976e-01 2.287188e-01 2.186908e-01 5.188587e-01
5.846530e-01 8.738890e-01 9.619104e-01 5.565968e-01 3.858843e-01 8.024953e-03 1.240179e-01 6.624928e-01
4.221561e-01 5.307681e-02 1.394470e-01 8.975978e-01 2.711707e-01 6.942072e-01 9.387133e-02 8.191577e-01
 ];
% Just before computing factorization.
% Just after computing factorization.
 Af = [
4.028131e+00 0.000000e+00 0.000000e+00 1.572611e-03 -1.254069e-04 -3.974468e-05 2.133971e-06 -1.261850e-08
0.000000e+00 1.560757e+00 0.000000e+00 9.447237e-02 8.679398e-03 -1.894195e-03 9.426621e-05 -4.129835e-07
0.000000e+00 0.000000e+00 1.127758e+00 -8.138325e-02 8.320477e-02 -1.471131e-03 6.790575e-05 4.722842e-07
0.000000e+00 0.000000e+00 0.000000e+00 1.025960e+00 0.000000e+00 0.000000e+00 2.974335e-04 -4.174366e-08
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 7.202730e-01 0.000000e+00 -1.437927e-03 6.341683e-06
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 4.495372e-01 -1.424684e-02 1.105057e-04
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 2.000458e-01 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 5.383423e-02
 ];
 Uf = [
-2.630485e-01 3.834227e-01 -2.286753e-01 -2.293101e-01 -1.112542e-01 7.791924e-01 1.490663e-01 1.928160e-01
-2.342793e-01 1.694648e-01 -5.202805e-01 3.851442e-01 -1.293763e-01 -1.561018e-01 3.835930e-01 -5.559833e-01

-3.009446e-01 -2.776689e-02 -2.385838e-01 -6.926807e-01 5.160826e-01 -2.203839e-01 -5.281333e-02
       -2.328798e-01
-4.670319e-01 4.505680e-01 3.805026e-01 3.632432e-01 4.575581e-01 -1.711215e-01 9.187044e-02 2.346479e-01
-4.397730e-01 -2.166848e-01 -4.699144e-01 2.052927e-01 -1.460808e-01 -1.311069e-01 -5.527670e-01
       3.906442e-01
-3.786614e-01 -4.531545e-01 3.560796e-01 1.733843e-01 8.848487e-02 4.309356e-01 -2.352114e-01 -4.955323e-01
-3.720799e-01 -4.925132e-01 1.246324e-01 -1.451726e-01 -2.230632e-01 -1.237276e-01 6.439406e-01 3.203964e-01

-3.033219e-01 3.544230e-01 3.349118e-01 -3.066698e-01 -6.451800e-01 -2.691409e-01 -2.094541e-01
       -2.088131e-01
 ];
 Vf = [
-3.232382e-01 6.700173e-03 -5.179460e-01 -1.896264e-01 -4.219098e-01 -7.662799e-03 5.096451e-01 3.917157e-01

-3.177619e-01 -6.368799e-01 -7.140105e-02 3.226619e-01 -3.606892e-01 2.385372e-01 -4.440570e-01 4.387743e-03

-4.178969e-01 -4.104571e-01 2.950856e-02 -3.003321e-01 4.943919e-01 1.989209e-01 3.662432e-01 -3.843046e-01
-3.343366e-01 1.490211e-01 5.511752e-01 1.397657e-01 -4.900513e-01 -2.881237e-01 2.779974e-01 -3.771406e-01
-2.759702e-01 1.323785e-02 4.797679e-01 3.049528e-01 3.029940e-01 1.273923e-01 1.520295e-01 6.850873e-01
-3.480325e-01 5.912583e-01 -5.326370e-02 -3.706634e-02 -5.341519e-02 6.733877e-01 -2.137079e-01
       -1.519893e-01
-3.403486e-01 2.141266e-01 -4.306222e-01 6.027210e-01 3.279437e-01 -3.884398e-01 -3.311543e-02 -1.734226e-01

-4.415907e-01 8.969370e-02 5.932280e-02 -5.408831e-01 7.152700e-02 -4.470919e-01 -5.116424e-01 1.846451e-01
 ];
% Computing residuals...
Res. || A - U * T * V' ||_F / || A ||_F = [
    1.15101e-15
 ];
Res. || I - U' * U ||_F / || U ||_F = [
    7.85750e-16
 ];
Res. || I - V' * V ||_F / || V ||_F = [
    5.97083e-16
 ];
Res. || sin.val.A - sin.val.T ||_F / || sin.val.A ||_F = [
    2.37709e-16
 ];
% End of Program
```

1.2 Basic installation steps for DM

The steps to install this subpackage are the following:

1. Uncompress the file.
2. Edit the first part of the "Makefile" file.

   The user must define the following variables (maybe replace the current values assigned to these variables):

   - the Fortran 90 compiler,
   - the Fortran 90 compiler flags,
   - the Fortran 90 loader or linker (usually the same as the compiler),
   - the Fortran 90 loader flags, and
   - the libraries to be employed:
     - the ScaLAPACK library (and the PBLAS and BLACS libraries, if not included inside it),
     - the LAPACK library, and
     - the BLAS library.

   There is no need to define the variable MKLROOT (and install the MKL library) if public-domain ScaLAPACK, LAPACK and BLAS libraries are employed. No more changes are required in this file.
3. Type "make" to compile and generate the executable file.
4. Execute the driver with one of the following commands: mpirun -np 4 pdttrandutv.x mpiexec -n 4 pdttrandutv.x

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

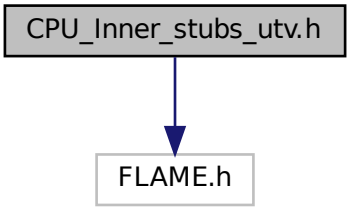
File Documentation

3.1 CPU_Inner_stubs_utv.h File Reference

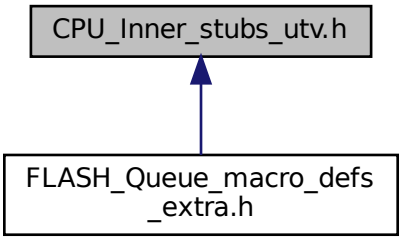
This file (together with the companion GPU_Inner_stubs_utv.c) contains the task implementations (kernels) to support randUTV. These routines are executed out-of-order, following a dataflow execution path.

```
#include "FLAME.h"
```

Include dependency graph for CPU_Inner_stubs_utv.h:



This graph shows which files directly or indirectly include this file:



Functions

- FLA_Error CPU_Compute_dense_QR_UT_inner_utv (FLA_Obj A, FLA_Obj S, dim_t nb_alg)

  Task implementation. Computes QR of dense matrix A.
- FLA_Error CPU_Apply_left_Qt_of_dense_QR_UT_inner_utv (FLA_Obj U, FLA_Obj S, FLA_Obj C, dim_t nb_alg)

  Task implementation. Applies block Householder transformations stored in U and S to matrix C from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error CPU_Apply_right_Q_of_dense_QR_UT_inner_utv (FLA_Obj U, FLA_Obj S, FLA_Obj C, dim_t nb_alg)

  Task implementation. Applies block Householder transformations stored in U and S to matrix C from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error CPU_Compute_td_QR_UT_inner_utv (FLA_Obj U, FLA_Obj D, FLA_Obj S, dim_t nb_alg)

  Task implementation. Compute QR factorization of [U; D], where U is upper triangular and D is dense.
- FLA_Error CPU_Apply_left_Qt_of_td_QR_UT_inner_utv (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G, dim_t nb_alg)

  Task implementation. Applies block Householder transformations stored in D and S to matrices F and G from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error CPU_Apply_right_Q_of_td_QR_UT_inner_utv (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G, dim_t nb_alg)

  Task implementation. Applies block Householder transformations stored in D and S to matrices F and G from right side. The orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error CPU_Keep_upper_triang_inner_utv (FLA_Obj A)

  Task implementation. Keep upper triangular part, and zero strictly lower triangular part.
- FLA_Error CPU_Set_to_zero_inner_utv (FLA_Obj A)

  Task implementation. Set matrix to zero.
- FLA_Error CPU_Gemm_tn_inner_utv (FLA_Obj alpha, FLA_Obj A, FLA_Obj B, FLA_Obj beta, FLA_Obj C)

  Task implementation. Computes C := alpha ∗ A′ ∗ B + C.
- FLA_Error CPU_Gemm_nn_inner_utv (FLA_Obj alpha, FLA_Obj A, FLA_Obj B, FLA_Obj beta, FLA_Obj C)

  Task implementation. Computes C := alpha ∗ A ∗ B + C.
- FLA_Error CPU_Compute_svd_inner_utv (FLA_Obj U, FLA_Obj A, FLA_Obj VT)

  Task implementation. Computes SVD of A.
- FLA_Error CPU_Gemm_abta_inner_utv (FLA_Obj A, FLA_Obj B)

  Task implementation. Computes A := B′ ∗ A.
- FLA_Error CPU_Gemm_aabt_inner_utv (FLA_Obj A, FLA_Obj B)

  Task implementation. Computes A := A ∗ B′.
- FLA_Error CPU_Gemm_aab_inner_utv (FLA_Obj A, FLA_Obj B)

  Task implementation. Computes A := A ∗ B.
- FLA_Error CPU_Mycopy_inner_utv (FLA_Obj A, FLA_Obj B)

  Task implementation. Copies the contents of A into the upper part of B. Unlike FLA_Copy, this routine allows that A.m can be strictly less than B.m.

3.1.1 Detailed Description

This file (together with the companion GPU_Inner_stubs_utv.c) contains the task implementations (kernels) to support randUTV. These routines are executed out-of-order, following a dataflow execution path.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)

Francisco D. Igual (figual@ucm.es)

Nathan Heavner (Nathan.Heavner@colorado.edu)

Per-Gunnar Martinsson (pgm@oden.utexas.edu)

Version

1.0

Date

2021-01-14

Copyright

3.1.2 Function Documentation

3.1.2.1 CPU_Apply_left_Qt_of_dense_QR_UT_inner_utv()

```
FLA_Error CPU_Apply_left_Qt_of_dense_QR_UT_inner_utv (
              FLA_Obj U,
              FLA_Obj S,
              FLA_Obj C,
              dim_t nb_alg )
```

Task implementation. Applies block Householder transformations stored in U and S to matrix C from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

C := (I - U_i ∗ S_i ∗ U_i') ∗ C.

Parameters

in	*U*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.1.2.2 CPU_Apply_left_Qt_of_td_QR_UT_inner_utv()

```
FLA_Error CPU_Apply_left_Qt_of_td_QR_UT_inner_utv (
              FLA_Obj D,
```

```
            FLA_Obj S,
            FLA_Obj F,
            FLA_Obj G,
            dim_t nb_alg )
```

Task implementation. Applies block Householder transformations stored in D and S to matrices F and G from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

[F; G] := (I - [I; D_i] ∗ S_i ∗ [I; D_i]') ∗ [F; G].

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.1.2.3 CPU_Apply_right_Q_of_dense_QR_UT_inner_utv()

```
FLA_Error CPU_Apply_right_Q_of_dense_QR_UT_inner_utv (
            FLA_Obj U,
            FLA_Obj S,
            FLA_Obj C,
            dim_t nb_alg )
```

Task implementation. Applies block Householder transformations stored in U and S to matrix C from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

C := C ∗ (I - U_i ∗ S_i' ∗ U_i')

Parameters

in	*U*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error.

3.1.2.4 CPU_Apply_right_Q_of_td_QR_UT_inner_utv()

```
FLA_Error CPU_Apply_right_Q_of_td_QR_UT_inner_utv (
              FLA_Obj D,
              FLA_Obj S,
              FLA_Obj F,
              FLA_Obj G,
              dim_t nb_alg )
```

Task implementation. Applies block Householder transformations stored in D and S to matrices F and G from right side. The orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

[F G] := [F G] ∗ (I - [I; D_i] ∗ S_i ∗ [I; D_i]').

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.1.2.5 CPU_Compute_dense_QR_UT_inner_utv()

```
FLA_Error CPU_Compute_dense_QR_UT_inner_utv (
              FLA_Obj A,
              FLA_Obj S,
              dim_t nb_alg )
```

Task implementation. Computes QR of dense matrix A.

Parameters

in,out	*A*	Matrix to be factorized. On output, it contains the factor R in the upper triangular part, and Householder vectors in the strictly lower triangular part.
out	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.1.2.6 CPU_Compute_svd_inner_utv()

```
FLA_Error CPU_Compute_svd_inner_utv (
            FLA_Obj U,
            FLA_Obj A,
            FLA_Obj VT )
```

Task implementation. Computes SVD of A.

Parameters

out	*U*	The left orthonormal matrix of the SVD factorization.
in,out	*A*	On input, the matrix to be factorized. On output, the matrix with the singular values on the diagonal.
out	*VT*	The transpose of the right orthonormal matrix of the SVD factorization.

Returns

FLA_Error

3.1.2.7 CPU_Compute_td_QR_UT_inner_utv()

```
FLA_Error CPU_Compute_td_QR_UT_inner_utv (
            FLA_Obj U,
            FLA_Obj D,
            FLA_Obj S,
            dim_t nb_alg )
```

Task implementation. Compute QR factorization of [U; D], where U is upper triangular and D is dense.

Parameters

in,out	*U*	Upper triangular matrix. On output, it contains the factor R in the upper triangular part. The strictly lower triangular part is not accessed nor modified.
in,out	*D*	Dense matrix. On output, it contains Householder vectors.
out	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.1.2.8 CPU_Gemm_aab_inner_utv()

```
FLA_Error CPU_Gemm_aab_inner_utv (
            FLA_Obj A,
            FLA_Obj B )
```

Task implementation. Computes A := A ∗ B.

Parameters

in,out	*A*	Target matrix.
in	*B*	Input matrix.

Returns

FLA_Error

3.1.2.9 CPU_Gemm_aabt_inner_utv()

```
FLA_Error CPU_Gemm_aabt_inner_utv (
            FLA_Obj A,
            FLA_Obj B )
```

Task implementation. Computes A := A ∗ B'.

Parameters

in,out	*A*	Target matrix.
in	*B*	Input matrix.

Returns

FLA_Error

3.1.2.10 CPU_Gemm_abta_inner_utv()

```
FLA_Error CPU_Gemm_abta_inner_utv (
            FLA_Obj A,
            FLA_Obj B )
```

Task implementation. Computes A := B' ∗ A.

Parameters

in,out	*A*	Target matrix.
in	*B*	Input matrix.

Returns

FLA_Error

3.1.2.11 CPU_Gemm_nn_inner_utv()

```
FLA_Error CPU_Gemm_nn_inner_utv (
             FLA_Obj alpha,
             FLA_Obj A,
             FLA_Obj B,
             FLA_Obj beta,
             FLA_Obj C )
```

Task implementation. Computes C := alpha ∗ A ∗ B + C.

Returns

FLA_Error

3.1.2.12 CPU_Gemm_tn_inner_utv()

```
FLA_Error CPU_Gemm_tn_inner_utv (
             FLA_Obj alpha,
             FLA_Obj A,
             FLA_Obj B,
             FLA_Obj beta,
             FLA_Obj C )
```

Task implementation. Computes C := alpha ∗ A' ∗ B + C.

Returns

FLA_Error

3.1.2.13 CPU_Keep_upper_triang_inner_utv()

```
FLA_Error CPU_Keep_upper_triang_inner_utv (
             FLA_Obj A )
```

Task implementation. Keep upper triangular part, and zero strictly lower triangular part.

Parameters

in,out	*A*	Matrix to be updated.

Returns

FLA_Error

3.1.2.14 CPU_Mycopy_inner_utv()

```
FLA_Error CPU_Mycopy_inner_utv (
            FLA_Obj A,
            FLA_Obj B )
```

Task implementation. Copies the contents of A into the upper part of B. Unlike FLA_Copy, this routine allows that A.m can be strictly less than B.m.

Parameters

in	*A*	Target matrix.
out	*B*	Input matrix.

Returns

FLA_Error

3.1.2.15 CPU_Set_to_zero_inner_utv()

```
FLA_Error CPU_Set_to_zero_inner_utv (
            FLA_Obj A )
```

Task implementation. Set matrix to zero.

Parameters

in,out	*A*	Matrix to be updated.

Returns

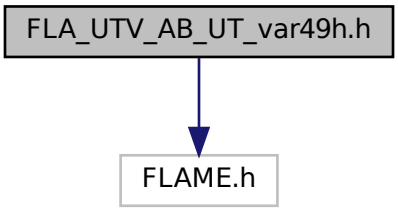
FLA_Error

3.2 FLA_UTV_AB_UT_var49h.h File Reference

This file (together with the companion FLA_UTV_AB_UT_var49h.c) contains the main routine for the algorithm-by-blocks for the randUTV factorization (FLA_UTV_AB_UT_var49h), together with the main building blocks that compute the factorization, as described in the companion manuscript.

```
#include "FLAME.h"
```

Include dependency graph for FLA_UTV_AB_UT_var49h.h:



Functions

- FLA_Error FLA_Compute_QR_of_column_panel (FLA_Obj B, FLA_Obj W, FLA_Obj S)

  Algorithm-by-blocks. Factorizes column block B, starting from top to bottom, block by block.
- FLA_Error FLA_Apply_left_Qt_of_QR_of_column_panel (FLA_Obj B, FLA_Obj W, FLA_Obj S, FLA_Obj C)

  Algorithm-by-blocks. Updates C with transformations stored in B, W and S.
- FLA_Error FLA_Apply_left_Qt_of_dense_QR_UT_to_row_panel (FLA_Obj B, FLA_Obj S, FLA_Obj C)

  Algorithm-by-blocks. Updates C with transformations stored in B and S.
- FLA_Error FLA_Apply_left_Qt_of_td_QR_UT_to_row_panels (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G)

  Algorithm-by-blocks. Updates [F; G] with transformations stored in D and S from the left. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Apply_right_Q_of_QR_of_column_panel (FLA_Obj B, FLA_Obj W, FLA_Obj S, FLA_Obj C)

  Algorithm-by-blocks. Updates C with transformations stored in B, W, and S from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_to_column_panel (FLA_Obj B, FLA_Obj S, FLA_Obj C)

  Algorithm-by-blocks. Updates C with transformations stored in B and S from the right.
- FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_to_column_panels (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G)

  Algorithm-by-blocks. Updates [F; G] with transformations stored in D and S from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Keep_upper_triangular_part_of_column_panel (FLA_Obj B)

  Algorithm-by-blocks. Keeps upper triangular part of panel B. The rest is set to zero.
- FLA_Error FLA_Set_column_panel_to_zero (FLA_Obj B)

  Algorithm-by-blocks. Set column panel B to zero.
- FLA_Error FLA_Generate_normal_column_panel (FLA_Obj A)

  Algorithm-by-blocks. Sets column panel to a normal random matrix.
- FLA_Error MyFLA_Gemm_tn_mp_oz (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes GEMM (Transpose-No transpose, Matrix-Panel, One-Zero).
- FLA_Error MyFLA_Gemm_tn_pb_oz (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes C := A'∗B (Panel-Block).
- FLA_Error MyFLA_Gemm_tn_pb_oo (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes C := A'∗B + C (Panel-Block).
- FLA_Error MyFLA_Gemm_nn_mp_oz (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes C := A∗B + C (Matrix-Panel).

- FLA_Error MyFLA_Gemm_nn_pb_oo (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes C := A∗B + C (Panel-Block).
- FLA_Error MyFLA_Gemm_nn_pb_oz (FLA_Obj A, FLA_Obj B, FLA_Obj C)

  Algorithm-by-blocks. Computes C := A∗B (Panel-Block).
- FLA_Error MyFLA_Compute_svd (FLA_Obj U, FLA_Obj A, FLA_Obj VT)

  Algorithm-by-blocks. Enqueues task for computing the SVD of A.
- FLA_Error MyFLA_Gemm_abta (FLA_Obj A, FLA_Obj B)

  Algorithm-by-blocks. Computes A := B' ∗ A, where A is a row panel.
- FLA_Error MyFLA_Gemm_aabt (FLA_Obj A, FLA_Obj B)

  Algorithm-by-blocks. Computes A := A ∗ B', where A is a column panel.
- FLA_Error MyFLA_Gemm_aab (FLA_Obj A, FLA_Obj B)

  Algorithm-by-blocks. Computes A := A ∗ B, where A is a column panel.
- FLA_Error FLA_UTV_AB_UT_var49h (int m_A, int n_A, FLA_Obj A, int build_u, FLA_Obj U, int build_v, F↩ LA_Obj V, dim_t nb_alg, int n_iter, int nt)

  Main routine for the FLAME implementation of randUTV.

3.2.1 Detailed Description

This file (together with the companion FLA_UTV_AB_UT_var49h.c) contains the main routine for the algorithm-by-blocks for the randUTV factorization (FLA_UTV_AB_UT_var49h), together with the main building blocks that compute the factorization, as described in the companion manuscript.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)

Francisco D. Igual (figual@ucm.es)

Nathan Heavner (Nathan.Heavner@colorado.edu)

Per-Gunnar Martinsson (pgm@oden.utexas.edu)

Version

1.0

Date

2021-01-14

Copyright

3.2.2 Function Documentation

3.2.2.1 FLA_Apply_left_Qt_of_dense_QR_UT_to_row_panel()

```
FLA_Error FLA_Apply_left_Qt_of_dense_QR_UT_to_row_panel (
              FLA_Obj B,
              FLA_Obj S,
              FLA_Obj C )
```

Algorithm-by-blocks. Updates C with transformations stored in B and S.

Parameters

in	*B*	The Householder vectors of the triangular-dense QR factorizations.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.

Returns

FLA_Error

3.2.2.2 FLA_Apply_left_Qt_of_QR_of_column_panel()

```
FLA_Error FLA_Apply_left_Qt_of_QR_of_column_panel (
              FLA_Obj B,
              FLA_Obj W,
              FLA_Obj S,
              FLA_Obj C )
```

Algorithm-by-blocks. Updates C with transformations stored in B, W and S.

Parameters

in	*B*	The Householder vectors of the triangular-dense QR factorizations.
in	*W*	The strictly lower triangular part contains the Housholder vectors of the dense QR factorization.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.

Returns

FLA_Error

3.2.2.3 FLA_Apply_left_Qt_of_td_QR_UT_to_row_panels()

```
FLA_Error FLA_Apply_left_Qt_of_td_QR_UT_to_row_panels (
              FLA_Obj D,
              FLA_Obj S,
              FLA_Obj F,
              FLA_Obj G )
```

Algorithm-by-blocks. Updates [F; G] with transformations stored in D and S from the left. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.

Returns

FLA_Error

3.2.2.4 FLA_Apply_right_Q_of_dense_QR_UT_to_column_panel()

```
FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_to_column_panel (
            FLA_Obj B,
            FLA_Obj S,
            FLA_Obj C )
```

Algorithm-by-blocks. Updates C with transformations stored in B and S from the right.

Parameters

in	*B*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.

Returns

FLA_Error

3.2.2.5 FLA_Apply_right_Q_of_dense_QR_UT_to_column_panels()

```
FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_to_column_panels (
            FLA_Obj D,
            FLA_Obj S,
            FLA_Obj F,
            FLA_Obj G )
```

Algorithm-by-blocks. Updates [F; G] with transformations stored in D and S from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.

Returns

FLA_Error

3.2.2.6 FLA_Apply_right_Q_of_QR_of_column_panel()

```
FLA_Error FLA_Apply_right_Q_of_QR_of_column_panel (
            FLA_Obj B,
            FLA_Obj W,
            FLA_Obj S,
            FLA_Obj C )
```

Algorithm-by-blocks. Updates C with transformations stored in B, W, and S from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.

Parameters

in	*B*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*W*	The strictly lower triangular part contains the Housholder vectors of the dense QR factorization.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.

Returns

FLA_Error

3.2.2.7 FLA_Compute_QR_of_column_panel()

```
FLA_Error FLA_Compute_QR_of_column_panel (
            FLA_Obj B,
            FLA_Obj W,
            FLA_Obj S )
```

Algorithm-by-blocks. Factorizes column block B, starting from top to bottom, block by block.

Parameters

in,out	*B*	Matrix to be factorized. On output, it contains the factor R in the upper triangular part, and Householder vectors in the strictly lower triangular part.
out	*W*	On output, a copy of the Householder vectors in the strictly lower triangular part to be used in future update tasks in order to avoid unnecessary dependencies with B.
out	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.

Returns

FLA_Error

3.2.2.8 FLA_Generate_normal_column_panel()

```
FLA_Error FLA_Generate_normal_column_panel (
            FLA_Obj A )
```

Algorithm-by-blocks. Sets column panel to a normal random matrix.

Parameters

in,out	*A*	Input column panel.

Returns

FLA_Error

3.2.2.9 FLA_Keep_upper_triangular_part_of_column_panel()

```
FLA_Error FLA_Keep_upper_triangular_part_of_column_panel (
              FLA_Obj B )
```

Algorithm-by-blocks. Keeps upper triangular part of panel B. The rest is set to zero.

Parameters

in,out	*B*	Panel to process.

Returns

FLA_Error

3.2.2.10 FLA_Set_column_panel_to_zero()

```
FLA_Error FLA_Set_column_panel_to_zero (
              FLA_Obj B )
```

Algorithm-by-blocks. Set column panel B to zero.

Parameters

out	*B*	Column panel to set to zero.

Returns

FLA_Error

3.2.2.11 FLA_UTV_AB_UT_var49h()

```
FLA_Error FLA_UTV_AB_UT_var49h (
            int m_A,
            int n_A,
            FLA_Obj A,
            int build_u,
            FLA_Obj U,
            int build_v,
            FLA_Obj V,
            dim_t nb_alg,
            int n_iter,
            int nt )
```

Main routine for the FLAME implementation of randUTV.

Parameters

in	*m_A*	Number of rows of input matrix A.
in	*n_A*	Number of columns of input matrix A.
in,out	*A*	Input matrix of dimensions m_A x n_A
in	*build_u*	Boolean to activate/deactivate the computation of U.
out	*U*	Orthonormal matrix U of dimensions m_A x m_A.
in	*build_v*	Boolean to activate/deactivate the computation of V.
out	*V*	Orthonormal matrix V of dimensions n_A x n_A.
in	*nb_alg*	Algorithmic block size.
in	*n_iter*	Number of power iterations (q).
in	*nt*	Number of threads.

Returns

FLA_Error

3.2.2.12 MyFLA_Compute_svd()

```
FLA_Error MyFLA_Compute_svd (
            FLA_Obj U,
            FLA_Obj A,
            FLA_Obj VT )
```

Algorithm-by-blocks. Enqueues task for computing the SVD of A.

Parameters

out	*U*	Ortogonal matrix U.
in,out	*A*	Input matrix. On output, return singular values in the matrix diagonal.
out	*VT*	Orthogonal matrix V.

Returns

FLA_Error

3.2.2.13 MyFLA_Gemm_aab()

```
FLA_Error MyFLA_Gemm_aab (
            FLA_Obj A,
            FLA_Obj B )
```

Algorithm-by-blocks. Computes A := A ∗ B, where A is a column panel.

Parameters

in,out	*A*	Input/output matrix. First operand.
in	*B*	Input matrix. Second operand.

Returns

FLA_Error

3.2.2.14 MyFLA_Gemm_aabt()

```
FLA_Error MyFLA_Gemm_aabt (
            FLA_Obj A,
            FLA_Obj B )
```

Algorithm-by-blocks. Computes A := A ∗ B', where A is a column panel.

Parameters

in,out	*A*	Input/output matrix. First operand.
in	*B*	Input matrix. Second operand.

Returns

FLA_Error

3.2.2.15 MyFLA_Gemm_abta()

```
FLA_Error MyFLA_Gemm_abta (
            FLA_Obj A,
            FLA_Obj B )
```

Algorithm-by-blocks. Computes A := B' ∗ A, where A is a row panel.

Parameters

in,out	*A*	Input/output matrix. Second operand.
in	*B*	Input matrix. First operand.

Returns

FLA_Error

3.2.2.16 MyFLA_Gemm_nn_mp_oz()

```
FLA_Error MyFLA_Gemm_nn_mp_oz (
              FLA_Obj A,
              FLA_Obj B,
              FLA_Obj C )
```

Algorithm-by-blocks. Computes C := A∗B + C (Matrix-Panel).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

3.2.2.17 MyFLA_Gemm_nn_pb_oo()

```
FLA_Error MyFLA_Gemm_nn_pb_oo (
              FLA_Obj A,
              FLA_Obj B,
              FLA_Obj C )
```

Algorithm-by-blocks. Computes C := A∗B + C (Panel-Block).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

3.2.2.18 MyFLA_Gemm_nn_pb_oz()

```
FLA_Error MyFLA_Gemm_nn_pb_oz (
            FLA_Obj A,
            FLA_Obj B,
            FLA_Obj C )
```

Algorithm-by-blocks. Computes C := A∗B (Panel-Block).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

3.2.2.19 MyFLA_Gemm_tn_mp_oz()

```
FLA_Error MyFLA_Gemm_tn_mp_oz (
            FLA_Obj A,
            FLA_Obj B,
            FLA_Obj C )
```

Algorithm-by-blocks. Computes GEMM (Transpose-No transpose, Matrix-Panel, One-Zero).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

3.2.2.20 MyFLA_Gemm_tn_pb_oo()

```
FLA_Error MyFLA_Gemm_tn_pb_oo (
            FLA_Obj A,
            FLA_Obj B,
            FLA_Obj C )
```

Algorithm-by-blocks. Computes C := A'∗B + C (Panel-Block).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

3.2.2.21 MyFLA_Gemm_tn_pb_oz()

```
FLA_Error MyFLA_Gemm_tn_pb_oz (
            FLA_Obj A,
            FLA_Obj B,
            FLA_Obj C )
```

Algorithm-by-blocks. Computes C := A'∗B (Panel-Block).

Parameters

in	*A*	Input matrix A.
in	*B*	Input matrix B.
in,out	*C*	Input/output matrix C.

Returns

FLA_Error

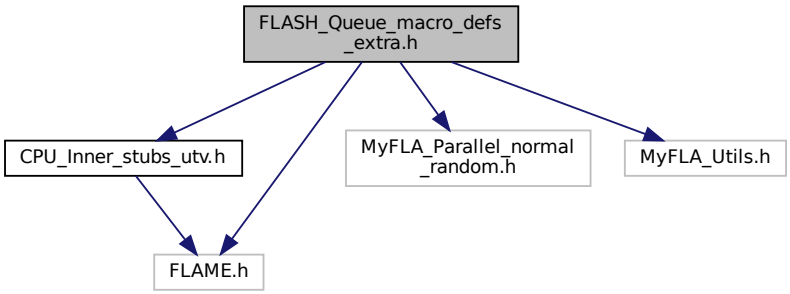
3.3 FLASH_Queue_macro_defs_extra.h File Reference

This file (provided as an extension of libflame) contains the enqueuing macros that add specific tasks to the libflame's SuperMatrix out-of-order execution queue.

```
#include "CPU_Inner_stubs_utv.h"
#include "MyFLA_Parallel_normal_random.h"
```

```
#include "MyFLA_Utils.h"
```

Include dependency graph for FLASH_Queue_macro_defs_extra.h:



Macros

- #define FLASH_OBJ_PTR_ID(A) (A).base->id
- #define ENQUEUE_FLASH_LU_piv_macro(A, p, cntl)
- #define ENQUEUE_FLASH_COMP_DENSE_QR_UT(B, T)
- #define ENQUEUE_FLASH_COMP_TD_QR_UT(U, D, S)
- #define ENQUEUE_FLASH_KEEP_UPPER_TRIANG(A)
- #define ENQUEUE_FLASH_SET_TO_ZERO(A)
- #define ENQUEUE_FLASH_APPLY_LEFT_QT_OF_DENSE_QR(U, S, C)
- #define ENQUEUE_FLASH_APPLY_LEFT_QT_OF_TD_QR(D, S, F, G)
- #define ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_DENSE_QR_UT(U, S, C)
- #define ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_TD_QR_UT(D, S, F, G)
- #define ENQUEUE_FLASH_NORMAL_RANDOM_MATRIX(A)
- #define ENQUEUE_SVD_OF_BLOCK(U, A, VT)
- #define ENQUEUE_FLASH_GEMM_ABTA(A, B)
- #define ENQUEUE_FLASH_GEMM_NN_OZ(C, A, B)
- #define ENQUEUE_FLASH_GEMM_AABT(A, B)
- #define ENQUEUE_FLASH_GEMM_AAB(A, B)
- #define ENQUEUE_FLASH_MYCOPY(A, B)

3.3.1 Detailed Description

This file (provided as an extension of libflame) contains the enqueuing macros that add specific tasks to the libflame's SuperMatrix out-of-order execution queue.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)

Francisco D. Igual (figual@ucm.es)

Nathan Heavner

Per-Gunnar Martinsson

Version

0.1

Date

2021-01-14

Copyright

3.3.2 Macro Definition Documentation

3.3.2.1 ENQUEUE_FLASH_APPLY_LEFT_QT_OF_DENSE_QR

```
#define ENQUEUE_FLASH_APPLY_LEFT_QT_OF_DENSE_QR(
                U,
                S,
                C )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Apply_left_Qt_of_dense_QR_UT_inner_utv, \
        NULL, \
        "Apply_left_qt_of_dense_qr", \
        FALSE, \
        0, 0, 2, 1, \
    U, S, C )
```

3.3.2.2 ENQUEUE_FLASH_APPLY_LEFT_QT_OF_TD_QR

```
#define ENQUEUE_FLASH_APPLY_LEFT_QT_OF_TD_QR(
                D,
                S,
                F,
                G )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Apply_left_Qt_of_td_QR_UT_inner_utv, \
        NULL, \
        "Apply_left_qt_of_td_qr", \
        FALSE, \
        0, 0, 2, 2, \
    D, S, F, G )
```

3.3.2.3 ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_DENSE_QR_UT

```
#define ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_DENSE_QR_UT(
                U,
                S,
                C )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Apply_right_Q_of_dense_QR_UT_inner_utv, \
        NULL, \
        "Apply_right_q_of_dense_qr_ut", \
        FALSE, \
        0, 0, 2, 1, \
    U, S, C )
```

3.3.2.4 ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_TD_QR_UT

```
#define ENQUEUE_FLASH_APPLY_RIGHT_Q_OF_TD_QR_UT(
                D,
                S,
                F,
                G )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Apply_right_Q_of_td_QR_UT_inner_utv, \
        NULL, \
        "Apply_right_q_of_td_qr_ut", \
        FALSE, \
        0, 0, 2, 2, \
    D, S, F, G )
```

3.3.2.5 ENQUEUE_FLASH_COMP_DENSE_QR_UT

```
#define ENQUEUE_FLASH_COMP_DENSE_QR_UT(
                B,
                T )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Compute_dense_QR_UT_inner_utv, \
        NULL, \
        "Compute_dense_QR_UT", \
        FALSE, \
        0, 0, 0, 2, \
    B, T )
```

3.3.2.6 ENQUEUE_FLASH_COMP_TD_QR_UT

```
#define ENQUEUE_FLASH_COMP_TD_QR_UT(
                U,
                D,
                S )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Compute_td_QR_UT_inner_utv, \
        NULL, \
        "Compute_td_QR", \
        FALSE, \
        0, 0, 0, 3, \
    U, D, S )
```

3.3.2.7 ENQUEUE_FLASH_GEMM_AAB

```
#define ENQUEUE_FLASH_GEMM_AAB(
                A,
                B )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Gemm_aab_inner_utv, \
        NULL, \
        " GEMM_aab", \
        FALSE, \
        0, 0, 1, 1, \
    A, B )
```

3.3.2.8 ENQUEUE_FLASH_GEMM_AABT

```
#define ENQUEUE_FLASH_GEMM_AABT(
                A,
                B )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Gemm_aabt_inner_utv, \
        NULL, \
        "GEMM_aabt", \
        FALSE, \
        0, 0, 1, 1, \
    A, B )
```

3.3.2.9 ENQUEUE_FLASH_GEMM_ABTA

```
#define ENQUEUE_FLASH_GEMM_ABTA(
                A,
                B )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Gemm_abta_inner_utv, \
        NULL, \
        "GEMM_abta", \
        FALSE, \
        0, 0, 1, 1, \
    A, B )
```

3.3.2.10 ENQUEUE_FLASH_GEMM_NN_OZ

```
#define ENQUEUE_FLASH_GEMM_NN_OZ(
                C,
                A,
                B )
```

Value:

```
        FLASH_Queue_push( (void*)FLA_GEMM, \
        NULL, \
        "GEMM_NN_OZ", \
        FALSE, \
        0, 0, 1, 2, \
    C, A, B )
```

3.3.2.11 ENQUEUE_FLASH_KEEP_UPPER_TRIANG

```
#define ENQUEUE_FLASH_KEEP_UPPER_TRIANG(
              A )
```

Value:

```
       FLASH_Queue_push( (void*)CPU_Keep_upper_triang_inner_utv, \
       NULL, \
       "Keep_upper_triang", \
       FALSE, \
       0, 0, 0, 1, \
   A )
```

3.3.2.12 ENQUEUE_FLASH_LU_piv_macro

```
#define ENQUEUE_FLASH_LU_piv_macro(
              A,
              p,
              cntl )
```

Value:

```
       FLASH_Queue_push( (void *) FLA_LU_piv_macro_task, \
                         (void *) cntl, \
                         "LU   ", \
                         FALSE, \
                         0, 0, 0, 2, \
                         A, p )
```

3.3.2.13 ENQUEUE_FLASH_MYCOPY

```
#define ENQUEUE_FLASH_MYCOPY(
              A,
              B )
```

Value:

```
       FLASH_Queue_push( (void *) CPU_Mycopy_inner_utv, \
                         NULL, \
                         "MYCOPY   ", \
                         FALSE, \
                         0, 0, 1, 1, \
                         A, B  )
```

3.3.2.14 ENQUEUE_FLASH_NORMAL_RANDOM_MATRIX

```
#define ENQUEUE_FLASH_NORMAL_RANDOM_MATRIX(
              A )
```

Value:

```
       FLASH_Queue_push( (void*)MyFLA_Normal_random_matrix, \
       NULL, \
       "Normal_random_matrix", \
       FALSE, \
       0, 0, 0, 1, \
   A )
```

3.3.2.15 ENQUEUE_FLASH_SET_TO_ZERO

```
#define ENQUEUE_FLASH_SET_TO_ZERO(
              A )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Set_to_zero_inner_utv, \
        NULL, \
        "Set_to_zero", \
        FALSE, \
        0, 0, 0, 1, \
    A )
```

3.3.2.16 ENQUEUE_SVD_OF_BLOCK

```
#define ENQUEUE_SVD_OF_BLOCK(
              U,
              A,
              VT )
```

Value:

```
        FLASH_Queue_push( (void*)CPU_Compute_svd_inner_utv, \
        NULL, \
        "SVD_of_block", \
        FALSE, \
        0, 0, 0, 3, \
    U, A, VT )
```

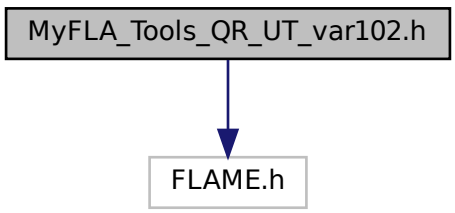
3.3.2.17 FLASH_OBJ_PTR_ID

```
#define FLASH_OBJ_PTR_ID(
              A ) ( A ).base->id
```

3.4 MyFLA_Tools_QR_UT_var102.h File Reference

This file contains the inner kernel implementations to support randUTV. These routines are executed out-of-order, following a dataflow execution path.

`#include "FLAME.h"`
Include dependency graph for MyFLA_Tools_QR_UT_var102.h:



Functions

- FLA_Error FLA_Compute_dense_QR_UT_var102 (FLA_Obj A, FLA_Obj t, FLA_Obj S, int nb_alg)

  Inner kernel implementation. Computes QR of dense matrix A.
- FLA_Error FLA_Apply_left_Qt_of_dense_QR_UT_var102 (FLA_Obj U, FLA_Obj S, FLA_Obj C, int nb_alg)

  Inner kernel implementation. Applies block Householder transformations stored in U and S to matrix C from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_var102 (FLA_Obj U, FLA_Obj S, FLA_Obj C, int nb_alg)

  Inner kernel implementation. Applies block Householder transformations stored in U and S to matrix C from the right. The orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Compute_td_QR_UT_var102 (FLA_Obj U, FLA_Obj D, FLA_Obj t, FLA_Obj S, int nb_alg)

  Inner kernel implementation. Compute QR factorization of [U; D], where U is upper triangular and D is dense.
- FLA_Error FLA_Apply_left_Qt_of_td_QR_UT_var102 (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G, int nb_alg)

  Inner kernel implementation. Applies block Householder transformations stored in D and S to matrices F and G from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.
- FLA_Error FLA_Apply_right_Q_of_td_QR_UT_var102 (FLA_Obj D, FLA_Obj S, FLA_Obj F, FLA_Obj G, int nb_alg)

  Inner kernel implementation. Applies block Householder transformations stored in D and S to matrices F and G from right side. The orthonormal matrix obtained in the previous factorization is applied without building it.

3.4.1 Detailed Description

This file contains the inner kernel implementations to support randUTV. These routines are executed out-of-order, following a dataflow execution path.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)

Francisco D. Igual (figual@ucm.es)

Nathan Heavner (Nathan.Heavner@colorado.edu)

Per-Gunnar Martinsson (pgm@oden.utexas.edu)

Version

1.0

Date

2021-01-14

Copyright

3.4.2 Function Documentation

3.4.2.1 FLA_Apply_left_Qt_of_dense_QR_UT_var102()

```
FLA_Error FLA_Apply_left_Qt_of_dense_QR_UT_var102 (
              FLA_Obj U,
              FLA_Obj S,
              FLA_Obj C,
              int nb_alg )
```

Inner kernel implementation. Applies block Householder transformations stored in U and S to matrix C from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

C := (I - U_i * S_i * U_i') * C.

Parameters

in	*U*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.4.2.2 FLA_Apply_left_Qt_of_td_QR_UT_var102()

```
FLA_Error FLA_Apply_left_Qt_of_td_QR_UT_var102 (
              FLA_Obj D,
              FLA_Obj S,
              FLA_Obj F,
              FLA_Obj G,
              int nb_alg )
```

Inner kernel implementation. Applies block Householder transformations stored in D and S to matrices F and G from left side. The transpose of the orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

[F; G] := (I - [I; D_i] * S_i * [I; D_i]') * [F; G].

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.4.2.3 FLA_Apply_right_Q_of_dense_QR_UT_var102()

```
FLA_Error FLA_Apply_right_Q_of_dense_QR_UT_var102 (
              FLA_Obj U,
              FLA_Obj S,
              FLA_Obj C,
              int nb_alg )
```

Inner kernel implementation. Applies block Householder transformations stored in U and S to matrix C from the right.
The orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

C := C ∗ (I - U_i ∗ S_i' ∗ U_i')

Parameters

in	*U*	Matrix with Householder vectors stored in the strictly lower triangular part.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*C*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error.

3.4.2.4 FLA_Apply_right_Q_of_td_QR_UT_var102()

```
FLA_Error FLA_Apply_right_Q_of_td_QR_UT_var102 (
              FLA_Obj D,
              FLA_Obj S,
              FLA_Obj F,
              FLA_Obj G,
              int nb_alg )
```

Inner kernel implementation. Applies block Householder transformations stored in D and S to matrices F and G from right side. The orthonormal matrix obtained in the previous factorization is applied without building it.

for i = 1, n_U/nb_alg

[F G] := [F G] ∗ (I - [I; D_i] ∗ S_i ∗ [I; D_i]').

Parameters

in	*D*	Matrix with Householder vectors.
in	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in,out	*F*	Matrix to be updated.
in,out	*G*	Matrix to be updated.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.4.2.5 FLA_Compute_dense_QR_UT_var102()

```
FLA_Error FLA_Compute_dense_QR_UT_var102 (
            FLA_Obj A,
            FLA_Obj t,
            FLA_Obj S,
            int nb_alg )
```

Inner kernel implementation. Computes QR of dense matrix A.

Parameters

in,out	*A*	Matrix to be factorized. On output, it contains the factor R in the upper triangular part, and Householder vectors in the strictly lower triangular part.
out	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

3.4.2.6 FLA_Compute_td_QR_UT_var102()

```
FLA_Error FLA_Compute_td_QR_UT_var102 (
            FLA_Obj U,
            FLA_Obj D,
            FLA_Obj t,
            FLA_Obj S,
            int nb_alg )
```

Inner kernel implementation. Compute QR factorization of [U; D], where U is upper triangular and D is dense.

Parameters

in,out	*U*	Upper triangular matrix. On output, it contains the factor R in the upper triangular part. The strictly lower triangular part is not accessed nor modified.
in,out	*D*	Dense matrix. On output, it contains Householder vectors.
out	*S*	Matrix containing a series of upper triangular factors of Householder block reflectors.
in	*nb_alg*	Algorithmic block size.

Returns

FLA_Error

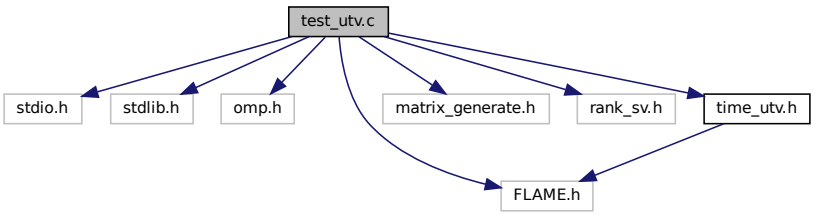
3.5 README.md File Reference

3.6 test_utv.c File Reference

This file contains the entry point to the test driver for the randUTV implementation.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include "FLAME.h"
#include "matrix_generate.h"
#include "rank_sv.h"
#include "time_utv.h"
```

Include dependency graph for test_utv.c:



Macros

- #define fla_utv_var49h_q0_no
- #define fla_utv_var49h_q1_no
- #define fla_utv_var49h_q2_no
- #define fla_utv_var49h_q0_or
- #define fla_utv_var49h_q1_or
- #define fla_utv_var49h_q2_or

Functions

- int main (int argc, char ∗argv[ ])

3.6.1 Detailed Description

This file contains the entry point to the test driver for the randUTV implementation.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)
Francisco D. Igual (figual@ucm.es)
Nathan Heavner (Nathan.Heavner@colorado.edu)
Per-Gunnar Martinsson (pgm@oden.utexas.edu)

Version

0.1

Date

2021-01-14

Copyright

3.6.2 Macro Definition Documentation

3.6.2.1 fla_utv_var49h_q0_no

```
#define fla_utv_var49h_q0_no
```

3.6.2.2 fla_utv_var49h_q0_or

```
#define fla_utv_var49h_q0_or
```

3.6.2.3 fla_utv_var49h_q1_no

```
#define fla_utv_var49h_q1_no
```

3.6.2.4 fla_utv_var49h_q1_or

```
#define fla_utv_var49h_q1_or
```

3.6.2.5 fla_utv_var49h_q2_no

```
#define fla_utv_var49h_q2_no
```

3.6.2.6 fla_utv_var49h_q2_or

```
#define fla_utv_var49h_q2_or
```

3.6.3 Function Documentation

3.6.3.1 main()

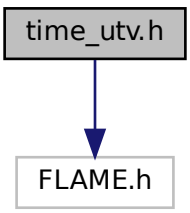
```
int main (
             int argc,
             char * argv[] )
```

3.7 time_utv.h File Reference

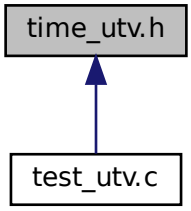
This file (together with the companion file time_utv.c) contains the main timing routine for the randUTV implementation.

```
#include "FLAME.h"
```

Include dependency graph for time_utv.h:



This graph shows which files directly or indirectly include this file:



Functions

- void time_utv (int variant, int num_threads, dim_t nb, int num_execs, int print_data, int check_result, FLA_Obj A, FLA_Obj Acopy, FLA_Obj sv, int build_ort_matrices, int q, double *dtime, double *gflops, double *res)

  Timing routine for randUTV.

3.7.1 Detailed Description

This file (together with the companion file time_utv.c) contains the main timing routine for the randUTV implementation.

Author

Gregorio Quintana-Ortí (gquintan@uji.es)

Francisco D. Igual (figual@ucm.es)

Nathan Heavner (Nathan.Heavner@colorado.edu)

Per-Gunnar Martinsson (pgm@oden.utexas.edu)

Version

0.1

Date

2021-01-14

Copyright

3.7.2 Function Documentation

3.7.2.1 time_utv()

```
void time_utv (
            int variant,
            int num_threads,
            dim_t nb,
            int num_execs,
            int print_data,
            int check_result,
            FLA_Obj A,
            FLA_Obj Acopy,
            FLA_Obj sv,
            int build_ort_matrices,
            int q,
            double * dtime,
            double * gflops,
            double * res )
```

Timing routine for randUTV.

Parameters

in	*variant*	Variant to test. In this driver, only variant 5049 is supported.
in	*num_threads*	Number of threads to evaluate.
in	*nb*	Algorithmic block size.
in	*num_execs*	Number of executions.
in	*print_data*	Print matrix data (only recommended for small matrices).
in	*check_result*	Activates result checking.
in	*A*	Input matrix.
in	*Acopy*	Space for a copy of A (useful for error checking).
in	*sv*	Vector to store singular values.
in	*build_ort_matrices*	Activation of the computation of orthonormal matrices.
in	*q*	Number of power iterations.
out	*dtime*	Measured execution time.
out	*gflops*	Measured GFLOPS.
out	*res*	Residual.