MIDAPACK - MIcrowave Data Analysis PACKage

1.1b

Generated by Doxygen 1.8.1.1

Sun Dec 2 2018 15:55:32

Contents

Chapter 1

General introduction

The goal of the **MIDAS** project is to provide high performance, middle-layer software tools, which would aid CMB data analysis efforts of current and planned CMB experiments to capitalize on the computational power of parallel (super)computers. The functionality provided by the library is supposed to fill in the gap in between available, low-level, high performance software packages such as Fast Fourier Transforms, dense and sparse linear algebra operations, etc, and the high-level data analysis pipelines, and thus to help the users to benefit from the former, while developing the latter in a more straightforward and transparent way. At the end of the project the library is supposed to provide functionality relevant to all main stages of the data analysis.

For more information about ANR MIDAS'09 project, and to find out how to contact us, see:

`http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html`

The first released installements of the library are

- Toeplitz algebra documentation
- Pointing operations documentation

which are described in this document.

Check here our funding sources and licensing info.

Chapter 2

Pointing operations documentation

2.1 Functionnality

2.1.1 Sparse matrix operations

Pointing and Unpointing are usefull operators in the CMB data analysis. It refers to the applications from time-signal-domain to sky-direction domain. It usually consist in recangular sparse matrices with few non-zero values. It is assumed that unpointing matrix has couples time more rows than columns. That means dimension of the time-signal domain(tod) is greater than the sky-direction domain(map). Pointing matrix is exactly the transposed of the unpointing matrix.

This module contains functionnalities for applying these operations. Which means creating sparse matrices into an efficient sparse matrix format, and also applying matrix vector multiplication or transposed matrix vector multiplication.

The two core functions provided are :

- unpointing product (matrix vector multiplication)
- pointing product (transposed matrix-vector multiplication);

2.1.2 Parallel execution

As CMB data analysis works with large data sets, it require the aibility to execute programs in parallel. Matrices can reach hundred billions of rows, hundred million of columns and be distributed over thousand cores. The aim is to deliver fast and highly scalable operation for sparse rectangular matrices. That's why midapack adopts a customized storage format and several communication schemes for pointing operators.

MIDAPACK parallel execution model is based on distributed memory architecture via the Message Passing Interface(MPI). Most of effort have been done for minmizing communication between processors. Especially we have developed algorithms for collective reduce operations. Moreover, most of the functions can also benefit from a sublevel parallellism using OpenMP. Thus programs build with MIDAPACK can works on big data sets and be runned on big computer in a multi-many-cores context.

2.2 Data structure and parallelism

2.2.1 Parallel sparse matrix

Considering a matrix A, parallelism assume A is row-distributed over processes. Each processor has into memory m rows of the global matrix. Reciprocally A^t is column-distributed, with m columns into memory. That is to say

$$A = \begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_{n_{prc}-1} \end{pmatrix}$$

Reciprocally

$$A^t = (A_0^t, A_1^t, A_2^t, ...A_{n_{prc}-1}^t)$$

.

As A is a sparse martix, it doesn't store zero values. Furthermore we assume A is exactly nnz nonzerovalues. Then building matrix A only require these non-zero values and theirs global columns indices, also called ELL format. Input data consists in two large tab of size m$*$nnz, where rows are concatenated. This input array have to be passed when calling matrix initializtaion function.

To well ballence the load over processes we have to ensure number of rows time number of non-zero per row is roughly the same on each processor

2.2.2 Input data

The two following examples illustrate the input data needs to build a matrix using MatInit. The first one is a sequential, the second consider 2 processors.

- sequential case : m=8, nnz=2, indices=[0 1 2 4 0 2 0 2 2 3 3 4 1 4 1 3], values=[1 7 2 8 5 3 5 6 2 9 8 6 1 3 6 4].

$$A = \begin{pmatrix} 1 & 7 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 8 \\ 5 & 0 & 3 & 0 & 0 \\ 5 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 9 & 0 \\ 0 & 0 & 0 & 8 & 6 \\ 0 & 1 & 0 & 0 & 3 \\ 0 & 6 & 0 & 4 & 0 \end{pmatrix}$$

- parallel case over 2 processors : input data on processor 0 is m=3, nnz=2, indices=[0 1 2 4 0 2 0 2], values=[1 7 2 8 5 3 5 6]. Input data on processor 1 is m=4, nnz=2, indices=[2 3 3 4 1 4 1 3], values=[2 9 8 6 1 3 6 4].

$$A = \begin{pmatrix} A_0 \\ A_1 \end{pmatrix}$$

$$A_0 = \begin{pmatrix} 1 & 7 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 8 \\ 5 & 0 & 3 & 0 & 0 \\ 5 & 0 & 2 & 0 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} 0 & 0 & 2 & 9 & 0 \\ 0 & 0 & 0 & 8 & 6 \\ 0 & 1 & 0 & 0 & 3 \\ 0 & 6 & 0 & 4 & 0 \end{pmatrix}$$

Two remarks about the input data structure (ELL format) :

- it happens that a row has more or less non-zero values that nnz. In this case we can choose the greater nnz and add zero wherever it is necessary with whatever index. For performance we advice to choose an index which has already a value in the row.
- ELL format is more general than DIA format since non-zero elements of given row do not need to be ordered. Thus permuting non-zero elements of given row in the input data do not change the matrix.

2.2.3 Internal data stucture

The internal structure is more sophisticated than the ELL format. Especially, to enhance matrix operations performance, a precomputation step reshapes the data structure into several arrays : global ordered columns indices, local indices, communication ...

When using MatInit function, precomputation is performed blindly. Nevertheless, for advanced user it is able to initialize a matrix in several steps. This enables to specify differents methods for the precomputations.

- set non-zero elements indices and values (MatSetIndices MatSetValues),
- reindex local matrix (MatLocalShape),
- create communication scheme (MatComShape).

2.3 Communication Algorithm

Transposed matrix vector multiplication is performed in two steps :

- Firstly, each processor i multiply a local vector by a local matrix, $x_i = A_i^t y_i$.
- Then processors communicated to update the local result vector, $x = \sum_{i=0}^{n_{prc}-1} x_i$

The second steps involved to communicate and sum elements of all the each local vectors. When size of the problem or number processors increases, this operation may become a bottleneck. To minimize the computationnal cost of this collective reduce operation, Midapack identifies the minimum parts of elements to communicate between processors. Once it is done, collective communication are executed using one of the custommized algorithms as Ring, Butterfly, Nonblocking, Noempty

The communication algorithm is specified when calling MatInit or MatComShape . An integer encodes all the communication algorithms (None=0, Ring=1, Butterfly=2, Nonblocking=3 Noempty=4).

2.4 Application example

Here is an application example of a least square problem resolution which is implemented in test_pcg_mapmat.c . Considering a problem formulated as $A^tAx = A^tb$, Solution x can be compute iteratively using conjutgate gradient. Instead computing and storing the whole A^tA matrix, we apply succesively pointing, A^t, and unpointing products, A, at each iterate.

Classic gradient conjugate algorithm has been slightly modified. As we explain A^t and A are applied succesively. Furtherwise dotproduct operations in the overlapped domain have been moved in the distributed domain. Espacially we use relation : $< A^t y, x >=< y, Ax >$.

Algorithm needs into memory 6 vectors :

- 3 in ovelapped domain(x, gradient, direction),
- 3 in distributed domain.

Complexity, counting operations at each iterate, is detailled as follow :

- 4 produits scalaires dans le domaine temporelle (communication = single MPI_Allreduce),
- 3 axpy dans le domaine de la carte (no communication),
- 3 multiplication par A (no communication),
- 1 multiplication par A^t (communication = MIDAPACK Communication scheme).

```
Mat A;
double *x, *g, *d;
double *Ax_b, *Ag, *Ad;

MatCreate(&A, m, nnz, MPI_COMM_WORLD);       //allocate matrix tabs
MatSetIndices(&A, m*nnz, 0, nnz, indices);  //copy indices
   into matrix structure
MatSetValues(&A, m*nnz, 0, nnz, values);     //copy values into
   matrix structure

MatLocalShape(&A, SORT_FLAG, OMP_FLAG);      //reindex data
   structure

MatComShape(&A, COM_SCHEME_FLAG);            //build
   communication pattern

//conjugate gradient initialization
//allocate vectors (overlapped domain)
g  = (double *) malloc(A.lcount*sizeof(double));       //g (gradient)
d  = (double *) malloc(A.lcount*sizeof(double));       //d (direction)

//allocate vector (distributed domain)
Ax_b = (double *) malloc(m*sizeof(double));            //Ax_b = Ax-b
Ad = (double *) malloc(m*sizeof(double));              //Ad = A d
Ag = (double *) malloc(m*sizeof(double));              //Ag = A g

MatVecProd(&A, x, Ax_b, 0);            //Ax_b = Ax-b
for(i=0; i<m; i++)                     //
  Ax_b[i] = Ax_b[i]-b[i];              //

TrMatVecProd(&A, Ax_b, d, 0);          //Ad = A d =  A A^t(Ax-b)
MatVecProd(&A, d, Ad, 0);              //

resnew=0.0;                            //initial residu, resnew =
   ||A^t(Ax-b)|| = <Ax_b, Ad>
localreduce=0.0;                       //
for(i=0; i<m; i++)                     //
  localreduce+=Ax_b[i]*Ad[i];          //
MPI_Allreduce(&localreduce, &resnew, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD
  );

//conjugate gradient iterate
for(k=0; k<KMAX ; k++){                //begin loop

  alpha=0.0;                           //alpha = <Ad, Ax_b>
  localreduce=0.0;                     //
  for(i=0; i<m; i++)                   //
    localreduce+=Ad[i]*Ax_b[i];        //
  MPI_Allreduce(&localreduce, &alpha, 1, MPI_DOUBLE, MPI_SUM,
  MPI_COMM_WORLD);

  gamma=0.0;                           //gamma = <Ad, Ad>
  localreduce=0.0;                     //
  for(i=0; i<m; i++)                   //
    localreduce+=Ad[i]*Ad[i];          //
  MPI_Allreduce(&localreduce, &gamma, 1, MPI_DOUBLE, MPI_SUM,
  MPI_COMM_WORLD);

  for(j=0; j<A.lcount; j++)            // x = x + (alpha/gamma) d
    x[j] = x[j] - (alpha/gamma)* d[j];//

  MatVecProd(&A, x, Ax_b, 0);          //Ax_b = Ax-b
  for(i=0; i<m; i++)                   //
    Ax_b[i] = Ax_b[i]-b[i];            //

  TrMatVecProd(&A, Ax_b, g, 0);        //g  = A^t(Ax-b)
  MatVecProd(&A, g, Ag, 0);            //Ag = AA^t(Ax-b)

  resold=resnew;                       //residu = ||g|| = <Ax-b, Ag>
  resnew=0.0;                          //
  localreduce=0.0;                     //
  for(i=0; i<m; i++)                   //
    localreduce+=Ax_b[i]*Ag[i];        //
  MPI_Allreduce(&localreduce, &resnew, 1, MPI_DOUBLE, MPI_SUM,
  MPI_COMM_WORLD);

  beta=0.0;                            //beta = <Ag, Ad>
  localreduce=0.0;                     //
  for(i=0; i<m; i++)                   //
    localreduce+=Ag[i]*Ad[i];          //
  MPI_Allreduce(&localreduce, &beta, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD
  );

  if(resnew<tol)                        //convergence test
     break;

  for(j=0; j<A.lcount; j++)            //d = -g + (beta/gamma) d
```

```
      d[j]= -g[j] + (beta/gamma)*d[j];  //

    MatVecProd(&A, d, Ad, 0);           //Ad = A d
  }
```

More information about pointing operator are detailled, in the pointing function synposis

Chapter 3

MIDAPACK development team

- Pierre Cargemel (developer);
- Frédéric Dauvergne (developer);
- Giulio Fabbian (validator);
- Laura Grigori (coordinator);
- Maude Le Jeune (senior developer);
- Antoine Rogier (developer - till Aug 31, 2011);
- Mikolaj Szydlarski (developer);
- Radek Stompor (coordinator).

Chapter 4

Acknowledgement

This work has been supported in part by French National Research Agency (ANR) through its COSINUS program (project MIDAS no. ANR-09-COSI-009).

High performance computing resources have been provided:

- in France by CCRT, TGCC, and IDRIS supercomputing centers under the GENCI program through projects: 2011-066647 and 2012-066647;
- in the US by the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Chapter 5

Licences

Chapter 6

Installation

Once you have downloaded Midapack_1.1.tgz, create a directory where you want to install it, for example midapackdir. Then extract it and move to midapackdir.

- mkdir midapackdir
- tar -xvzf Midapack_1.1.tgz -C midapackdir
- cd midapackdir

Here are several files and directories. Please refer to files README, LICENSE and INSTALL, to see under which conditions your distribution of Midapack is licensed and how to install it.

Midapack requires few libraries : mpi, openmp, fftw3. Ensure these libraries are available on your system. To build the library, run the following commands :

- ./configure
- make

The first command generate Makefiles. The second one compiles sources and build the library, libmidapack.a, in midapackdir/lib

Chapter 7

Bug tracking and reporting

To report bugs please use the following website:

http://code.google.com/p/cmb-da-library/

Chapter 8

Toeplitz algebra documentation

8.1 Introduction

The purpose of the Toeplitz algebra package of the MIDAPACK libary is to provide efficient massively, parallel routines, performing products of some special structured matrices and an arbitrary matrix. The special matrices considered here are related to **Toeplitz** matrices.

Toeplitz matrices are ubiquitous in the CMB data analysis as they describe correlation properties of stationary time-domain processes (usually instrumental noise). The matrices relevant are therefore **symmetric** and **non-negative** definite. They are also **band-diagonal** as the noise correlation length, i.e., the band-width in the parlance of Toeplitz algebra, is typically much shorter than length of the data. A useful and important generalization of those include :

- **symmetric**, **Toeplitz block-diagonal** matrices - describing piece-wise stationary processes, each of the blocks is in turn a symmetric, band-diagonal Toeplitz matrix, which can be different for the different blocks. The performance of the routines included in the package is expected to be the best, whenever the bandwidth of each block is much smaller than its size.

- **symmetric**, **Toeplitz block-diagonal**, **gappy** matrices - which are just symmetric, Toeplitz block-diagonal matrices but with some of their rows (and corresponding columns) removed. Such matrices describe piece-wise stationary processes, which have some short sequences of samples, referred hereafter as **gaps**, removed. The gaps are common in the analysis of any real CMB data sets and can arise due to cosmic rays hits or some short-term instrumental transients. If a gap happens to be longer than the matrix correlation length accounting on it, i.e., removing relevant rows and columns of the initial matrix, will result in a new block of the matrix, which remains symmetric, block-diagonal.

The library provides distributed (MPI) and sequential/multithreaded (openMP) routines, which are based on common low level sequential/openMP functions. The Fourier Transforms are performed with help of the FFTW library http://www.fftw.org/.

The overall structure of the library is as follows:

8.2 Functionality and definitions

The Toeplitz algebra package described here provides functionality for calculating **products** of a **generalized Toeplitz matrix** (understood as one of those defined earlier) and a **general matrix**. The latter is referred to hereafter typically as a **data matrix**. This is the latter matrix, which defines the reference for the global indexing scheme adopted throughout the package, which is based on its global row number. The data matrices are always stored as vectors in the **column-wise** order.

In MPI-cases the data matrices are assumed to be distributed over the processes as defined in Section on data distribution.

Toeplitz matrices are defined by a first row of the Toeplitz matrix trimmed to its half-bandwidth $+1$ denoted hereafter as λ, which one therefore includes half-bandwidth and the diagonal element), and three integer numbers giving λ, indices of the first and last row to which the matrix should be applied. The last two numbers are global row numbers of the data matrix. The size of the Toeplitz matrix is then implictly given as, `last_row_index - first_row_index + 1`. We will refer to an interval defined by these two indices as **Toeplitz matrix range**.

The list of specific functions provided is as follows:

- symmetric Toeplitz matrix-matrix product (stmm routines);
- symmetric Toeplitz block-diagonal matrix-matrix product (stbmm routines);
- symmetric, gappy, Toeplitz block-diagonal matrix-matrix product (gstbmm routines).

8.2.1 stmm routines

`stmm` routines multiply a symmetric banded Toeplitz matrix by the data matrix. The Toeplitz matrix size is assumed to be equal to that of the data matrix. If this is not the case use `stbmm` routines instead. If the data matrix is distributed over multiple processes the Toeplitz matrix has to be defined on each of them and has to be the same on each of them, including the first and the last row indices used to defined its region.

8.2.2 stbmm routines

`stbmm` routines multiply a symmetric, block-diagonal matrix, with Toeplitz banded blocks by a data matrix. This operation is in fact construed as a product of a series of Toeplitz blocks by the data matrix. The Toeplitz blocks are each defined as any Toeplitz matrix (see, e.g., Functionality and definitions above) and have in particular defined a range - an interval of the data matrix rows to which it should be applied. **The ranges of any two Toeplitz matrices must not overlap**. The limits of the interval can be looked at as defining the position of each Toeplitz block within a single Toeplitz-block diagonal matrix of the size equal that of a number of rows of the data matrix. What the routine then does it is to multiply each block by a respective subblock of the data matrix. The rows of the data matrix, which do not belong to the interval of any block are copied unchanged, meaning that the corresponding blocks of the Toeplitz matrix are implicitly assumed to be equal to 1.

In the MPI implementation each processor needs only those of all the Toeplitz blocks which have ranges overlapping with part of the data matrix assigned to it. If more are defined on the input they are ignored. Note that this is the user's responsibility to ensure that the Toeplitz matrices are assigned to different processes in a consistent way meaning they represent a product of a series of diagonal Toeplitz blocks. It is important to observe that the block distribution will in general depend on the assumed distribution pattern for the data matrix. This is discussed in the examples below.

If a Toeplitz block corresponds to the data matrix parts assigned to two (or more) processes, the Toeplitz block parameters have to be the same on all implicated processes, as each process will use them to define the amount of

data which needs communicate to its neighbors. In general, each process calculates products of the Toeplitz blocks by the corresponding part of the data matrix. If a Toeplitz block corresponds to data of more than one process then the communication is used to copy necessary data to enable such a multiplication locally.

Examples.

8.2.2.1 Examples

Example #1: Column-wise data distribution.

This figure illustrates the operations performed by the `mpi_stbmm` routine. On the input the routine requires a definition of the Toeplitz-blocks and the data matrix. The latter is assumed to be distributed in the column-wise order - here between 3 MPI-processes, as marked by three different colors. The routine will multiply each Toeplitz-block by the corresponding set of rows of each column of the data matrix. The rows of the data matrix which do not correspond to any of the blocks will be copied without change to the resulting matrix. The dashed lines mark the data divisions between the processes which will require some MPI communication to be performed. The communication will affect only the data of a single column which happens to be shared between two processes. We note that for the result to be correct each of the three processes considered here has to have both Toeplitz blocks defined in the same way on each of them.

Example #2: Row-wise data distribution.

This figure corresponds to the analogous operation as the one in Example #1 just assuming that the data matrix is distributed between 3 processes in the row-wise order. The dashed lines indicate communication instances between the processes. Note that unlike in the previous case this time the data for *all* columns have to be exchanged. In this case for the operation to be consistent process #0 needs only the first Toeplitz block in its memory, process #2 - only the second, and process #1 - both.

8.2.3 gstbmm routines

`gstbmm` routines multiply a symmetric Toeplitz block diagonal matrix, i.e., as used by `stbmm` routines, but with some sets of rows and corresponding columns removed, by an arbitrary data matrix. Such an *effective* matrix, referred hereafter as a `gstb` (a *gappy*, *symmetric*, *Toeplitz* block matrix), in general will not be made of only Toeplitz blocks anymore. (See the example below). On the input this matrix is however still represented as a corresponding `stb` matrix complemented by a list of column (or row) which are to be neglected (or effectively excised). The operation is then construed also as a `stbmm` operation, which the `sbt` matrix representating the `gstb` and, and the data matrix with the corresponding rows set to zero. Note that the data matrix is assumed to have all the rows on the input including the ones to be neglected.

On the output the routine will produce a matrix of the same type as the input one with the values in the rows to be neglected set to zero.

Example.

8.2.3.1 Example

The figure shows a product of a Toeplitz block matrix with a number of columns and rows to be excised by a data matrix. The excised columns (and rows) are marked by the dashed grids and correspond to the grayish areas of the input and output data matrices. These grayish sets of rows will be set to zero in the final result. We note that two of the three excised intervals, hereafter called *gaps*, of columns are broad enough that they effectively lead only to change of the size of the Toeplitz blocks including splitting one of them into two smaller ones. The third (rightmost) gap however destroys the Toeplitz structure of the second block. Indeed the *effective* matrix, by which white shaded part of the data matrix are multiplied by, corresponds to only dark blue areas, and does not have a Toeplitz block structure.

8.3 Numerical algorithms

The package implements two algorithms for performing the operations.

The **first** algorithm is based on a *shift-and-overlap* approach, where a product of a single band-diagonal Toeplitz matrix by an arbitrary matrix is done as a sequence of products of a submatrix of the initial Toeplitz matrix by overlapping blocks of the arbitrary matrix. Each of the latter products is performed in turn by embedding the Toeplitz subblock in a minimal circulant matrix and performing the multiplication via Fast Fourier transforms. The size of the subblock can be set appropriately to optimize the calculation and typically is a function of the bandwith. Denoting by λ a half bandwith, i.e., the full bandwidth is $2\lambda+1$, the overall complexity of the operation is $\mathcal{O}(n\ln\lambda)$, where n is the size of the initial Toeplitz matrix.

Check Shift and overlap algorithm for more details.

The **second** algorithm is just a *direct real space* multiplication of a Toeplitz matrix by an arbitrary one. This approach has complexity $\mathcal{O}(n\lambda)$ but much better prefactors and therefore can have superior performance over the previous one for very narrow bands.

All the other operations implemented in the package are then expressed as the product of a Toeplitz matrix times a general matrix. This may in turn require on occasions some data objects reformatting (or as called hereafter - reshaping) operations.

The inter-process communication is required whenever the same Toeplitz matrix (or a Toeplitz block for Toeplitz block-diagonal cases) is to be applied to a segment of the data distributed over more than one process. These are implemented using MPI calls.

More details can be found here MPI communication patterns

8.3.1 Shift and overlap algorithm

This algorithm exploits explicitly the fact that considered Toeplitz matrices are band-diagonal with a narrow band, i.e., $\lambda \ll n$ and cuts the complexity of the operation down to $\mathcal{O}|(n\ln\lambda)$ from $\mathcal{O}(2n\ln 2n)$, where the latter is obtained assuming embedding of the full Toeplitz matrix of a rank n into a circulant matrix of a twice larger rank and performing the product via Fast Fourier transforms.

The shift and overlap algorithm performs the same task as a series of products of a smaller circulant matrix with a rank b, where $b > 2\lambda$, by a corresponding, overlapping segments of the arbitrary matrix. The circulant matrix embeds a Toeplitz matrix, which is just the inital matrix trimmed to the size b. The schematic of the algorithm is shown in the figure below.

Here a product of a Toeplitz matrix marked in black by a vector is split into three products of a circulant matrix of a rank b by three overlapping segments of the input vector. Each product is marked by a different color, however the circulanr matrix by which the vector segments are multiplied is always the same. The overlaps are clearly necessary to avoid contributions from the circulant corners of the matrix. At the end the entries of the final vector which are biased by the corner contributions are removed from the result and the remainders combined together. Note that the edge segments need to be padded by zeros. The padding is done in the way that the circulant block size used is always the same. This helps to save the time needed for FFT related precomputation (FFT plans etc) and optimize a number of required FFTs.

The generalization of the algorithm for the case of a general matrix instead of a vector, as shown in the figure, is straightforward. We note that each of the elemental products of the circulant matrix times a general matrix subblock could in principle be performed in a single step using an FFT, which permits a computation of many identical FFTs simultanenously rather than it being implemented as a series of the products of the circulant matrix by subblock columns. Given that the gain in using multi-vector is not clear in current implementations of the FFTs we looked at and, if present, it is probably at the best limited to a relatively small number of the vectors, the adopted solution in the package represents the product of the circulant block by a general matrix subblocks as series of products each involving the circulant matrix by a subset of all columns of the general matrix. The number of the columns is set by the `toeplitz_init` routine.

In general given the size of the input problem n the cost of the computation is:

$$n/(n-2\lambda)\times b\ln b \sim n\ln b \sim n\ln\lambda$$

where the first factor of the leftmost term gives a number of products to be performed and the latter the cost of each of them. Here we did not account on any gains from a multi-vector FFT, e.g., we have assumed that a simulatenous FFT of k-vectors is as costly as k FFTs of a single vector.

8.3.2 MPI communication patterns

The inter-process communication is needed in the MPI routines of the package whenever boundaries of the distributed data matrix do not coincide with those of the Toeplitz block. The communication pattern is **local** in a sense that it involves only neighboring processes and is therefore expected to scale well with a number of MPI processes (and it indeed does in the regime in which the tests have been done.) It involves each process sending to and receiving from a neighboring process a vector of data of the length defined by the half-bandwidth of the Toeplitz block, λ, shared between them. This provides sufficient information to enable each process to compute a part of the Toeplitz-vector product corresponding to its input data on its own without any need for further data exchanges. In particular we note that all the FFT calls used by the package are either sequential or threaded.

The communication pattern as implemented is either *non-blocking* and then instituted with help of `MPI_Isend` and `MPI_Irecv` calls used twice to send to and receive from left and right, i.e.,

what is followed by a series of respective `MPI_Wait` calls, i.e.,

or *blocking* implemented with help `MPI_Sendrecv` calls, i.e.,

The choice between the two is made with help of the global flag `FLAG_BLOCKINGCOMM`, which by default is set to 0 (non-blocking communication).

8.4 Programming models

The Toeplitz algebra library routines allow the user to take advantage of both **multithreaded** and **memory-distributed** programming paradigms and are therefore adapted to run efficiently on heteregeneous computer architectures. The multithreading is implemented using **openMP** directives, while distributed programming uses **MPI**. Both shared and/or distributed parallelism can be switched of, at the compilation time, if so desired. Moreover, the user has always access to two versions of each of the routines: openMP/MPI and openMP-only.

8.5 Data distribution

In the memory-distributed (MPI) running modes, the data input matrix is assumed to be distributed in between the MPI processes (nodes, processors, etc). The library routines allow essentially for two different data distributions as well as one inermediate option.

The first distribution is called hereafter a **column-wise** distribution. In this case the data matrix is treated as a vector made of columns of the data matrix concatenated together. A valid data distribution can be then nearly any partition of the vector into consecutive segments, which are then assigned one-by-one to the processes. It is then assumed that the neighboring processes receive consecutive segments. Moreover, each process has to have at least as many data points as a half-bandwith of a Toeplitz block corresponding to them, if it has only one Toeplitz block assigned, which does not start or end within the data ranges.

The second distribution is called hereafter a **row-wise** distribution and it corresponds to dividing the data matrix into subblocks with a number of columns as in the full matrix. This time neighboring processess have to have blocks corresponding to the consecutive rows of the data matrix and each process has to have at least as many rows as

the band-width of the corresponding Toeplitz blocks, unless one of the Toeplitz blocks assigned to that set of rows starts or end within the rows interval.

The **hybrid** data distribution first represents the data matrix as a matrix of k columns, where $1 \leq k \leq$ *# of columns of the data matrix*. This is obtained by concatenating first k columns of the data matrix, by following k etc ... - note that the total number of columns of the data matrix has to divide by k - and then chopping such a matrix into segments assigned to different MPI process. The requirements as above also apply.

What data layout is used is defined by the input parameters of the MPI routines.

For all the routines of the package, **the layout of the output coincides with that of the input**.

8.6 Load balancing

In the case of the MPI routines the load balancing will be generally dependent on the adopted data distribution and therefore it is left to the user to select the latter to ensure the former. On the other hand, the library is designed to work, albeit potentially not very efficiently, whether such a goal is achieved or not and therefore also in circumstances far from the load balancing.

8.7 Availability and bug tracking

You can download the last release from the official website of the ANR-MIDAS'09 project at http://www.apc.-univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/software/midapack/

Please report any bugs via bug tracker at: http://code.google.com/p/cmb-da-library/

8.8 Installation

This software is reported to work on several Linux distributions and should work on any modern Unix-like system after minimal porting efforts.

The source code is delivered in a set of directories :

- The /src directory contains the sources files for the core library. It's composed by the differents modules of the MIDAS CMB DA library (please refer to the website for more details). You can directly compile theses files and link the generated binaries with your own program.

- The /test directory contains some Utility/demonstration programs to show some examples of how to use the library fonctionnalities.

8.9 User example

Here is a short example showing how to use it:

```
// sequential use
fftw_complex *V_fft, *T_fft;
double *V_rfft;
fftw_plan plan_f, plan_b;
tpltz_init(v1_size, lambda , &nfft, &blocksize, &T_fft, T, &V_fft, &
     V_rfft, &plan_f, &plan_b);
stmm(V, n, m, id0, local_V_size, T_fft, lambda, V_fft, V_rfft, plan_f,
     plan_b, blocksize, nfft);
tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);


// MPI use
MPI_Scatterv(V, nranks, displs, MPI_DOUBLE, Vrank, maxsize, MPI_DOUBLE, 0,
     MPI_COMM_WORLD);
```

```
flag_stgy_init_auto(&flag_stgy);
mpi_stbmm(&Vrank, nrow, m, m_rowwise, tpltzblocks, nb_blocks,
     nb_blocks, id0, local_V_size, flag_stgy, MPI_COMM_WORLD);
MPI_Gatherv(Vrank, nranks[rank], MPI_DOUBLE, TV, nranks, displs, MPI_DOUBLE, 0
     , MPI_COMM_WORLD);
```

Chapter 9

Todo List

Global MatLoad (Mat *mat, char *filename)

Implement to read several file formats as basic ASCII, XML, HDF5...

Global ring_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int steps, MPI_Comm comm)

Ring loop and ring table are set from index 1 to size. Should be shift and be set from index 0 to size-1.

Chapter 10

Module Index

10.1 Modules

Here is a list of all modules:

Chapter 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

Chapter 13

Module Documentation

13.1 Pointing module

Modules

- user interface (API)
- internal routines

13.1.1 Detailed Description

Pointing operations module

13.2 user interface (API)

Modules

- utility routines
- main routines

13.2.1 Detailed Description

These are routines "officially" accessible by a user.

13.3 utility routines

Functions

- int MatInit (Mat ∗A, int m, int nnz, int ∗indices, double ∗values, int flag#ifdef W_MPI, MPI_Comm comm#endif)
- void MatSetIndices (Mat ∗A, int m, int nnz, int ∗indices)
- void MatSetValues (Mat ∗A, int m, int nnz, double ∗values)
- void MatFree (Mat ∗A)
- int MatLoad (Mat ∗mat, char ∗filename)
- int MatSave (Mat ∗mat, char ∗filename)
- int MatLocalShape (Mat ∗A, int sflag)
- int MatComShape (Mat ∗A, int flag, MPI_Comm comm)
- int TrMatVecProd (Mat ∗A, double ∗y, double ∗x, int pflag)
- int MatInfo (Mat ∗mat, int verbose, char ∗filename)

  Print information about a matrix.

13.3.1 Detailed Description

These are auxiliary utility routines. They may be used in both memory-shared/sequential and dsitributed contexts, though their syntax may change depending on that.

13.3.2 Function Documentation

13.3.2.1 int MatInit (Mat ∗ *A*, int *m*, int *nnz*, int ∗ *indices*, double ∗ *values*, int flag#ifdef *W_MPI*, MPI_Comm comm# *endif*)

Create a matrix specifying the number of local rows m, the number of non-zero elements per row nnz , indices tab, values tab, flag for communication and a communicator comm. indices and values tabs must be allocated and contain at least m∗nnz elements. It represents column indices of the nonzero elements. Respectively values tab represents the non-zero values. After call MatInit, all precomputation are done and the matrix structure is ready to use. That means you can start applying matrix operation. Another way to initialize a matrix structure is to apply step by step :

- MatSetIndices
- MatSetValues
- MatLocalShape
- MatComShape

Warning

do not modify indices tab until you will use the matrix structure.
[MPI COMM!] with Midapack sequential version, there is no communicator argument.

Parameters

A	pointer to a Mat struct
m	number of local rows
nnz	number of non-zero per row
indices	input tab (modified)
values	input tab
flag	communication flag
comm	MPI communicator

See also

MatFree

Definition at line 43 of file mapmat.c.

13.3.2.2 void MatSetIndices (Mat * A, int m, int nnz, int * indices)

Set column indices of the nonzero elements. indices tab must be allocated and contains at least m*nnz elements.

Parameters

A	pointer to a Mat struct
m	number of local rows
nnz	number of non-zero per row
indices	input tab

Returns

void

Definition at line 70 of file mapmat.c.

13.3.2.3 void MatSetValues (Mat * A, int m, int nnz, double * values)

Set values of the nonzero elements. values tab must be allocated and contains at least m*nnz values.

Parameters

A	pointer to a Mat struct
m	number of local rows
nnz	number of non-zero per row
values	input tab

Returns

void

Definition at line 85 of file mapmat.c.

13.3.2.4 void MatFree (Mat * A)

Free allocated tabs of a matrix structure including local indices tab and communication tabs. Do not free indices and values which user is responsible for.

Parameters

A	pointer to a Mat struct

Returns

void

See also

MatInit MatLocalShape

Definition at line 317 of file mapmat.c.

13.3.2.5 int MatLoad (Mat * *mat*, char * *filename*)

Load matrix from a file. This is MatSave dual routine which loads data into matrix reading a specified file (or several specified files). Number of files should equal number of processor. File format should be ascii files (for examples look at files generated by MapMatSave routine).

Todo Implement to read several file formats as basic ASCII, XML, HDF5...

Parameters

mat	pointer to the Mat
filename	basename of a file, actually data are loaded from several files denotes by "basename + processor number"

Returns

error code

Definition at line 403 of file mapmat.c.

13.3.2.6 int MatSave (Mat * *mat*, char * *filename*)

Write matrix into files This is the dual routine of MatLoad. It saves matrix data into files, and can be usefull to check that data are well stored. Obviously it performs IO, moreover with one file per processor. Therfore, just call this function in a development phase, with few data and few processors.

Parameters

A	pointer to the Mat
filename	file basename, for instance passing "toto" should produce the output files named "toto_$(rank)"

Returns

error code

Definition at line 451 of file mapmat.c.

13.3.2.7 int MatLocalShape (Mat * *A*, int *sflag*)

Compute a local indices into a dense vector, lindices, and reindices indices tab according the this local dense vector. For this three steps are performed :

- sort and merge indices tab,
- allocate lindices of size lcount and copy the sorted indices
- reindex indices according the local indices

Warning

lindices is internally allocated (to free it, use MatFree)

See also

MatComShape MatFree MatSetIndices

Definition at line 489 of file mapmat.c.

13.3.2.8 int MatComShape (Mat * A, int flag, MPI_Comm comm)

Transform the matrix data structure, identifying columns shared by several processors

Warning

[MPI ONLY!] this function does not exist in Midapack sequential version

See also

MatLocalShape MatInit TrMatVecProd

Definition at line 515 of file mapmat.c.

13.3.2.9 int TrMatVecProd (Mat * A, double * y, double * x, int pflag)

Perform a transposed matrix-vector multiplication, $x \leftarrow A^t y$ using a precomputed communication scheme. Before calling this routine, the communication structure should have been set, calling MatInit or MatComShape. The routine can be divided in two steps :

- a local matrix vector multiplication
- a collective-reduce. it consits in a sum reduce over all processes.

The collective reduce is performed using algorithm previously defined : ring, butterfly ...

See also

MatVecProd MatComShape TrMatVecProd_Naive MatInit

Parameters

A	a pointer to a Mat
y	local input vector (distributed)
x	local output vector (overlapped)

Definition at line 723 of file mapmat.c.

13.3.2.10 int MatInfo (Mat * mat, int verbose, char * filename)

Print information about a matrix.

Usefull function to check, debug or bench. It prints matrix array sizes.

See also

MatSave

Parameters

A	pointer to the Mat

Definition at line 758 of file mapmat.c.

13.4 main routines

Modules

- multithreaded/sequential routines
- distributed memory (MPI) routines

13.4.1 Detailed Description

These are core routines of the pointing library. They come in two flavors:

- shared-memory: multithreaded (openMP/sequential) routines
- distributed-memory (MPI) routines

13.5 multithreaded/sequential routines

Functions

- int MatVecProd (Mat *A, double *x, double *y, int pflag)

13.5.1 Detailed Description

These are sequential and/or shared-memory routines.

13.5.2 Function Documentation

13.5.2.1 int MatVecProd (Mat * *A*, double * *x*, double * *y*, int *pflag*)

Perform matrix-vector multiplication, $y \leftarrow Ax$.

Parameters

A	pointer to a Mat
x	input vector (overlapped)
y	output vector (distributed)

Definition at line 630 of file mapmat.c.

13.6 distributed memory (MPI) routines

Functions

- int TrMatVecProd_Naive (Mat *A, double *y, double *x, int pflag)

13.6.1 Detailed Description

These are distributed-memory routines.

Note that if the MPI flag is not set on the compilation stage, most of the routines listed here will run sequentially unless explicitly stated to the contrary in their description.

13.6.2 Function Documentation

13.6.2.1 int TrMatVecProd_Naive (Mat * *A*, double * *y*, double * *x*, int *pflag*)

Perform transposed matrix-vector multiplication, $x \leftarrow A^t y$. This naive version does not require a precomputed communication structure. But communication volumes may be significant. Consequently in most of the cases is not optimized.

Warning

[MPI ONLY!] this function does not exist in Midapack sequential version

See also

TrMatVecProd MatLocalShape

Parameters

mat	pointer
y	local input vector (distributed)
x	local output vector (overlapped)

Definition at line 658 of file mapmat.c.

13.7 internal routines

Modules

- low-level computation routines
- lower internal routines

13.7.1 Detailed Description

These are auxiliary, internal routines, not intended to be used by no-expert user. They are divided in two groups:

- low level computation routines
- internal routines

13.8 low-level computation routines

These are low-level computation routines not really expected to be used by users, e.g., the syntax may evolve unexpectedly, but may be be found on occasions useful ...

13.9 lower internal routines

Functions

- int m2m (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)
- int m2m_sum (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)
- int card_or (int *A1, int n1, int *A2, int n2)
- int card_and (int *A1, int n1, int *A2, int n2)
- int set_or (int *A1, int n1, int *A2, int n2, int *A1orA2)
- int set_and (int *A1, int n1, int *A2, int n2, int *A1andA2)
- int butterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.
- int butterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int butterfly_blocking_2instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int butterfly_blocking_1instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- double truebutterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int sindex (int *T, int nT, int *A, int nA)
- int omp_pindex (int *T, int nT, int *A, int nA)
- int ssort (int *indices, int count, int flag)
- int omp_psort (int *A, int nA, int flag)
- int ring_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int steps, MPI_Comm comm)

  Initialize tables for ring-like communication scheme.
- int ring_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.
- int alltoallv_reduce (int **R, int *nR, int nRtot, int **S, int *nS, int nStot, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using an MPI-Alltoallv call.
- int ring_nonblocking_reduce (int **R, int *nR, int **S, int *nS, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking communication scheme.
- int ring_noempty_reduce (int **R, int *nR, int nneR, int **S, int *nS, int nneS, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking no-empty communication scheme.
- int ring_noempty_step_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.
- int truebutterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

Initialize tables for butterfly-like communication scheme (true means pair wise) This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

13.9.1 Detailed Description

These are low level internal routines. These are generally not to be used by external users.

13.9.2 Function Documentation

13.9.2.1 int m2m (double * *vA1*, int * *A1*, int *n1*, double * *vA2*, int * *A2*, int *n2*)

Function m2m for "map to map" Extract values from one map (A1, vA1), and for each pixel shared with an other map (A2, vA2), assign pixel value in vA1 and to pixel value in vA2.

Returns

a number of elements shared between A1 and A2

See also

m2m_sum

Definition at line 93 of file alm.c.

13.9.2.2 int m2m_sum (double * *vA1*, int * *A1*, int *n1*, double * *vA2*, int * *A2*, int *n2*)

Function m2m_sum for "sum map to map" Extract values from one map (A1, vA1), and for each pixel shared with an other map (A2, vA2), sum pixel value in vA1 to pixel value in vA2.

Returns

a number of elements shared between A1 and A2

See also

m2m

Definition at line 118 of file alm.c.

13.9.2.3 int card_or (int * *A1*, int *n1*, int * *A2*, int *n2*)

Compute $card(A_1 \cup A_2)$ A1 and A2 should be two ascending ordered monotmony sets. of size n1 and n2.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices

Returns

size of the union

Definition at line 60 of file als.c.

13.9.2.4 int card_and (int * *A1*, int *n1*, int * *A2*, int *n2*)

Compute $card(A_1 \cap A_2)$ A1 and A2 should be two ascending ordered monotony sets, of size n1 and n2.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices

Returns

size of the intersection

Definition at line 89 of file als.c.

13.9.2.5 int set_or (int * *A1*, int *n1*, int * *A2*, int *n2*, int * *A1orA2*)

Compute $A1 \cup A_2$ A1 and A2 should be two ascending ordered sets. It requires the sizes of these two sets, n1 and n2. A1andA2 has to be previouly allocated.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices
address	to the set A1orA2

Returns

number of elements in A1orA2

Definition at line 118 of file als.c.

13.9.2.6 int set_and (int * *A1*, int *n1*, int * *A2*, int *n2*, int * *A1andA2*)

Compute $A_1 \cap A_2$ A1 and A2 should be two monotony sets in ascending order. It requires the sizes of these two sets, n1 and n2. A1andA2 has to be previously allocated.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices
address	to the set A1andA2

Returns

number of elements in A1andA2

Definition at line 162 of file als.c.

13.9.2.7 int butterfly_init (int * *indices*, int *count*, int ** *R*, int * *nR*, int ** *S*, int * *nS*, int ** *com_indices*, int * *com_count*, int *steps*, MPI_Comm *comm*)

Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

Parameters

indices	set of indices(monotony) handle by a process.
count	number of elements
R	pointer to receiving maps
nR	array of number of elements in each receiving map
S	pointer to sending maps
nS	array of number of elements in each sending map
com_indices	set of indices(monotony) communicated by a process
com_count	number of elements
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 37 of file butterfly.c.

13.9.2.8 double butterfly_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 209 of file butterfly.c.

13.9.2.9 int butterfly_blocking_2instr_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 258 of file butterfly.c.

13.9.2.10 int butterfly_blocking_1instr_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 303 of file butterfly.c.

13.9.2.11 int truebutterfly_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme (true means pairwise)

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 430 of file butterfly_extra.c.

13.9.2.12 int sindex (int * *T*, int *nT*, int * *A*, int *nA*)

Sequential reindexing

Parameters

T	monotony array
nT	number of index
A	tab to reindex
nA	number of element to reindex

Returns

array of indices

Definition at line 18 of file cindex.c.

13.9.2.13 int omp_pindex (int * *T*, int *nT*, int * *A*, int *nA*)

Multithread (OpenMP) reindexing

Parameters

T	monotony array
nT	number of index
A	tab to reindex
nA	inumber of element to reindex

Returns

array of indices

Definition at line 36 of file cindex.c.

13.9.2.14 int ssort (int * *indices*, int *count*, int *flag*)

Sort and merge redundant elements of a set of indices using a specified method. The indices tab, initially an arbitrary set of integers, becomes a monotony set. Available methods :

- quick sort
- bubble sort
- insertion sort
- counting sort
- shell sort

Parameters

indices	tab (modified)
count	number of indices
flag	method

Returns

number of sorted elements

Definition at line 161 of file csort.c.

13.9.2.15 int omp_psort (int * *A*, int *nA*, int *flag*)

Sort and merge redundant elements of a set of indices, using openMP. The indices tab, initially an arbitrary set of integers, becomes a monotony set. Algorithm is diivided in two steps :

- each thread sorts, in parallel, a subpart of the set using a specified method.
- subsets obtained are merged successively in a binary tree manner.

Available methods for the fully parallel step :

- quick sort
- bubble sort
- insertion sort
- counting sort
- shell sort

Parameters

indices	tab (modified)
count	number of elements to sort

Returns

flag method

Definition at line 291 of file csort.c.

13.9.2.16 int ring_init (int * *indices*, int *count*, int ** *R*, int * *nR*, int ** *S*, int * *nS*, int *steps*, MPI_Comm *comm*)

Initialize tables for ring-like communication scheme.

This routine set up needed tables for the ring communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in ring scheme). Double pointer are partially allocated, the last allocation is performed inside the routine (only for R S are just pointer).

Parameters

indices	set of indices(monotony) handle by a process.
count	number of elements
R	pointer to receiving maps
nR	array of number of elements in each receiving map
S	pointer to sending maps
nS	array of number of elements in each sending map
com_indices	set of indices(monotony) communicated by a process
com_count	number of elements
steps	number of communication exchange in the ring scheme
comm	MPI communicator

Todo Ring loop and ring table are set from index 1 to size. Should be shift and be set from index 0 to size-1.

Returns

0 if no error

Definition at line 33 of file ring.c.

13.9.2.17 int ring_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 83 of file ring.c.

13.9.2.18 int alltoallv_reduce (int ** *R*, int * *nR*, int *nRtot*, int ** *S*, int * *nS*, int *nStot*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using an MPI-Alltoallv call.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRtot	size of the receive buffer
S	pointer to sending maps
nS	array of number of elements in each sending map
nStot	size of the send buffer
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 129 of file ring.c.

13.9.2.19 int ring_nonblocking_reduce (int ** *R*, int * *nR*, int ** *S*, int * *nS*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
S	pointer to sending maps
nS	array of number of elements in each sending map
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 199 of file ring.c.

13.9.2.20 int ring_noempty_reduce (int ** *R*, int * *nR*, int *nneR*, int ** *S*, int * *nS*, int *nneS*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking no-empty communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nneR	number of no-empty receiving messages
S	pointer to sending maps
nS	array of number of elements in each sending map
nneS	number of no-empty sending messages
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 257 of file ring.c.

13.9.2.21 int ring_noempty_step_reduce (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.

Parameters

R	pointer to receiving maps
nR	array of number of elements in each receiving map
nRmax	maximum size of received message
S	pointer to sending maps
nS	array of number of elements in each sending map
nSmax	maximum size of sent message
val	set of values (typically values associated to communicated indices)
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 331 of file ring.c.

13.9.2.22 int truebutterfly_init (int * *indices*, int *count*, int ** *R*, int * *nR*, int ** *S*, int * *nS*, int ** *com_indices*, int * *com_count*, int *steps*, MPI_Comm *comm*)

Initialize tables for butterfly-like communication scheme (true means pair wise) This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

Parameters

indices	set of indices(monotony) handle by a process.
count	number of elements
R	pointer to receiving maps
nR	array of number of elements in each receiving map
S	pointer to sending maps
nS	array of number of elements in each sending map
com_indices	set of indices(monotony) communicated by a process
com_count	number of elements
steps	number of communication exchange in the butterfly scheme
comm	MPI communicator

Returns

0 if no error

Definition at line 37 of file truebutterfly.c.

13.10 TOEPLITZ module

Modules

- user interface (API)
- internal routines

13.10.1 Detailed Description

Toeplitz matrix algebra module

13.11 user interface (API)

Modules

- wizard routines
- multithreaded/sequential routines
- distributed memory (MPI) routines

13.11.1 Detailed Description

These routines provide main functionality of the Toeplitz algebra library. They are divided in two groups:

- shared-memory: multithreaded (openMP/sequential) routines
- distributed-memory (MPI) routines

13.12 wizard routines

Functions

- int stbmmProd (Tpltz Nm1, double *V)

  Performs the product of a Toeplitz matrix by a general matrix either sequentially or using MPI. The complexity is hidden in the input structure, which needs to be defined by a user.

13.12.1 Detailed Description

These are easy to use drivers for the Toeplitz routines.

13.12.2 Function Documentation

13.12.2.1 int stbmmProd (Tpltz *Nm1*, double * *V*)

Performs the product of a Toeplitz matrix by a general matrix either sequentially or using MPI. The complexity is hidden in the input structure, which needs to be defined by a user.

Definition at line 63 of file toeplitz_wizard.c.

13.13 multithreaded/sequential routines

Functions

- int tpltz_init (int n, int lambda, int *nfft, int *blocksize, fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, Flag flag_stgy)

  Sets a block size and initializes all fftw arrays and plans needed for the computation.
- int tpltz_cleanup (fftw_complex **T_fft, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b)

  Cleans fftw workspace used in the Toeplitz matrix matrix product's computation.
- int reset_gaps (double **V, int id0, int local_V_size, int m, int nrow, int m_rowwise, int64_t *id0gap, int *lgap, int ngap)

  Set the data to zeros at the gaps location.
- int gap_masking (double **V, int l, int *id0gap, int *lgap, int ngap)

  Reduce the vector and mask the defined gaps.
- int gap_filling (double **V, int l, int *id0gap, int *lgap, int ngap)

  Extend the vector and add zeros on the gaps locations.
- int stmm (double **V, int n, int m, double *T, int lambda, Flag flag_stgy)

  Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm.
- int gstbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix with gaps, T, by an arbitrary matrix, V, distributed over processes.

13.13.1 Detailed Description

These are shared-memory routines.

13.13.2 Function Documentation

13.13.2.1 int tpltz_init (int *n*, int *lambda*, int * *nfft*, int * *blocksize*, fftw_complex ** *T_fft*, double * *T*, fftw_complex ** *V_fft*, double ** *V_rfft*, fftw_plan * *plan_f*, fftw_plan * *plan_b*, Flag *flag_stgy*)

Sets a block size and initializes all fftw arrays and plans needed for the computation.

Initializes the fftw arrays and plans is necessary before any computation of the Toeplitz matrix matrix product. Use tpltz_cleanup afterwards.

See also

tpltz_cleanup

Parameters

n	row size of the matrix used for later product
lambda	Toeplitz band width
nfft	maximum number of FFTs you want to compute at the same time
blocksize	optimal block size used in the sliding window algorithm to compute an optimize value)
T_fft	complex array used for FFTs
T	Toeplitz matrix
V_fft	complex array used for FFTs
V_rfft	real array used for FFTs
plan_f	fftw plan forward (r2c)
plan_b	fftw plan backward (c2r)

Definition at line 257 of file toeplitz.c.

13.13.2.2 int tpltz_cleanup (fftw_complex ** *T_fft*, fftw_complex ** *V_fft*, double ** *V_rfft*, fftw_plan * *plan_f*, fftw_plan * *plan_b*)

Cleans fftw workspace used in the Toeplitz matrix matrix product's computation.

Destroy fftw plans, free memory and reset fftw workspace.

See also

tpltz_init

Parameters

T_fft	complex array used for FFTs
V_fft	complex array used for FFTs
V_rfft	real array used for FFTs
plan_f	fftw plan forward (r2c)
plan_b	fftw plan backward (c2r)

Definition at line 435 of file toeplitz.c.

13.13.2.3 int reset_gaps (double ** *V*, int *id0*, int *local_V_size*, int *m*, int *nrow*, int *m_rowwise*, int64_t * *id0gap*, int * *lgap*, int *ngap*)

Set the data to zeros at the gaps location.

The datas located on a gap are set to zeros. The gaps are defined in the time space, meaning their indexes are defined in the row dimension.

Definition at line 189 of file toeplitz_gappy.c.

13.13.2.4 int gap_masking (double ** *V*, int *l*, int * *id0gap*, int * *lgap*, int *ngap*)

Reduce the vector and mask the defined gaps.

Parameters

V	input/output vector
l	length of the vector
id0gap	gap first index
lgap	gap length
ngap	number of gaps

Definition at line 255 of file toeplitz_gappy_seq.dev.c.

13.13.2.5 int gap_filling (double ** *V*, int *l*, int * *id0gap*, int * *lgap*, int *ngap*)

Extend the vector and add zeros on the gaps locations.

Parameters

V	input/output vector
l	length of the extend vector
id0gap	gap first index
lgap	gap length
ngap	number of gaps

Definition at line 293 of file toeplitz_gappy_seq.dev.c.

13.13.2.6 int stmm (double ** *V*, int *n*, int *m*, double * *T*, int *lambda*, Flag *flag_stgy*)

Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm.

This is a simplifed call of the sequential product including initialization and cleaning. This use the flag parameters to set the comutational options.

Parameters

V	[input] data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
n	number of rows of V
m	number of columns of V
m	number of columns of V
T	Toeplitz matrix data composed of the non-zero entries of his first row
lambda	number of non-zero in the first row of the Toeplitz and size of T
flag_stgy	flag strategy for the product computation

Definition at line 72 of file toeplitz_seq.c.

13.13.2.7 int gstbmm (double ** *V*, int *nrow*, int *m_cw*, int *m_rw*, Block * *tpltzblocks*, int *nb_blocks*, int64_t *idp*, int *local_V_size*, int64_t * *id0gap*, int * *lgap*, int *ngap*, Flag *flag_stgy*)

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix with gaps, T, by an arbitrary matrix, V, distributed over processes.

We first rebuild the Toeplitz block matrix structure to reduce the computation cost and skip the computations of the values on the defined gaps.

The parameters are :

Parameters

V	[input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
nrow	number of rows of the global data matrix V
m	number of columns for the data matrix V in the global rowwise order
m_rowwise	number of columns for the data matrix V in the rowwise order per processor
tpltzblocks	list of the toeplitz blocks struture with its own parameters (idv, n, T_block, lambda) : • idv is the global row index defining for each Toeplitz block as stored in the vector T ; • n size of each Toeplitz block • T_block location of each Toeplitz matrix data composed of the non-zero entries of the first row ; • lambda size of each Toeplitz block data T_block. The bandwith size is then equal to lambda∗2-1
nb_blocks_all	number of all Toeplitz block on the diagonal of the full Toeplitz matrix
nb_blocks_local	number of Toeplitz blocks as stored in T
idp	global index of the first element of the local part of V
local_V_size	a number of all elements in local V
id0gap	index of the first element of each defined gap
lgap	length of each defined gaps
ngap	number of defined gaps
flag_stgy	flag strategy for the product computation

Definition at line 170 of file toeplitz_seq.c.

13.14 distributed memory (MPI) routines

Functions

- int mpi_stmm (double **V, int n, int m, int id0, int l, double *T, int lambda, Flag flag_stgy, MPI_Comm comm)

  Performs the product of a Toeplitz matrix by a general matrix using MPI. We assume that the matrix has already been scattered. (a USER routine)
- int mpi_gstbmm (double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.
- int stbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

13.14.1 Detailed Description

These are distributed-memory routines.

13.14.2 Function Documentation

13.14.2.1 int mpi_stmm (double ** *V,* int *n,* int *m,* int *id0,* int *l,* double * *T,* int *lambda,* Flag *flag_stgy,* MPI_Comm *comm*)

Performs the product of a Toeplitz matrix by a general matrix using MPI. We assume that the matrix has already been scattered. (a USER routine)

The multiplication is performed using FFT applied to circulant matrix in order to diagonalized it. The parameters are :

Parameters

V	[input] distributed data matrix (with the convention V(i,j)=V[i+j*n]); [out] result of the product TV
n	number of rows of V
m	number of columns of V
id0	first index of scattered V
l	length of the scattered V
T	Toeplitz matrix.
lambda	Toeplitz band width.
flag_stgy	flag strategy for the product computation
comm	communicator (usually MPI_COMM_WORLD)

Definition at line 980 of file toeplitz.c.

13.14.2.2 int mpi_gstbmm (double ** *V,* int *nrow,* int *m,* int *m_rowwise,* Block * *tpltzblocks,* int *nb_blocks_local,* int *nb_blocks_all,* int *id0p,* int *local_V_size,* int64_t * *id0gap,* int * *lgap,* int *ngap,* Flag *flag_stgy,* MPI_Comm *comm*)

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.

We first rebuild the Toeplitz block matrix structure to reduce the computation cost and skip the computations of the values on the defined gaps. then, each process performs the multiplication sequentially for each of the gappy block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. The parameters are :

Parameters

V	[input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
nrow	number of rows of the global data matrix V
m	number of columns for the data matrix V in the global rowwise order
m_rowwise	number of columns for the data matrix V in the rowwise order per processor
tpltzblocks	list of the toeplitz blocks struture with its own parameters (idv, n, T_block, lambda) : • idv is the global row index defining for each Toeplitz block as stored in the vector T ; • n size of each Toeplitz block • T_block location of each Toeplitz matrix data composed of the non-zero entries of the first row ; • lambda size of each Toeplitz block data T_block. The bandwith size is then equal to lambda∗2-1
nb_blocks_all	number of all Toeplitz block on the diagonal of the full Toeplitz matrix
nb_blocks_local	number of Toeplitz blocks as stored in T
idp	global index of the first element of the local part of V
local_V_size	a number of all elements in local V
id0gap	index of the first element of each defined gap
lgap	length of each defined gaps
ngap	number of defined gaps
flag_stgy	flag strategy for the product computation
comm	MPI communicator

Definition at line 88 of file toeplitz_gappy.c.

13.14.2.3 int stbmm (double ∗∗ *V*, int *nrow*, int *m_cw*, int *m_rw*, Block ∗ *tpltzblocks*, int *nb_blocks*, int64_t *idp*, int *local_V_size*, Flag *flag_stgy*)

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

Each process performs the multiplication sequentially for each diagonal block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. Each of the diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block. The parameters are :

Parameters

V	[input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
nrow	number of rows of the global data matrix V
m	number of columns for the data matrix V in the global rowwise order
m_rowwise	number of columns for the data matrix V in the rowwise order per processor
tpltzblocks	list of the toeplitz blocks struture with its own parameters (idv, n, T_block, lambda) :
nb_blocks	number of Toeplitz blocks as stored in T
idp	global index of the first element of the local part of V
local_V_size	a number of all elements in local V
flag_stgy	flag strategy for the product computation

Definition at line 121 of file toeplitz_seq.c.

13.15 internal routines

Modules

- low-level routines
- lower internal routines

13.15.1 Detailed Description

These are auxiliary, internal routines, not intended to be used by no-expert user. They are divided in two groups:

- low level routines
- internal routines

13.16 low-level routines

Functions

- int define_blocksize (int n, int lambda, int bs_flag, int fixed_bs)

  Defines an optimal size of the block used in the sliding windows algorithm.
- int define_nfft (int n_thread, int flag_nfft, int fixed_nfft)

  Defines the number of simultaneous ffts for the Toeplitz matrix product computation.
- int fftw_init_omp_threads (int fftw_n_thread)

  Initialize omp threads for fftw plans.
- int rhs_init_fftw (int *nfft, int fft_size, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag)

  Initializes fftw array and plan for the right hand side, general matrix V.
- int circ_init_fftw (double *T, int fft_size, int lambda, fftw_complex **T_fft)

  Initializes fftw array and plan for the circulant matrix T_circ obtained from T.
- int scmm_direct (int fft_size, int nfft, fftw_complex *C_fft, int ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.
- int scmm_basic (double **V, int blocksize, int m, fftw_complex *C_fft, double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix by a matrix using FFT's (an INTERNAL routine)
- int stmm_core (double **V, int n, int m, double *T, fftw_complex *T_fft, int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f, fftw_plan plan_b, int flag_offset, int flag_nofft)

  Performs the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm. (an INTERNAL routine)
- int stmm_main (double **V, int n, int m, int id0, int l, double *T, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy)

  Performs the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping. (an INTERNAL routine)
- int build_gappy_blocks (int nrow, int m, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t *id0gap, int *lgap, int ngap, Block *tpltzblocks_gappy, int *nb_blocks_gappy_final, int flag_param_distmin_fixed)

  Build the gappy Toeplitz block structure to optimise the product computation at gaps location.
- int stmm_simple_basic (double **V, int n, int m, double *T, int lambda, double **TV)

  Perform the product of a Toeplitz matrix by a matrix without using FFT's.
- int stmm_simple_core (double **V, int n, int m, double *T, int blocksize, int lambda, int nfft, int flag_offset)

  Perform the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm.
- int flag_stgy_init_auto (Flag *flag_stgy)

  Set the flag to automatic paramaters.
- int flag_stgy_init_zeros (Flag *flag_stgy)

  Set the flag parameters to zeros. This is almost the same as automatic.
- int flag_stgy_init_defined (Flag *flag_stgy)

  Set the parameters flag to the defined ones.
- int print_flag_stgy_init (Flag flag_stgy)

  Print the flag parameters values.

13.16.1 Detailed Description

These are low-level routines.

13.16.2 Function Documentation

13.16.2.1 int define_blocksize (int *n*, int *lambda*, int *bs_flag*, int *fixed_bs*)

Defines an optimal size of the block used in the sliding windows algorithm.

The optimal block size is computed as the minimum power of two above 3∗lambda, i.e. the smallest value equal to 2^x, where x is an integer, and above 3∗lambda. If bs_flag is set to one, a different formula is used to compute the optimal block size (see MADmap: A MASSIVELY PARALLEL MAXIMUM LIKELIHOOD COSMIC MICROWAVE BACKGROUND MAP-MAKER, C. M. Cantalupo, J. D. Borrill, A. H. Jaffe, T. S. Kisner, and R. Stompor, The Astrophysical Journal Supplement Series, 187:212–227, 2010 March). To avoid using block size much bigger than the matrix, the block size is set to 3∗lambda when his previous computed size is bigger than the matrix size n. This case append mostly for small matrix compared to his bandwith.

Parameters

n	matrix row dimension
lambda	half bandwidth of the Toeplitz matrix
bs_flag	flag to use a different formula for optimal block size computation
fixed_bs	fixed blocksize value if needed

Definition at line 130 of file toeplitz.c.

13.16.2.2 int define_nfft (int *n_thread*, int *flag_nfft*, int *fixed_nfft*)

Defines the number of simultaneous ffts for the Toeplitz matrix product computation.

Parameters

n_thread	number of omp threads
flag_nfft	flag to set the strategy to define nfft
fixed_nfft	fixed nfft value if nedeed (used for the case where flag_nfft=1)

Definition at line 220 of file toeplitz.c.

13.16.2.3 int fftw_init_omp_threads (int *fftw_n_thread*)

Initialize omp threads for fftw plans.

Initialize omp threads for fftw plans. The number of threads used for ffts (define by the variable n_thread) is read from OMP_NUM_THREAD environment variable. fftw multithreaded option is controlled by fftw_MULTITHREADING macro.

Definition at line 333 of file toeplitz.c.

13.16.2.4 int rhs_init_fftw (int ∗ *nfft*, int *fft_size*, fftw_complex ∗∗ *V_fft*, double ∗∗ *V_rfft*, fftw_plan ∗ *plan_f*, fftw_plan ∗ *plan_b*, int *fftw_flag*)

Initializes fftw array and plan for the right hand side, general matrix V.

Initialize fftw array and plan for the right hand side matrix V.

Parameters

nfft	maximum number of FFTs you want to compute at the same time
fft_size	effective FFT size for the general matrix V (usually equal to blocksize)
V_fft	complex array used for FFTs
V_rfft	real array used for FFTs
plan_f	fftw plan forward (r2c)

plan_b	fftw plan backward (c2r)
fftw_flag	fftw plan allocation flag

Definition at line 365 of file toeplitz.c.

13.16.2.5 int circ_init_fftw (double * *T*, int *fft_size*, int *lambda*, fftw_complex ** *T_fft*)

Initializes fftw array and plan for the circulant matrix T_circ obtained from T.

Builds the circulant matrix T_circ from T and initilizes its fftw arrays and plans. Use tpltz_cleanup afterwards.

See also

tpltz_cleanup

Parameters

T	Toeplitz matrix.
fft_size	effective FFT size for the circulant matrix (usually equal to blocksize)
lambda	Toeplitz band width.
T_fft	complex array used for FFTs.

Definition at line 392 of file toeplitz.c.

13.16.2.6 int scmm_direct (int *fft_size*, int *nfft*, fftw_complex * *C_fft*, int *ncol*, double * *V_rfft*, double ** *CV*, fftw_complex * *V_fft*, fftw_plan *plan_f_V*, fftw_plan *plan_b_CV*)

Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.

Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans: forward - plan_f_V; and backward - plan_b_CV. C_fft is a Fourier (complex representation of the circulant matrix) of length fft_size/2+1; V_rfft is a matrix with ncol columns and fft_size rows; V_fft is a workspace of fft_size/2+1 complex numbers as required by the backward FFT (plan_b_CV); CV is the output matrix of the same size as the input V_rfft one. The FFTs transform ncol vectors simultanously.

Parameters

	fft_size	row dimension
	nfft	number of simultaneous FFTs
	C_fft	complex array used for FFTs
	ncol	column dimension
	V_rfft	real array used for FFTs
out	*CV*	product of the circulant matrix C_fft by the matrix V_rfft
	V_fft	complex array used for FFTs
	plan_f_V	fftw plan forward (r2c)
	plan_b_CV	fftw plan backward (c2r)

Definition at line 509 of file toeplitz.c.

13.16.2.7 int scmm_basic (double ** *V*, int *blocksize*, int *m*, fftw_complex * *C_fft*, double ** *CV*, fftw_complex * *V_fft*, double * *V_rfft*, int *nfft*, fftw_plan *plan_f_V*, fftw_plan *plan_b_CV*)

Performs the product of a circulant matrix by a matrix using FFT's (an INTERNAL routine)

This routine multiplies a circulant matrix, represented by C_fft, by a general matrix V, and stores the output as a matrix CV. In addition the routine requires two workspace objects, V_fft and V_rfft, to be allocated prior to a call to

it as well as two fftw plans: one forward (plan_f_V), and one backward (plan_b_TV). The sizes of the input general matrix V and the ouput CV are given by blocksize rows and m columns. They are stored as a vector in the column-wise order. The circulant matrix, which is assumed to be band-diagonal with a band-width lambda, is represented by a Fourier transform with its coefficients stored in a vector C_fft (length blocksize). blocksize also defines the size of the FFTs, which will be performed and therefore this is the value which has to be used while creating the fftw plans and allocating the workspaces. The latter are given as: nfft∗(blocksize/2+1) for V_fft and nfft∗blocksize for V_rfft. The fftw plans should correspond to doing the transforms of nfft vectors simultaneously. Typically, the parameters of this routine are fixed by a preceding call to Toeplitz_init(). The parameters are :

Parameters

	V	matrix (with the convention V(i,j)=V[i+j∗n])
	blocksize	row dimension of V
	m	column dimension of V
	C_fft	complex array used for FFTs (FFT of the Toeplitz matrix)
out	*CV*	product of the circulant matrix C_fft by the matrix V_rfft
	V_fft	complex array used for FFTs
	V_rfft	real array used for FFTs
	nfft	number of simultaneous FFTs
	plan_f_V	fftw plan forward (r2c)
	plan_b_CV	fftw plan backward (c2r)

Definition at line 587 of file toeplitz.c.

13.16.2.8 int stmm_core (double ∗∗ *V*, int *n*, int *m*, double ∗ *T*, fftw_complex ∗ *T_fft*, int *blocksize*, int *lambda*, fftw_complex ∗ *V_fft*, double ∗ *V_rfft*, int *nfft*, fftw_plan *plan_f*, fftw_plan *plan_b*, int *flag_offset*, int *flag_nofft*)

Performs the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm. (an INTERNAL routine)

The product is performed block-by-block with a defined block size or a computed optimized block size that reflects a trade off between cost of a single FFT of a length block_size and a number of blocks needed to perform the mutiplicaton. The latter determines how many spurious values are computed extra due to overlaps between the blocks. Use flag_offset=0 for "classic" algorithm and flag_offset=1 to put an offset to avoid the first and last lambdas terms. Usefull when a reshaping was done before with optimal column for a nfft. Better be inside the arguments of the routine. The parameters are:

Parameters

V	[input] data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
n	number of rows of V
m	number of columns of V
T	Toeplitz matrix data composed of the non-zero entries of his first row
T_fft	complex array used for FFTs
blocksize	block size used in the sliding window algorithm
lambda	Toeplitz band width
V_fft	complex array used for FFTs
V_rfft	real array used for FFTs
nfft	number of simultaneous FFTs
plan_f	fftw plan forward (r2c)
plan_b	fftw plan backward (c2r)
flag_offset	flag to avoid extra 2∗lambda padding to zeros on the edges
flag_nofft	flag to do product without using fft

Definition at line 659 of file toeplitz.c.

13.16.2.9 int stmm_main (double ** V, int n, int m, int id0, int l, double * T, fftw_complex * T_fft, int lambda, fftw_complex * V_fft, double * V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy)

Performs the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping. (an INTERNAL routine)

The input matrix is formatted into an optimized matrix depending on the block size and the number of simultaneous ffts (defined with the variable nfft). The obtained number of columns represent the number of vectors FFTs of which are computed simulatenously. The multiplication is then performed block-by-block with the chosen block size using the core routine. The parameters are :

Parameters

V	[input] data matrix (with the convention V(i,j)=V[i+j*n]) ; [out] result of the product TV
n	number of rows of V
m	number of columns of V
id0	first index of V
l	length of V
T	Toeplitz matrix data composed of the non-zero entries of his first row
T_fft	complex array used for FFTs
lambda	Toeplitz band width
V_fft	complex array used for FFTs
V_rfft	real array used for FFTs
plan_f	fftw plan forward (r2c)
plan_b	fftw plan backward (c2r)
blocksize	block size
nfft	number of simultaneous FTTs
flag_stgy	flag strategy for the product computation

Definition at line 803 of file toeplitz.c.

13.16.2.10 int build_gappy_blocks (int nrow, int m, Block * tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t * id0gap, int * lgap, int ngap, Block * tpltzblocks_gappy, int * nb_blocks_gappy_final, int flag_param_distmin_fixed)

Build the gappy Toeplitz block structure to optimise the product computation at gaps location.

Considering the significant gaps, the blocks to which they belong are cut and split between the gap's edges to reduce the total row size of the flotting blocks. It take into consideration the minimum correlation length and a parameter that allows us to control the minimum gap size allowed to split the blocks. In some cases, the gap can be partially reduce to fit the minimum block size needed for computation or just for performance criteria. This is based on the fact that the gaps are previously set to zeros before calling this routine.

Parameters

nrow	number of rows of the global data matrix V
m	number of columns for the data matrix V in the global rowwise order
tpltzblocks	list of the toeplitz blocks struture with its own parameters (idv, n, T_block, lambda).
nb_blocks_local	number of Toeplitz blocks as stored in T
nb_blocks_all	number of all Toeplitz block on the diagonal of the full Toeplitz matrix
id0gap	index of the first element of each defined gap
lgap	length of each defined gaps
ngap	number of defined gaps
tpltzblocks_gappy	list of the gappy toeplitz blocks struture with its own parameters
nb_blocks_gappy_final	real number of obtained gappy Toeplitz blocks
flag_param_distmin_fixed	flag to defined the minimum gap value allowed to split a Toeplitz block

Definition at line 231 of file toeplitz_gappy.c.

13.16.2.11 int stmm_simple_basic (double ** V, int n, int m, double * T, int lambda, double ** TV)

Perform the product of a Toeplitz matrix by a matrix without using FFT's.

This routine multiplies the values directly between them. This exploit the fact that the bandwith is small compared to the matrix size. The number of operation is then no more than (lambda*2-1) multiplications and (lambda*2-1)-1 additions per row.

Definition at line 73 of file toeplitz_nofft.c.

13.16.2.12 int stmm_simple_core (double ** V, int n, int m, double * T, int blocksize, int lambda, int nfft, int flag_offset)

Perform the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm.

The product is performed block-by-block with a defined block size or a computed optimized blocksize. This routine is not used by th API.

Parameters

V	[input] data matrix (with the convention V(i,j)=V[i+j*n]) ; [out] result of the product TV
n	number of rows of V
m	number of columns of V
T	Toeplitz matrix data composed of the non-zero entries of his first row
blocksize	block size used in the sliding window algorithm
lambda	Toeplitz band width
nfft	number of simultaneous FFTs
flag_offset	flag to avoid extra 2*lambda padding to zeros on the edges

Definition at line 128 of file toeplitz_nofft.c.

13.16.2.13 int flag_stgy_init_auto (Flag * flag_stgy)

Set the flag to automatic paramaters.

Parameters

flag_stgy	flag strategy for the product computation

Definition at line 73 of file toeplitz_params.c.

13.16.2.14 int flag_stgy_init_zeros (Flag * flag_stgy)

Set the flag parameters to zeros. This is almost the same as automatic.

Parameters

flag_stgy	flag strategy for the product computation

Definition at line 91 of file toeplitz_params.c.

13.16.2.15 int flag_stgy_init_defined (Flag * flag_stgy)

Set the parameters flag to the defined ones.

Parameters

flag_stgy	flag strategy for the product computation

Definition at line 106 of file toeplitz_params.c.

13.16.2.16 int print_flag_stgy_init (Flag *flag_stgy*)

Print the flag parameters values.

Parameters

flag_stgy	flag strategy for the product computation

Definition at line 131 of file toeplitz_params.c.

13.17 lower internal routines

Functions

- int print_error_message (int error_number, char const *file, int line)

  Prints error message corresponding to an error number.
- int copy_block (int ninrow, int nincol, double *Vin, int noutrow, int noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int outrow, int outcol, double norm, int set_zero_flag)

  Copies (and potentially reshapes) a selected block of the input matrix to a specified position of the output matrix.
- int mpi_stbmm (double **V, int64_t nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int local_V_size, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.
- int gap_reduce (double **V, int id0, int l, int lambda, int *id0gap, int *lgap, int ngap, int *newl, int id0out)

  ...convert the data vector structure into a matrix structure optimized for nfft

13.17.1 Detailed Description

These are lower internal routines.

13.17.2 Function Documentation

13.17.2.1 int print_error_message (int *error_number,* char const * *file,* int *line*)

Prints error message corresponding to an error number.

Parameters

error_number	error number
file	file name
line	line number

Definition at line 93 of file toeplitz.c.

13.17.2.2 int copy_block (int *ninrow,* int *nincol,* double * *Vin,* int *noutrow,* int *noutcol,* double * *Vout,* int *inrow,* int *incol,* int *nblockrow,* int *nblockcol,* int *outrow,* int *outcol,* double *norm,* int *set_zero_flag*)

Copies (and potentially reshapes) a selected block of the input matrix to a specified position of the output matrix.

Copy a matrix block of a size nblockrow x nblockcol from the input matrix Vin (size ninrow x nincol) starting with the element (inrow, incol) to the output matrix Vout (size notrow x noutcol) starting with the element (outrow, outcol) after multiplying by norm. If the output matrix is larger than the block the extra elements are either left as they were on the input or zeroed if zero_flag is set to 1. If the block to be copied is larger than either the input or the output matrix an error occurs.

Definition at line 459 of file toeplitz.c.

13.17.2.3 int mpi_stbmm (double ** *V,* int64_t *nrow,* int *m,* int *m_rowwise,* Block * *tpltzblocks,* int *nb_blocks_local,* int *nb_blocks_all,* int64_t *idp,* int *local_V_size,* Flag *flag_stgy,* MPI_Comm *comm*)

Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

Each process performs the multiplication sequentially for each diagonal block and based on the sliding window algorithm. Prior to that MPI calls are used to exchange data between neighboring process. Each of the diagonal

blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block. The parameters are :

Parameters

V	[input] distributed data matrix (with the convention V(i,j)=V[i+j∗n]) ; [out] result of the product TV
nrow	number of rows of the global data matrix V
m	number of columns for the data matrix V in the global rowwise order
m_rowwise	number of columns for the data matrix V in the rowwise order per processor
tpltzblocks	list of the toeplitz blocks struture with its own parameters (idv, n, T_block, lambda) : • idv is the global row index defining for each Toeplitz block as stored in the vector T ; • n size of each Toeplitz block • T_block location of each Toeplitz matrix data composed of the non-zero entries of the first row ; • lambda size of each Toeplitz block data T_block. The bandwith size is then equal to lambda∗2-1
nb_blocks_all	number of all Toeplitz block on the diagonal of the full Toeplitz matrix
nb_blocks_local	number of Toeplitz blocks as stored in T
idp	global index of the first element of the local part of V
local_V_size	a number of all elements in local V
flag_stgy	flag strategy for the product computation
comm	MPI communicator

Select the useful flotting blocks for the local data of the current processor. parameters (idpnew, local_V_size_new, nnew) for the computation. ide the local range are set with a size nnew equal to zero.

```
This compute the right parameters (idpnew, local_V_size_new, nnew) for the computation.
All the block outside the local range are set with a size nnew equal to zero. local_V_size_new
correspond to the size without the shift between the global rank index and the global index of
the first flotting block. idnew is then set to the index of this first flotting block.
```

Definition at line 87 of file toeplitz_block.c.

13.17.2.4 int gap_reduce (double ∗∗ *V,* int *id0,* int *l,* int *lambda,* int ∗ *id0gap,* int ∗ *lgap,* int *ngap,* int ∗ *newl,* int *id0out*)

...convert the data vector structure into a matrix structure optimized for nfft

....Copy the data vector structure into an equivalent matrix with nfft column. Thus, the obtained matrix is optimize for the nfft multithreading algorithm use. The middle part is a direct copy of the data vector and we copy on the edges of each column the lambda terms needed to fullfill the correlation of theses data.

Definition at line 163 of file toeplitz_gappy_seq.dev.c.

Chapter 14

Data Structure Documentation

14.1 Block Struct Reference

`#include <toeplitz.h>`

Data Fields

- int64_t idv
- double * T_block
- int lambda
- int n

14.1.1 Detailed Description

Definition at line 145 of file toeplitz.h.

14.1.2 Field Documentation

14.1.2.1 int64_t Block::idv

Definition at line 146 of file toeplitz.h.

14.1.2.2 double* Block::T_block

Definition at line 147 of file toeplitz.h.

14.1.2.3 int Block::lambda

Definition at line 148 of file toeplitz.h.

14.1.2.4 int Block::n

Definition at line 149 of file toeplitz.h.

The documentation for this struct was generated from the following file:

- toeplitz.h

14.2 CMat Struct Reference

Matrix structure

A∗ = (A0∗ | A1∗ | ... | Ap-1∗)

`#include <mapmatc.h>`

Data Fields

- int flag
- int r
- int ∗ m
- int ∗ nnz
- int ∗ disp
- int ∗∗ indices
- double ∗∗ values
- int lcount
- int ∗ lindices
- MPI_Comm comm
- int ∗ com_indices
- int com_count
- int steps
- int ∗ nS
- int ∗ nR
- int ∗∗ R
- int ∗∗ S

14.2.1 Detailed Description

Matrix structure

A∗ = (A0∗ | A1∗ | ... | Ap-1∗)

Definition at line 23 of file mapmatc.h.

14.2.2 Field Documentation

14.2.2.1 int CMat::flag

Definition at line 24 of file mapmatc.h.

14.2.2.2 int CMat::r

Definition at line 25 of file mapmatc.h.

14.2.2.3 int∗ CMat::m

Definition at line 26 of file mapmatc.h.

14.2.2.4 int∗ CMat::nnz

Definition at line 27 of file mapmatc.h.

14.2.2.5 int* CMat::disp

Definition at line 28 of file mapmatc.h.

14.2.2.6 int** CMat::indices

Definition at line 29 of file mapmatc.h.

14.2.2.7 double** CMat::values

Definition at line 30 of file mapmatc.h.

14.2.2.8 int CMat::lcount

Definition at line 32 of file mapmatc.h.

14.2.2.9 int* CMat::lindices

Definition at line 33 of file mapmatc.h.

14.2.2.10 MPI_Comm CMat::comm

Definition at line 35 of file mapmatc.h.

14.2.2.11 int* CMat::com_indices

Definition at line 37 of file mapmatc.h.

14.2.2.12 int CMat::com_count

Definition at line 37 of file mapmatc.h.

14.2.2.13 int CMat::steps

Definition at line 38 of file mapmatc.h.

14.2.2.14 int* CMat::nS

Definition at line 39 of file mapmatc.h.

14.2.2.15 int * CMat::nR

Definition at line 39 of file mapmatc.h.

14.2.2.16 int** CMat::R

Definition at line 40 of file mapmatc.h.

14.2.2.17 int ** CMat::S

Definition at line 40 of file mapmatc.h.

The documentation for this struct was generated from the following file:

- mapmatc.h

14.3 Flag Struct Reference

`#include <toeplitz.h>`

Data Fields

- int flag_bs
- int flag_nfft
- int flag_fftw
- int flag_no_rshp
- int flag_nofft
- int flag_blockingcomm
- int fixed_nfft
- int fixed_bs
- int flag_verbose
- int flag_skip_build_gappy_blocks
- int flag_param_distmin_fixed
- int flag_precompute_lvl

14.3.1 Detailed Description

Definition at line 163 of file toeplitz.h.

14.3.2 Field Documentation

14.3.2.1 int Flag::flag_bs

Definition at line 164 of file toeplitz.h.

14.3.2.2 int Flag::flag_nfft

Definition at line 165 of file toeplitz.h.

14.3.2.3 int Flag::flag_fftw

Definition at line 166 of file toeplitz.h.

14.3.2.4 int Flag::flag_no_rshp

Definition at line 167 of file toeplitz.h.

14.3.2.5 int Flag::flag_nofft

Definition at line 168 of file toeplitz.h.

14.3.2.6 int Flag::flag_blockingcomm

Definition at line 169 of file toeplitz.h.

14.3.2.7 int Flag::fixed_nfft

Definition at line 170 of file toeplitz.h.

14.3.2.8 int Flag::fixed_bs

Definition at line 171 of file toeplitz.h.

14.3.2.9 int Flag::flag_verbose

Definition at line 172 of file toeplitz.h.

14.3.2.10 int Flag::flag_skip_build_gappy_blocks

Definition at line 173 of file toeplitz.h.

14.3.2.11 int Flag::flag_param_distmin_fixed

Definition at line 174 of file toeplitz.h.

14.3.2.12 int Flag::flag_precompute_lvl

Definition at line 175 of file toeplitz.h.

The documentation for this struct was generated from the following file:

- toeplitz.h

14.4 Gap Struct Reference

`#include <toeplitz.h>`

Data Fields

- int64_t * id0gap
- int * lgap
- int ngap

14.4.1 Detailed Description

Definition at line 178 of file toeplitz.h.

14.4.2 Field Documentation

14.4.2.1 int64_t* Gap::id0gap

Definition at line 179 of file toeplitz.h.

14.4.2.2 int* Gap::lgap

Definition at line 180 of file toeplitz.h.

14.4.2.3 int Gap::ngap

Definition at line 181 of file toeplitz.h.

The documentation for this struct was generated from the following file:

- toeplitz.h

14.5 Mat Struct Reference

Matrix structure

A* = (A0* | A1* | ... | Ap-1*)

```
#include <mapmat.h>
```

Data Fields

- int flag
- int m
- int nnz
- int * indices
- double * values
- int lcount
- int * lindices
- MPI_Comm comm
- int * com_indices
- int com_count
- int steps
- int * nS
- int * nR
- int ** R
- int ** S

14.5.1 Detailed Description

Matrix structure

A* = (A0* | A1* | ... | Ap-1*)

Definition at line 34 of file mapmat.h.

14.5.2 Field Documentation

14.5.2.1 int Mat::flag

Definition at line 35 of file mapmat.h.

14.5.2.2 int Mat::m

Definition at line 36 of file mapmat.h.

14.5.2.3 int Mat::nnz

Definition at line 37 of file mapmat.h.

14.5.2.4 int* Mat::indices

Definition at line 38 of file mapmat.h.

14.5.2.5 double* Mat::values

Definition at line 39 of file mapmat.h.

14.5.2.6 int Mat::lcount

Definition at line 41 of file mapmat.h.

14.5.2.7 int* Mat::lindices

Definition at line 42 of file mapmat.h.

14.5.2.8 MPI_Comm Mat::comm

Definition at line 44 of file mapmat.h.

14.5.2.9 int* Mat::com_indices

Definition at line 46 of file mapmat.h.

14.5.2.10 int Mat::com_count

Definition at line 46 of file mapmat.h.

14.5.2.11 int Mat::steps

Definition at line 47 of file mapmat.h.

14.5.2.12 int* Mat::nS

Definition at line 48 of file mapmat.h.

14.5.2.13 int * Mat::nR

Definition at line 48 of file mapmat.h.

14.5.2.14 int** Mat::R

Definition at line 49 of file mapmat.h.

14.5.2.15 int ** Mat::S

Definition at line 49 of file mapmat.h.

The documentation for this struct was generated from the following file:

- mapmat.h

14.6 Tpltz Struct Reference

`#include <toeplitz.h>`

Data Fields

- int64_t nrow
- int m_cw
- int m_rw
- Block * tpltzblocks
- int nb_blocks_loc
- int nb_blocks_tot
- int64_t idp
- int local_V_size
- Flag flag_stgy
- MPI_Comm comm

14.6.1 Detailed Description

Definition at line 185 of file toeplitz.h.

14.6.2 Field Documentation

14.6.2.1 int64_t Tpltz::nrow

Definition at line 186 of file toeplitz.h.

14.6.2.2 int Tpltz::m_cw

Definition at line 187 of file toeplitz.h.

14.6.2.3 int Tpltz::m_rw

Definition at line 188 of file toeplitz.h.

14.6.2.4 Block* Tpltz::tpltzblocks

Definition at line 189 of file toeplitz.h.

14.6.2.5 int Tpltz::nb_blocks_loc

Definition at line 190 of file toeplitz.h.

14.6.2.6 int Tpltz::nb_blocks_tot

Definition at line 191 of file toeplitz.h.

14.6.2.7 int64_t Tpltz::idp

Definition at line 192 of file toeplitz.h.

14.6.2.8 int Tpltz::local_V_size

Definition at line 193 of file toeplitz.h.

14.6.2.9 Flag Tpltz::flag_stgy

Definition at line 194 of file toeplitz.h.

14.6.2.10 MPI_Comm Tpltz::comm

Definition at line 195 of file toeplitz.h.

The documentation for this struct was generated from the following file:

- toeplitz.h

Chapter 15

File Documentation

15.1 alm.c File Reference

Implementation of subroutines handling maps, distributions or functions. That means, almost all structures describes as sets of indices associated to sets of values).

Functions

- void m2s (double *mapval, double *submapval, int *subset, int count)

  Set some map values into a submap values array
- void lmatvecprod (int *ind, double *val, int m, int nnz, double *in, double *out)

  Local mat vec prod.
- void s2m_sum (double *mapval, double *submapval, int *subset, int count)

  Sum submap values the submap values array.
- void s2m (double *mapval, double *submapval, int *subset, int count)

  assign submap values the submap values array
- void cnt_nnz_dot_prod (double *out, double *in, int cnt, int *ind, double *val, int nnz)

  Sum submap values the submap values array.
- void omp_cnt_nnz_dot_prod (double *out, double *in, int cnt, int *ind, double *val, int nnz)

  Sum submap values the submap values array.
- int m2m (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)
- int m2m_sum (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)

15.1.1 Detailed Description

Implementation of subroutines handling maps, distributions or functions. That means, almost all structures describes as sets of indices associated to sets of values).

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

April 2012

Definition in file alm.c.

15.1.2 Function Documentation

15.1.2.1 void m2s (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

Set some map values into a submap values array

Set some map values into a submap values array

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 16 of file alm.c.

15.1.2.2 void lmatvecprod (int * *ind*, double * *val*, int *m*, int *nnz*, double * *in*, double * *out*)

Local mat vec prod.

Parameters

indm	table of integers apval array of values
submapval	array of values

Returns

void

Definition at line 29 of file alm.c.

15.1.2.3 void s2m_sum (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

Sum submap values the submap values array.

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 45 of file alm.c.

15.1.2.4 void s2m (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

assign submap values the submap values array

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 58 of file alm.c.

15.1.2.5 void cnt_nnz_dot_prod (double * *out*, double * *in*, int *cnt*, int * *ind*, double * *val*, int *nnz*)

Sum submap values the submap values array.

Returns

void

Definition at line 67 of file alm.c.

15.1.2.6 void omp_cnt_nnz_dot_prod (double * *out*, double * *in*, int *cnt*, int * *ind*, double * *val*, int *nnz*)

Sum submap values the submap values array.

Returns

void

Definition at line 78 of file alm.c.

15.2 alm.c

```
00001
00010 #include <stdlib.h>
00011
00016 void m2s(double *mapval, double *submapval, int *subset, int count){
00017   int i;
00018
00019 //#pragma omp parallel for
00020   for(i=0; i< count; i++){
00021     submapval[i]=mapval[subset[i]];
00022   }
00023 }
00024
00029 void lmatvecprod(int *ind, double *val, int m, int nnz, double *in,
      double *out){
00030   int i, j, k;
00031   k=0;
00032   for(i=0; i<m; i++){                              /*<local transform reduce*/
```

```
00033     for(j=0; j<nnz; j++){
00034       out[i]+=val[k]*in[ind[k]];
00035       k++;
00036     }
00037   }
00038 }
00039
00040
00045 void s2m_sum(double *mapval, double *submapval, int *subset, int count){
00046   int i;
00047 //#pragma omp parallel for
00048   for(i=0; i< count; i++){
00049     mapval[subset[i]] += submapval[i];
00050   }
00051 }
00052
00053
00058 void s2m(double *mapval, double *submapval, int *subset, int count){
00059   int i;
00060   for(i=0; i< count; i++){
00061     mapval[subset[i]] = submapval[i];
00062   }
00063 }
00064
00067 void cnt_nnz_dot_prod(double *out, double *in, int cnt, int *
      ind, double *val, int nnz){
00068   int i, j, k;
00069   k=0;
00070   for(i=0; i<cnt; i++)                                    /*<local transform
       reduce*/
00071     for(j=0; j<nnz; j++)
00072       out[ind[k]]+=val[k]*in[i];
00073 }
00074
00075 #if OPENMP
00076
00078 void omp_cnt_nnz_dot_prod(double *out, double *in, int cnt,
       int *ind, double *val, int nnz){
00079   int i, j, k;
00080   k=0;
00081   for(i=0; i<cnt; i++)                                    /*<local transform
       reduce*/
00082     for(j=0; j<nnz; j++)
00083       out[ind[k]]+=val[k]*in[i];
00084 }
00085 #endif
00086
00093 int m2m(double *vA1, int *A1, int n1, double *vA2, int *A2, int n2){
00094   int i=0, j=0, k= 0;
00095   while( i<n1 && j<n2){
00096     if(A1[i] < A2[j]){
00097       i++;
00098     }
00099     else if(A1[i] > A2[j]){
00100       j++;
00101     }
00102     else{
00103       vA2[j]=vA1[i];
00104       k++;
00105       i++;
00106       j++;
00107     }
00108   }
00109   return k;
00110 }
00111
00118 int m2m_sum(double *vA1, int *A1, int n1, double *vA2, int *A2, int n2){
00119   int i=0, j=0, k= 0;
00120   while( i<n1 && j<n2){
00121     if(A1[i] < A2[j]){
00122       i++;
00123     }
00124     else if(A1[i] > A2[j]){
00125       j++;
00126     }
00127     else{
00128       vA2[j]+=vA1[i];
00129       k++;
00130       i++;
00131       j++;
00132     }
00133   }
00134   return k;
00135 }
00136
```

15.3 alm.h File Reference

Declaration of subroutines handling maps (associated sets of indices and values).

Functions

- void m2s (double *mapval, double *submapval, int *subset, int count)
  Set some map values into a submap values array
- void s2m_sum (double *mapval, double *submapval, int *subset, int count)
  Sum submap values the submap values array.
- void s2m (double *mapval, double *submapval, int *subset, int count)
  assign submap values the submap values array
- void cnt_nnz_dot_prod (double *out, double *in, int cnt, int *ind, double *val, int nnz)
  Sum submap values the submap values array.
- void lmatvecprod (int *ind, double *val, int m, int nnz, double *in, double *out)
  Local mat vec prod.
- void omp_cnt_nnz_dot_prod (double *out, double *in, int cnt, int *ind, double *val, int nnz)
  Sum submap values the submap values array.
- void omp_lmatvecprod (int *ind, double *val, int m, int nnz, double *in, double *out)
- int m2m (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)
- int m2m_sum (double *vA1, int *A1, int n1, double *vA2, int *A2, int n2)

15.3.1 Detailed Description

Declaration of subroutines handling maps (associated sets of indices and values).

Note

Copyright (C) 2010 APC CNRS Université Paris Diderot
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/gpl.html

Author

Pierre Cargemel

Date

April 2012

Definition in file alm.h.

15.3.2 Function Documentation

15.3.2.1 void m2s (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

Set some map values into a submap values array

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Set some map values into a submap values array

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 16 of file alm.c.

15.3.2.2 void s2m_sum (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

Sum submap values the submap values array.

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 45 of file alm.c.

15.3.2.3 void s2m (double * *mapval*, double * *submapval*, int * *subset*, int *count*)

assign submap values the submap values array

Parameters

mapval	array of values
submapval	array of values

Returns

array of indices

Definition at line 58 of file alm.c.

15.3.2.4 void cnt_nnz_dot_prod (double * *out*, double * *in*, int *cnt*, int * *ind*, double * *val*, int *nnz*)

Sum submap values the submap values array.

Returns

void

Definition at line 67 of file alm.c.

15.3.2.5 void lmatvecprod (int * *ind*, double * *val*, int *m*, int *nnz*, double * *in*, double * *out*)

Local mat vec prod.

Parameters

indm	table of integers apval array of values
submapval	array of values

Returns

void

Definition at line 29 of file alm.c.

15.3.2.6 void omp_cnt_nnz_dot_prod (double * *out*, double * *in*, int *cnt*, int * *ind*, double * *val*, int *nnz*)

Sum submap values the submap values array.

Returns

void

Definition at line 78 of file alm.c.

15.3.2.7 void omp_lmatvecprod (int * *ind*, double * *val*, int *m*, int *nnz*, double * *in*, double * *out*)

15.4 alm.h

```
00001
00011 void m2s(double *mapval, double *submapval, int *subset, int count);
00012
00013 void s2m_sum(double *mapval, double *submapval, int *subset, int count);
00014 void s2m(double *mapval, double *submapval, int *subset, int count);
00015
00016 void cnt_nnz_dot_prod(double *out, double *in, int cnt, int *
      ind, double *val, int nnz);
00017 void lmatvecprod(int *ind, double *val, int m, int nnz, double *in,
      double *out);
00018
00019 #if OPENMP
00020 void omp_cnt_nnz_dot_prod(double *out, double *in, int cnt,
       int *ind, double *val, int nnz);
00021 void omp_lmatvecprod(int *ind, double *val, int m, int nnz,
      double *in, double *out);
00022 #endif
00023 int m2m(double *vA1, int *A1, int n1, double *vA2, int *A2, int n2);
00024
00025 int m2m_sum(double *vA1, int *A1, int n1, double *vA2, int *A2, int n2);
```

15.5 als.c File Reference

Implementation of subroutines handling sets of inidices (union, intersection, merge, cardinal ...)

Functions

- int card (int ∗A, int nA)
- void merge (int ∗A, int nA, int ∗B)
- int card_or (int ∗A1, int n1, int ∗A2, int n2)
- int card_and (int ∗A1, int n1, int ∗A2, int n2)
- int set_or (int ∗A1, int n1, int ∗A2, int n2, int ∗A1orA2)
- int set_and (int ∗A1, int n1, int ∗A2, int n2, int ∗A1andA2)
- int map_and (int ∗A1, int n1, int ∗A2, int n2, int ∗mapA1andA2)

  Compute map A1 and A2 / A1.
- void subset2map (int ∗A, int nA, int ∗subA, int nsubA)

15.5.1 Detailed Description

Implementation of subroutines handling sets of inidices (union, intersection, merge, cardinal ...)

Note

Copyright (c) 2010-2012 APC CNRS Université Paris Diderot. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/lgpl.html
For more information about ANR MIDAS'09 project see http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

April 2012

Definition in file als.c.

15.5.2 Function Documentation

15.5.2.1 int card (int ∗ *A*, int *nA*)

Compute cardinal(A) Perform a loop onto elements of a set, counting different elements. The array should be in ascending order with possible redundancy.

Parameters

n	number of elemnets in A
A	set of indices

Returns

number of elements in A

Definition at line 17 of file als.c.

15.5.2.2 void merge (int * *A*, int *nA*, int * *B*)

Merge redundant elements. Fill a new array containing elements of A, merging redundant elements. The array A should be in ascending order with possible redundancy. The array B has to be allocated.

Parameters

nA	number of elemnets in A
A	set of indices
B	output array

Returns

void

Definition at line 39 of file als.c.

15.5.2.3 int map_and (int * *A1*, int *n1*, int * *A2*, int *n2*, int * *mapA1andA2*)

Compute map A1 and A2 / A1.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices
address	to the set A1andA2

Returns

number of elements in A1andA2

Definition at line 188 of file als.c.

15.5.2.4 void subset2map (int * *A*, int *nA*, int * *subA*, int *nsubA*)

Transform a subset into a mapper array Parse a subset and replace element by his index in the larger set. A and subA should be two monotony sets in ascending ordered. subA has to belong to A.

Parameters

A	a set of indices(monotony)
nA	number of elemnets in A
subA	a subset of A(monotony)
nsubA	number of elemnets in A

Returns

void

Definition at line 217 of file als.c.

15.6 als.c

```
00001
00009 #include <stdlib.h>
00010
00017 int card(int *A, int nA){
00018   int i;
00019   int tmp=A[0];
00020   int c=1;
00021   for(i=1; i<nA; i++){
00022     if(A[i] != tmp){
00023       c++;
00024       tmp=A[i];
00025     }
00026   }
00027   return c;
00028 }
00029
00030
00039 void merge(int *A, int nA, int *B){
00040   int i=0, j=0;
00041   B[0]=A[0];
00042   for(i=1; i<nA; i++){
00043     if(A[i] != B[j]){
00044       j++;
00045       B[j]=A[i];
00046     }
00047   }
00048 }
00049
00050
00060 int card_or(int *A1, int n1, int *A2, int n2){
00061   int i=0, j=0, k= 0;
00062   while( i<n1 || j<n2){
00063     if(A1[i] < A2[j]){
00064       if(i<n1){ i++; }
00065       else{ j++; }
00066     }
00067     else if(A1[i] > A2[j]){
00068       if(j<n2){ j++; }
00069       else{ i++; }
00070     }
00071     else{
00072       if(i<n1){ i++; }
00073       if(j<n2){ j++; }
00074     }
00075     k++;
00076   }
00077   return k;
00078 }
00079
00089 int card_and(int *A1, int n1, int *A2, int n2){
00090   int i=0, j=0, k= 0;
00091   while( i<n1 && j<n2){
00092     if(A1[i] < A2[j]){
00093       i++;
00094     }
00095     else if(A1[i] > A2[j]){
00096       j++;
00097     }
00098     else{
00099       k++;
00100       i++;
00101       j++;
00102     }
00103   }
00104   return k;
00105 }
00106
00118 int set_or(int *A1, int n1, int *A2, int n2, int *A1orA2){
00119   int i=0, j=0, k= 0;
00120   while( i<n1 || j<n2){
00121     if(A1[i] < A2[j]){
00122       if(i<n1){
00123         A1orA2[k]=A1[i];
00124         i++;
```

```
00125       }
00126       else{
00127         A1orA2[k]=A2[j];
00128         j++;
00129       }
00130     }
00131     else if(A1[i] > A2[j]){
00132       if(j<n2){
00133         A1orA2[k]=A2[j];
00134         j++;
00135       }
00136       else{
00137         A1orA2[k]=A1[i];
00138         i++;
00139       }
00140     }
00141     else{
00142       A1orA2[k]=A1[i];
00143       i++;
00144       j++;
00145     }
00146     k++;
00147   }
00148   return k;
00149 }
00150
00162 int set_and(int *A1, int n1, int *A2, int n2, int *A1andA2){
00163   int i=0, j=0, k= 0;
00164   while( i<n1 && j<n2){
00165     if(A1[i] < A2[j]){
00166       i++;
00167     }
00168     else if(A1[i] > A2[j]){
00169       j++;
00170     }
00171     else{
00172       A1andA2[k]=A1[i];
00173       k++;
00174       i++;
00175       j++;
00176     }
00177   }
00178   return k;
00179 }
00180
00188 int map_and(int *A1, int n1, int *A2, int n2, int *mapA1andA2){
00189   int i=0, j=0, k= 0;
00190   while( i<n1 && j<n2){
00191     if(A1[i] < A2[j]){
00192       i++;
00193     }
00194     else if(A1[i] > A2[j]){
00195       j++;
00196     }
00197     else{
00198       mapA1andA2[k]=i;
00199       k++;
00200       i++;
00201       j++;
00202     }
00203   }
00204   return k;
00205 }
00206
00207
00217 void subset2map(int *A, int nA, int *subA, int nsubA){
00218   int i=0, j=0;
00219   while( i<nA && j<nsubA){
00220     if(A[i] < subA[j]){
00221       i++;
00222     }
00223     else{
00224       subA[j]=i;
00225       i++;
00226       j++;
00227     }
00228   }
00229 }
00230
```

15.7 als.h File Reference

Declaration of subroutines handling sets of indices or sets of values.

Functions

- int card (int ∗A, int nA)
- void merge (int ∗A, int nA, int ∗B)
- int card_or (int ∗A1, int n1, int ∗A2, int n2)
- int card_and (int ∗A1, int n1, int ∗A2, int n2)
- int map_and (int ∗A1, int n1, int ∗A2, int n2, int ∗mapA1andA2)

  Compute map A1 and A2 / A1.
- int set_or (int ∗A1, int n1, int ∗A2, int n2, int ∗A1orA2)
- int set_and (int ∗A1, int n1, int ∗A2, int n2, int ∗A1andA2)
- void subset2map (int ∗A, int nA, int ∗subA, int nsubA)

15.7.1 Detailed Description

Declaration of subroutines handling sets of indices or sets of values.

Note

Author

Pierre Cargemel

Date

April 2012

Definition in file als.h.

15.7.2 Function Documentation

15.7.2.1 int card (int ∗ *A*, int *nA*)

Compute cardinal(A) Perform a loop onto elements of a set, counting different elements. The array should be in ascending order with possible redundancy.

Parameters

n	number of elemnets in A
A	set of indices

Returns

number of elements in A

Definition at line 17 of file als.c.

15.7.2.2 void merge (int * *A*, int *nA*, int * *B*)

Merge redundant elements. Fill a new array containing elements of A, merging redundant elements. The array A should be in ascending order with possible redundancy. The array B has to be allocated.

Parameters

nA	number of elemnets in A
A	set of indices
B	output array

Returns

void

Definition at line 39 of file als.c.

15.7.2.3 int map_and (int * *A1*, int *n1*, int * *A2*, int *n2*, int * *mapA1andA2*)

Compute map A1 and A2 / A1.

Parameters

n1	number of elemnets in A1
A1	set of indices
n2	number of elemnets in A2
A2	set of indices
address	to the set A1andA2

Returns

number of elements in A1andA2

Definition at line 188 of file als.c.

15.7.2.4 void subset2map (int * *A*, int *nA*, int * *subA*, int *nsubA*)

Transform a subset into a mapper array Parse a subset and replace element by his index in the larger set. A and subA should be two monotony sets in ascending ordered. subA has to belong to A.

Parameters

A	a set of indices(monotony)
nA	number of elemnets in A
subA	a subset of A(monotony)
nsubA	number of elemnets in A

Returns

void

Definition at line 217 of file als.c.

15.8 als.h

```
00001
```

```
00008 int card(int *A, int nA);
00009
00010 void merge(int *A, int nA, int *B);
00011
00012 int card_or(int *A1, int n1, int *A2, int n2);
00013
00014 int card_and(int *A1, int n1, int *A2, int n2);
00015
00016 int map_and(int *A1, int n1, int *A2, int n2, int *mapA1andA2);
00017
00018 int set_or(int *A1, int n1, int *A2, int n2, int *A1orA2);
00019
00020 int set_and(int *A1, int n1, int *A2, int n2, int *A1andA2);
00021
00022 void subset2map(int *A, int nA, int *subA, int nsubA);
```

15.9 bitop.c File Reference

Functions

- int is_pow_2 (int n)
- int pow_2 (int k)
- int log_2 (int n)

15.9.1 Detailed Description

Note

Copyright (c) 2010-2012 APC CNRS Université Paris Diderot. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/lgpl.html

Definition in file bitop.c.

15.9.2 Function Documentation

15.9.2.1 int is_pow_2 (int *n*)

Definition at line 5 of file bitop.c.

15.9.2.2 int pow_2 (int *k*)

Definition at line 10 of file bitop.c.

15.9.2.3 int log_2 (int *n*)

Definition at line 20 of file bitop.c.

15.10 bitop.c

```
00001
00005 int is_pow_2(int n){
00006     return((n & -n) ^ n);
```

```
00007 }
00008
00009
00010 int pow_2(int k){
00011    int n=1;
00012    while(k!=0){
00013      n = n << 1;
00014      k--;
00015    }
00016    return n;
00017 }
00018
00019
00020 int log_2(int n){
00021    int k=0;
00022    while(n!=1 && k<32){
00023      n= n >> 1;
00024      k++;
00025    }
00026    return k;
00027 }
00028
```

15.11 bitop.h File Reference

Functions

- int is_pow_2 (int n)
- int pow_2 (int k)
- int log_2 (int n)

15.11.1 Function Documentation

15.11.1.1 int is_pow_2 (int *n*)

Definition at line 5 of file bitop.c.

15.11.1.2 int pow_2 (int *k*)

Definition at line 10 of file bitop.c.

15.11.1.3 int log_2 (int *n*)

Definition at line 20 of file bitop.c.

15.12 bitop.h

```
00001
00002
00003 int is_pow_2(int n);
00004
00005 int pow_2(int k);
00006
00007 int log_2(int n);
```

15.13 butterfly.c File Reference

Implementation of routines for butterfly-like communication scheme.

Functions

- int butterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.
- int butterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int butterfly_blocking_2instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int butterfly_blocking_1instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

15.13.1 Detailed Description

Implementation of routines for butterfly-like communication scheme.

Note

Copyright (c) 2010-2012 APC CNRS Université Paris Diderot. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/lgpl.html
For more information about ANR MIDAS'09 project see http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

April 2012

Definition in file butterfly.c.

15.14 butterfly.c

```
00001
00009 #ifdef W_MPI
00010 #include <mpi.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
```

```
00014
00037 int butterfly_init(int *indices, int count, int **R, int *nR, int
      **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm){
00038
00039   int i, k, p2k;
00040   int rank, size, rk, sk;
00041   int tag;
00042   MPI_Request s_request, r_request;
00043   int nbuf, *buf;
00044   int **I, *nI;
00045   int **J, *nJ;
00046
00047   MPI_Comm_size(comm, &size);
00048   MPI_Comm_rank(comm, &rank);
00049
00050   I = (int **) malloc(steps * sizeof(int*));
00051   nI = (int *) malloc(steps * sizeof(int));
00052   tag=0;
00053   p2k=size/2;
00054
00055   for(k=0; k<steps; k++){               //butterfly first pass : bottom up
      (fill tabs nI and I)
00056     sk=(rank+size-p2k)%size;
00057     rk=(rank+p2k)%size;
00058
00059     if(k==0){                                          //S^0 := A
00060       nS[k] = count;
00061       S[k] = (int *) malloc(nS[k] * sizeof(int));
00062       memcpy( S[k], indices, nS[k]*sizeof(int));
00063     }
00064     else{                                              //S^k := S^{k-1} \cup
      R^{k-1}
00065       nS[k] = card_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k]);
00066       S[k] = (int *) malloc(nS[k] * sizeof(int));
00067       set_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k], S[k]);
00068     }
00069
00070     MPI_Irecv(&nI[steps-k-1], 1, MPI_INT, rk, tag, comm, &r_request);   //
     receive number of indices
00071     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);           //send
      number of indices
00072     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00073     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00074
00075     I[steps-k-1]= (int *) malloc(nI[steps-k-1] * sizeof(int));
00076
00077     tag++;
00078     MPI_Irecv(I[steps-k-1], nI[steps-k-1], MPI_INT, rk, tag, comm, &r_request);
     //receive indices
00079     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request);
     //send indices
00080     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00081     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00082
00083     p2k/=2;
00084     tag++;
00085   }
00086
00087   J = (int **) malloc(steps * sizeof(int*));
00088   nJ = (int *) malloc(steps * sizeof(int));
00089
00090   tag=0;
00091   p2k=1;
00092   for(k=0; k<steps; k++){               //buuterfly second pass : top down
      (fill tabs nJ and J)
00093     free(S[k]);
00094     sk=(rank+p2k)%size;
00095     rk=(rank+size-p2k)%size;
00096     if(k==0){
00097       nJ[k] = count;
00098       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00099       memcpy( J[k], indices, nJ[k]*sizeof(int));
00100     }
00101     else{
00102       nJ[k] = card_or(J[k-1], nJ[k-1], R[k-1], nR[k-1]);
00103       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00104       set_or(J[k-1], nJ[k-1], R[k-1], nR[k-1], J[k]);  //J^k=R^k-1 \cup
      J^k-1
00105       free(R[k-1]);
00106     }
00107     if(k!=steps-1){
00108     MPI_Irecv(&nR[k], 1, MPI_INT, rk, tag, comm, &r_request);
00109     MPI_Isend(&nJ[k], 1, MPI_INT, sk, tag, comm, &s_request);
00110     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00111     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00112
00113     R[k]= (int *) malloc( nR[k] * sizeof(int));
```

```
00114     tag++;
00115
00116     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request);
00117     MPI_Isend(J[k], nJ[k], MPI_INT, sk, tag, comm, &s_request);
00118     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00119     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00120     }
00121     p2k*=2;
00122     tag++;
00123   }
00124
00125
00126   tag=0;
00127   p2k=1;
00128   for(k=0; k<steps; k++){               //butterfly last pass : know that
      Sending tab is S = I \cap J, so send S and we'll get R
00129     sk=(rank+p2k)%size;
00130     rk=(rank+size-p2k)%size;
00131
00132     nS[k] = card_and(I[k], nI[k], J[k], nJ[k]);
00133     S[k] = (int *) malloc(nJ[k] *sizeof(int));
00134     set_and( I[k], nI[k], J[k], nJ[k], S[k]);    //S^k=I^k \cap J^k
00135
00136     free(I[k]);
00137     free(J[k]);
00138
00139     MPI_Irecv(&nR[k],1, MPI_INT, rk, tag, comm, &r_request);    //receive size
00140     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);   //send size
00141     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00142     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00143
00144     R[k]= (int *) malloc( nR[k] * sizeof(int));
00145     tag++;
00146
00147     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request); //receive
      indices
00148     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request); //send indices
00149     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00150     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00151
00152     p2k*=2;
00153     tag++;
00154   }
00155
00156   //Now we work locally
00157   int **USR, *nUSR, **U, *nU;
00158
00159   USR = (int **) malloc(steps*sizeof(int *));
00160   nUSR = (int *) malloc(steps*sizeof(int));
00161   U = (int **) malloc(steps*sizeof(int *));
00162   nU = (int *) malloc(steps*sizeof(int));
00163
00164   for(k=0; k<steps; k++){
00165     nUSR[k] = card_or(S[k], nS[k], R[k], nR[k]);
00166     USR[k] = (int *) malloc(nUSR[k]*sizeof(int));
00167     set_or(S[k], nS[k], R[k], nR[k], USR[k]);
00168   }
00169   for(k=0; k<steps; k++){
00170     if(k==0){
00171       nU[k]=nUSR[k];
00172       U[k] = (int *) malloc(nU[k] * sizeof(int));
00173       memcpy( U[k], USR[k], nU[k]*sizeof(int));
00174     }
00175     else{
00176       nU[k] = card_or(U[k-1], nU[k-1], USR[k], nUSR[k]);
00177       U[k] = (int *) malloc(nU[k]*sizeof(int *));
00178       set_or(U[k-1], nU[k-1], USR[k], nUSR[k], U[k]);
00179     }
00180   }
00181   *com_count=nU[steps-1];
00182   *com_indices = (int *) malloc(*com_count * sizeof(int));
00183   memcpy(*com_indices, U[steps-1], *com_count * sizeof(int));
00184   //=====================================================================
00185
00186   for(k=0; k<steps; k++){
00187     subset2map(*com_indices, *com_count, S[k], nS[k]);
00188     subset2map(*com_indices, *com_count, R[k], nR[k]);
00189   }
00190   free(USR);
00191   free(U);
00192
00193   return 0;
00194 }
00195
00196
00209 int butterfly_reduce(int **R, int *nR, int nRmax, int **S, int
      *nS, int nSmax, double *val, int steps, MPI_Comm comm){
```

```
00210   //double st, t;
00211   //t=0.0;
00212   int k, p2k, tag;
00213   int rank, size, rk, sk;
00214   MPI_Request s_request, r_request;
00215   double *sbuf, *rbuf;
00216
00217   MPI_Comm_size(comm, &size);
00218   MPI_Comm_rank(comm, &rank);
00219
00220   sbuf = (double *) malloc(nSmax * sizeof(double));
00221   rbuf = (double *) malloc(nRmax * sizeof(double));
00222   tag=0;
00223   p2k=1;
00224
00225   for(k=0; k<steps; k++){
00226     //st=MPI_Wtime();
00227     rk=(rank+size-p2k)%size;
00228     MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00229     sk=(rank+p2k)%size;
00230     m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00231     MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00232     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00233     s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
      //nR[k] floating sum
00234     p2k*=2;
00235     tag++;
00236     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00237     //t=t+MPI_Wtime()-st;
00238   }
00239   free(sbuf);
00240   free(rbuf);
00241   return 0;
00242 }
00243
00244 //================================================Modification of the code by
      Sebastien Cayrols : 01/09/2015 ; Berkeley
00245
00258 int butterfly_blocking_2instr_reduce(int **R,
      int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm
      comm){
00259   //double st, t;
00260   //t=0.0;
00261   int k, p2k, tag;
00262   int rank, size, rk, sk;
00263   double *sbuf, *rbuf;
00264   MPI_Status status;
00265
00266   MPI_Comm_size(comm, &size);
00267   MPI_Comm_rank(comm, &rank);
00268
00269   sbuf = (double *) malloc(nSmax * sizeof(double));
00270   rbuf = (double *) malloc(nRmax * sizeof(double));
00271   tag=0;
00272   p2k=1;
00273
00274   for(k=0; k<steps; k++){
00275     //st=MPI_Wtime();
00276     sk=(rank+p2k)%size;
00277     m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00278     MPI_Send(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm);
00279     rk=(rank+size-p2k)%size;
00280     MPI_Recv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm,&status);
00281     s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
      //nR[k] floating sum
00282     p2k*=2;
00283     tag++;
00284     //t=t+MPI_Wtime()-st;
00285   }
00286   free(sbuf);
00287   free(rbuf);
00288   return 0;
00289 }
00290
00303 int butterfly_blocking_1instr_reduce(int **R,
      int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm
      comm){
00304   //double st, t;
00305   //t=0.0;
00306   int k, p2k, tag;
00307   int rank, size, rk, sk;
00308   double *sbuf, *rbuf;
00309   MPI_Status status;
00310
00311   MPI_Comm_size(comm, &size);
00312   MPI_Comm_rank(comm, &rank);
00313
```

```
00314   sbuf = (double *) malloc(nSmax * sizeof(double));
00315   rbuf = (double *) malloc(nRmax * sizeof(double));
00316   tag=0;
00317   p2k=1;
00318
00319   for(k=0; k<steps; k++){
00320     //st=MPI_Wtime();
00321     sk=(rank+p2k)%size;
00322     rk=(rank+size-p2k)%size;
00323     m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00324     MPI_Sendrecv(sbuf, nS[k], MPI_DOUBLE, sk, tag,rbuf, nR[k], MPI_DOUBLE, rk,
      tag, comm, &status);
00325     s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
       //nR[k] floating sum
00326     p2k*=2;
00327     tag++;
00328     //t=t+MPI_Wtime()-st;
00329   }
00330   free(sbuf);
00331   free(rbuf);
00332   return 0;
00333 }
00334 #endif
00335
```

15.15 butterfly.h File Reference

Declaration of routines for butterfly-like communication scheme.

Functions

- int butterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

- int butterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

- int butterfly_blocking_1instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

- int butterfly_blocking_2instr_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

- double butterfly_reduce_b (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int b, int steps, MPI_Comm comm)

15.15.1 Detailed Description

Declaration of routines for butterfly-like communication scheme.

Note

Copyright (C) 2010 APC CNRS Université Paris Diderot program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. should have received a copy of the GNU General Public License along with this program; if not, see http://www.-gnu.org/licenses/gpl.html

Author

Pierre Cargemel

Date

April 2012

Definition in file butterfly.h.

15.15.2 Function Documentation

15.15.2.1 double butterfly_reduce_b (int ** *R*, int * *nR*, int *nRmax*, int ** *S*, int * *nS*, int *nSmax*, double * *val*, int *b*, int *steps*, MPI_Comm *comm*)

15.16 butterfly.h

```
00001
00008 int butterfly_init(int *indices, int count, int **R, int *nR, int
       **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm);
00009
00010 int butterfly_reduce(int **R, int *nR, int  nRmax, int **S, int
       *nS, int nSmax, double *val, int steps, MPI_Comm comm);
00011
00012 int butterfly_blocking_1instr_reduce(int **R,
      int *nR, int  nRmax, int **S, int *nS, int nSmax, double *val, int steps,
      MPI_Comm comm);
00013 int butterfly_blocking_2instr_reduce(int **R,
      int *nR, int  nRmax, int **S, int *nS, int nSmax, double *val, int steps,
      MPI_Comm comm);
00014
00015 double butterfly_reduce_b(int **R, int *nR, int nRmax, int **
      S, int *nS, int nSmax, double *val, int b, int steps, MPI_Comm comm);
```

15.17 butterfly_extra.c File Reference

Functions

- int butterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

- int butterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_-Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

- int truebutterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)
- double truebutterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

15.17.1 Function Documentation

15.17.1.1 int truebutterfly_init (int * *indices*, int *count*, int ** *R*, int * *nR*, int ** *S*, int * *nS*, int ** *com_indices*, int * *com_count*, int *steps*, MPI_Comm *comm*)

Definition at line 251 of file butterfly_extra.c.

15.18 butterfly_extra.c

```
00001
00009 #ifdef W_MPI
00010 #include <mpi.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014
00037 int butterfly_init(int *indices, int count, int **R, int *nR, int
       **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm){
00038
00039   int i, k, p2k;
00040   int rank, size, rk, sk;
00041   int tag;
00042   MPI_Request s_request, r_request;
00043   int nbuf, *buf;
00044   int **I, *nI;
00045   int **J, *nJ;
00046
00047   MPI_Comm_size(comm, &size);
00048   MPI_Comm_rank(comm, &rank);
00049
00050   I = (int **) malloc(steps * sizeof(int*));
00051   nI = (int *) malloc(steps * sizeof(int));
00052   tag=0;
00053   p2k=size/2;
00054
00055   for(k=0; k<steps; k++){               //butterfly first pass : bottom up
       (fill tabs nI and I)
00056     sk=(rank+size-p2k)%size;
00057     rk=(rank+p2k)%size;
00058
00059     if(k==0){                                          //S^0 := A
00060       nS[k] = count;
00061       S[k] = (int *) malloc(nS[k] * sizeof(int));
00062       memcpy( S[k], indices, nS[k]*sizeof(int));
00063     }
00064     else{                                              //S^k := S^{k-1} \cup
       R^{k-1}
00065       nS[k] = card_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k]);
00066       S[k] = (int *) malloc(nS[k] * sizeof(int));
00067       set_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k], S[k]);
00068     }
00069
00070     MPI_Irecv(&nI[steps-k-1], 1, MPI_INT, rk, tag, comm, &r_request);   //
      receive number of indices
00071     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);           //send
       number of indices
00072     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00073     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00074
00075     I[steps-k-1]= (int *) malloc(nI[steps-k-1] * sizeof(int));
00076
00077     tag++;
00078     MPI_Irecv(I[steps-k-1], nI[steps-k-1], MPI_INT, rk, tag, comm, &r_request);
      //receive indices
00079     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request);
      //send indices
00080     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00081     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00082
00083     p2k/=2;
```

```
00084     tag++;
00085   }
00086
00087   J = (int **) malloc(steps * sizeof(int*));
00088   nJ = (int *) malloc(steps * sizeof(int));
00089
00090   tag=0;
00091   p2k=1;
00092   for(k=0; k<steps; k++){               //buuterfly second pass : top down
       (fill tabs nJ and J)
00093     free(S[k]);
00094     sk=(rank+p2k)%size;
00095     rk=(rank+size-p2k)%size;
00096     if(k==0){
00097       nJ[k] = count;
00098       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00099       memcpy( J[k], indices, nJ[k]*sizeof(int));
00100     }
00101     else{
00102       nJ[k] = card_or(J[k-1], nJ[k-1], R[k-1], nR[k-1]);
00103       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00104       set_or(J[k-1], nJ[k-1], R[k-1], nR[k-1], J[k]);  //J^k=R^k-1 \cup
       J^k-1
00105       free(R[k-1]);
00106     }
00107     if(k!=steps-1){
00108     MPI_Irecv(&nR[k], 1, MPI_INT, rk, tag, comm, &r_request);
00109     MPI_Isend(&nJ[k], 1, MPI_INT, sk, tag, comm, &s_request);
00110     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00111     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00112
00113     R[k]= (int *) malloc( nR[k] * sizeof(int));
00114     tag++;
00115
00116     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request);
00117     MPI_Isend(J[k], nJ[k], MPI_INT, sk, tag, comm, &s_request);
00118     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00119     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00120     }
00121     p2k*=2;
00122     tag++;
00123   }
00124
00125
00126   tag=0;
00127   p2k=1;
00128   for(k=0; k<steps; k++){               //butterfly last pass : know that
       Sending tab is S = I \cap J, so send S and we'll get R
00129     sk=(rank+p2k)%size;
00130     rk=(rank+size-p2k)%size;
00131
00132     nS[k] = card_and(I[k], nI[k], J[k], nJ[k]);
00133     S[k] = (int *) malloc(nJ[k] *sizeof(int));
00134     set_and( I[k], nI[k], J[k], nJ[k], S[k]);    //S^k=I^k \cap J^k
00135
00136     free(I[k]);
00137     free(J[k]);
00138
00139     MPI_Irecv(&nR[k],1, MPI_INT, rk, tag, comm, &r_request);    //receive size
00140     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);   //send size
00141     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00142     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00143
00144     R[k]= (int *) malloc( nR[k] * sizeof(int));
00145     tag++;
00146
00147     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request); //receive
       indices
00148     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request); //send indices
00149     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00150     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00151
00152     p2k*=2;
00153     tag++;
00154   }
00155
00156   //Now we work locally
00157   int **USR, *nUSR, **U, *nU;
00158
00159   USR = (int **) malloc(steps*sizeof(int *));
00160   nUSR = (int *) malloc(steps*sizeof(int));
00161   U = (int **) malloc(steps*sizeof(int *));
00162   nU = (int *) malloc(steps*sizeof(int));
00163
00164   for(k=0; k<steps; k++){
00165     nUSR[k] = card_or(S[k], nS[k], R[k], nR[k]);
00166     USR[k] = (int *) malloc(nUSR[k]*sizeof(int));
```

```
00167     set_or(S[k], nS[k], R[k], nR[k], USR[k]);
00168   }
00169   for(k=0; k<steps; k++){
00170     if(k==0){
00171       nU[k]=nUSR[k];
00172       U[k] = (int *) malloc(nU[k] * sizeof(int));
00173       memcpy( U[k], USR[k], nU[k]*sizeof(int));
00174     }
00175     else{
00176       nU[k] = card_or(U[k-1], nU[k-1], USR[k], nUSR[k]);
00177       U[k] = (int *) malloc(nU[k]*sizeof(int *));
00178       set_or(U[k-1], nU[k-1], USR[k], nUSR[k], U[k]);
00179     }
00180   }
00181   *com_count=nU[steps-1];
00182   *com_indices = (int *) malloc(*com_count * sizeof(int));
00183   memcpy(*com_indices, U[steps-1], *com_count * sizeof(int));
00184   //================================================================
00185
00186   for(k=0; k<steps; k++){
00187     subset2map(*com_indices, *com_count, S[k], nS[k]);
00188     subset2map(*com_indices, *com_count, R[k], nR[k]);
00189   }
00190   free(USR);
00191   free(U);
00192
00193  return 0;
00194 }
00195
00196
00209 double butterfly_reduce(int **R, int *nR, int nRmax, int **S,
      int *nS, int nSmax, double *val, int steps, MPI_Comm comm){
00210   double st, t;
00211   t=0.0;
00212   int k, p2k, tag;
00213   int rank, size, rk, sk;
00214   MPI_Request s_request, r_request;
00215   double *sbuf, *rbuf;
00216
00217   MPI_Comm_size(comm, &size);
00218   MPI_Comm_rank(comm, &rank);
00219
00220   sbuf = (double *) malloc(nSmax * sizeof(double));
00221   rbuf = (double *) malloc(nRmax * sizeof(double));
00222   tag=0;
00223   p2k=1;
00224
00225   for(k=0; k<steps; k++){
00226     sk=(rank+p2k)%size;
00227     rk=(rank+size-p2k)%size;
00228
00229     m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00230
00231     st=MPI_Wtime();
00232     MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00233     MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00234
00235     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00236     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00237
00238     t=t+MPI_Wtime()-st;
00239
00240     s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
00241
00242     p2k*=2;
00243     tag++;
00244
00245   }
00246   free(sbuf);
00247   free(rbuf);
00248   return t;
00249 }
00250
00251 int truebutterfly_init(int *indices, int count, int **R, int
      *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm
      comm){
00252
00253   int i, k, p2k, p2k1;
00254   int rank, size, rk, sk;
00255   int tag;
00256   MPI_Request s_request, r_request;
00257   int nbuf, *buf;
00258   int **I, *nI;
00259   int **J, *nJ;
00260
00261   MPI_Comm_size(comm, &size);
00262   MPI_Comm_rank(comm, &rank);
```

```
00263
00264   I = (int **) malloc(steps * sizeof(int*));
00265   nI = (int *) malloc(steps * sizeof(int));
00266   tag=0;
00267   p2k=size/2;
00268   p2k1=2*p2k;
00269
00270   for(k=0; k<steps; k++){               //butterfly first pass : bottom up
       (fill tabs nI and I)
00271
00272     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00273
00274     if(k==0){                                          //S^0 := A
00275       nS[k] = count;
00276       S[k] = (int *) malloc(nS[k] * sizeof(int));
00277       memcpy( S[k], indices, nS[k]*sizeof(int));
00278     }
00279     else{                                              //S^k := S^{k-1} \cup
       R^{k-1}
00280       nS[k] = card_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k]);
00281       S[k] = (int *) malloc(nS[k] * sizeof(int));
00282       set_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k], S[k]);
00283     }
00284
00285     MPI_Irecv(&nI[steps-k-1], 1, MPI_INT, rk, tag, comm, &r_request);   //
      receive number of indices
00286     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);           //send
       number of indices
00287     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00288     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00289
00290     I[steps-k-1]= (int *) malloc(nI[steps-k-1] * sizeof(int));
00291
00292     tag++;
00293     MPI_Irecv(I[steps-k-1], nI[steps-k-1], MPI_INT, rk, tag, comm, &r_request);
      //receive indices
00294     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request);
      //send indices
00295     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00296     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00297
00298     p2k/=2;
00299     p2k1/=2;
00300     tag++;
00301   }
00302
00303   J = (int **) malloc(steps * sizeof(int*));
00304   nJ = (int *) malloc(steps * sizeof(int));
00305
00306   tag=0;
00307   p2k=1;
00308   p2k1=p2k*2;
00309   for(k=0; k<steps; k++){               //buuterfly second pass : top down
       (fill tabs nJ and J)
00310     free(S[k]);
00311
00312     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00313
00314     if(k==0){
00315       nJ[k] = count;
00316       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00317       memcpy( J[k], indices, nJ[k]*sizeof(int));
00318     }
00319     else{
00320       nJ[k] = card_or(J[k-1], nJ[k-1], R[k-1], nR[k-1]);
00321       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00322       set_or(J[k-1], nJ[k-1], R[k-1], nR[k-1], J[k]);  //J^k=R^k-1 \cup
       J^k-1
00323       free(R[k-1]);
00324     }
00325     if(k!=steps-1){
00326     MPI_Irecv(&nR[k], 1, MPI_INT, rk, tag, comm, &r_request);
00327     MPI_Isend(&nJ[k], 1, MPI_INT, sk, tag, comm, &s_request);
00328     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00329     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00330
00331     R[k]= (int *) malloc( nR[k] * sizeof(int));
00332     tag++;
00333
00334     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request);
00335     MPI_Isend(J[k], nJ[k], MPI_INT, sk, tag, comm, &s_request);
00336     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00337     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00338     }
00339     p2k*=2;
00340     p2k1*=2;
00341     tag++;
```

```
00342   }
00343
00344
00345   tag=0;
00346   p2k=1;
00347   p2k1=p2k*2;
00348   for(k=0; k<steps; k++){               //butterfly last pass : know that
      Sending tab is S = I \cap J, so send S and we'll get R
00349
00350     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00351
00352     nS[k] = card_and(I[k], nI[k], J[k], nJ[k]);
00353     S[k] = (int *) malloc(nJ[k] *sizeof(int));
00354     set_and( I[k], nI[k], J[k], nJ[k], S[k]);    //S^k=I^k \cap J^k
00355
00356     free(I[k]);
00357     free(J[k]);
00358
00359     MPI_Irecv(&nR[k],1, MPI_INT, rk, tag, comm, &r_request);    //receive size
00360     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);   //send size
00361     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00362     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00363
00364     R[k]= (int *) malloc( nR[k] * sizeof(int));
00365     tag++;
00366
00367     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request); //receive
    indices
00368     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request); //send indices
00369     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00370     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00371
00372     p2k*=2;
00373     p2k1*=2;
00374     tag++;
00375   }
00376
00377   //Now we work locally
00378   int **USR, *nUSR, **U, *nU;
00379
00380   USR = (int **) malloc(steps*sizeof(int *));
00381   nUSR = (int *) malloc(steps*sizeof(int));
00382   U = (int **) malloc(steps*sizeof(int *));
00383   nU = (int *) malloc(steps*sizeof(int));
00384
00385   for(k=0; k<steps; k++){
00386     nUSR[k] = card_or(S[k], nS[k], R[k], nR[k]);
00387     USR[k] = (int *) malloc(nUSR[k]*sizeof(int));
00388     set_or(S[k], nS[k], R[k], nR[k], USR[k]);
00389   }
00390   for(k=0; k<steps; k++){
00391     if(k==0){
00392       nU[k]=nUSR[k];
00393       U[k] = (int *) malloc(nU[k] * sizeof(int));
00394       memcpy( U[k], USR[k], nU[k]*sizeof(int));
00395     }
00396     else{
00397       nU[k] = card_or(U[k-1], nU[k-1], USR[k], nUSR[k]);
00398       U[k] = (int *) malloc(nU[k]*sizeof(int *));
00399       set_or(U[k-1], nU[k-1], USR[k], nUSR[k], U[k]);
00400     }
00401   }
00402   *com_count=nU[steps-1];
00403   *com_indices = (int *) malloc(*com_count * sizeof(int));
00404   memcpy(*com_indices, U[steps-1], *com_count * sizeof(int));
00405   //====================================================================
00406
00407   for(k=0; k<steps; k++){
00408     subset2map(*com_indices, *com_count, S[k], nS[k]);
00409     subset2map(*com_indices, *com_count, R[k], nR[k]);
00410   }
00411   free(USR);
00412   free(U);
00413
00414  return 0;
00415 }
00416
00417
00430 double truebutterfly_reduce(int **R, int *nR, int nRmax,
     int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm){
00431   double st, t;
00432   t=0.0;
00433   int k, p2k, p2k1, tag;
00434   int rank, size, rk, sk;
00435   MPI_Status status;
00436   MPI_Request s_request, r_request;
00437   double *sbuf, *rbuf;
```

```
00438
00439   MPI_Comm_size(comm, &size);
00440   MPI_Comm_rank(comm, &rank);
00441
00442   sbuf = (double *) malloc(nSmax * sizeof(double));
00443   rbuf = (double *) malloc(nRmax * sizeof(double));
00444   tag=0;
00445   p2k=1;
00446   p2k1=p2k*2;
00447
00448   for(k=0; k<steps; k++){
00449
00450     if( rank%p2k1 < p2k){
00451
00452       sk=rk=rank+p2k;
00453
00454       st=MPI_Wtime();
00455
00456       //      MPI_Sendrecv(sbuf, nS[k], MPI_DOUBLE, sk, tag, rbuf, nR[k],
       MPI_DOUBLE, rk, tag, comm, &status);
00457
00458       m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00459       MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00460       MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00461
00462       MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00463       MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00464       s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
00465
00466
00467       t=t+MPI_Wtime()-st;
00468
00469     } else {
00470
00471       sk=rk=rank-p2k;
00472
00473       st=MPI_Wtime();
00474
00475       MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00476       m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00477       MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00478
00479       MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00480       s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
00481
00482       MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00483
00484       // MPI_Sendrecv(sbuf, nS[k], MPI_DOUBLE, sk, tag, rbuf, nR[k],
       MPI_DOUBLE, rk, tag, comm, &status);
00485
00486       t=t+MPI_Wtime()-st;
00487
00488     }
00489
00490     p2k*=2;
00491     p2k1*=2;
00492     tag++;
00493
00494   }
00495   free(sbuf);
00496   free(rbuf);
00497   return t;
00498 }
00499
00500 #endif
00501
00502
```

15.19 butterfly_extra.h File Reference

Functions

- int butterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between pro-

cessors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.

- double butterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- int truebutterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme (true means pair wise) This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.
- double truebutterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.
- double butterfly_reduce_b (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int b, int steps, MPI_Comm comm)

15.19.1 Function Documentation

15.19.1.1 double butterfly_reduce_b (int ** *R,* int * *nR,* int *nRmax,* int ** *S,* int * *nS,* int *nSmax,* double * *val,* int *b,* int *steps,* MPI_Comm *comm*)

15.20 butterfly_extra.h

```
00001
00008 int butterfly_init(int *indices, int count, int **R, int *nR, int
       **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm);
00009
00010 double butterfly_reduce(int **R, int *nR, int  nRmax, int **S,
      int *nS, int nSmax, double *val, int steps, MPI_Comm comm);
00011
00012 int truebutterfly_init(int *indices, int count, int **R, int
      *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm
      comm);
00013
00014 double truebutterfly_reduce(int **R, int *nR, int  nRmax,
      int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm);
00015
00016 double butterfly_reduce_b(int **R, int *nR, int nRmax, int **
      S, int *nS, int nSmax, double *val, int b, int steps, MPI_Comm comm);
```

15.21 cindex.c File Reference

Indexing subroutines implemetation.

Functions

- int sindex (int *T, int nT, int *A, int nA)
- int omp_pindex (int *T, int nT, int *A, int nA)
- int dichotomy (int nT, int *T, int e)

15.21.1 Detailed Description

Indexing subroutines implemetation.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

May 2012

Definition in file cindex.c.

15.21.2 Function Documentation

15.21.2.1 int dichotomy (int *nT,* int * *T,* int *e*)

dichotmic search of an integer in a monotony array

Parameters

number	elemnent array of values
monotony	array
element	to search

Returns

index of searched element

Definition at line 88 of file cindex.c.

15.22 cindex.c

```
00001
00010 #include <stdlib.h>
00018 int sindex(int *T, int nT, int *A, int nA){
00019   int i, tmp;
00020   i=0;
00021   for(i=0; i<nA; i++){
00022     tmp = A[i];
00023     A[i] =dichotomy(nT, T, tmp);
00024   }
00025 }
00026
00027
00028 #ifdef W_OPENMP
00029
00036 int omp_pindex(int *T, int nT, int *A, int nA){
00037 //  printf("\nomp_pindex");
00038   int i;
```

```
00039   int *count, *disp;
00040   int q, r;
00041   int tid, nths;
00042
00043   #pragma omp parallel private(tid) shared(nths)
00044   {//---fork---just to get the number of threads
00045     nths = omp_get_num_threads();
00046     tid = omp_get_thread_num();
00047 //    printf("\ntid %d nths %d", tid, nths);
00048   }//---join---
00049
00050   q = nA/nths;
00051   r = nA%nths;
00052
00053   count = (int *) malloc(nths *sizeof(int));
00054   disp = (int *) malloc(nths *sizeof(int));
00055
00056   for(i=0; i<nths; i++){
00057     if(i<r){
00058       count[i] = q+1;
00059     }
00060     else{
00061       count[i] = q;
00062     }
00063   }
00064
00065   disp[0] = 0;
00066   for(i=0; i<nths-1; i++){
00067     disp[i+1] = disp[i] + count[i];
00068   }
00069
00070   #pragma omp parallel private(tid) shared(T, nT, A, disp, count)
00071   {//---fork---1st step, sort on local chunk
00072     tid = omp_get_thread_num();
00073     sindex(T, nT, A+disp[tid], count[tid]);
00074   }//---join---
00075   free(count);
00076   free(disp);
00077   return 0;
00078 }
00079 #endif
00080
00081
00082
00088 int dichotomy(int nT, int *T, int e){
00089   int min, max, pivot;
00090   min=0;
00091   max=nT-1;
00092   pivot=(max-min)/2;
00093   while(e != T[pivot] && max > min ){
00094     if(T[pivot]<e){
00095       min=pivot+1;
00096     }
00097     else{
00098       max=pivot;
00099     }
00100     pivot= min + (max-min)/2;
00101   }
00102   return pivot;
00103 }
00104
```

15.23 cindex.h File Reference

Functions

- int sindex (int *array1, int nb1, int *array2, int nb2)
- int omp_pindex (int *array1, int nb1, int *array2, int nb2)

15.23.1 Detailed Description

Author

Pierre Cargemel

Date

May 2011

Definition in file cindex.h.

15.24 cindex.h

```
00001
00007 int sindex(int *array1, int nb1, int *array2, int nb2);
00008
00009 #if OPENMP
00010 int omp_pindex(int *array1, int nb1, int *array2, int nb2);
00011 #endif
```

15.25 csort.c File Reference

subroutines for sequential or parallel sort and/or merge sets of integer.

Functions

- void insertion_sort (int *indices, int count)
- void quick_sort (int *indices, int left, int right)
- void bubble_sort (int *indices, int count)
- int counting_sort (int *indices, int count)
- void shell_sort (int *a, int n)
- int ssort (int *indices, int count, int flag)
- int omp_psort_opt (int *A, int nA, int flag)
- int omp_psort (int *A, int nA, int flag)
- int sorted (int *indices, int count)

  ==========Checker Routines==========
- int monotony (int *indices, int count)

15.25.1 Detailed Description

subroutines for sequential or parallel sort and/or merge sets of integer.

Note

Copyright (c) 2010-2012 APC CNRS Université Paris Diderot. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/lgpl.html
For more information about ANR MIDAS'09 project see http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

April 2012

Definition in file csort.c.

15.25.2 Function Documentation

15.25.2.1 void insertion_sort (int * *indices*, int *count*)

Insertion sort : complexity is n square; ascending order

Parameters

indices	array of integer
count	number of elements in indices array

Returns

void

Definition at line 18 of file csort.c.

15.25.2.2 void quick_sort (int * *indices*, int *left*, int *right*)

Quick sort : complexity n square (Average is n log(n)). Sort in ascending order indices between left index and right index. Notice that this algorithm uses recursive calls, and can lead to stack overflow.

Parameters

indices	array of integer
first	element number
last	element number

Returns

void

Definition at line 40 of file csort.c.

15.25.2.3 void bubble_sort (int * *indices*, int *count*)

Bubble sort : complexity n square

Parameters

indices	array of integer
count	number of elements in indices array

Returns

void

Definition at line 73 of file csort.c.

15.25.2.4 int counting_sort (int * *indices*, int *count*)

Counting sort : sort indices in strictly ascending order (monotony). If the array initially ows reundants elements, they will be merged. Thus the returned number of sorted elements is less than the initial number of elements. Assuming elements belong to [min, max], complexity is n+(max-min+1). Due to allocation, memory overhead is (max-min+1)sizeof(element)

Parameters

indices	array of integer
count	number of elements in indices array

Returns

number of sorted elements

Definition at line 95 of file csort.c.

15.25.2.5 void shell_sort (int * *a*, int *n*)

Shell sort

Parameters

indices	array of integer
count	number of elements in indices array

Returns

void

Definition at line 129 of file csort.c.

15.25.2.6 int omp_psort_opt (int * *A*, int *nA*, int *flag*)

Definition at line 197 of file csort.c.

15.25.2.7 int sorted (int * *indices*, int *count*)

==========Checker Routines==========

Definition at line 364 of file csort.c.

15.25.2.8 int monotony (int * *indices*, int *count*)

Definition at line 377 of file csort.c.

15.26 csort.c

```
00001
00008 #include <stdlib.h>
00009 #include <string.h>
00010 //int per page size
00011 #define PAGE 1024
00012
00018 void insertion_sort(int *indices, int count){
```

```
00019   int i, j;
00020   int tmp;
00021   for(i=0; i<count-1 ; i++){
00022     tmp = indices[i+1];
00023     j=i;
00024     while(j != -1 && tmp < indices[j]){
00025       indices[j+1] = indices[j];
00026       indices[j]=tmp;
00027       j--;
00028     }
00029   }
00030 }
00031
00040 void quick_sort(int *indices, int left, int right){
00041   int pivot;
00042   int tmp, key;
00043   int i,j;
00044   if (left<right){
00045     key=indices[left];
00046     i=left+1;
00047     j=right;
00048     while(i<=j){
00049       while((i<=right) && (indices[i]<=key))  i++;
00050       while ((j>left) && (indices[j]>key))   j--;
00051       if(i<j){
00052         tmp=indices[i];
00053         indices[i] = indices[j];
00054         indices[j] = tmp;
00055         i++;
00056         j--;
00057       }
00058     }
00059     tmp=indices[left];
00060     indices[left] = indices[j];
00061     indices[j] = tmp;
00062     pivot = j;
00063     quick_sort(indices, left, pivot-1);
00064     quick_sort(indices, pivot+1, right);
00065   }
00066 }
00067
00073 void bubble_sort(int *indices, int count){
00074   int i, j, tmp;
00075   for (i=(count-1); i>0; i--){
00076     for (j = 1; j <= i; j++){
00077       if (indices[j-1] > indices[j]){
00078         tmp = indices[j-1];
00079         indices[j-1] = indices[j];
00080         indices[j] = tmp;
00081       }
00082     }
00083   }
00084 }
00085
00095 int counting_sort(int *indices, int count){
00096   int *buf;
00097   int i, j, k;
00098   int min, max;
00099   min=indices[0];
00100   max=indices[0];
00101   for(i=1; i<count ; i++){
00102     if(indices[i]>max){
00103       max=indices[i];
00104     }
00105     else{
00106      if(indices[i]<min)
00107        min=indices[i];
00108     }
00109   }
00110   buf = (int *) calloc(max-min+1, sizeof(int));
00111   for(i=0; i<count ; i++){
00112     buf[indices[i]-min]=1;
00113   }
00114   j=0;
00115   for(i=0; i<(max-min+1) ; i++){
00116     if(buf[i]==1){
00117       indices[j]=min+i;
00118       j++;
00119     }
00120   }
00121   free(buf);
00122   return j;
00123 }
00124
00129 void shell_sort(int *a,int n){
00130   int j,i,k,m,mid;
00131   for(m = n/2;m>0;m/=2){
```

```
00132     for(j = m;j< n;j++){
00133       for(i=j-m;i>=0;i-=m){
00134         if(a[i+m]>=a[i])
00135           break;
00136         else{
00137           mid = a[i];
00138           a[i] = a[i+m];
00139           a[i+m] = mid;
00140         }
00141       }
00142     }
00143   }
00144 }
00145
00146
00161 int ssort(int *indices, int count, int flag){
00162   int i, n;
00163   int *ptr_i, *ptr_o;
00164   switch(flag){
00165     case 0 :
00166       quick_sort(indices, 0, count-1);
00167       break;
00168     case 1 :
00169       bubble_sort(indices, count);
00170       break;
00171     case 2 :
00172       insertion_sort(indices, count);
00173       break;
00174     case 3 :
00175       n=counting_sort(indices, count);
00176       return n;
00177     case 4 :
00178       shell_sort(indices, count);
00179       break;
00180   }
00181   ptr_i = indices;
00182   ptr_o = indices;
00183   n=1;
00184   for(i=0; i<count-1; i++){
00185     ptr_i++;
00186     if(*ptr_i != *ptr_o){
00187       ptr_o++;
00188       n++;
00189       *ptr_o = *ptr_i;
00190     }
00191   }
00192   return n;
00193 }
00194
00195 //optimized version is faster than the other implementation but there is a bug
     !!!
00196 #ifdef W_OPENMP
00197 int omp_psort_opt(int *A, int nA, int flag){
00198   int i;
00199   int *count, *disp;
00200   int q, r;
00201   int p, l;
00202   int tid, nths;
00203   int *buf;
00204   int *ptr_i, *ptr_o;
00205   int n, k, d;
00206
00207   #pragma omp parallel shared(nths)
00208   {//---fork---just to get the number of threads
00209     nths = omp_get_num_threads();
00210   }//---join---
00211
00212   p = nA/PAGE; //number of full pages
00213   q = p/nths;   //full pages per thread
00214   l = p%nths;   //full pages left
00215   r = nA%PAGE;   //number of elements the last page not full
00216
00217   count = (int *) malloc(nths *sizeof(int));
00218   disp = (int *) malloc(nths *sizeof(int));
00219
00220   for(i=0; i<nths; i++){
00221     count[i] = q*PAGE;
00222     if(i<l)
00223       count[i] += PAGE;
00224     if(i==l)
00225       count[i] += r;
00226   }
00227
00228   disp[0] = 0;
00229   for(i=0; i<nths-1; i++){
00230     disp[i+1] = disp[i] + count[i];
00231   }
```

```
00232
00233   #pragma omp parallel private(tid, n, k, d, buf) shared(nths, A, nA, disp,
      count)
00234   {//---fork---1st step, sort on local chunk
00235     tid = omp_get_thread_num();
00236
00237     buf = (int *) malloc(nA*sizeof(int));
00238     //buf = (int *) malloc(count[tid]*sizeof(int));
00239     memcpy(buf, A+disp[tid], count[tid]*sizeof(int));
00240
00241     n = ssort(buf, count[tid], flag);
00242     count[tid]=n;
00243
00244     memcpy(A+disp[tid], buf, n*sizeof(int));
00245
00246     nths = omp_get_num_threads();
00247
00248
00249     k = nths;
00250     d = 1;
00251     while(k>1){
00252       #pragma omp barrier
00253       if(tid%(2*d)==0 && tid+d<nths){
00254         set_or(A+disp[tid], count[tid] , A+disp[tid+d], count[tid+d], buf
      );
00255         count[tid]=n;
00256         memcpy(A+disp[tid], buf, n*sizeof(int));
00257         d*=2;
00258         k/=2;
00259       }
00260     }
00261     free(buf);
00262   }//---join---
00263
00264   nA=count[0];
00265 //  printf("\nA :");
00266 //  for(i=0; i<nA; i++){
00267 //    printf(" %d",A[i]);
00268 //  }
00269   free(count);
00270   free(disp);
00271   return nA;
00272 }
00273
00291 int omp_psort(int *A, int nA, int flag){
00292   int i;
00293   int *count, *disp;
00294   int q, r;
00295   int tid, nths;
00296   int *buf;
00297   int *ptr_i, *ptr_o;
00298   int n, k, d;
00299
00300   #pragma omp parallel private(tid) shared(nths)
00301   {//---fork---just to get the number of threads
00302     nths = omp_get_num_threads();
00303   }//---join---
00304
00305   q = nA/nths;
00306   r = nA%nths;
00307
00308   count = (int *) malloc(nths *sizeof(int));
00309   disp = (int *) malloc(nths *sizeof(int));
00310
00311   for(i=0; i<nths; i++){
00312     if(i<r){
00313       count[i] = q+1;
00314     }
00315     else{
00316       count[i] = q;
00317     }
00318   }
00319
00320   disp[0] = 0;
00321   for(i=0; i<nths-1; i++){
00322     disp[i+1] = disp[i] + count[i];
00323   }
00324
00325   #pragma omp parallel private(tid, n) shared(A, disp, count)
00326   {//---fork---1st step, sort on local chunk
00327     tid = omp_get_thread_num();
00328     n = ssort(A+disp[tid], count[tid], flag);
00329     count[tid]=n;
00330   }//---join---
00331
00332     buf = (int *) malloc(nA*sizeof(int));
00333
```

```
00334   #pragma omp parallel private(tid, n, k, d) shared(nths, nA, A, disp, count,
       buf)
00335   {//---fork---2nd step, gathering with a binary tree scheme
00336     tid = omp_get_thread_num();
00337     nths = omp_get_num_threads();
00338     k = nths;
00339     d = 1;
00340     while(k>1){
00341       if(tid%(2*d)==0 && tid+d<nths){
00342 //        printf("\nd %d, thread %d, count+ %d, disp %d",d , tid, count[tid],
       disp[tid]);
00343         n = card_or(A+disp[tid], count[tid] , A+disp[tid+d], count[tid+d
      ]);
00344         set_or(A+disp[tid], count[tid] , A+disp[tid+d], count[tid+d], buf
      +disp[tid]);
00345         count[tid]=n;
00346         memcpy(A+disp[tid], buf+disp[tid], n*sizeof(int));
00347         d*=2;
00348         k/=2;
00349       }
00350       #pragma omp barrier
00351     }
00352   }//---join---
00353
00354   nA=count[0];
00355   free(buf);
00356   free(count);
00357   free(disp);
00358   return nA;
00359 }
00360 #endif
00361
00362
00364 int sorted(int *indices, int count){
00365   int i=0;
00366   while(i<count-2){
00367     if(indices[i]>indices[i+1]){
00368       return 1;
00369     }
00370     else{
00371       i++;
00372     }
00373   }
00374   return 0;
00375 }
00376
00377 int monotony(int *indices, int count){
00378   int i=0;
00379   while(i<count-2){
00380     if(indices[i]>=indices[i+1]){
00381       return 1;
00382     }
00383     else{
00384       i++;
00385     }
00386   }
00387   return 0;
00388 }
```

15.27 csort.h File Reference

Functions

- int ssort (int *indices, int count, int flag)
- int omp_psort (int *indices, int count, int flag)
- int sorted (int *indices, int count)

  ==========Checker Routines==========
- int monotony (int *indices, int count)

15.27.1 Detailed Description

Note

Author

Pierre Cargemel

Date

March 2012

Definition in file csort.h.

15.27.2 Function Documentation

15.27.2.1 int sorted (int * *indices*, int *count*)

==========Checker Routines==========

Definition at line 364 of file csort.c.

15.27.2.2 int monotony (int * *indices*, int *count*)

Definition at line 377 of file csort.c.

15.28 csort.h

```
00001
00006 int ssort(int *indices, int count, int flag);
00007
00008 #if OPENMP
00009 int omp_psort(int *indices, int count, int flag);
00010 #endif
00011
00012 int sorted(int *indices, int count);
00013
00014 int monotony(int *indices, int count);
```

15.29 mapmat.c File Reference

Matrix routines implementation.

Functions

- int MatInit (Mat *A, int m, int nnz, int *indices, double *values, int flag#ifdef W_MPI, MPI_Comm comm#endif)
- void MatSetIndices (Mat *A, int m, int nnz, int *indices)
- void MatSetValues (Mat *A, int m, int nnz, double *values)
- void CommInfo (Mat *A)

- void MatReset (Mat *A)
- void MatFree (Mat *A)
- int MatLoad (Mat *mat, char *filename)
- int MatSave (Mat *mat, char *filename)
- int MatLocalShape (Mat *A, int sflag)
- int MatComShape (Mat *A, int flag, MPI_Comm comm)
- int MatVecProd (Mat *A, double *x, double *y, int pflag)
- int TrMatVecProd_Naive (Mat *A, double *y, double *x, int pflag)
- int TrMatVecProd (Mat *A, double *y, double *x, int pflag)
- int MatInfo (Mat *mat, int verbose, char *filename)

  Print information about a matrix.
- int greedyreduce (Mat *A, double *x)

15.29.1 Detailed Description

Matrix routines implementation.

Note

Copyright (c) 2010-2012 APC CNRS Université Paris Diderot. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/lgpl.html
For more information about ANR MIDAS'09 project see http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

November 2011

Definition in file mapmat.c.

15.29.2 Function Documentation

15.29.2.1 void CommInfo (Mat * A)

Definition at line 93 of file mapmat.c.

15.29.2.2 void MatReset (Mat * A)

Definition at line 250 of file mapmat.c.

15.29.2.3 int greedyreduce (Mat * A, double * x)

Definition at line 894 of file mapmat.c.

15.30 mapmat.c

```
00001
00009 #ifdef W_MPI
00010  #include <mpi.h>
00011 #endif
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include "mapmat.h"
00016
00017
00043 int MatInit(Mat *A, int m, int nnz, int *indices, double *values, int
       flag
00044 #ifdef W_MPI
00045 , MPI_Comm comm
00046 #endif
00047 ){
00048   int err;
00049   MatSetIndices(A, m, nnz, indices);
00050
00051   MatSetValues(A, m, nnz, values);
00052
00053   err = MatLocalShape(A, 3);               // compute lindices
       (local columns) (method 3 = counting sort)
00054
00055 #ifdef W_MPI
00056   err = MatComShape(A, flag, comm);          // build communication
       scheme
00057 #endif
00058   return err;
00059 }
00060
00061
00070 void MatSetIndices(Mat *A, int m, int nnz, int *indices){
00071   A->m    = m;                          // set number of local rows
00072   A->nnz  = nnz;                      // set number of non-zero values per
       row
00073   A->indices = indices;                   // point to indices
00074 }
00075
00076
00085 void MatSetValues(Mat *A, int m, int nnz, double *values){
00086   int err;
00087   A->m    = m;                          // set number of local rows
00088   A->nnz  = nnz;                      // set number of non-zero values per
       row
00089   A->values  = values;                    // point to values
00090 }
00091
00092 //==================Part added by Sebastien Cayrols to get amount of memory
       needed by communication algoritms
00093 void CommInfo(Mat *A){
00094 #if W_MPI
00095   int i=0,size,rank;
00096   double maxSizeR = 0.0;
00097   double maxSizeS = 0.0;
00098   double amountSizeR = 0.0;
00099   double amountSizeS = 0.0;
00100   double stepSum=0.0,stepAvg=0.0;
00101   //this value is based on data sent
00102   double *amountSizeByStep = NULL;
00103   double minStep=0.0, maxStep=0.0;
00104   double *s=NULL;
00105   double *r=NULL;
00106   MPI_Comm comm = MPI_COMM_WORLD;
00107   MPI_Comm_rank(comm,&rank);
00108   MPI_Comm_size(comm,&size);
00109   s = malloc(4*sizeof(double));
00110   r = malloc(4*3*sizeof(double));
00111   amountSizeByStep = malloc(A->steps*sizeof(double));
00112   switch(A->flag){
00113     case NONE :
00114       break;
00115     case BUTTERFLY :
00116       for(i=0;i<A->steps;i++){
00117         amountSizeR +=A->nR[i];
00118         amountSizeS +=A->nS[i];
00119         if(A->nR[i]>maxSizeR)
00120           maxSizeR=A->nR[i];
00121         if(A->nS[i]>maxSizeS)
00122           maxSizeS=A->nS[i];
00123       }
00124     break;
00125     //=========================Modification added by Sebastien Cayrols :
       01/09/2015 , Berkeley
```

```
00126     case BUTTERFLY_BLOCKING_1 :
00127       for(i=0;i<A->steps;i++){
00128         amountSizeR +=A->nR[i];
00129         amountSizeS +=A->nS[i];
00130         if(A->nR[i]>maxSizeR)
00131           maxSizeR=A->nR[i];
00132         if(A->nS[i]>maxSizeS)
00133           maxSizeS=A->nS[i];
00134       }
00135     break;
00136     case BUTTERFLY_BLOCKING_2 :
00137       for(i=0;i<A->steps;i++){
00138         amountSizeR +=A->nR[i];
00139         amountSizeS +=A->nS[i];
00140         if(A->nR[i]>maxSizeR)
00141           maxSizeR=A->nR[i];
00142         if(A->nS[i]>maxSizeS)
00143           maxSizeS=A->nS[i];
00144       }
00145     break;
00146     case NOEMPTYSTEPRING :
00147       for(i=0;i<A->steps;i++){
00148         amountSizeR +=A->nR[i];
00149         amountSizeS +=A->nS[i];
00150         if(A->nR[i]>maxSizeR)
00151           maxSizeR=A->nR[i];
00152         if(A->nS[i]>maxSizeS)
00153           maxSizeS=A->nS[i];
00154       }
00155     break;
00156     //=========================End modification
00157     case RING :
00158       for(i=0;i<A->steps;i++){
00159         amountSizeR +=A->nR[i];
00160         amountSizeS +=A->nS[i];
00161         if(A->nR[i]>maxSizeR)
00162           maxSizeR=A->nR[i];
00163         if(A->nS[i]>maxSizeS)
00164           maxSizeS=A->nS[i];
00165       }
00166     break;
00167     case NONBLOCKING :
00168       for(i=0;i<A->steps;i++){
00169         amountSizeR +=A->nR[i];
00170         amountSizeS +=A->nS[i];
00171         if(A->nR[i]>maxSizeR)
00172           maxSizeR=A->nR[i];
00173         if(A->nS[i]>maxSizeS)
00174           maxSizeS=A->nS[i];
00175       }
00176     break;
00177     case NOEMPTY :
00178       for(i=0;i<A->steps;i++){
00179         amountSizeR +=A->nR[i];
00180         amountSizeS +=A->nS[i];
00181         if(A->nR[i]>maxSizeR)
00182           maxSizeR=A->nR[i];
00183         if(A->nS[i]>maxSizeS)
00184           maxSizeS=A->nS[i];
00185       }
00186     break;
00187     case ALLTOALLV :          // added -- rs 2015/02/04
00188           for(i=0;i<A->steps;i++){
00189              amountSizeR +=A->nR[i];
00190              amountSizeS +=A->nS[i];
00191           }
00192     break;
00193     case ALLREDUCE :
00194       amountSizeR = A->com_count;
00195       amountSizeS = A->com_count;
00196       maxSizeR    = A->com_count;
00197       maxSizeS    = A->com_count;
00198     break;
00199   }
00200
00201   if(A->flag != ALLREDUCE && A->flag != ALLTOALLV ){
00202     double *t=NULL;
00203
00204     t=malloc(A->steps*sizeof(double));
00205     // Copy int array into double array
00206     for(i=0;i<A->steps;i++)
00207       t[i]=A->nS[i];
00208
00209     MPI_Reduce(t,amountSizeByStep,A->steps,MPI_DOUBLE,MPI_SUM,0,comm);
00210
00211     free(t);
00212
```

```
00213     if(rank==0){
00214       stepSum=minStep=maxStep=amountSizeByStep[0];
00215       printf("\n[MEMORY]Step n°%4d, message size : %e",0,amountSizeByStep[0]);
00216       for(i=1;i<A->steps;i++){
00217         printf("\n[MEMORY]Step n°%4d, message size : %e",i,amountSizeByStep[i])
      ;
00218         if(minStep>amountSizeByStep[i])
00219           minStep=amountSizeByStep[i];
00220         else if(maxStep<amountSizeByStep[i])
00221           maxStep=amountSizeByStep[i];
00222         stepSum+=amountSizeByStep[i];
00223       }
00224       stepAvg=stepSum/A->steps;
00225     }
00226   }
00227   s[0]=amountSizeR;
00228   s[1]=amountSizeS;
00229   s[2]=maxSizeR;
00230   s[3]=maxSizeS;
00231   MPI_Reduce(s,r,4,MPI_DOUBLE,MPI_SUM,0,comm);
00232   if(rank==0)
00233     for(i=0;i<4;i++)
00234       r[i]/=size;
00235   MPI_Reduce(s,&r[4],4,MPI_DOUBLE,MPI_MIN,0,comm);
00236   MPI_Reduce(s,&r[8],4,MPI_DOUBLE,MPI_MAX,0,comm);
00237   if(rank==0){
00238     printf("\n[MEMORY]Step average             : %e\t[%e,%e]",stepAvg,minStep,
    maxStep);
00239     printf("\n[MEMORY]Amount of data received  : %e\t[%e,%e]", r[0],r[4],r[8]);
00240     printf("\n[MEMORY]Amount of data sent      : %e\t[%e,%e]", r[1],r[5],r[9]);
00241     printf("\n[MEMORY]Message size received    : %e\t[%e,%e]",r[2],r[6],r[10]);
00242     printf("\n[MEMORY]Message size sent        : %e\t[%e,%e]\n",r[3],r[7],r[11]
    );
00243   }
00244   free(s);
00245   free(r);
00246   free(amountSizeByStep );
00247 #endif
00248 }
00249
00250 void MatReset(Mat *A){
00251 #if W_MPI
00252   switch(A->flag){
00253     case NONE :
00254       break;
00255     case BUTTERFLY :
00256       free(A->R);              //
00257       free(A->nR);            //
00258       free(A->S);              //
00259       free(A->nS);
00260     break;
00261     case BUTTERFLY_BLOCKING_1 :
00262       free(A->R);              //
00263       free(A->nR);            //
00264       free(A->S);              //
00265       free(A->nS);
00266     break;
00267     case BUTTERFLY_BLOCKING_2 :
00268       free(A->R);              //
00269       free(A->nR);            //
00270       free(A->S);              //
00271       free(A->nS);
00272     break;
00273     case NOEMPTYSTEPRING :
00274       free(A->R);              //
00275       free(A->nR);            //
00276       free(A->S);              //
00277       free(A->nS);
00278     break;
00279     case RING :
00280       free(A->R);              //
00281       free(A->nR);            //
00282       free(A->S);              //
00283       free(A->nS);
00284     break;
00285     case NONBLOCKING :
00286       free(A->R);              //
00287       free(A->nR);            //
00288       free(A->S);              //
00289       free(A->nS);
00290     break;
00291     case NOEMPTY :
00292       free(A->R);              //
00293       free(A->nR);            //
00294       free(A->S);              //
00295       free(A->nS);
00296     break;
```

```
00297     case ALLTOALLV :    // added -- rs 2015/02/04
00298           free(A->R);           //
00299           free(A->nR);                //
00300           free(A->S);           //
00301           free(A->nS);
00302     break;
00303     case ALLREDUCE :
00304     break;
00305   }
00306 #endif
00307 }
00308
00309 //==================End
00310
00317 void MatFree(Mat *A){
00318
00319   //get information about communication size
00320   CommInfo(A);
00321
00322   free(A->lindices);
00323 #if W_MPI
00324   switch(A->flag){
00325     case NONE :
00326       break;
00327     case BUTTERFLY :
00328       free(A->com_indices);  //
00329       free(A->R);               //
00330       free(A->nR);             //
00331       free(A->S);               //
00332       free(A->nS);
00333     break;
00334     //==========================Modification added by Sebastien Cayrols :
       01/09/2015 , Berkeley
00335     case BUTTERFLY_BLOCKING_1 :
00336       free(A->com_indices);  //
00337       free(A->R);               //
00338       free(A->nR);             //
00339       free(A->S);               //
00340       free(A->nS);
00341     break;
00342     case BUTTERFLY_BLOCKING_2 :
00343       free(A->com_indices);  //
00344       free(A->R);               //
00345       free(A->nR);             //
00346       free(A->S);               //
00347       free(A->nS);
00348     break;
00349     case NOEMPTYSTEPRING :
00350       free(A->R);               //
00351       free(A->nR);             //
00352       free(A->S);               //
00353       free(A->nS);
00354     break;
00355     //==========================End modification
00356     case RING :
00357       free(A->R);               //
00358       free(A->nR);            //
00359       free(A->S);               //
00360       free(A->nS);
00361     break;
00362     case NONBLOCKING :
00363       free(A->R);               //
00364       free(A->nR);            //
00365       free(A->S);               //
00366       free(A->nS);
00367     break;
00368     case NOEMPTY :
00369       free(A->R);               //
00370       free(A->nR);            //
00371       free(A->S);               //
00372       free(A->nS);
00373     break;
00374     case ALLTOALLV :    // Added: rs 2015/02/04
00375           free(A->R);           //
00376           free(A->nR);                //
00377           free(A->S);           //
00378           free(A->nS);
00379     break;
00380     case ALLREDUCE :
00381       free(A->com_indices);  //
00382 //==================================Modification from Sebastien Cayrols :
       comment of these lines to avoid SEGSIGV
00383 //      free(A->R);             //
00384 //      free(A->nR);            //
00385 //      free(A->S);             //
00386 //      free(A->nS);
00387 //==================================End modif
```

```
00388     break;
00389   }
00390 #endif
00391 }
00392
00393
00403 int MatLoad(Mat *mat, char *filename){
00404   int err;
00405   int rank;
00406 #if W_MPI
00407   MPI_Comm_rank(mat->comm, &rank);
00408 #else
00409   rank=0;
00410 #endif
00411   FILE *in;
00412   char fn[100];
00413   int i=0;
00414   sprintf(fn, "%s_%d.dat", filename, rank);
00415   printf("%s", fn);
00416   in=fopen(fn,"r");
00417   if(in==NULL){
00418      printf("cannot open file %s", fn);
00419      return 1;
00420   }
00421   while(feof(in)== 0 && i< (mat->m * mat->nnz)){
00422     if(mat->nnz==1){
00423       fscanf(in, "%d %lf", &(mat->indices[i]), &(mat->values[i]));
00424     }
00425     else if(mat->nnz==2){
00426       fscanf(in, "%d %lf %d %lf", &(mat->indices[i]), &(mat->values
      [i]), &(mat->indices[i+1]), &(mat->values[i+1]));
00427     }
00428     else{
00429       return 1;             //(nnz > 2) not implement
00430     }
00431     i+=mat->nnz;
00432   }
00433   if(i!= mat->m * mat->nnz ){
00434     printf("WARNNING data size doesn't fit\n");
00435   }
00436   fclose(in);
00437   return 0;
00438 }
00439
00440
00441
00451 int MatSave(Mat *mat, char *filename){
00452   FILE *out;
00453   char fn [100];
00454   int i,j;
00455   int rank;
00456 #if W_MPI
00457   MPI_Comm_rank(mat->comm, &rank);
00458 #else
00459   sprintf(fn,"%s_%d.dat", filename, rank);
00460 #endif
00461   out=fopen(fn,"w");
00462   if(out==NULL){
00463      printf("cannot open file %s", fn);
00464      return 1;
00465   }
00466   for(i=0; i < (mat->nnz * mat->m); i+=mat->nnz){
00467     for(j=0; j< mat->nnz ; j++){
00468       fprintf(out,"%d ",mat->indices[i+j]);
00469       fprintf(out,"%f ", mat->values[i+j]);
00470     }
00471     fprintf(out, "\n");
00472   }
00473   fclose(out);
00474   return 0;
00475 }
00476
00477
00478
00489 int MatLocalShape(Mat *A, int sflag){
00490   int *tmp_indices;
00491
00492   tmp_indices = (int *) malloc((int64_t) (A->m) * A->nnz * sizeof(int));
      //allocate a tmp copy of indices tab to sort
00493   memcpy(tmp_indices, A->indices, (int64_t) (A->m) * A->nnz * sizeof
      (int));  //copy
00494
00495 //  A->lcount = omp_psort(tmp_indices, A->m * A->nnz, sflag);
      //sequential sort tmp_indices
00496   A->lcount = ssort(tmp_indices, A->m * A->nnz, sflag);
      //sequential sort tmp_indices
00497
```

```
00498   A->lindices = (int *) malloc( A->lcount * sizeof(int));
00499   memcpy(A->lindices, tmp_indices, A->lcount * sizeof(int));
      //copy tmp_indices into lindices and free
00500   free(tmp_indices);
00501
00502   sindex(A->lindices, A->lcount, A->indices, A->nnz
       * A->m);
00503   return 0;
00504 }
00505
00506
00507
00508
00509
00510 #if W_MPI
00511
00515 int MatComShape(Mat *A, int flag, MPI_Comm comm){
00516   int size;
00517   int i, min, max, j;
00518   A->comm = comm;                    // set communivcator
00519   A->flag=flag;
00520   MPI_Comm_size(A->comm, &size);
00521   if((A->flag==BUTTERFLY || A->flag==BUTTERFLY_BLOCKING_1 || A->flag
      ==BUTTERFLY_BLOCKING_2) && is_pow_2(size)!=0)
00522     A->flag=RING;
00523   switch(A->flag){
00524     case BUTTERFLY :
00525       A->steps = log_2(size);
00526       A->S = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate sending maps tab
00527       A->R = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate receiving maps tab
00528       A->nS = (int* ) malloc(A->steps * sizeof(int));
      //allocate sending map sizes tab
00529       A->nR = (int* ) malloc(A->steps * sizeof(int));
      //allocate receiving map size tab
00530       butterfly_init(A->lindices, A->lcount, A->R,
       A->nR, A->S, A->nS, &(A->com_indices), &(A->com_count)
      , A->steps, A->comm);
00531       break;
00532     //=========================Modification added by Sebastien Cayrols :
       01/09/2015 , Berkeley
00533     case BUTTERFLY_BLOCKING_1 :
00534       A->steps = log_2(size);
00535       A->S = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate sending maps tab
00536       A->R = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate receiving maps tab
00537       A->nS = (int* ) malloc(A->steps * sizeof(int));
      //allocate sending map sizes tab
00538       A->nR = (int* ) malloc(A->steps * sizeof(int));
      //allocate receiving map size tab
00539       butterfly_init(A->lindices, A->lcount, A->R,
       A->nR, A->S, A->nS, &(A->com_indices), &(A->com_count)
      , A->steps, A->comm);
00540       break;
00541     case BUTTERFLY_BLOCKING_2 :
00542       A->steps = log_2(size);
00543       A->S = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate sending maps tab
00544       A->R = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate receiving maps tab
00545       A->nS = (int* ) malloc(A->steps * sizeof(int));
      //allocate sending map sizes tab
00546       A->nR = (int* ) malloc(A->steps * sizeof(int));
      //allocate receiving map size tab
00547       butterfly_init(A->lindices, A->lcount, A->R,
       A->nR, A->S, A->nS, &(A->com_indices), &(A->com_count)
      , A->steps, A->comm);
00548       break;
00549     case NOEMPTYSTEPRING :
00550       A->steps = size;
00551       A->S = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate sending maps tab
00552       A->R = (int** ) malloc(A->steps * sizeof(int* ));
      //allocate receiving maps tab
00553       A->nS = (int* ) malloc(A->steps * sizeof(int));
      //allocate sending map sizes tab
00554       A->nR = (int* ) malloc(A->steps * sizeof(int));
      //allocate receiving map size tab
00555       ring_init(A->lindices, A->lcount, A->R, A->nR,
      A->S, A->nS, A->steps, A->comm);
00556       A->com_count = A->lcount;
00557       A->com_indices = A->lindices;
00558       break;
00559     //=========================End modification
00560     case RING :
```

```
00561       A->steps = size;
00562       A->S = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate sending maps tab
00563       A->R = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate receiving maps tab
00564       A->nS = (int* ) malloc(A->steps * sizeof(int));
    //allocate sending map sizes tab
00565       A->nR = (int* ) malloc(A->steps * sizeof(int));
    //allocate receiving map size tab
00566       ring_init(A->lindices, A->lcount, A->R, A->nR,
    A->S, A->nS, A->steps, A->comm);
00567       A->com_count = A->lcount;
00568       A->com_indices = A->lindices;
00569       break;
00570     case NONBLOCKING :
00571       A->steps = size;
00572       A->S = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate sending maps tab
00573       A->R = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate receiving maps tab
00574       A->nS = (int* ) malloc(A->steps * sizeof(int));
    //allocate sending map sizes tab
00575       A->nR = (int* ) malloc(A->steps * sizeof(int));
    //allocate receiving map size tab
00576       ring_init(A->lindices, A->lcount, A->R, A->nR,
    A->S, A->nS, A->steps, A->comm);
00577       A->com_count = A->lcount;
00578       A->com_indices = A->lindices;
00579       break;
00580     case NOEMPTY :
00581       A->steps = size;
00582       A->S = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate sending maps tab
00583       A->R = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate receiving maps tab
00584       A->nS = (int* ) malloc(A->steps * sizeof(int));
    //allocate sending map sizes tab
00585       A->nR = (int* ) malloc(A->steps * sizeof(int));
    //allocate receiving map size tab
00586       ring_init(A->lindices, A->lcount, A->R, A->nR,
    A->S, A->nS, A->steps, A->comm);
00587       A->com_count = A->lcount;
00588       A->com_indices = A->lindices;
00589       break;
00590     case ALLTOALLV :
00591       A->steps = size;
00592       A->S = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate sending maps tab
00593       A->R = (int** ) malloc(A->steps * sizeof(int* ));
    //allocate receiving maps tab
00594       A->nS = (int* ) malloc(A->steps * sizeof(int));
    //allocate sending map sizes tab
00595       A->nR = (int* ) malloc(A->steps * sizeof(int));
    //allocate receiving map size tab
00596       ring_init(A->lindices, A->lcount, A->R, A->nR,
    A->S, A->nS, A->steps, A->comm);
00597       A->com_count = A->lcount;
00598       A->com_indices = A->lindices;
00599       break;
00600     case ALLREDUCE :
00601       MPI_Allreduce(&A->lindices[A->lcount-1], &max, 1, MPI_INT,
    MPI_MAX, A->comm);   //maximum index
00602       MPI_Allreduce(&A->lindices[0], &min, 1, MPI_INT, MPI_MIN, A->comm
    );       //
00603       A->com_count=(max-min+1);
00604       A->com_indices = (int *) malloc(A->lcount * sizeof(int))
    ; //warning
00605       i=0;
00606       j=0;
00607       while( j<A->com_count && i<A->lcount){ //same as subsetmap for a
     coutiguous set
00608         if(min+j < A->lindices[i]){
00609           j++;
00610         }
00611         else{
00612           A->com_indices[i]=j;
00613           i++;
00614           j++;
00615         }
00616       }
00617       break;
00618   }
00619  return 0;
00620 }
00621 #endif
00622
00623
```

```
00630 int MatVecProd(Mat *A, double *x, double* y, int pflag){
00631   int i, j, e;
00632   for(i=0; i<A->m; i++)                                          //

00633       y[i] = 0.0;
00634
00635   e=0;
00636   for(i=0; i<A->m*A->nnz; i+=A->nnz){
      //
00637     for(j=0; j<A->nnz; j++){                                    //
00638       y[e] += A->values[i+j] * x[A->indices[i+j]];
00639     }
00640     e++;
00641   }
00642   return 0;
00643 };
00644
00645
00646 #ifdef W_MPI
00647
00658 int TrMatVecProd_Naive(Mat *A, double *y, double* x, int
      pflag){
00659   int i, j, e, rank, size;
00660   int *rbuf, rbufcount;
00661   double *rbufvalues, *lvalues;
00662   int p, rp, sp, tag;
00663   MPI_Request s_request, r_request;
00664   MPI_Status status;
00665
00666   MPI_Comm_rank(A->comm, &rank);                               //get rank and
       size of the communicator
00667   MPI_Comm_size(A->comm, &size);                               //
00668   lvalues = (double *) malloc( A->lcount *sizeof(double));        //
      allocate and set local values to 0.0
00669   for(i=0; i < A->lcount; i++)                                //
00670     lvalues[i]=0.0;                                              //
00671
00672   e=0;
00673   for(i=0; i<A->m; i++){                                         //local
       transform reduces
00674     for(j=0; j<A->nnz; j++){                                   //
00675       lvalues[A->indices[i*A->nnz+j]] += (A->values[i*A->nnz
      +j]) * y[i];
00676     }
00677   }
00678
00679   memcpy(x, lvalues, (A->lcount)*sizeof(double));
      //copy local values into the result*/
00680   MPI_Allreduce(&(A->lcount), &(rbufcount), 1, MPI_INT, MPI_MAX, A->comm
      );       //find the max communication buffer sizes, and allocate
00681
00682   rbuf = (int *)    malloc(rbufcount * sizeof(int));
00683   rbufvalues  = (double *) malloc(rbufcount * sizeof(double));
00684
00685   tag=0;
00686   for (p=1; p < size; p++){     //loop : collective global reduce in ring-like
       fashion
00687     rp = (size + rank - p)%size;
00688     sp = (rank + p)%size;
00689     MPI_Send(&(A->lcount), 1, MPI_INT, sp, 0, A->comm);
      //exchange sizes
00690     MPI_Recv(&rbufcount, 1, MPI_INT, rp, 0, A->comm, &status);
00691     tag++;
00692     MPI_Irecv(rbuf, rbufcount, MPI_INT, rp, tag, A->comm, &r_request);
      //exchange local indices
00693     MPI_Isend(A->lindices, A->lcount, MPI_INT, sp, tag, A->comm
      , &s_request);
00694     MPI_Wait(&r_request, &status);
00695     MPI_Wait(&s_request, &status);
00696     tag++;
00697     MPI_Irecv(rbufvalues, rbufcount, MPI_DOUBLE, rp, tag, A->comm, &
      r_request);     //exchange local values
00698     MPI_Isend(lvalues, A->lcount, MPI_DOUBLE, sp, tag, A->comm, &
      s_request);
00699     tag++;
00700     MPI_Wait(&r_request, &status);
00701     m2m_sum(rbufvalues, rbuf, rbufcount, x, A->lindices, A->
      lcount);          //sum in the result
00702     MPI_Wait(&s_request, &status);
00703   }
00704   free(lvalues);
00705   return 0;
00706 }
00707 #endif
00708
00709
00723 int TrMatVecProd(Mat *A, double *y, double* x, int pflag){
```

```
00724   double *sbuf, *rbuf;
00725   int i, j, k, e;
00726   int nSmax, nRmax;
00727   double *lvalues;
00728
00729   for(i=0; i < A->lcount; i++)                              //refresh
       vector
00730     x[i]=0.0;                                              //
00731
00732   e=0;
00733   for(i=0; i< A->m*A->nnz; i+=A->nnz){                     //local
       transform reduce
00734     for(j=0; j< A->nnz; j++){
00735       x[A->indices[i+j]] += A->values[i+j] * y[e];         //
00736     }                                                      //
00737     e++;                                                   //
00738   }                                                        //
00739
00740 #ifdef W_MPI
00741   greedyreduce(A, x);                                       //
      global reduce
00742 #endif
00743   return 0;
00744 }
00745
00746
00747
00748
00749
00750
00751 #ifdef W_MPI
00752
00758 int MatInfo(Mat *mat, int verbose, char *filename){
00759   FILE *out;
00760   int *n;
00761   int *sr;
00762   int *s;
00763   int nnzline, sparsity, maxstep, maxsize, sumline, total;
00764   int i, j, k;
00765   char fn [100];
00766   int rank, size;
00767   int master=0;
00768   MPI_Comm_rank(mat->comm, &rank);
00769   MPI_Comm_size(mat->comm, &size);
00770
00771
00772   if(rank==master){                     //master process saves data into
       filename_info.txt
00773     sprintf(fn,"%s_%s", filename, "info.txt");
00774     out=fopen(fn,"w");
00775     if(out==NULL){
00776       printf("cannot open file %s\n", fn);
00777       return 1;
00778     }
00779     printf("open file %s ...", fn);
00780     fprintf(out, "flag %d\n", mat->flag);   //print matirx main description
       : flag (communication scheme),
00781     fprintf(out, "rows %d\n ", mat->m);        //rows per process,
00782     fprintf(out, "nnz %d\n", mat->nnz);      //nnz (number of non zero per
       row).
00783     fprintf(out, "\n"); //separator
00784   }
00785
00786   /*n = (int* ) calloc(mat->lcount,sizeof(int));               //allocate
00787   //printf("before gather %d\n", rank);
00788   MPI_Gather(&(mat->lcount), 1, MPI_INT, n, 1, MPI_INT, master, mat->comm);
      //gather nbnonempty cols
00789   //printf("after gather %d\n", rank);
00790
00791   if(rank==master){                     //master process saves data into
       filename_info.txt
00792     fprintf(out, "cols :\n");   //nnz (number of non zero per row).
00793     for(i=0; i<size; i++)               //
00794       fprintf(out, "%d ", n[i]);        //non-empty columns per process.
00795     fprintf(out, "\n");                 //
00796   }
00797   free(n); */                           //free allocated tabs
00798
00799   nnzline = 0;                          //compute communication sparsity and
       maximum message size
00800   sumline = 0;
00801   for(i=0; i<mat->steps; i++){             //
00802     sumline+=mat->nS[i];
00803     if(mat->nS[i]==0){                        //
00804       nnzline +=1;                      //
00805     }                                   //
00806   }                                     //
```

```
00807   MPI_Reduce(&nnzline, &sparsity, 1, MPI_INT, MPI_SUM, 0, mat->comm);
      //sparsity
00808   MPI_Reduce(&sumline, &total, 1, MPI_INT, MPI_SUM, 0, mat->comm);  //
      sparsity
00809   if(rank==master){                    //master process saves data into
       filename_info.txt
00810     fprintf(out, "sparsity %d\n", sparsity);    //
00811     fprintf(out, "total %d\n", total);  //
00812   }
00813
00814   maxsize =0;
00815   for(i=0; i<mat->steps; i++){             //
00816     MPI_Reduce(&(mat->nS[i]), &maxstep, 1, MPI_INT, MPI_MAX, 0, mat->comm
      );       //maximum message size
00817     maxsize+=maxstep;                  //
00818   }                          //
00819   if(rank==master){                              //master process saves data
       into filename_info.txt
00820     fprintf(out, "maxsize %d\n ", maxsize);     //
00821     fprintf(out, "\n");                          //separator
00822   }                                              //
00823
00824  /* s = (int* ) calloc((mat->steps),sizeof(int));       //allocate steps
00825   MPI_Reduce(mat->nS, s, mat->steps, MPI_INT, MPI_SUM, 0, mat->comm);   //
      imaximum message size
00826
00827   if(rank==master){                    //master process saves data into
       filename_info.txt
00828     fprintf(out, "sumsteps :\n");       //nnz (number of non zero per row).
00829     for(i=0; i<mat->steps; i++)         //
00830       fprintf(out, "%d ", s[i]);        //non-empty columns per process.
00831     fprintf(out, "\n");                 //
00832   }
00833   free(s);
00834
00835   if(verbose==1){
00836     sr = (int* ) calloc((mat->steps)*size,sizeof(int)); //allocate send/receive
       matrix
00837     //printf("before gather %d\n", rank);
00838     MPI_Gather(mat->nS, mat->steps, MPI_INT, sr, mat->steps, MPI_INT, master,
       mat->comm);       //gather nbnonempty cols
00839     //printf("after gather %d\n", rank);
00840
00841     if(rank==master){                  //master process saves data into
       filename_info.txt
00842     fprintf(out, "send/receive matrix\n");      //separator
00843       for(i=0; i<size; i++){           //print collective description :
00844         if(mat->flag==BUTTERFLY){              //send-receive matrix
00845           for(j=0; j<size; j++){               //print send/receive matrix
00846             if(j>i){
00847               if(is_pow_2(j-i)==0)
00848                 fprintf(out,"%d ", sr[i*(mat->steps)+log_2(j-i)]);
00849               else
00850                 fprintf(out,"%d ", 0);
00851             }
00852             else if(i>j){
00853               if(is_pow_2(size+j-i)==0)
00854                 fprintf(out,"%d ", sr[i*(mat->steps)+log_2(size+j-i)]);
00855               else
00856                 fprintf(out,"%d ", 0);
00857             }
00858             else{
00859               fprintf(out,"%d ", 0);
00860             }
00861           }
00862           fprintf(out, "\n");
00863         }
00864         else{
00865           for(j=0; j<size; j++){               //print send/receive matrix
00866             if(j>i){
00867               fprintf(out,"%d ", sr[i*(mat->steps)+j-i]);
00868             }
00869             else if(i>j){
00870               fprintf(out,"%d ", sr[(i+1)*(mat->steps)-i+j]);
00871             }
00872             else{
00873               fprintf(out,"%d ", 0);
00874             }
00875           }
00876           fprintf(out, "\n");
00877         }
00878       }
00879     }
00880     free(sr);
00881   }*/
00882
00883   if(rank==master){                    //master process saves data into
```

```
      filename_info.txt
00884     fclose(out);
00885     printf("close %s\n", fn);
00886   }
00887   return 0;
00888 }
00889 #endif
00890
00891
00892
00893 #if W_MPI
00894 int greedyreduce(Mat *A, double* x){
00895   int i, j, k;
00896   int nSmax, nRmax, nStot, nRtot;
00897   double *lvalues;
00898   lvalues = (double *) malloc(A->lcount *sizeof(double)); //allocate and
      set to 0.0 local values
00899   memcpy(lvalues, x, (A->lcount) *sizeof(double));                //copy
      local values into result values
00900   double *com_val;
00901   double *out_val;
00902   int ne=0;
00903   switch(A->flag){
00904     case BUTTERFLY :
00905       for(k=0; k< A->steps; k++)                                    //
    compute max communication buffer size
00906         if(A->nR[k] > nRmax)
00907           nRmax = A->nR[k];
00908       for(k=0; k< A->steps; k++)
00909         if(A->nS[k] > nSmax)
00910           nSmax = A->nS[k];
00911       com_val=(double *) malloc( A->com_count *sizeof(double));
00912       for(i=0; i < A->com_count; i++)
00913         com_val[i]=0.0;
00914       m2m(lvalues, A->lindices, A->lcount, com_val, A->
    com_indices, A->com_count);
00915       butterfly_reduce(A->R, A->nR, nRmax, A->S, A->nS,
    nSmax, com_val, A->steps, A->comm);
00916       m2m(com_val, A->com_indices, A->com_count, x, A->
    lindices, A->lcount);
00917       free(com_val);
00918       break;
00919     //=========================Modification added by Sebastien Cayrols :
      01/09/2015 , Berkeley
00920     case BUTTERFLY_BLOCKING_1 :
00921       for(k=0; k< A->steps; k++)                                    //
    compute max communication buffer size
00922         if(A->nR[k] > nRmax)
00923           nRmax = A->nR[k];
00924       for(k=0; k< A->steps; k++)
00925         if(A->nS[k] > nSmax)
00926           nSmax = A->nS[k];
00927       com_val=(double *) malloc( A->com_count *sizeof(double));
00928       for(i=0; i < A->com_count; i++)
00929         com_val[i]=0.0;
00930       m2m(lvalues, A->lindices, A->lcount, com_val, A->
    com_indices, A->com_count);
00931       butterfly_blocking_linstr_reduce(A->R, A
    ->nR, nRmax, A->S, A->nS, nSmax, com_val, A->steps, A->comm);
00932       m2m(com_val, A->com_indices, A->com_count, x, A->
    lindices, A->lcount);
00933       free(com_val);
00934       break;
00935     case BUTTERFLY_BLOCKING_2 :
00936       for(k=0; k< A->steps; k++)                                    //
    compute max communication buffer size
00937         if(A->nR[k] > nRmax)
00938           nRmax = A->nR[k];
00939       for(k=0; k< A->steps; k++)
00940         if(A->nS[k] > nSmax)
00941           nSmax = A->nS[k];
00942       com_val=(double *) malloc( A->com_count *sizeof(double));
00943       for(i=0; i < A->com_count; i++)
00944         com_val[i]=0.0;
00945       m2m(lvalues, A->lindices, A->lcount, com_val, A->
    com_indices, A->com_count);
00946       butterfly_blocking_linstr_reduce(A->R, A
    ->nR, nRmax, A->S, A->nS, nSmax, com_val, A->steps, A->comm);
00947       m2m(com_val, A->com_indices, A->com_count, x, A->
    lindices, A->lcount);
00948       free(com_val);
00949       break;
00950     case NOEMPTYSTEPRING :
00951       for(k=1; k< A->steps; k++)                                //compute max
      communication buffer size
00952         if(A->nR[k] > nRmax)
00953           nRmax = A->nR[k];
```

```
00954       nSmax = nRmax;
00955       ring_noempty_step_reduce(A->R, A->nR, nRmax, A
    ->S, A->nS, nSmax, lvalues, x, A->steps, A->comm);
00956       break;
00957     //=========================End modification
00958     case RING :
00959       for(k=1; k< A->steps; k++)                          //compute max
     communication buffer size
00960         if(A->nR[k] > nRmax)
00961           nRmax = A->nR[k];
00962       nSmax = nRmax;
00963       ring_reduce(A->R, A->nR, nRmax, A->S, A->nS, nSmax,
    lvalues, x, A->steps, A->comm);
00964       break;
00965     case NONBLOCKING :
00966       ring_nonblocking_reduce(A->R, A->nR, A->S, A->
    nS, lvalues, x, A->steps, A->comm);
00967       break;
00968     case NOEMPTY :
00969       for(k=1; k< A->steps; k++)
00970         if(A->nR[k]!=0)
00971           ne++;
00972       ring_noempty_reduce(A->R, A->nR, ne, A->S, A->nS
    , ne, lvalues, x, A->steps, A->comm);
00973       break;
00974     case ALLREDUCE :
00975       com_val=(double *) malloc( A->com_count *sizeof(double));
00976       out_val=(double *) malloc( A->com_count *sizeof(double));
00977       for(i=0; i < A->com_count; i++){
00978         com_val[i]=0.0;
00979         out_val[i]=0.0;
00980       }
00981       s2m(com_val, lvalues, A->com_indices, A->lcount);
00982       /*for(i=0; i < A->com_count; i++){
00983          printf("%lf ", com_val[i]);
00984       } */
00985       MPI_Allreduce(com_val, out_val, A->com_count, MPI_DOUBLE,
    MPI_SUM, A->comm); //maximum index
00986       /*for(i=0; i < A->com_count; i++){
00987          printf("%lf ", out_val[i]);
00988       } */
00989       m2s(out_val, x, A->com_indices, A->lcount);
                          //sum receive buffer into values
00990       free(com_val);
00991       free(out_val);
00992       break;
00993       case ALLTOALLV :
00994          nRtot=nStot=0;
00995          for(k=0; k< A->steps; k++){                              //
    compute buffer sizes
00996               nRtot += A->nR[k];               // to receive
00997               nStot += A->nS[k];               // to send
00998        }
00999
01000       alltoallv_reduce(A->R, A->nR, nRtot, A->S, A->nS,
    nStot, lvalues, x, A->steps, A->comm);
01001       break;
01002   }
01003   free(lvalues);
01004   return 0;
01005 }
01006 #endif
01007
01008
```

15.31 mapmat.dox File Reference

15.32 mapmat.h File Reference

Declarations of the matrix type and his associated routines.

these routines are developed to handle sparse matrices. Typically, in the CMB Data Analysis context, it is especially developed handle pointing or unpointing matrices. Thus, the unpointing matrix *A* can be defined as a MIDAS_Mat. Operating with the pointing matrices can be done without redefining a new matrix.

Data Structures

- struct Mat

  Matrix structure
  A∗ = (A0∗ | A1∗ | ... | Ap-1∗)

Functions

- int MatInit (Mat ∗A, int m, int nnz, int ∗indices, double ∗values, int flag#ifdef W_MPI, MPI_Comm comm#endif)
- void MatSetIndices (Mat ∗A, int m, int nnz, int ∗indices)
- void MatSetValues (Mat ∗A, int m, int nnz, double ∗values)
- void MatFree (Mat ∗A)
- int MatLocalShape (Mat ∗A, int sflag)
- int MatComShape (Mat ∗A, int flag, MPI_Comm comm)
- int MatVecProd (Mat ∗A, double ∗x, double ∗y, int pflag)
- int TrMatVecProd (Mat ∗A, double ∗y, double ∗x, int pflag)
- int TrMatVecProd_Naive (Mat ∗A, double ∗y, double ∗x, int pflag)
- int MatLoad (Mat ∗A, char ∗filename)
- int MatSave (Mat ∗A, char ∗filename)
- int MatInfo (Mat ∗A, int master, char ∗filename)

  Print information about a matrix.
- int greedyreduce (Mat ∗A, double ∗x)

15.32.1 Detailed Description

Declarations of the matrix type and his associated routines.

these routines are developed to handle sparse matrices. Typically, in the CMB Data Analysis context, it is especially developed handle pointing or unpointing matrices. Thus, the unpointing matrix *A* can be defined as a MIDAS_Mat. Operating with the pointing matrices can be done without redefining a new matrix.

Author

Pierre Cargemel

Date

November 2011

Definition in file mapmat.h.

15.32.2 Function Documentation

15.32.2.1 int greedyreduce (Mat ∗ *A*, double ∗ *x*)

Definition at line 894 of file mapmat.c.

15.33 mapmat.h

```
00001
00013 #define NONE 0
00014 #define RING 1
00015 #define BUTTERFLY 2
00016 #define NONBLOCKING 3
00017 #define NOEMPTY 4
```

```
00018 #define ALLTOALLV 5
00019 #define ALLREDUCE 6
00020 //================Modification introduced by Sebastien Cayrols : 01/09/2015 ;
       Berkeley
00021 #define BUTTERFLY_BLOCKING_1 7
00022 #define BUTTERFLY_BLOCKING_2 8
00023 #define NOEMPTYSTEPRING 9
00024 //================End modification
00025 #define SEQ 0
00026 #define OMP 1
00027 #define GPU 2
00028
00029
00030
00034 typedef struct {
00035   int           flag;                       // flag for communication
       scheme (NONE, RING, BUTTERFLY ...)
00036   int           m;                    // number local rows
00037   int           nnz;                   // number non-zero per rows
00038   int           *indices;               // column indices tab; size = m
       * nnz; can be a global or local numbering
00039   double        *values;          // non-zero values tab; size = m * nnz
00040   //--------local shaping---------------
00041   int           lcount;
00042   int           *lindices;              // local indices tab (monotony
       with global numbering); size = lcount
00043 #ifdef W_MPI
00044   MPI_Comm      comm;                   // MPI communicator
00045   //--------com shaping-----------------
00046   int           *com_indices, com_count;// communicated indices tab,
       and size
00047   int           steps;                     // number of steps in the
       communication scheme
00048   int           *nS, *nR;             // number of indices (to send and to
       receive); size = steps
00049   int           **R, **S;              // sending or receiving indices tab
00050 #endif
00051 }Mat;
00052
00053
00054 int MatInit(Mat *A, int m, int nnz, int *indices, double *values, int
       flag
00055 #ifdef W_MPI
00056 , MPI_Comm comm
00057 #endif
00058 );
00059
00060 void MatSetIndices(Mat *A, int m, int nnz, int *indices);
00061
00062 void MatSetValues(Mat *A, int m, int nnz, double *values);
00063
00064 void MatFree(Mat *A);
00065
00066 int MatLocalShape(Mat *A, int sflag);
00067
00068 #if W_MPI
00069 int MatComShape(Mat *A, int flag,  MPI_Comm comm);
00070 #endif
00071
00072 int MatVecProd(Mat *A, double *x, double *y, int pflag);
00073
00074 int TrMatVecProd(Mat *A, double *y, double* x, int pflag);
00075
00076
00077 #if W_MPI
00078 int TrMatVecProd_Naive(Mat *A, double *y, double* x, int
      pflag);
00079 #endif
00080
00081 int MatLoad(Mat *A, char *filename);
00082
00083 int MatSave(Mat *A, char* filename);
00084
00085 #if W_MPI
00086 int MatInfo(Mat *A, int master, char* filename);
00087 #endif
00088
00089 int greedyreduce(Mat *A, double* x);
00090
00091
00092 // Doxygen definitions
00093
```

15.34 mapmatc.c File Reference

Functions

- int CMatInit (CMat *A, int r, int *m, int *nnz, int **indices, double **values, int flag#ifdef W_MPI, MPI_Comm comm#endif)
- int CMatFree (CMat *A)
- int CMatComShape (CMat *mat, int flag)
- int CMatVecProd (CMat *A, double *xvalues, double *yvalues, int pflag)
- int CTrMatVecProd (CMat *A, double *in_values, double *out_values, int pflag)

15.34.1 Function Documentation

15.34.1.1 int CMatInit (CMat * *A*, int *r*, int * *m*, int * *nnz*, int ** *indices*, double ** *values*, int flag#ifdef *W_MPI*, MPI_Comm comm# *endif*)

Definition at line 17 of file mapmatc.c.

15.34.1.2 int CMatFree (CMat * *A*)

Definition at line 75 of file mapmatc.c.

15.34.1.3 int CMatComShape (CMat * *mat*, int *flag*)

Definition at line 96 of file mapmatc.c.

15.34.1.4 int CMatVecProd (CMat * *A*, double * *xvalues*, double * *yvalues*, int *pflag*)

Definition at line 139 of file mapmatc.c.

15.34.1.5 int CTrMatVecProd (CMat * *A*, double * *in_values*, double * *out_values*, int *pflag*)

Definition at line 159 of file mapmatc.c.

15.35 mapmatc.c

```
00001
00009 #ifdef W_MPI
00010  #include <mpi.h>
00011 #endif
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include "mapmatc.h"
00016
00017 int CMatInit(CMat *A, int r, int *m, int *nnz, int **indices,
      double **values, int flag
00018 #ifdef W_MPI
00019   ,MPI_Comm comm
00020 #endif
00021   ){
00022   int M, k, *tmp_indices;
00023   A->r    = r;                                    // set number of local
       rows
00024   A->m    = m;                                    //
00025   A->nnz  = nnz;                                 //
00026   A->disp = (int *) malloc((A->r+1)*sizeof(int));          // allocate
       disp array
00027   A->disp[0]=0;
00028 //  printf(" %d\t%d\t%d\n", A->m[0], A->nnz[0], A->disp[0]);
```

```
00029   for(k=1; k<=A->r; k++){
00030     A->disp[k]=A->disp[k-1]+A->m[k-1]*A->nnz[k-1];
00031   //  if(k!=A->r)
00032   //    printf(" %d\t%d\t", A->m[k], A->nnz[k]);
00033   //  printf(" %d\n", A->disp[k]);
00034   }
00035   A->indices = indices;                                    //
00036   A->values  = values;
00037   /*int i, j;
00038   for(k=0; k<A->r; k++){
00039     for(i=0; i<A->m[k]*A->nnz[k]; i+=A->nnz[k]){
00040       for(j=0; j<A->nnz[k]; j++){
00041         printf(" %d ", A->indices[k][i+j]);
00042       }
00043     }
00044     printf("\n");
00045   }*/
00046   tmp_indices = (int *) malloc(A->disp[A->r]*sizeof(int)); // allocate a
       tmp copy of indices tab to sort
00047   for(k=0; k<A->r; k++){
00048     memcpy(tmp_indices+A->disp[k], A->indices[k], A->m[k]*A->nnz
      [k]*sizeof(int));        // copy
00049   }
00050
00051   A->lcount = ssort(tmp_indices, A->disp[A->r], 0);
            // sequential sort tmp_indices (flag:3=counting sort)
00052   A->lindices = (int *) malloc((A->lcount)*sizeof(int));      //

00053   memcpy(A->lindices, tmp_indices, (A->lcount) *sizeof(int));
      // copy tmp_indices into lindices and free
00054   free(tmp_indices);
00055
00056   for(k=0; k<A->r; k++){
00057     sindex(A->lindices, A->lcount, A->indices[k], A
      ->nnz[k]*A->m[k]);    // transform indices tab in local indices tab
00058   }
00059   /*for(k=0; k<A->r; k++){
00060     for(i=0; i<A->m[k]*A->nnz[k]; i+=A->nnz[k]){
00061       for(j=0; j<A->nnz[k]; j++){
00062         printf(" %d ", A->indices[k][i+j]);
00063       }
00064     }
00065     printf("\n");
00066   }*/
00067   //printf("cmat init 0\n");
00068 #ifdef W_MPI
00069   A->comm = comm;                                    // link communivcator
00070   return CMatComShape(A, flag);                      // build
       communication scheme
00071 #endif
00072 }
00073
00074
00075 int CMatFree(CMat *A){
00076   free(A->disp);
00077   free(A->lindices);
00078 #ifdef W_MPI
00079   if(A->flag != NONE){              //if necessary free communication tab
00080     if(A->R)                    //
00081       free(A->R);               //
00082     if(A->nR)                  //
00083       free(A->nR);             //
00084     if(A->S)                    //
00085       free(A->S);               //
00086     if(A->nS)
00087       free(A->nS);
00088   }
00089 #endif
00090   return 0;
00091 }
00092
00093
00094
00095 #ifdef W_MPI
00096 int CMatComShape(CMat *mat, int flag){
00097   //printf("commshape 0\n");
00098   int size;
00099   mat->flag = flag;
00100   MPI_Comm_size(mat->comm, &size);
00101   if(flag==BUTTERFLY){
00102     if(is_pow_2(size)==0){
00103     mat->flag=flag;
00104     mat->steps = log_2(size);
00105     }
00106     else{
00107       mat->flag=RING;
00108       mat->steps = size;
```

```
00109     }
00110   }
00111   else if(flag==NONE){
00112     mat->flag=flag;
00113     return 0;
00114     }
00115   else{
00116     mat->flag=flag;
00117     mat->steps = size;
00118   }
00119   mat->S = (int** ) malloc(mat->steps * sizeof(int* ));
      /*<allocate sending maps tab*/
00120   mat->R = (int** ) malloc(mat->steps * sizeof(int* ));
      /*<allocate receiving maps tab*/
00121   mat->nS = (int* ) malloc(mat->steps * sizeof(int));
      /*<allocate sending map sizes tab*/
00122   mat->nR = (int* ) malloc(mat->steps * sizeof(int));
      /*<allocate receiving map size tab*/
00123
00124   if(mat->flag == BUTTERFLY){
00125     butterfly_init(mat->lindices, mat->lcount, mat
      ->R, mat->nR, mat->S, mat->nS, &(mat->com_indices), &(mat->
      com_count), mat->steps, mat->comm);
00126   }
00127   else{
00128     ring_init(mat->lindices, mat->lcount, mat->R, mat->
      nR, mat->S, mat->nS, mat->steps, mat->comm);
00129     mat->com_count = mat->lcount;
00130     mat->com_indices = mat->lindices;
00131   }
00132   //printf("commshape 1\n");
00133  return 0;
00134 }
00135 #endif
00136
00137
00138
00139 int CMatVecProd(CMat *A, double *xvalues, double* yvalues, int
      pflag){
00140   int i, j, k;
00141   int l;
00142   for(i=0; i<A->disp[A->r]; i++)

00143       yvalues[i] = 0.0;
00144   l=0;
00145   for(k=0; k<A->r; k++){                                          //
      coarse levels
00146     for(i=0; i<A->m[k]; i+=A->nnz[k]){
      //rows
00147       for(j=0; j<A->nnz[k]; j++){                                 //
      non-zero per row
00148         yvalues[l] += A->values[k][i+j] * xvalues[A->indices[k][i+
      j]];
00149       }
00150       l++;
00151     }
00152   }
00153   return 0;
00154 }
00155
00156
00157
00158
00159 int CTrMatVecProd(CMat *A, double *in_values, double*
      out_values, int pflag){
00160   int i, j, k;
00161   int l;
00162   int nSmax, nRmax;
00163   double *lvalues;
00164
00165   lvalues = (double *) malloc(A->lcount *sizeof(double));    /*<allocate
       and set to 0.0 local values*/
00166   for(i=0; i < A->lcount; i++)
00167     lvalues[i]=0.0;
00168
00169   l=0;
00170   for(k=0; k<A->r; k++){                                          //
      coarse levels
00171     for(i=0; i<A->m[k]; i+=A->nnz[k]){
      //rows
00172       for(j=0; j<A->nnz[k]; j++){                                 //
      non-zero per row
00173         lvalues[A->indices[k][i+j]] += A->values[k][i+j] *
      in_values[l];
00174       }
00175       l++;
00176     }
```

```
00177   }
00178   memcpy(out_values, lvalues, (A->lcount) *sizeof(double)); /*<copy local
       values into result values*/
00179 #ifdef W_MPI
00180   nRmax=0;
00181   nSmax=0;
00182
00183   if(A->flag  == BUTTERFLY){                                  /*<branch
       butterfly*/
00184     for(k=0; k< A->steps; k++)                                /*compute
       max communication buffer size*/
00185       if(A->nR[k] > nRmax)
00186         nRmax = A->nR[k];
00187     for(k=0; k< A->steps; k++)
00188       if(A->nS[k] > nSmax)
00189         nSmax = A->nS[k];
00190
00191     double *com_val;
00192     com_val=(double *) malloc( A->com_count *sizeof(double));
00193     for(i=0; i < A->com_count; i++){
00194       com_val[i]=0.0;
00195     }
00196     m2m(lvalues, A->lindices, A->lcount, com_val, A->
      com_indices, A->com_count);
00197     butterfly_reduce(A->R, A->nR, nRmax, A->S, A->nS,
      nSmax, com_val, A->steps, A->comm);
00198     m2m(com_val, A->com_indices, A->com_count,
      out_values, A->lindices, A->lcount);
00199     free(com_val);
00200   }
00201   else if(A->flag == RING){
00202     for(k=1; k< A->steps; k++)                                  /*compute
       max communication buffer size*/
00203       if(A->nR[k] > nRmax)
00204         nRmax = A->nR[k];
00205
00206     nSmax = nRmax;
00207     ring_reduce(A->R, A->nR, nRmax, A->S, A->nS, nSmax,
      lvalues, out_values, A->steps, A->comm);
00208   }
00209   else if(A->flag == NONBLOCKING){
00210     ring_nonblocking_reduce(A->R, A->nR, A->S, A->nS
      , lvalues, out_values, A->steps, A->comm);
00211   }
00212   else if(A->flag == NOEMPTY){
00213     int ne=0;
00214     for(k=1; k< A->steps; k++)
00215       if(A->nR[k]!=0)
00216         ne++;
00217     ring_noempty_reduce(A->R, A->nR, ne, A->S, A->nS,
      ne, lvalues, out_values, A->steps, A->comm);
00218   }
00219   else{
00220     return 1;
00221   }
00222 #endif
00223   free(lvalues);
00224   return 0;
00225 }
00226
00227
00228
```

15.36 mapmatc.h File Reference

Data Structures

- struct CMat

  Matrix structure
  A∗ = (A0∗ | A1∗ | ... | Ap-1∗)

Functions

- int CMatInit (CMat ∗A, int r, int ∗m, int ∗nnz, int ∗∗indices, double ∗∗values, int flag#ifdef W_MPI, MPI_Comm comm#endif)
- int CMatFree (CMat ∗A)

- int CMatComShape (CMat *A, int flag)
- int CMatVecProd (CMat *A, double *x, double *y, int pflag)
- int CTrMatVecProd (CMat *A, double *y, double *x, int pflag)

15.36.1 Detailed Description

Author

Pierre Cargemel

Date

October 2012

Definition in file mapmatc.h.

15.36.2 Function Documentation

15.36.2.1 int CMatInit (CMat * *A*, int *r*, int * *m*, int * *nnz*, int ** *indices*, double ** *values*, int flag#ifdef *W_MPI*, MPI_Comm comm# *endif*)

Definition at line 17 of file mapmatc.c.

15.36.2.2 int CMatFree (CMat * *A*)

Definition at line 75 of file mapmatc.c.

15.36.2.3 int CMatComShape (CMat * *A*, int *flag*)

Definition at line 96 of file mapmatc.c.

15.36.2.4 int CMatVecProd (CMat * *A*, double * *x*, double * *y*, int *pflag*)

Definition at line 139 of file mapmatc.c.

15.36.2.5 int CTrMatVecProd (CMat * *A*, double * *y*, double * *x*, int *pflag*)

Definition at line 159 of file mapmatc.c.

15.37 mapmatc.h

```
00001
00009 #define NONE 0
00010 #define RING 1
00011 #define BUTTERFLY 2
00012 #define NONBLOCKING 3
00013 #define NOEMPTY 4
00014 #define SEQ 0
00015 #define OMP 1
00016 #define GPU 2
00017
00018
00019
00023 typedef struct {
00024   int           flag;                       // flag for communication
       scheme (NONE, RING, BUTTERFLY ...)
00025   int           r;                    // number of local coarse space
```

```
00026   int           *m;                    // table containing number of rows in
       each caorse space
00027   int           *nnz;                   // number non-zero per row in each
       corse space
00028   int           *disp;                  // displacement
00029   int           **indices;               // column rows indices tab;
00030   double        **values;         // non-zero values tab;
00031   //--------local shaping---------------
00032   int           lcount;
00033   int           *lindices;              // local indices tab (monotony
       with global numbering); size = lcount
00034 #ifdef W_MPI
00035   MPI_Comm      comm;                   // MPI communicator
00036   //--------com shaping-----------------
00037   int           *com_indices, com_count;// communicated indices tab,
       and size
00038   int           steps;                     // number of steps in the
       communication scheme
00039   int           *nS, *nR;             // number of indices (to send and to
       receive); size = steps
00040   int           **R, **S;               // sending or receiving indices tab
00041 #endif
00042 }CMat;
00043
00044
00045 int CMatInit(CMat *A, int r, int *m, int *nnz, int **indices,
      double **values, int flag
00046 #ifdef W_MPI
00047   ,MPI_Comm comm
00048 #endif
00049 );
00050
00051 int CMatFree(CMat *A);
00052
00053 #ifdef W_MPI
00054 int CMatComShape(CMat *A, int flag);
00055 #endif
00056
00057 int CMatVecProd(CMat *A, double *x, double *y, int pflag);
00058
00059 int CTrMatVecProd(CMat *A, double *y, double* x, int pflag);
00060
00061
```

15.38 midapack.dox File Reference

15.39 mkdoc.dox File Reference

Functions

- v s you should set this option
  to YES to or you want to show
  the then doxygen will reuse
  the documentation of the first
  this can be done per class
  using the nosubgrouping
  command a typedef of a or enum
  is documented as struct union
  or enum with the name of the
  typedef So typedef struct
  TypeS projects (< 1000 input files) the default value isSYMBOL_CACHE_SIZE=0EXTRACT_ALL
- namespace and class members
  alphabetically by member name
  If set to NO (the default) the members will appear inSORT_BRIEF_DOCS

Variables

- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a struct
- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a union
- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS
- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS TypeT
- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS file
- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct

TypeS namespace

- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS referenced

- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS TYPEDEF_HIDES_STRUCT

- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS which are defined in the implementation section but not in EXTRACT_LOCAL_METHODS

- v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS which are defined in the implementation section but not in the members of anonymous namespaces will be EXTRACT_ANON_NSPACES

- v s you should set this option

to YES to or you want to show
the then doxygen will reuse
the documentation of the first
this can be done per class
using the nosubgrouping
command a typedef of a or enum
is documented as struct union
or enum with the name of the
typedef So typedef struct
TypeS which are defined in the
implementation section but not
in the members of anonymous
namespaces will be Doxygen
will hide all HIDE_UNDOC_MEMBERS

- v s you should set this option
  to YES to or you want to show
  the then doxygen will reuse
  the documentation of the first
  this can be done per class
  using the nosubgrouping
  command a typedef of a or enum
  is documented as struct union
  or enum with the name of the
  typedef So typedef struct
  TypeS which are defined in the
  implementation section but not
  in the members of anonymous
  namespaces will be Doxygen
  will hide all Doxygen will
  hide all HIDE_UNDOC_CLASSES

- v s you should set this option
  to YES to or you want to show
  the then doxygen will reuse
  the documentation of the first
  this can be done per class
  using the nosubgrouping
  command a typedef of a or enum
  is documented as struct union
  or enum with the name of the
  typedef So typedef struct
  TypeS which are defined in the
  implementation section but not
  in the members of anonymous
  namespaces will be Doxygen
  will hide all Doxygen will
  hide all Doxygen will hide all HIDE_FRIEND_COMPOUNDS

- v s you should set this option

to YES to or you want to show
the then doxygen will reuse
the documentation of the first
this can be done per class
using the nosubgrouping
command a typedef of a or enum
is documented as struct union
or enum with the name of the
typedef So typedef struct
TypeS which are defined in the
implementation section but not
in the members of anonymous
namespaces will be Doxygen
will hide all Doxygen will
hide all Doxygen will hide all
Doxygen will hide any HIDE_IN_BODY_DOCS

- DOXYFILE_ENCODING
- it will be relative to the location OUTPUT_DIRECTORY
- it will be relative to the
location then doxygen will
create CREATE_SUBDIRS
- it will be relative to the
location then doxygen will
create other supported
languages are
- it will be relative to the
location then doxygen will
create other supported
languages Arabic
- it will be relative to the
location then doxygen will
create other supported
languages Brazilian
- it will be relative to the
location then doxygen will
create other supported
languages Catalan
- it will be relative to the
location then doxygen will
create other supported
languages Chinese
- it will be relative to the
location then doxygen will
create other supported
languages Chinese Traditional
- it will be relative to the
location then doxygen will
create other supported
languages Chinese OUTPUT_LANGUAGE
- it will be relative to the
location then doxygen will
create other supported
languages Chinese the REPEAT_BRIEF
- it will be relative to the
location then doxygen will
create other supported
languages Chinese the if found
as the leading text of the

brief description

- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be ABBREVIATE_BRIEF
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all INLINE_INHERITED_MEMB
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells STRIP_FROM_INC_PATH
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter SHORT_NAMES
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc JAVADOC_AUTOBRIEF
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments QT_AUTOBRIEF
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce SEPARATE_MEMBER_PAGES

- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which ALIASES
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list OPTIMIZE_OUTPUT_FOR_C
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list namespaces will be presented as packages
- it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list namespaces will be presented as qualified OPTIMIZE_OUTPUT_JAVA
- v s BUILTIN_STL_SUPPORT
- v s you should set this option to YES to CPP_CLI_SUPPORT
- v s you should set this option to YES to or you want to show the IDL_PROPERTY_SUPPORT

- v s you should set this option
  to YES to or you want to show
  the then doxygen will reuse
  the documentation of the first DISTRIBUTE_GROUP_DOC
- v s you should set this option
  to YES to or you want to show
  the then doxygen will reuse
  the documentation of the first
  this can be done per class
  using the nosubgrouping
  command SUBGROUPING

15.39.1 Function Documentation

15.39.1.1 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS projects ( ) [pure virtual]

15.39.1.2 namespace and class members alphabetically by member name If set to the class list will be sorted by fully qualified names including namespaces If set to NO (the *default*)

15.39.2 Variable Documentation

15.39.2.1 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a struct

Definition at line 264 of file mkdoc.dox.

15.39.2.2 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a union

Definition at line 264 of file mkdoc.dox.

15.39.2.3 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS

Definition at line 266 of file mkdoc.dox.

15.39.2.4 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS TypeT

15.39.2.5 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS file

15.39.2.6 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS namespace

15.39.2.7 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as **struct union** or enum with the name of the typedef So typedef **struct TypeS** referenced

TYPEDEF_HIDES_STRUCT

15.39.2.8 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as **struct union** or enum with the name of the typedef So typedef **struct TypeS** which are defined in the implementation section but not in EXTRACT_LOCAL_METHODS

Initial value:

```
 NO

# If this flag is set to YES
```

Definition at line 324 of file mkdoc.dox.

15.39.2.9 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as **struct union** or enum with the name of the typedef So typedef **struct TypeS** which are defined in the implementation section but not in the members of anonymous namespaces will be EXTRACT_ANON_NSPACES

Initial value:

```
 NO

# If the HIDE_UNDOC_MEMBERS tag is set to YES
```

Definition at line 332 of file mkdoc.dox.

15.39.2.10 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as **struct union** or enum with the name of the typedef So typedef **struct TypeS** which are defined in the implementation section but not in the members of anonymous namespaces will be Doxygen will hide all HIDE_UNDOC_MEMBERS

Initial value:

```
 NO

# If the HIDE_UNDOC_CLASSES tag is set to YES
```

Definition at line 340 of file mkdoc.dox.

15.39.2.11 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as **struct union** or enum with the name of the typedef So typedef **struct TypeS** which are defined in the implementation section but not in the members of anonymous namespaces will be Doxygen will hide all Doxygen will hide all HIDE_UNDOC_CLASSES

Initial value:

```
 NO

# If the HIDE_FRIEND_COMPOUNDS tag is set to YES
```

Definition at line 347 of file mkdoc.dox.

15.39.2.12 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS which are defined in the implementation section but not in the members of anonymous namespaces will be Doxygen will hide all Doxygen will hide all Doxygen will hide all HIDE_FRIEND_COMPOUNDS

Initial value:

```
 NO

# If the HIDE_IN_BODY_DOCS tag is set to YES
```

Definition at line 354 of file mkdoc.dox.

15.39.2.13 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command a typedef of a or enum is documented as struct union or enum with the name of the typedef So typedef struct TypeS which are defined in the implementation section but not in the members of anonymous namespaces will be Doxygen will hide all Doxygen will hide all Doxygen will hide all Doxygen will hide any HIDE_IN_BODY_DOCS

Definition at line 361 of file mkdoc.dox.

15.39.2.14 DOXYFILE_ENCODING

Initial value:

```
 UTF-8

# The PROJECT_NAME tag is a single word (or a sequence of words surrounded
# by quotes) that should identify the project.

PROJECT_NAME           = ANR-MIDAS'09_CMB_DA_library

# The PROJECT_NUMBER tag can be used to enter a project or revision number.
# This could be handy for archiving the generated documentation or
# if some version control system is used.

PROJECT_NUMBER         = 1.0


# The OUTPUT_DIRECTORY tag is used to specify the (relative or absolute)
# base path where the generated documentation will be put.
# If a relative path is entered
```

Definition at line 23 of file mkdoc.dox.

15.39.2.15 it will be relative to the location OUTPUT_DIRECTORY

Initial value:

```
 .

# If the CREATE_SUBDIRS tag is set to YES
```

Definition at line 42 of file mkdoc.dox.

15.39.2.16 it will be relative to the location then doxygen will create CREATE_SUBDIRS

Initial value:

```
 NO

# The OUTPUT_LANGUAGE tag is used to specify the language in which all
# documentation generated by doxygen is written. Doxygen will use this
# information to generate all constant output in the proper language.
# The default language is English
```

Definition at line 51 of file mkdoc.dox.

15.39.2.17 it will be relative to the location then doxygen will create other supported languages are

Definition at line 51 of file mkdoc.dox.

15.39.2.18 it will be relative to the location then doxygen will create other supported languages Arabic

Definition at line 51 of file mkdoc.dox.

15.39.2.19 it will be relative to the location then doxygen will create other supported languages Brazilian

Definition at line 51 of file mkdoc.dox.

15.39.2.20 it will be relative to the location then doxygen will create other supported languages Catalan

Definition at line 51 of file mkdoc.dox.

15.39.2.21 it will be relative to the location then doxygen will create other supported languages Chinese

Definition at line 51 of file mkdoc.dox.

15.39.2.22 it will be relative to the location then doxygen will create other supported languages Chinese Traditional

Definition at line 51 of file mkdoc.dox.

15.39.2.23 it will be relative to the location then doxygen will create other supported languages Chinese OUTPUT_LANGUAGE

Initial value:

```
 English

# If the BRIEF_MEMBER_DESC tag is set to YES (the default) Doxygen will
# include brief member descriptions after the members that are listed in
# the file and class documentation (similar to JavaDoc).
# Set to NO to disable this.

BRIEF_MEMBER_DESC      = YES

# If the REPEAT_BRIEF tag is set to YES (the default) Doxygen will prepend
# the brief description of a member or function before the detailed
       description.
# Note: if both HIDE_UNDOC_MEMBERS and BRIEF_MEMBER_DESC are set to NO
```

Definition at line 64 of file mkdoc.dox.

15.39.2.24 it will be relative to the location then doxygen will create other supported languages Chinese the REPEAT_BRIEF

Initial value:

```
 YES

# This tag implements a quasi-intelligent brief description abbreviator
# that is used to form the text in various listings. Each string
# in this list
```

Definition at line 78 of file mkdoc.dox.

15.39.2.25 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief description

Definition at line 78 of file mkdoc.dox.

15.39.2.26 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be ABBREVIATE_BRIEF

Initial value:

```
# If the ALWAYS_DETAILED_SEC and REPEAT_BRIEF tags are both set to YES then
# Doxygen will generate a detailed section even if there is only a brief
# description.

ALWAYS_DETAILED_SEC    = NO

# If the INLINE_INHERITED_MEMB tag is set to YES
```

Definition at line 90 of file mkdoc.dox.

15.39.2.27 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all INLINE_INHERITED_MEMB

Initial value:

```
 NO

# If the FULL_PATH_NAMES tag is set to YES then Doxygen will prepend the full
# path before files name in the file list and in the header files. If set
# to NO the shortest path that makes the file name unique will be used.

FULL_PATH_NAMES        = NO

# If the FULL_PATH_NAMES tag is set to YES then the STRIP_FROM_PATH tag
# can be used to strip a user-defined part of the path. Stripping is
# only done if one of the specified strings matches the left-hand part of
# the path. The tag can be used to show relative paths in the file list.
# If left blank the directory from which doxygen is run is used as the
# path to strip.

STRIP_FROM_PATH        =

# The STRIP_FROM_INC_PATH tag can be used to strip a user-defined part of
# the path mentioned in the documentation of a class
```

Definition at line 103 of file mkdoc.dox.

15.39.2.28 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells STRIP_FROM_INC_PATH

Initial value:

```
# If the SHORT_NAMES tag is set to YES
```

Definition at line 127 of file mkdoc.dox.

15.39.2.29 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter SHORT_NAMES

Initial value:

```
 NO

# If the JAVADOC_AUTOBRIEF tag is set to YES then Doxygen
# will interpret the first line (until the first dot) of a JavaDoc-style
# comment as the brief description. If set to NO
```

Definition at line 133 of file mkdoc.dox.

15.39.2.30 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc JAVADOC_AUTOBRIEF

Initial value:

```
 NO

# If the QT_AUTOBRIEF tag is set to YES then Doxygen will
# interpret the first line (until the first dot) of a Qt-style
# comment as the brief description. If set to NO
```

Definition at line 141 of file mkdoc.dox.

15.39.2.31 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments QT_AUTOBRIEF

Initial value:

```
 NO

# The MULTILINE_CPP_IS_BRIEF tag can be set to YES to make Doxygen
# treat a multi-line C++ special comment block (i.e. a block of
# comments) as a brief description. This used to be the default behaviour.
# The new default is to treat a multi-line C++ comment block as a detailed
# description. Set this tag to YES if you prefer the old behaviour instead.

MULTILINE_CPP_IS_BRIEF = NO

# If the INHERIT_DOCS tag is set to YES (the default) then an undocumented
# member inherits the documentation from any documented member that it
# re-implements.

INHERIT_DOCS           = YES

# If the SEPARATE_MEMBER_PAGES tag is set to YES
```

Definition at line 149 of file mkdoc.dox.

15.39.2.32 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce SEPARATE_MEMBER_PAGES

Initial value:

```
"sideeffect=\par Side Effects:\n" will allow you to
# put the command \sideeffect (or @sideeffect) in the documentation
```

Definition at line 169 of file mkdoc.dox.

15.39.2.33 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which ALIASES

Initial value:

```
# Set the OPTIMIZE_OUTPUT_FOR_C tag to YES if your project consists of C
# sources only. Doxygen will then generate output that is more tailored for C.
# For instance
```

Definition at line 183 of file mkdoc.dox.

15.39.2.34 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list OPTIMIZE_OUTPUT_FOR_C

Initial value:

```
 YES

# Set the OPTIMIZE_OUTPUT_JAVA tag to YES if your project consists of Java
# sources only. Doxygen will then generate output that is more tailored for
# Java. For instance
```

Definition at line 190 of file mkdoc.dox.

15.39.2.35 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list namespaces will be presented as packages

Definition at line 190 of file mkdoc.dox.

15.39.2.36 it will be relative to the location then doxygen will create other supported languages Chinese the if found as the leading text of the brief will be doxygen will show all which tells doxygen will generate much shorter the JavaDoc the comments then doxygen will produce which some of the names that are used will be different The list namespaces will be presented as qualified OPTIMIZE_OUTPUT_JAVA

Initial value:

```
 NO

# Set the OPTIMIZE_FOR_FORTRAN tag to YES if your project consists of Fortran
# sources only. Doxygen will then generate output that is more tailored for
# Fortran.

OPTIMIZE_FOR_FORTRAN   = NO

# Set the OPTIMIZE_OUTPUT_VHDL tag to YES if your project consists of VHDL
# sources. Doxygen will then generate output that is tailored for
# VHDL.

OPTIMIZE_OUTPUT_VHDL   = NO

# Doxygen selects the parser to use depending on the extension of the files it
      parses.
# With this tag you can assign which parser to use for a given extension.
# Doxygen has a built-in mapping
```

Definition at line 197 of file mkdoc.dox.

15.39.2.37 v s BUILTIN_STL_SUPPORT

Initial value:

```
 NO

# If you use Microsoft's C++/CLI language
```

Definition at line 229 of file mkdoc.dox.

15.39.2.38 v s you should set this option to YES to CPP_CLI_SUPPORT

Initial value:

```
 NO

# Set the SIP_SUPPORT tag to YES if your project consists of sip sources only.
# Doxygen will parse them like normal C++ but will assume all classes use
       public
# instead of private inheritance when no explicit protection keyword is
       present.

SIP_SUPPORT            = NO

# For Microsoft's IDL there are propget and propput attributes to indicate
       getter
# and setter methods for a property. Setting this option to YES (the default)
# will make doxygen to replace the get and set methods by a property in the
# documentation. This will only work if the methods are indeed getting or
# setting a simple type. If this is not the case
```

Definition at line 234 of file mkdoc.dox.

15.39.2.39 v s you should set this option to YES to or you want to show the IDL_PROPERTY_SUPPORT

Initial value:

```
 YES

# If member grouping is used in the documentation and the DISTRIBUTE_GROUP_DOC
# tag is set to YES
```

Definition at line 249 of file mkdoc.dox.

15.39.2.40 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first DISTRIBUTE_GROUP_DOC

Initial value:

```
 NO

# Set the SUBGROUPING tag to YES (the default) to allow class member groups of
# the same type (for instance a group of public functions) to be put as a
# subgroup of that type (e.g. under the Public Functions section). Set it to
# NO to prevent subgrouping. Alternatively
```

Definition at line 256 of file mkdoc.dox.

15.39.2.41 v s you should set this option to YES to or you want to show the then doxygen will reuse the documentation of the first this can be done per class using the nosubgrouping command SUBGROUPING

Initial value:

```
 YES

# When TYPEDEF_HIDES_STRUCT is enabled
```

Definition at line 264 of file mkdoc.dox.

15.40 ring.c File Reference

Implementation of routines for ring-like communication scheme.

Functions

- int ring_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int steps, MPI_Comm comm)

  Initialize tables for ring-like communication scheme.
- int ring_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.
- int alltoallv_reduce (int **R, int *nR, int nRtot, int **S, int *nS, int nStot, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using an MPI-Alltoallv call.
- int ring_nonblocking_reduce (int **R, int *nR, int **S, int *nS, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking communication scheme.
- int ring_noempty_reduce (int **R, int *nR, int nneR, int **S, int *nS, int nneS, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking no-empty communication scheme.
- int ring_noempty_step_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.

15.40.1 Detailed Description

Implementation of routines for ring-like communication scheme.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Pierre Cargemel

Date

April 2012

Definition in file ring.c.

15.41 ring.c

```
00001
00008 #ifdef W_MPI
00009
00010 #include <mpi.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013 #include <stdio.h>
00014
00033 int ring_init(int *indices, int count, int **R, int *nR, int **S, int
      *nS, int steps, MPI_Comm comm){
00034   int err, p, tag;
00035   int size, rank, sp, rp;
00036   int *buf, nbuf;
00037   MPI_Request s_request, r_request;
00038
00039   MPI_Comm_size(comm, &size);
00040   MPI_Comm_rank(comm, &rank);
00041   MPI_Allreduce( &count, &nbuf, 1, MPI_INT, MPI_MAX, comm);             //
      compute the buffer size : max(count)_{comm}
00042   buf = (int* ) malloc(nbuf*sizeof(int));                               //
      allocate buffer
00043   tag=0;
00044   for (p=1; p < steps; p++){            //communication phase to get nb shared
       indices between peer of preocesses
00045     sp=(rank+p)%size;
00046     rp=(rank+size-p)%size;
00047     MPI_Isend( &count, 1, MPI_INT, sp  , 0, comm, &s_request);          //send
       my number of indices
00048     MPI_Irecv( &nbuf, 1, MPI_INT, rp, 0, comm, &r_request);             //
      receive a number of indices
00049     tag++;
00050     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00051     MPI_Irecv( buf, nbuf, MPI_INT, rp, tag, comm, &r_request);          //
      receive indices tab
00052     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00053     MPI_Isend( indices, count, MPI_INT, sp, tag, comm, &s_request);     //send
       indices tab
00054     tag++;
00055
00056     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00057     nR[p] = card_and(indices, count, buf, nbuf);
      //compute number of shared indices
00058     nS[steps-p]=nR[p];
00059     R[p] = (int* ) malloc(nR[p]*sizeof(int));                           //
      allocate receiving tab
00060     S[steps-p] = (int* ) malloc(nS[steps-p]*sizeof(int));               //
      allocate sanding tab
00061     map_and(indices, count, buf, nbuf, R[p]);
      //fill receiving tab
00062     S[steps-p]=R[p];                                                    //
00063   }
00064   free(buf);
00065   nS[0]=0; //
00066   nR[0]=0; //
00067   return 0;
00068 }
00069
00083 int ring_reduce(int **R, int *nR, int nRmax, int **S, int *nS, int
      nSmax, double *val, double *res_val, int steps, MPI_Comm comm){
00084   int tag, rank, size, p;
00085   MPI_Request s_request, r_request;
00086   int sp, rp;
00087   double *sbuf, *rbuf;
00088
00089   MPI_Comm_size(comm, &size);
00090   MPI_Comm_rank(comm, &rank);
00091   tag=0;
00092
00093   rbuf = (double *) malloc(nRmax * sizeof(double));
00094   sbuf = (double *) malloc(nSmax * sizeof(double));
00095
00096   for (p=1; p < steps; p++){
00097     rp=(rank+size-p)%size;
00098     MPI_Irecv(rbuf, nR[p], MPI_DOUBLE, rp, tag, comm, &r_request);
```

```
00099     sp=(rank+p)%size;
00100     m2s(val, sbuf, S[p], nS[p]); //fill the sending buffer
00101     MPI_Isend(sbuf, nS[p], MPI_DOUBLE, sp, tag, comm, &s_request);
00102
00103     tag++;
00104
00105     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00106     s2m_sum(res_val, rbuf, R[p], nR[p]); //sum receive buffer into
      values
00107
00108     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00109   }
00110   free(sbuf);
00111   free(rbuf);
00112   return 0;
00113 }
00114
00115
00129 int alltoallv_reduce(int **R, int *nR, int nRtot, int **S, int
      *nS, int nStot, double *val, double *res_val, int steps, MPI_Comm comm){
00130         int rank, size, p;
00131         MPI_Request s_request, r_request;
00132         int sp, rp, *rindx, *sindx, *rdisp, *sdisp;
00133         double *sbuf, *rbuf;
00134
00135
00136         MPI_Comm_size(comm, &size);  // N.B. size and steps must be equal,
      shall we check for this ?! -- rs
00137         MPI_Comm_rank(comm, &rank);
00138
00139         rbuf = (double *) malloc(nRtot * sizeof(double));
00140         sbuf = (double *) malloc(nStot * sizeof(double));
00141
00142         rindx = (int *)calloc( size, sizeof(int));
00143         sindx = (int *)calloc( size, sizeof(int));
00144
00145         rdisp = (int *)calloc( size, sizeof(int));
00146         sdisp = (int *)calloc( size, sizeof(int));
00147
00148         // compute shifts ...
00149
00150         for (p=0; p<steps; p++){    // starts with 0 !
00151                 rp=(rank+size-p)%size;
00152                 rindx[rp] = nR[p];
00153                 sp=(rank+p)%size;
00154                 sindx[sp] = nS[p];
00155         }
00156
00157         for( p=1; p<size; p++) {
00158           sdisp[p] = sdisp[p-1]+sindx[p-1];
00159           rdisp[p] = rdisp[p-1]+rindx[p-1];
00160         }
00161
00162         // prepare data to send ...
00163
00164         for (p=0; p<steps; p++){
00165                 sp=(rank+p)%size;
00166                 m2s(val, &sbuf[sdisp[sp]], S[p], nS[p]); //fill the sending
      buffer
00167         }
00168
00169         MPI_Alltoallv(sbuf,sindx,sdisp,MPI_DOUBLE,rbuf,rindx,rdisp,MPI_DOUBLE,
      comm);
00170
00171         // accumulate contributions ...
00172
00173         for (p=0; p<steps; p++){
00174                 rp=(rank+size-p)%size;
00175                 s2m_sum(res_val, &rbuf[rdisp[rp]], R[p], nR[p]); // sum
      receive buffer into values
00176         }
00177
00178         free(sdisp);
00179         free(rdisp);
00180         free(sindx);
00181         free(rindx);
00182         free(sbuf);
00183         free(rbuf);
00184
00185         return 0;
00186 }
00187
00199 int ring_nonblocking_reduce(int **R, int *nR, int **S,
      int *nS, double *val, double *res_val, int steps, MPI_Comm comm){
00200   int tag, rank, size, p;
00201   MPI_Request *s_request, *r_request;
00202   int sp, rp;
```

```
00203   double **sbuf, **rbuf;
00204
00205   MPI_Comm_size(comm, &size);
00206   MPI_Comm_rank(comm, &rank);
00207     //printf("\n non_blocking rank %d", rank);
00208
00209   s_request =  (MPI_Request *) malloc((steps-1) * sizeof(MPI_Request));
00210   r_request =  (MPI_Request *) malloc((steps-1) * sizeof(MPI_Request));
00211
00212   rbuf = (double **) malloc((steps-1) * sizeof(double *));
00213   sbuf = (double **) malloc((steps-1) * sizeof(double *));
00214
00215   for (p=1; p < steps; p++){
00216     //printf("\n buf alloc %d", p);
00217     rbuf[p-1] = (double *) malloc(nR[p] * sizeof(double));
00218     sbuf[p-1] = (double *) malloc(nS[p] * sizeof(double));
00219     m2s(val, sbuf[p-1], S[p], nS[p]); //fill the sending buffer
00220   }
00221
00222   tag=0;
00223   for (p=1; p < steps; p++){
00224     //printf("\n isend  %d", p);
00225     sp=(rank+p)%size;
00226     rp=(rank+size-p)%size;
00227
00228     MPI_Irecv(rbuf[p-1], nR[p], MPI_DOUBLE, rp, tag, comm, &r_request[p-1]);
00229     MPI_Isend(sbuf[p-1], nS[p], MPI_DOUBLE, sp, tag, comm, &s_request[p-1]);
00230     tag++;
00231   }
00232   MPI_Waitall(size-1, r_request, MPI_STATUSES_IGNORE);
00233
00234   for (p=1; p < steps; p++){
00235     s2m_sum(res_val, rbuf[p-1], R[p], nR[p]); //sum receive buffer into
      values
00236   }
00237   MPI_Waitall(size-1, s_request, MPI_STATUSES_IGNORE);
00238   free(r_request);
00239   free(s_request);
00240   free(sbuf);
00241   free(rbuf);
00242   return 0;
00243 }
00244
00257 int ring_noempty_reduce(int **R, int *nR, int nneR, int **S,
      int *nS, int nneS, double *val, double *res_val, int steps, MPI_Comm comm){
00258   int tag, rank, size, p;
00259   MPI_Request *s_request, *r_request;
00260   int sp, rp, nesi, neri;
00261   double **sbuf, **rbuf;
00262
00263   MPI_Comm_size(comm, &size);
00264   MPI_Comm_rank(comm, &rank);
00265     //printf("\n non_blocking rank %d", rank);
00266
00267   s_request =  (MPI_Request *) malloc(nneS * sizeof(MPI_Request));
00268   r_request =  (MPI_Request *) malloc(nneR * sizeof(MPI_Request));
00269
00270   rbuf = (double **) malloc(nneR * sizeof(double *));
00271   sbuf = (double **) malloc(nneS * sizeof(double *));
00272
00273   nesi=0;
00274   for (p=1; p < steps; p++){
00275     if(nS[p] != 0){
00276       sbuf[nesi] = (double *) malloc(nS[p] * sizeof(double));
00277       m2s(val, sbuf[nesi], S[p], nS[p]); //fill the sending buffer
00278       nesi++;
00279     }
00280   }
00281
00282   tag=0;
00283   nesi=0;
00284   neri=0;
00285   for (p=1; p < steps; p++){
00286     sp=(rank+p)%size;
00287     rp=(rank+size-p)%size;
00288     if(nR[p] != 0){
00289       rbuf[neri] = (double *) malloc(nR[p] * sizeof(double));
00290       MPI_Irecv(rbuf[neri], nR[p], MPI_DOUBLE, rp, tag, comm, &r_request[neri])
    ;
00291       neri++;
00292     }
00293     if(nS[p] != 0){
00294       MPI_Isend(sbuf[nesi], nS[p], MPI_DOUBLE, sp, tag, comm, &s_request[nesi])
    ;
00295       nesi++;
00296     }
00297     tag++;
```

```
00298   }
00299   MPI_Waitall(nneR, r_request, MPI_STATUSES_IGNORE);
00300
00301   neri=0;
00302   for (p=1; p < steps; p++){
00303     if(nR[p] != 0){
00304       s2m_sum(res_val, rbuf[neri], R[p], nR[p]); //sum receive buffer
       into values
00305       neri++;
00306     }
00307   }
00308   MPI_Waitall(nneS, s_request, MPI_STATUSES_IGNORE);
00309   free(r_request);
00310   free(s_request);
00311   free(sbuf);
00312   free(rbuf);
00313   return 0;
00314 }
00315
00316 //=========================================================Modification added by
       Sebastien Cayrols : 01/09/2015 ; Berkeley
00317
00331 int ring_noempty_step_reduce(int **R, int *nR, int
      nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps,
      MPI_Comm comm){
00332   int tag, rank, size, p;
00333   MPI_Request s_request, r_request;
00334   int sp, rp;
00335   double *sbuf, *rbuf;
00336
00337   MPI_Comm_size(comm, &size);
00338   MPI_Comm_rank(comm, &rank);
00339   tag=0;
00340
00341   rbuf = (double *) malloc(nRmax * sizeof(double));
00342   sbuf = (double *) malloc(nSmax * sizeof(double));
00343
00344   for (p=1; p < steps; p++){
00345     rp=(rank+size-p)%size;
00346     if(nR[p] != 0)
00347       MPI_Irecv(rbuf, nR[p], MPI_DOUBLE, rp, tag, comm, &r_request);
00348     sp=(rank+p)%size;
00349     if(nS[p] != 0){
00350       m2s(val, sbuf, S[p], nS[p]); //fill the sending buffer
00351       MPI_Isend(sbuf, nS[p], MPI_DOUBLE, sp, tag, comm, &s_request);
00352     }
00353     tag++;
00354
00355     if(nR[p] != 0){
00356       MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00357       s2m_sum(res_val, rbuf, R[p], nR[p]); //sum receive buffer into
       values
00358     }
00359     if(nS[p] != 0)
00360       MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00361   }
00362   free(sbuf);
00363   free(rbuf);
00364   return 0;
00365 }
00366
00367 #endif
```

15.42 ring.h File Reference

Declaration of routines for ring-like communication scheme.

Functions

- int ring_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int steps, MPI_Comm comm)

  Initialize tables for ring-like communication scheme.
- int ring_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.
- int ring_noempty_step_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps, MPI_Comm comm)

Perform a sparse sum reduction (or mapped reduction) using a ring-like communication scheme.

- int ring_nonblocking_reduce (int **R, int *nR, int **S, int *nS, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a ring-like non-blocking communication scheme.
- int ring_noniempty_reduce (int **R, int *nR, int nneR, int **S, int *nS, int nneS, double *val, double *res_val, int steps, MPI_Comm comm)
- int alltoallv_reduce (int **R, int *nR, int nRtot, int **S, int *nS, int nStot, double *val, double *res_val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using an MPI-Alltoallv call.

15.42.1 Detailed Description

Declaration of routines for ring-like communication scheme.

Note

Copyright (C) 2010 APC CNRS Université Paris Diderot program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. should have received a copy of the GNU General Public License along with this program; if not, see http://www.gnu.org/licenses/gpl.html

Author

Pierre Cargemel

Date

April 2012

Definition in file ring.h.

15.42.2 Function Documentation

15.42.2.1 int ring_noniempty_reduce (int ** *R*, int * *nR*, int *nneR*, int ** *S*, int * *nS*, int *nneS*, double * *val*, double * *res_val*, int *steps*, MPI_Comm *comm*)

15.43 ring.h

```
00001
00007 int ring_init(int *indices, int count, int **R, int *nR, int **S, int
      *nS, int steps, MPI_Comm comm);
00008
00009 int ring_reduce(int **R, int *nR, int nRmax, int **S, int *nS, int
      nSmax, double *val, double *res_val, int steps, MPI_Comm comm);
00010
00011 int ring_noempty_step_reduce(int **R, int *nR, int
      nRmax, int **S, int *nS, int nSmax, double *val, double *res_val, int steps,
      MPI_Comm comm);
00012
00013 int ring_nonblocking_reduce(int **R, int *nR, int **S,
      int *nS, double *val, double *res_val, int steps, MPI_Comm comm);
00014
00015 int ring_noniempty_reduce(int **R, int *nR, int nneR, int
      **S, int *nS, int nneS, double *val, double *res_val, int steps, MPI_Comm comm);
00016
00017 int alltoallv_reduce(int **R, int *nR, int nRtot, int **S, int
      *nS, int nStot, double *val, double *res_val, int steps, MPI_Comm comm);
```

15.44 toeplitz.c File Reference

Contains the main part of the sequential routines for Toeplitz algebra.

Functions

- int print_error_message (int error_number, char const *file, int line)

  Prints error message corresponding to an error number.
- int define_blocksize (int n, int lambda, int bs_flag, int fixed_bs)

  Defines an optimal size of the block used in the sliding windows algorithm.
- int define_nfft (int n_thread, int flag_nfft, int fixed_nfft)

  Defines the number of simultaneous ffts for the Toeplitz matrix product computation.
- int tpltz_init (int n, int lambda, int *nfft, int *blocksize, fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, Flag flag_stgy)

  Sets a block size and initializes all fftw arrays and plans needed for the computation.
- int fftw_init_omp_threads (int fftw_n_thread)

  Initialize omp threads for fftw plans.
- int rhs_init_fftw (int *nfft, int fft_size, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag)

  Initializes fftw array and plan for the right hand side, general matrix V.
- int circ_init_fftw (double *T, int fft_size, int lambda, fftw_complex **T_fft)

  Initializes fftw array and plan for the circulant matrix T_circ obtained from T.
- int tpltz_cleanup (fftw_complex **T_fft, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b)

  Cleans fftw workspace used in the Toeplitz matrix matrix product's computation.
- int copy_block (int ninrow, int nincol, double *Vin, int noutrow, int noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int outrow, int outcol, double norm, int set_zero_flag)

  Copies (and potentially reshapes) a selected block of the input matrix to a specified position of the output matrix.
- int scmm_direct (int fft_size, int nfft, fftw_complex *C_fft, int ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.
- int scmm_basic (double **V, int blocksize, int m, fftw_complex *C_fft, double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix by a matrix using FFT's (an INTERNAL routine)
- int stmm_core (double **V, int n, int m, double *T, fftw_complex *T_fft, int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f, fftw_plan plan_b, int flag_offset, int flag_nofft)

  Performs the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm. (an INTERNAL routine)
- int stmm_main (double **V, int n, int m, int id0, int l, double *T, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy)

  Performs the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping. (an INTERNAL routine)
- int mpi_stmm (double **V, int n, int m, int id0, int l, double *T, int lambda, Flag flag_stgy, MPI_Comm comm)

  Performs the product of a Toeplitz matrix by a general matrix using MPI. We assume that the matrix has already been scattered. (a USER routine)

Variables

- int VERBOSE

  Verbose mode.
- int VERBOSE_FIRSTINIT = 1
- int PRINT_RANK = -1

15.44.1 Detailed Description

Contains the main part of the sequential routines for Toeplitz algebra. version 1.2b, November 2012

Author

Frederic Dauvergne, Maude Le Jeune, Antoine Rogier, Radek Stompor

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz.c.

15.44.2 Variable Documentation

15.44.2.1 int VERBOSE

Verbose mode.

Prints some informative messages during the computation.

Definition at line 78 of file toeplitz.c.

15.44.2.2 int VERBOSE_FIRSTINIT = 1

Definition at line 79 of file toeplitz.c.

15.44.2.3 int PRINT_RANK = -1

Definition at line 82 of file toeplitz.c.

15.45 toeplitz.c

```
00001
00059 #include "toeplitz.h"
00060
00061 //r1.2 - Frederic Dauvergne (APC)
00062 //This file contains the main part of the Toeplitz algebra module. This include
00063 //the elementary product routines (using FFT) and initialization routines.
00064 //This also contains the mpi version of the Toeplitz matrix product with global
00065 //row-wise order distribution of the data.
00066 //
00067 //todo:
00068 //- add in stmm non blocking communication as it is done for the stbmm routine
00069 //- scmm_direct dont need nfft parameter
00070
00071
00072 //================================================================================
00073 //Global parameters
00074
00076
00078 int VERBOSE;
00079 int VERBOSE_FIRSTINIT=1;
00080
00081 //Parameter just to know the rank for printing when VERBOSE mode is on
00082 int PRINT_RANK = -1;
00083
00084
00085 //================================================================================
00086
00088
00093 int print_error_message(int error_number, char const *file
      , int line)
00094 {
00095   char *str_mess;
00096   str_mess = (char *) malloc(100 * sizeof(char));
00097   if(error_number == 1)
00098     sprintf (str_mess, "Error on line %d of %s. Toeplitz band width > vector
       size\n", line, file);
00099   if(error_number == 2)
00100     sprintf (str_mess, "Error on line %d of %s. Bad allocation.\n", line, file)
      ;
00101   if(error_number == 3)
00102     sprintf (str_mess, "Error on line %d of %s. Error at fftw multithread
       initialization.\n", line, file);
00103   if(error_number == 7)
00104     sprintf (str_mess, "Error on line %d of %s.\n", line, file);
00105   fprintf(stderr, "%s", str_mess);
00106   printf("%s", str_mess);
00107   return error_number;
00108
00109 }
00110
00111
00112 //================================================================================
00113
00115
```

```
00130 int define_blocksize(int n, int lambda, int bs_flag, int
      fixed_bs)
00131 {
00132   int bs;  //computed optimal block size
00133   int min_bs;  //minimum block size used for the computation
00134   int min_pow2;  //minimum power of two index used for the block size
      computation
00135
00136   //cheating
00137 //  bs_flag = 5;//1;//5;
00138 //  fixed_bs = pow(2,15);  //2^14 winner because smaller block than 2^15 (as
      same speed)
00139
00140   if (bs_flag==1) {
00141     bs = fixed_bs;
00142
00143   }
00144   else if (bs_flag==2) {  //this formula need to be check - seems there is a pb
00145     min_bs = 2*lambda;    //when bs = 2 lambda. Not enough data left in the
      middle
00146     min_pow2 = (int) ceil( log(min_bs)/log(2) );
00147     bs = pow(2, min_pow2);
00148     if (bs > n)       //This is to avoid block size much bigger than the
      matrix. Append mostly
00149       bs = min_bs;  //when the matrix is small compared to his bandwith
00150
00151   }
00152   else if (bs_flag==3) {
00153     min_bs = 3*lambda;
00154     min_pow2 = (int) ceil( log(min_bs)/log(2) );
00155     bs = pow(2, min_pow2);
00156     if (bs > n)       //This is to avoid block size much bigger than the
      matrix. Append mostly
00157       bs = min_bs;  //when the matrix is small compared to his bandwith
00158
00159   }
00160   else if (bs_flag==4 || bs_flag==0) {
00161     min_bs = 4*lambda;
00162     min_pow2 = (int) ceil( log(min_bs)/log(2) );
00163     bs = pow(2, min_pow2);
00164     if (bs > n)       //This is to avoid block size much bigger than the
      matrix. Append mostly
00165       bs = min_bs;  //when the matrix is small compared to his bandwith
00166
00167   }
00168   else if (bs_flag==5) {
00169     //Different formula to compute the optimal block size
00170     bs=1;
00171     while(bs < 2*(lambda-1)*log(bs+1) && bs<n) {
00172       bs = bs*2;  }
00173
00174   }
00175   else if (bs_flag==6) { //the same as bs_flag==5 but with constrain on the
      minimal size
00176                                       // and the number of subblocks.
00177
00178     min_bs = 4*lambda;
00179     min_pow2 = (int) ceil( log(min_bs)/log(2) );
00180
00181     min_pow2 = max(min_pow2, pow(2,14)); //add condition to have a minimum size
      2^14 for bs
00182                                         //This is based on empirical
      estimation and can be justified
00183                                         //by the speed benchmark of FFTW3 (see
      the FFTW official website).
00184     bs = pow(2, min_pow2);
00185
00186     if (bs > n)       //This is to avoid block size much bigger than the
      matrix. Append mostly
00187       bs = min_bs;  //when the matrix is small compared to his bandwith
00188
00189 //test if enough subblock for sliding windows algorithm:
00190 //    int nbloc_bs = ceil( (1.0*n)/(bs-2*distcorrmin));
00191 //    if (nbloc_bs<8) //Empirical condition to avoid small number of subblocks
00192 //      bs = 0;   //Switch to no sliding windows algorithm
00193
00194   }
00195   else {
00196     printf("Error. Wrong value for bs_flag. Set to auto mode.\n");
00197     min_bs = 4*lambda;
00198     min_pow2 = (int) ceil( log(min_bs)/log(2) );
00199     bs = pow(2, min_pow2);
00200     if (bs > n)       //This is to avoid block size much bigger than the
      matrix. Append mostly
00201       bs = min_bs;  //when the matrix is small compared to his bandwith
00202   }
00203
```

```
00204
00205   if(PRINT_RANK==0 && VERBOSE>1)
00206     printf("Computed optimal blocksize is %d (with lambda = %d)\n", bs, lambda)
      ;
00207
00208   return bs;
00209 }
00210
00211
00212 //=========================================================================
00213
00215
00220 int define_nfft(int n_thread, int flag_nfft, int fixed_nfft)
00221 {
00222   int nfft;
00223
00224   if (flag_nfft==0)
00225     nfft = NFFT_DEFAULT;
00226   else if (flag_nfft==1)
00227     nfft = fixed_nfft;
00228   else if (flag_nfft==2)
00229     nfft = n_thread;
00230   else {
00231     printf("Error. Wrong value for flag_nfft. Set to auto mode.\n");
00232     nfft = NFFT_DEFAULT;
00233   }
00234
00235   return nfft;
00236 }
00237
00238
00239 //=========================================================================
00240
00242
00257 int tpltz_init(int n, int lambda, int *nfft, int *blocksize,
      fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *
      plan_f, fftw_plan *plan_b, Flag flag_stgy)
00258 {
00259   int n_thread;
00260   double t1, t2;
00261
00262   //Set the VERBOSE global variable
00263   VERBOSE = flag_stgy.flag_verbose;
00264
00265
00266   //initialize block size
00267   *blocksize = define_blocksize(n, lambda, flag_stgy.flag_bs
      , flag_stgy.fixed_bs);
00268
00269
00270   //if (bs==0)
00271 //     flag_stgy.flag_bs = 9999 //swich to noslidingwindowsalgo
00272
00273
00274 //#pragma omp parallel
00275 //{  n_thread = omp_get_num_threads(); }
00276
00277 //  if ((NB_OMPTHREADS <= n_thread) && (NB_OMPTHREADS != 0))
00278 //    omp_set_num_threads(NB_OMPTHREADS);
00279
00280   n_thread = omp_get_max_threads();
00281
00282
00283   //initialize nfft
00284   *nfft = define_nfft(n_thread, flag_stgy.flag_nfft,
      flag_stgy.fixed_nfft);   //*nfft=n_thread;
00285
00286
00287   if(PRINT_RANK==0 && VERBOSE>0 && VERBOSE_FIRSTINIT
      ==1) {
00288     printf("Using %d threads\n", n_thread);
00289     printf("nfft = %d\n", *nfft);
00290   }
00291
00292   //initialize fftw plan allocation flag
00293   int fftw_flag = flag_stgy.flag_fftw; //FFTW_FLAG;
00294
00295   //initialize fftw for omp threads
00296 #ifdef fftw_MULTITHREADING
00297   fftw_init_omp_threads(n_thread);
00298 #endif
00299
00300   //initialize fftw array and plan for T (and make it circulant first)
00301 // t1=MPI_Wtime();
00302   circ_init_fftw(T, (*blocksize), lambda, T_fft);
00303 //  t2=  MPI_Wtime();
00304
```

```
00305 //  if (PRINT_RANK==0 && VERBOSE>0)
00306 //    printf("time circ_init_fftw=%f\n", t2-t1);
00307
00308   //initialize fftw array and plan for V
00309 // t1=MPI_Wtime();
00310   rhs_init_fftw(nfft, (*blocksize), V_fft, V_rfft, plan_f, plan_b,
       fftw_flag);
00311 //  t2=  MPI_Wtime();
00312
00313 //  if (PRINT_RANK==0 && VERBOSE>0)
00314 //    printf("time rhs_init_fftw=%f\n", t2-t1);
00315
00316   if(PRINT_RANK==0 && VERBOSE>1)
00317     printf("Initialization finished successfully\n");
00318
00319   VERBOSE_FIRSTINIT=0;
00320
00321   return 0;
00322 }
00323
00324
00325 //=========================================================================
00326
00328
00333 int fftw_init_omp_threads(int fftw_n_thread)
00334 {
00335   int status;
00336
00337   //initialize fftw omp threads
00338   status = fftw_init_threads();
00339   if (status==0)
00340     return print_error_message (3, __FILE__, __LINE__);
00341
00342   //set the number of FFTW threads
00343   fftw_plan_with_nthreads(fftw_n_thread);
00344
00345   if(PRINT_RANK==0 && VERBOSE>1 && VERBOSE_FIRSTINIT
      ==1)
00346     printf("Using multithreaded FFTW with %d threads\n", fftw_n_thread);
00347
00348   return 0;
00349 }
00350
00351
00352 //=========================================================================
00353
00355
00365 int rhs_init_fftw(int *nfft, int fft_size, fftw_complex **V_fft,
      double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag)
00366 {
00367   //allocate fftw arrays and plans for V
00368   *V_fft  = (fftw_complex*) fftw_malloc((*nfft)*(fft_size/2+1) * sizeof(
      fftw_complex) );
00369   *V_rfft = (double*) fftw_malloc((*nfft)*fft_size * sizeof(double) );
00370   if (*V_fft==0 || *V_rfft==0)
00371     return print_error_message (2, __FILE__, __LINE__);
00372
00373   *plan_f = fftw_plan_many_dft_r2c(1, &fft_size, (*nfft), *V_rfft, &fft_size, 1
      , fft_size, *V_fft, NULL, 1, fft_size/2+1, fftw_flag );
00374   *plan_b = fftw_plan_many_dft_c2r(1, &fft_size, (*nfft), *V_fft, NULL, 1,
      fft_size/2+1, *V_rfft, &fft_size, 1, fft_size, fftw_flag );
00375
00376
00377   return 0;
00378 }
00379
00380
00381 //=========================================================================
00382
00384
00392 int circ_init_fftw(double *T, int fft_size, int lambda,
      fftw_complex **T_fft)
00393 {
00394   //routine variable
00395   int i;
00396   int circ_fftw_flag = FFTW_ESTIMATE;
00397   //allocation for T_fft
00398   *T_fft = (fftw_complex*) fftw_malloc( (fft_size/2+1) * sizeof(fftw_complex) )
      ;
00399   if (*T_fft==0)
00400     return print_error_message (2, __FILE__, __LINE__);
00401   double *T_circ = (double*) (*T_fft);
00402
00403   //inplace fft
00404   fftw_plan plan_f_T;
00405   plan_f_T   = fftw_plan_dft_r2c_1d( fft_size, T_circ, *T_fft, circ_fftw_flag )
      ;
```

```
00406
00407   //make T circulant
00408 #pragma omp parallel for
00409   for(i=0; i<fft_size+2;i++)
00410     T_circ[i] = 0.0;
00411
00412   T_circ[0] = T[0];
00413   for(i=1;i<lambda;i++) {
00414     T_circ[i] = T[i];
00415     T_circ[fft_size-i] = T[i];    }
00416
00417   fftw_execute(plan_f_T);
00418   fftw_destroy_plan(plan_f_T);
00419
00420   return 0;
00421 }
00422
00423
00424 //=========================================================================
00425
00427
00435 int tpltz_cleanup(fftw_complex **T_fft, fftw_complex **V_fft,
      double **V_rfft,fftw_plan *plan_f, fftw_plan *plan_b){
00436   fftw_destroy_plan(*plan_f);
00437   fftw_destroy_plan(*plan_b);
00438   fftw_free(*T_fft);
00439   fftw_free(*V_fft);
00440   fftw_free(*V_rfft);
00441 #ifdef fftw_MULTITHREADING
00442   fftw_cleanup_threads();
00443 #endif
00444   fftw_cleanup();
00445 }
00446
00447
00448 //=========================================================================
00449
00451
00459 int copy_block(int ninrow, int nincol, double *Vin, int noutrow, int
      noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int
      outrow, int outcol, double norm, int set_zero_flag)
00460 {
00461   int i, j, p, offsetIn, offsetOut;
00462
00463   //do some size checks first
00464   if( (nblockcol > nincol) || (nblockrow > ninrow) || (nblockcol > noutcol) ||
      (nblockrow > noutrow)) {
00465     printf("Error in routine copy_block. Bad size setup.\n");
00466     return print_error_message(7, __FILE__, __LINE__);
00467   }
00468
00469   if(set_zero_flag) {
00470 #pragma omp parallel for //private(i) num_threads(NB_OMPTHREADS_CPBLOCK)
00471     for(i=0;i<noutrow*noutcol;i++)  //could use maybe memset but how about
       threading
00472       Vout[i] = 0.0;
00473   }
00474
00475   offsetIn = ninrow*incol+inrow;
00476   offsetOut = noutrow*outcol+outrow;
00477
00478 //#pragma omp parallel for private(i,j,p) num_threads(NB_OMPTHREADS_CPBLOCK)
00479   for(i=0;i<nblockcol*nblockrow;i++) {    //copy the block
00480     j = i/nblockrow;
00481     p = i%nblockrow;
00482     Vout[offsetOut+j*noutrow+p] = Vin[offsetIn+j*ninrow+p]*norm;
00483   }
00484
00485   return 0;
00486 }
00487
00488
00489 //=========================================================================
00490
00492
00509 int scmm_direct(int fft_size, int nfft, fftw_complex *C_fft, int
      ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V,
      fftw_plan plan_b_CV)
00510 {
00511   //routine variables
00512   int sizeT = fft_size/2+1;
00513   int i, idx;
00514
00515   //perform forward FFT
00516   fftw_execute(plan_f_V); //input in V_rfft; output in V_fft
00517
00518 //  printf("ncol=%d, fft_size=%d, sizeT=%d\n", ncol, fft_size, sizeT);
```

```
00519
00520 //double t1, t2;
00521 //  t1=MPI_Wtime();
00522
00523 #pragma omp parallel for private(idx) //num_threads(nfft)
00524   for(i=0;i<ncol*sizeT;i++) {
00525     idx = i%sizeT;
00526     V_fft[i][0] = C_fft[idx][0]*V_fft[i][0]-C_fft[idx][1]*V_fft[i][1];
00527     V_fft[i][1] = C_fft[idx][0]*V_fft[i][1]+C_fft[idx][1]*V_fft[i][0];    }
00528
00529 //  t2=  MPI_Wtime();
00530 //  printf("Computation time : %lf s.\n", t2-t1);
00531
00532
00533 // This is wrong :
00534 /*
00535 int icol;
00536 double t1, t2;
00537   t1=MPI_Wtime();
00538 #pragma omp parallel for private(i, idx)
00539   for(icol=0;icol<ncol;icol++) {
00540   for(idx=0;idx<sizeT;idx++) {
00541     i=icol*idx;
00542     V_fft[i][0] = C_fft[idx][0]*V_fft[i][0]-C_fft[idx][1]*V_fft[i][1];
00543     V_fft[i][1] = C_fft[idx][0]*V_fft[i][1]+C_fft[idx][1]*V_fft[i][0];
00544   }}
00545   t2=  MPI_Wtime();
00546 */
00547 //  printf("Computation time : %lf s.\n", t2-t1);
00548
00549
00550
00551   //perform  backward FFts
00552   fftw_execute(plan_b_CV); //input in V_fft; output in V_rfft
00553
00554   return 0;
00555 }
00556
00557
00558 //=========================================================================
00559
00561
00587 int scmm_basic(double **V, int blocksize, int m, fftw_complex *C_fft,
       double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V,
       fftw_plan plan_b_CV)
00588 {
00589   //routine variables
00590   int i,k; //loop index
00591   int nloop = (int) ceil((1.0*m)/nfft);  //number of subblocks
00592
00593   // Loop over set of columns
00594   int ncol = min(nfft, m);  //a number of columns to be copied from the data to
       working matrix
00595                             //equal the number of simultaneous FFTs
00596
00597
00598 #pragma omp parallel for //num_threads(NB_OMPTHREADS_BASIC)//
      schedule(dynamic,1)
00599   for( i=0;i<blocksize*ncol;i++)
00600     V_rfft[i] = 0.0;  //could use maybe memset but how about threading
00601
00602
00603 //bug fixed conflit between num_threads and nfft
00604 //#pragma omp parallel for schedule(dynamic,1) num_threads(8)
       //num_threads(nfft)
00605   for(k=0;k<nloop;k++) {   //this is the main loop over the set of columns
00606     if (k==nloop-1)   //last loop ncol may be smaller than nfft
00607       ncol = m-(nloop-1)*nfft;
00608
00609   //init fftw matrices.
00610   //extracts a block of ncol full-length columns from the data matrix and
       embeds in a bigger
00611   //matrix padding each column with lambda zeros. Note that all columns will be
       zero padded
00612   //thanks to the "memset" call above
00613
00614   copy_block(blocksize, m, (*V), blocksize, ncol, V_rfft, 0, k*nfft,
      blocksize, ncol, 0, 0, 1.0, 0);
00615   //note: all nfft vectors are transformed below ALWAYS in a single go (if ncol
       < nfft) the extra
00616   //useless work is done.
00617
00618   scmm_direct(blocksize, nfft, C_fft, ncol, V_rfft, CV, V_fft,
      plan_f_V, plan_b_CV);
00619   //note: the parameter CV is not really used
00620
00621   //extract the relevant part from the result
```

```
00622   copy_block(blocksize, ncol, V_rfft, blocksize, m, (*CV), 0, 0,
      blocksize, ncol, 0, k*nfft, 1.0/((double) blocksize), 0);
00623
00624   }  //end of loop over the column-sets
00625
00626
00627   return 0;
00628 }
00629
00630
00631 //========================================================================
00632
00634
00659 int stmm_core(double **V, int n, int m, double *T, fftw_complex *T_fft
      , int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft,
      fftw_plan plan_f, fftw_plan plan_b, int flag_offset, int flag_nofft)
00660 {
00661
00662   double t1,t2;
00663
00664   t1=  MPI_Wtime();
00665
00666   //cheating:
00667 //  flag_offset = 1;
00668
00669   //routine variable
00670   int status;
00671   int i,j,k,p;  //loop index
00672   int currentsize;
00673   int distcorrmin= lambda-1;
00674
00675   int blocksize_eff = blocksize-2*distcorrmin;  //just a good part after
       removing the overlaps
00676   int nbloc;  //number of subblock of slide/overlap algorithm
00677
00678   if (flag_offset==1)
00679     nbloc = ceil((1.0*(n-2*distcorrmin))/blocksize_eff);
00680   else
00681     nbloc = ceil( (1.0*n)/blocksize_eff);  //we need n because of reshaping
00682
00683   if(PRINT_RANK==0 && VERBOSE>0)
00684     printf("nbloc=%d, n=%d, m=%d, blocksize=%d, blocksize_eff=%d\n", nbloc, n,
      m, blocksize, blocksize_eff);
00685
00686   double *V_bloc, *TV_bloc;
00687   V_bloc  = (double *) calloc(blocksize*m, sizeof(double));
00688   TV_bloc = (double *) calloc(blocksize*m, sizeof(double));
00689   if((V_bloc==0)||(TV_bloc==0))
00690     return print_error_message(2, __FILE__, __LINE__);
00691
00692   int offset=0;
00693   if (flag_offset==1)
00694     offset=distcorrmin;
00695
00696   int iV = 0;  //"-distcorrmin+offset";  //first index in V
00697   int iTV = offset;  //first index in TV
00698
00699   //"k=0";
00700   //first subblock separately as it requires some padding. prepare the block of
       the data vector
00701   //with the overlaps on both sides
00702   currentsize = min( blocksize-distcorrmin+offset, n-iV);
00703   //note: if flag_offset=0, pad first distcorrmin elements with zeros (for the
       first subblock only)
00704   // and if flag_offset=1 there is no padding with zeros.
00705   copy_block( n, m, *V, blocksize, m, V_bloc, 0, 0, currentsize, m,
      distcorrmin-offset, 0, 1.0, 0);
00706
00707   //do block computation
00708   if (flag_nofft==1)
00709     status = stmm_simple_basic(&V_bloc, blocksize, m, T,
      lambda, &TV_bloc);
00710   else
00711     status = scmm_basic(&V_bloc, blocksize, m, T_fft, &TV_bloc, V_fft
      , V_rfft, nfft, plan_f, plan_b);
00712
00713   if (status!=0) {
00714     printf("Error in stmm_core.");
00715     return print_error_message(7, __FILE__, __LINE__);  }
00716
00717
00718   //now copy first the new chunk of the data matrix **before** overwriting the
       input due to overlaps !
00719   iV = blocksize_eff-distcorrmin+offset;
00720
00721   if(nbloc > 1) {
00722     currentsize  = min( blocksize, n-iV);  //not to overshoot
```

```
00723
00724     int flag_reset = (currentsize!=blocksize);  //with flag_reset=1, always
      "memset" the block.
00725     copy_block( n, m, *V, blocksize, m, V_bloc, iV, 0, currentsize, m
     , 0, 0, 1.0, flag_reset);
00726   }
00727
00728   //and now store the ouput back in V
00729   currentsize  = min( blocksize_eff, n-iTV);       // to trim the extra rows
00730   copy_block( blocksize, m, TV_bloc, n, m, *V, distcorrmin, 0,
      currentsize, m, iTV, 0, 1.0, 0);
00731
00732
00733   iTV += blocksize_eff;
00734   //now continue with all the other subblocks
00735   for(k=1;k<nbloc;k++) {
00736
00737     //do bloc computation
00738   if (flag_nofft==1)
00739     status = stmm_simple_basic(&V_bloc, blocksize, m, T,
      lambda, &TV_bloc);
00740   else
00741     status = scmm_basic(&V_bloc, blocksize, m, T_fft, &TV_bloc, V_fft
     , V_rfft, nfft, plan_f, plan_b);
00742
00743   if (status!=0) break;
00744
00745
00746     iV += blocksize_eff;
00747     //copy first the next subblock to process
00748     if(k != nbloc-1) {
00749       currentsize = min(blocksize, n-iV);  //not to overshoot
00750
00751       int flag_resetk = (currentsize!=blocksize);  //with flag_reset=1, always
      "memset" the block.
00752       copy_block( n, m, *V, blocksize, m, V_bloc, iV, 0, currentsize,
      m, 0, 0, 1.0, flag_resetk);
00753     }
00754
00755     //and then store the output in V
00756     currentsize  = min( blocksize_eff, n-iTV);  //not to overshoot

00757     copy_block( blocksize, m, TV_bloc, n, m, *V, distcorrmin, 0,
      currentsize, m, iTV, 0, 1.0, 0);
00758     iTV += blocksize_eff;
00759
00760   }//end bloc computation
00761
00762
00763   free(V_bloc);
00764   free(TV_bloc);
00765
00766
00767   t2=  MPI_Wtime();
00768
00769   if (PRINT_RANK==0 && VERBOSE>0)
00770     printf("time stmm_core=%f\n", t2-t1);
00771
00772   return status;
00773 }
00774
00775
00776 //========================================================================
00777
00779
00803 int stmm_main(double **V, int n, int m, int id0, int l, double *T,
      fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan
      plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy)
00804 {
00805
00806   //routine variable
00807   int i,j,k,p;  //loop index
00808   int distcorrmin= lambda-1;
00809   int flag_prod_strategy_nofft=0;  //0: ffts   1: no ffts
00810   int flag_shortcut_m_eff_eq_1=1;//1;//1;
00811   int flag_shortcut_nbcol_eq_1=1;//1;//1;
00812   int flag_nfullcol_in_middle=0;//0; //in the case where m=1 can be good to
      direct stmm_core too
00813   int flag_optim_offset_for_nfft=0;
00814   int flag_no_rshp=flag_stgy.flag_no_rshp;//0;
00815   int flag_nofft=flag_stgy.flag_nofft;//1;
00816
00817   int m_eff     = (id0+l-1)/n - id0/n + 1 ;  //number of columns
00818   int nfullcol;
00819   int nloop_middle;  //change it to number of full column to improve memory
00820
00821   FILE *file;
```

```
00822   file = stdout;
00823
00824   if (l<distcorrmin) //test to avoid communications errors
00825     return print_error_message (1, __FILE__, __LINE__);
00826
00827
00828 //shortcut for m==1 if flag_shortcut_m_eff_eq_1==1  && nfft==1 ??
00829   if (m_eff==1 && flag_shortcut_m_eff_eq_1==1 && nfft==1 || flag_no_rshp==1 &&
      id0==0 && l==n*m) {
00830
00831     int flag_offset=0;
00832
00833 //  if (flag_prod_strategy_nofft==1)   //need to have T as input to make it
       work
00834     // stmm_simple_core(V, n, m, T, blocksize, lambda, nfft, flag_offset);
00835 //  else
00836     int nr=min(l,n);
00837     stmm_core(V, nr, m_eff, T, T_fft, blocksize, lambda, V_fft, V_rfft
      , nfft, plan_f, plan_b, flag_offset, flag_nofft);
00838
00839
00840     return 0;
00841   }//End shortcut for m==1
00842
00843
00844 //the middle
00845   int m_middle;
00846
00847 //define splitting for the product computation
00848   nfullcol = max(0, (l-(n-id0%n)%n-(id0+l)%n)/n );  //check how many full
       columns input data we have
00849
00850   if (flag_nfullcol_in_middle==1)
00851     nloop_middle = ceil(1.0*(nfullcol)/nfft);
00852   else
00853     nloop_middle = (nfullcol)/nfft;
00854
00855   if (flag_nfullcol_in_middle==1)
00856     m_middle = nfullcol;
00857   else
00858     m_middle = nfft*nloop_middle;
00859
00860
00861   int vmiddle_size = n*m_middle;
00862
00863
00864   if(PRINT_RANK==0 && VERBOSE>2)
00865     printf("nloop_middle=%d , m_middle=%d\n", nloop_middle, m_middle);
00866
00867
00868 //compute the middle if needed
00869   if (nloop_middle>0) {
00870     double *Vmiddle;
00871     int offset_middle = (n-id0%n)%n;
00872     Vmiddle = (*V)+offset_middle;
00873
00874     int flag_offset=0;
00875     stmm_core(&Vmiddle, n, m_middle, T, T_fft, blocksize, lambda,
      V_fft, V_rfft, nfft, plan_f, plan_b, flag_offset, flag_nofft);
00876
00877   } //(nloop_middle>0)
00878
00879
00880 //edge  (first+last columns + extra column from the euclidian division)
00881   int v1edge_size = min(l,(n-id0%n)%n);
00882   int v2edge_size = max( l-(v1edge_size+vmiddle_size) , 0);
00883   int vedge_size = v1edge_size+v2edge_size;
00884
00885 //compute the edges if needed
00886   if (vedge_size>0) {
00887
00888   int m_v1edge, m_v2edge;
00889   m_v1edge = (v1edge_size>0)*1;  //m_v1 = 1 or 0 cannot be more
00890   m_v2edge = m-(m_v1edge+m_middle);
00891   int nbcol = m_v1edge+m_v2edge;
00892   int *nocol;
00893   nocol = (int *) calloc(nbcol, sizeof(double));
00894
00895   //define the columns for the edge computation
00896   if (m_v1edge==1)
00897     nocol[0]=0;
00898   for(i=(m_v1edge);i<nbcol;i++)
00899     nocol[i]=m_middle+i;
00900
00901   if(PRINT_RANK==0 && VERBOSE>2)
00902     printf("nbcol=%d , m_v1edge=%d , m_v2edge=%d\n", nbcol, m_v1edge, m_v2edge)
      ;
```

```
00903
00904 //shorcut for nbcol==1
00905   if (nbcol==1 && nfft==1 && flag_shortcut_nbcol_eq_1==1) {
00906             //this is the case where no reshaping is needed. This is equivalent
       to flag_format_rshp==0
00907     double *Vedge;
00908     int offset_edge = n*nocol[0];//work because all the previous columns are
       obligatory full
00909     Vedge = (*V)+offset_edge;
00910     int flag_offset=0;
00911     stmm_core(&Vedge, vedge_size, nbcol, T, T_fft, blocksize, lambda,
      V_fft, V_rfft, nfft, plan_f, plan_b, flag_offset, flag_nofft);
00912
00913   }
00914   else {  //general case to compute de edges
00915
00916   double *Vin;
00917   Vin = (*V);
00918
00919 //size for the different kinds of reshaping
00920   int lconc = vedge_size; //another way to compute : lconc = n*nbcol -
       (nocol[0]==0)*(id0%n) - (nocol[nbcol-1]==(m-1))*(n-(id0+l)%n);
00921   int v1_size=lconc+(distcorrmin)*(nbcol-1);
00922   int fft_size = ceil(1.0*v1_size/nfft)+2*distcorrmin;
00923
00924   int flag_format_rshp = (nfft>1)*2 + (nfft==1 && nbcol>1)*1 + (nfft==1 &&
      nbcol==1)*0;
00925   int nrshp, mrshp, lrshp;
00926
00927   define_rshp_size(flag_format_rshp, fft_size, nfft, v1_size,
      vedge_size, &nrshp, &mrshp, &lrshp);
00928
00929 //allocate Vrshp for computation
00930   double *Vrshp;
00931   Vrshp = (double *) calloc(lrshp, sizeof(double));
00932   double *Vout;
00933   Vout = (*V);
00934
00935   if(PRINT_RANK==0 && VERBOSE>2) {
00936     fprintf(file, "nrshp=%d , mrshp=%d , lrshp=%d\n", nrshp, mrshp, lrshp);
00937     fprintf(file, "flag_format_rshp=%d\n", flag_format_rshp);
00938   }
00939
00940   build_reshape(Vin, nocol, nbcol, lconc, n, m, id0, l, lambda,
      nfft, Vrshp, nrshp, mrshp, lrshp, flag_format_rshp);
00941
00942   int flag_offset;
00943   if (flag_format_rshp==2 && flag_optim_offset_for_nfft==1)
00944     flag_offset=1;
00945   else
00946     flag_offset=0;
00947
00948 //compute Vrshp
00949     stmm_core(&Vrshp, nrshp, mrshp, T, T_fft, blocksize, lambda, V_fft
      , V_rfft, nfft, plan_f, plan_b, flag_offset, flag_nofft);
00950
00951     extract_result(Vout, nocol, nbcol, lconc, n, m, id0, l,
      lambda, nfft, Vrshp, nrshp, mrshp, lrshp, flag_format_rshp);
00952
00953
00954   }//End general case to compute de edges
00955   }//End (vedge_size>0)
00956
00957
00958   return 0;
00959 }
00960
00961
00962 //========================================================================
00963 #ifdef W_MPI
00964
00965
00980 int mpi_stmm(double **V, int n, int m, int id0, int l, double *T, int
      lambda, Flag flag_stgy, MPI_Comm comm)
00981 {
00982
00983   //mpi variables
00984   int rank;    //rank process
00985   int size;    //number of processes
00986   MPI_Status status;
00987   MPI_Comm_rank(comm, &rank);
00988   MPI_Comm_size(comm, &size);
00989
00990
00991   //routine variables
00992   int i,j,k; // some index
00993   int idf = id0+l;  // first index of scattered V for rank "rank + 1";
```

```
00994   int cfirst = id0/n;   // first column index
00995   int clast  = idf/n; // last column index
00996   int clast_r = (idf-1)/n;
00997   int m_eff  = clast_r - cfirst + 1 ;
00998   double *V1, *Lambda;
00999
01000   // Mpi communication conditions
01001   // Mpi comm is needed when columns are truncated
01002   int right = rank + 1;
01003   int left  = rank - 1;
01004   int v1_size = l + 2*lambda; // size including comm
01005   if (rank==0 || cfirst*n==id0){ // no left comm
01006     v1_size -= lambda;
01007     left = MPI_PROC_NULL;}
01008   if (rank==(size-1) || clast*n==idf){ // no right comm
01009     v1_size -= lambda;
01010     right = MPI_PROC_NULL;}
01011
01012   // init data to send
01013   Lambda=(double *) malloc(2*lambda * sizeof(double));
01014   if (Lambda==0)
01015     return print_error_message (2, __FILE__, __LINE__);
01016
01017   for(i=0;i<lambda;i++)    {
01018     Lambda[i]=(*V)[i];
01019     Lambda[i+lambda]=(*V)[i+l-lambda];    }
01020
01021   if(PRINT_RANK==0 && VERBOSE>2)
01022     printf("[rank %d] Left comm with %d | Right comm with %d\n", rank, left,
      right);
01023
01024   //send and receive data
01025   MPI_Sendrecv_replace(Lambda, lambda, MPI_DOUBLE, left, MPI_USER_TAG, right,
      MPI_USER_TAG, comm, &status);  //1st comm
01026   MPI_Sendrecv_replace((Lambda+lambda), lambda, MPI_DOUBLE, right, MPI_USER_TAG
      , left, MPI_USER_TAG, comm, &status);  //2nd comm
01027
01028
01029   if (l<lambda)  //After sendrecv to avoid problems of communication for others
       processors
01030     return print_error_message (1, __FILE__, __LINE__);
01031
01032   //copy received data
01033   if(left==MPI_PROC_NULL && right==MPI_PROC_NULL)  // 0--0 : nothing to do
01034     V1 = *V;
01035   else if(left==MPI_PROC_NULL) { // 0--1 : realloc
01036     *V = realloc(*V, v1_size * sizeof(double));
01037     if(*V == NULL)
01038       return print_error_message (2, __FILE__, __LINE__);
01039     V1 = *V;  }
01040   else  // 1--1 or 1--0 : new allocation
01041     V1  = (double *) malloc(v1_size * sizeof(double));
01042
01043   if (left!=MPI_PROC_NULL){
01044     for(i=0;i<lambda;i++)
01045       V1[i] = Lambda[i+lambda];
01046     id0 -= lambda;}
01047   if (right!=MPI_PROC_NULL){
01048     for(i=0;i<lambda;i++)
01049       V1[i+v1_size-lambda] = Lambda[i];      }
01050
01051   // Copy input matrix V
01052   int offset = 0;
01053   if (left!=MPI_PROC_NULL){
01054     offset = lambda;
01055 #pragma omp parallel for
01056     for(i=offset;i<l+offset;i++)
01057       V1[i] = (*V)[i-offset]; }
01058
01059   fftw_complex *V_fft, *T_fft;
01060   double *V_rfft;
01061   fftw_plan plan_f, plan_b;
01062
01063
01064   //Compute matrix product
01065   int nfft,  blocksize;
01066
01067   tpltz_init(v1_size, lambda , &nfft, &blocksize, &T_fft, T, &V_fft,
      &V_rfft, &plan_f, &plan_b, flag_stgy);
01068
01069   if(PRINT_RANK==0 && VERBOSE>1)
01070     printf("[rank %d] Before middle-level call : blocksize=%d, nfft=%d\n", rank
      , blocksize, nfft);
01071
01072   stmm_main(&V1, n, m, id0, v1_size, T, T_fft, lambda, V_fft, V_rfft,
      plan_f, plan_b, blocksize, nfft, flag_stgy);
01073
```

```
01074
01075   tpltz_cleanup(&T_fft, &V_fft, &V_rfft,&plan_f, &plan_b );
01076
01077   // Copy output matrix TV
01078   offset = 0;
01079   if (left!=MPI_PROC_NULL)
01080     offset = lambda;
01081   if(left==MPI_PROC_NULL && right==MPI_PROC_NULL) // 0--0
01082     *V=V1;
01083   else if(left==MPI_PROC_NULL) { // 0--1
01084     V1 = realloc(V1, l * sizeof(double));
01085     if(V1 == NULL)
01086       return print_error_message (2, __FILE__, __LINE__);
01087     *V = V1;      }
01088   else { // 1--0 or 1--1
01089 #pragma omp parallel for
01090     for(i=offset;i<l+offset;i++)
01091       (*V)[i-offset] = V1[i];     }
01092
01093   if(left!=MPI_PROC_NULL)
01094     free(V1);
01095
01096   return 0;
01097 }
01098
01099 #endif
01100
01101
```

15.46 toeplitz.dox File Reference

15.47 toeplitz.h File Reference

Header file with main definitions and declarations for the Toeplitz algebra module.

Data Structures

- struct Block
- struct Flag
- struct Gap
- struct Tpltz

Typedefs

- typedef struct Block Block
- typedef struct Flag Flag
- typedef struct Gap Gap
- typedef struct Tpltz Tpltz

Functions

- int stbmmProd (Tpltz Nm1, double ∗V)

  Performs the product of a Toeplitz matrix by a general matrix either sequentially or using MPI. The complexity is hidden in the input structure, which needs to be defined by a user.

- int tpltz_init (int n, int lambda, int ∗nfft, int ∗blocksize, fftw_complex ∗∗T_fft, double ∗T, fftw_complex ∗∗V_fft, double ∗∗V_rfft, fftw_plan ∗plan_f, fftw_plan ∗plan_b, Flag flag_stgy)

  Sets a block size and initializes all fftw arrays and plans needed for the computation.

- int tpltz_cleanup (fftw_complex ∗∗T_fft, fftw_complex ∗∗V_fft, double ∗∗V_rfft, fftw_plan ∗plan_f, fftw_plan ∗plan_b)

  Cleans fftw workspace used in the Toeplitz matrix matrix product's computation.

- int stmm_core (double **V, int n, int m, double *T, fftw_complex *T_fft, int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f, fftw_plan plan_b, int flag_offset, int flag_nofft)

  Performs the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm. (an INTERNAL routine)
- int stmm_main (double **V, int n, int m, int id0, int l, double *T, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy)

  Performs the product of a Toeplitz matrix by a general matrix using the sliding window algorithm with optimize reshaping. (an INTERNAL routine)
- int stmm (double **V, int n, int m, double *T, int lambda, Flag flag_stgy)

  Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm.
- int stbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.
- int gstbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix with gaps, T, by an arbitrary matrix, V, distributed over processes.
- int reset_gaps (double **V, int id0, int local_V_size, int m, int nrow, int m_rowwise, int64_t *id0gap, int *lgap, int ngap)

  Set the data to zeros at the gaps location.
- int mpi_stmm (double **V, int n, int m, int id0, int l, double *T, int lambda, Flag flag_stgy, MPI_Comm comm)

  Performs the product of a Toeplitz matrix by a general matrix using MPI. We assume that the matrix has already been scattered. (a USER routine)
- int mpi_stbmm (double **V, int64_t nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int local_V_size, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.
- int mpi_gstbmm (double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.
- int flag_stgy_init_auto (Flag *flag_stgy)

  Set the flag to automatic paramaters.
- int flag_stgy_init_zeros (Flag *flag_stgy)

  Set the flag parameters to zeros. This is almost the same as automatic.
- int flag_stgy_init_defined (Flag *flag_stgy)

  Set the parameters flag to the defined ones.
- int print_flag_stgy_init (Flag flag_stgy)

  Print the flag parameters values.
- int define_blocksize (int n, int lambda, int bs_flag, int fixed_bs)

  Defines an optimal size of the block used in the sliding windows algorithm.
- int define_nfft (int n_thread, int flag_nfft, int fixed_nfft)

  Defines the number of simultaneous ffts for the Toeplitz matrix product computation.
- int fftw_init_omp_threads ()
- int rhs_init_fftw (int *nfft, int fft_size, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag)

  Initializes fftw array and plan for the right hand side, general matrix V.
- int circ_init_fftw (double *T, int fft_size, int lambda, fftw_complex **T_fft)

  Initializes fftw array and plan for the circulant matrix T_circ obtained from T.

- int scmm_direct (int fft_size, int nfft, fftw_complex *C_fft, int ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix C_fft by a matrix V_rfft using fftw plans.
- int scmm_basic (double **V, int blocksize, int m, fftw_complex *C_fft, double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V, fftw_plan plan_b_CV)

  Performs the product of a circulant matrix by a matrix using FFT's (an INTERNAL routine)
- int stmm_simple_basic (double **V, int n, int m, double *T, int lambda, double **TV)

  Perform the product of a Toeplitz matrix by a matrix without using FFT's.
- int build_gappy_blocks (int nrow, int m, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t *id0gap, int *lgap, int ngap, Block *tpltzblocks_gappy, int *nb_blocks_gappy_final, int flag_param_distmin_fixed)

  Build the gappy Toeplitz block structure to optimise the product computation at gaps location.
- int print_error_message (int error_number, char const *file, int line)

  Prints error message corresponding to an error number.
- int copy_block (int ninrow, int nincol, double *Vin, int noutrow, int noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int outrow, int outcol, double norm, int set_zero_flag)

  Copies (and potentially reshapes) a selected block of the input matrix to a specified position of the output matrix.
- int vect2nfftblock (double *V1, int v1_size, double *V2, int fft_size, int nfft, int lambda)
- int nfftblock2vect (double *V2, int fft_size, int nfft, int lambda, double *V1, int v1_size)
- int get_overlapping_blocks_params (int nbloc, Block *tpltzblocks, int local_V_size, int64_t nrow, int64_t idp, int64_t *idpnew, int *local_V_size_new, int *nnew, int *ifirstBlock, int *ilastBlock)

Variables

- int VERBOSE

  Verbose mode.
- int PRINT_RANK

15.47.1 Detailed Description

Header file with main definitions and declarations for the Toeplitz algebra module. version 1.2b, November 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research
Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz.h.

15.47.2 Typedef Documentation

15.47.2.1 typedef struct Block Block

15.47.2.2 typedef struct Flag Flag

15.47.2.3 typedef struct Gap Gap

15.47.2.4 typedef struct Tpltz Tpltz

15.47.3 Function Documentation

15.47.3.1 int fftw_init_omp_threads ( )

15.47.3.2 int vect2nfftblock (double ∗ *V1,* int *v1_size,* double ∗ *V2,* int *fft_size,* int *nfft,* int *lambda*)

15.47.3.3 int nfftblock2vect (double ∗ *V2,* int *fft_size,* int *nfft,* int *lambda,* double ∗ *V1,* int *v1_size*)

15.47.3.4 int get_overlapping_blocks_params (int *nbloc,* Block ∗ *tpltzblocks,* int *local_V_size,* int64_t *nrow,* int64_t *idp,* int64_t ∗ *idpnew,* int ∗ *local_V_size_new,* int ∗ *nnew,* int ∗ *ifirstBlock,* int ∗ *ilastBlock*)

15.47.4 Variable Documentation

15.47.4.1 int VERBOSE

Verbose mode.

Prints some informative messages during the computation.

Definition at line 78 of file toeplitz.c.

15.47.4.2 int PRINT_RANK

Definition at line 82 of file toeplitz.c.

15.48 toeplitz.h

```
00001
00059 #ifndef         TOEPLITZ_H_
00060 #define         TOEPLITZ_H_
00061
00062 #ifdef W_MPI
00063 #include <mpi.h>
00064 #endif
00065
00066 #ifdef W_OPENMP
00067 #include <omp.h>
00068 #endif
00069
00070 #include <fftw3.h>
00071 #include <stdlib.h>
00072 #include <stdio.h>
00073 #include <math.h>
00074 #include <string.h>
00075
00076 //=========================================================================
00077 //Basic functions definition
00078 #define max(a,b) (((a) > (b)) ? (a) : (b))
00079 #define min(a,b) (((a) < (b)) ? (a) : (b))
00080
00081 //=========================================================================
00082 //Fixed parameters
00083
00084
00086
00088 #ifndef MPI_USER_TAG
00089 #define MPI_USER_TAG 123
00090 #endif
00091
00092 //Define this parameter to use fftw multithreading
00093 //This is not fully tested
00094 #ifndef fftw_MULTITHREADING
00095 #define fftw_MULTITHREADING
00096 #endif
00097
00099
00101 #ifndef NFFT_DEFAULT
00102 #define NFFT_DEFAULT 1 /*1*/
00103 #endif
00104
00105
00107
00111 #ifndef FFTW_FLAG_AUTO
00112 #define FFTW_FLAG_AUTO FFTW_ESTIMATE
00113 #endif
00114
00115
00116 //Parameters to define the computational strategy
00117 #ifndef FLAG_STGY
00118 #define FLAG_STGY
00119
00120 #define FLAG_BS 0  //0:auto  1:fixed  2:zero  3:3lambda  4:4lambda  5:formula2
00121 #define FLAG_NFFT 0  //0:auto  1:fixed  2:numthreads  3:fftwthreads
00122 #define FLAG_FFTW FFTW_FLAG_AUTO  //ESTIMATE, MEASURE, PATIENT, EXHAUSTIVE.
       Default is MEASURE
00123 #define FLAG_NO_RSHP 0  //0:auto  1:yes  1:no
00124 #define FLAG_NOFFT 0 //0:auto  1:yes  1:no
00125 #define FLAG_BLOCKINGCOMM 0  //0:auto  1:noblocking  2:blocking
00126 #define FIXED_NFFT 0  //fixed init value for nfft
00127 #define FIXED_BS 0   //fixed init value for blockside
00128 #define FLAG_VERBOSE 0
```

```
00129 #define FLAG_SKIP_BUILD_GAPPY_BLOCKS 0
00130 #define FLAG_PARAM_DISTMIN_FIXED 0
00131 #define FLAG_PRECOMPUTE_LVL 0   //0: no precompute  1: precompute plans  2:
       precomputes Toeplitz and plans
00132
00133 #endif
00134
00135 //============================================================================
00136 //Global parameters
00137
00138 extern int VERBOSE;
00139 extern int PRINT_RANK;
00140
00141 //============================================================================
00142 //Strutures definition
00143
00144
00145 typedef struct Block {
00146      int64_t idv;
00147      double *T_block;  //pointer of the Toeplitz data
00148      int lambda;
00149      int n;
00150 /* For precomputed fftw
00151      int bs;
00152      int nfft;
00153      fftw_complex *T_fft;
00154      fftw_complex *V_fft;
00155      double *V_rfft;
00156      fftw_plan plan_f;
00157      fftw_plan plan_b;
00158      fftw_plan plan_f_T;
00159 */
00160 } Block;
00161
00162
00163 typedef struct Flag {
00164        int flag_bs;  //bs used formula
00165        int flag_nfft;
00166        int flag_fftw;
00167        int flag_no_rshp;  //with or without
00168        int flag_nofft;
00169        int flag_blockingcomm;
00170        int fixed_nfft;  //init value for nfft
00171        int fixed_bs;   //long long int
00172        int flag_verbose;
00173        int flag_skip_build_gappy_blocks;
00174        int flag_param_distmin_fixed;
00175        int flag_precompute_lvl;
00176 } Flag;
00177
00178 typedef struct Gap {
00179   int64_t *id0gap;
00180   int *lgap;
00181   int ngap;
00182 } Gap;
00183
00184
00185 typedef struct Tpltz {
00186      int64_t nrow;  //n total
00187      int m_cw;  //V column number in the linear row-wise order (vect
       row-wise order)
00188      int m_rw;  //V column number in the uniform row-wise order (matrix
       row-wise order)
00189      Block *tpltzblocks;
00190      int nb_blocks_loc;
00191      int nb_blocks_tot;
00192      int64_t idp;
00193      int local_V_size;
00194      Flag flag_stgy;
00195      MPI_Comm comm;
00196 } Tpltz;
00197
00198
00199 //============================================================================
00200 //Groups definition for documentation
00201
00248 //============================================================================
00249 // User routines definition (API)
00250
00251 //Wizard routines
00252 int stbmmProd( Tpltz Nm1, double *V);
00253
00254
00255 //Sequential routines (group 11)
00256 int tpltz_init(int n, int lambda, int *nfft, int *blocksize,
      fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *
      plan_f, fftw_plan *plan_b, Flag flag_stgy);
```

```
00257
00258 int tpltz_cleanup(fftw_complex **T_fft, fftw_complex **V_fft,
      double **V_rfft,fftw_plan *plan_f, fftw_plan *plan_b);
00259
00260 int stmm_core(double **V, int n, int m, double *T, fftw_complex *T_fft
      , int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft,
      fftw_plan plan_f, fftw_plan plan_b, int flag_offset, int flag_nofft);
00261
00262 int stmm_main(double **V, int n, int m, int id0, int l, double *T,
      fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan
      plan_f, fftw_plan plan_b, int blocksize, int nfft, Flag flag_stgy);
00263
00264 int stmm(double **V, int n, int m, double *T, int lambda, Flag
      flag_stgy);
00265
00266 //int stbmm(double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks, int
       nb_blocks_local, int nb_blocks_all, int idp, int local_V_size, Flag flag_stgy);
00267
00268 //int gstbmm(double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks,
       int nb_blocks_local, int nb_blocks_all, int id0p, int local_V_size, int *id0gap,
       int *lgap, int ngap,Flag flag_stgy);
00269
00270 int stbmm(double **V, int nrow, int m_cw, int m_rw, Block *
      tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, Flag flag_stgy);
00271
00272 int gstbmm(double **V, int nrow, int m_cw, int m_rw, Block *
      tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, int64_t *id0gap, int *lgap,
      int ngap, Flag flag_stgy);
00273
00274
00275 int reset_gaps(double **V, int id0,int local_V_size, int m, int nrow,
       int m_rowwise, int64_t *id0gap, int *lgap, int ngap);
00276
00277
00278 //Mpi routines (group 12)
00279 #ifdef W_MPI
00280 int mpi_stmm(double **V, int n, int m, int id0, int l, double *T, int
      lambda, Flag flag_stgy, MPI_Comm comm);
00281
00282 int mpi_stbmm(double **V, int64_t nrow, int m, int m_rowwise, Block
       *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int
      local_V_size, Flag flag_stgy, MPI_Comm comm);
00283
00284 int mpi_gstbmm(double **V, int nrow, int m, int m_rowwise, Block
       *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int
      local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy, MPI_Comm comm);
00285
00286 #endif
00287
00288
00289 //========================================================================
00290 // User routines definition
00291
00292 //Low level routines (group 21)
00293 int flag_stgy_init_auto(Flag *flag_stgy);
00294
00295 int flag_stgy_init_zeros(Flag *flag_stgy);
00296
00297 int flag_stgy_init_defined(Flag *flag_stgy);
00298
00299 int print_flag_stgy_init(Flag flag_stgy);
00300
00301 int define_blocksize(int n, int lambda, int bs_flag, int
      fixed_bs);
00302
00303 int define_nfft(int n_thread, int flag_nfft, int fixed_nfft);
00304
00305 int fftw_init_omp_threads();
00306
00307 int rhs_init_fftw(int *nfft, int fft_size, fftw_complex **V_fft,
      double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag);
00308
00309 int circ_init_fftw(double *T, int fft_size, int lambda,
      fftw_complex **T_fft);
00310
00311 int scmm_direct(int fft_size, int nfft, fftw_complex *C_fft, int
      ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V,
      fftw_plan plan_b_CV);
00312
00313 int scmm_basic(double **V, int blocksize, int m, fftw_complex *C_fft,
       double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V,
       fftw_plan plan_b_CV);
00314
00315 int stmm_simple_basic(double **V, int n, int m, double *T, int
       lambda, double **TV);
00316
00317 int build_gappy_blocks(int nrow, int m, Block *
```

```
      tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t *id0gap, int *lgap, int ngap
      , Block *tpltzblocks_gappy, int *nb_blocks_gappy_final, int
      flag_param_distmin_fixed);
00318
00319
00320 //Internal routines (group 22)
00321 int print_error_message(int error_number, char const *file
      , int line);
00322
00323 int copy_block(int ninrow, int nincol, double *Vin, int noutrow, int
      noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int
      outrow, int outcol, double norm, int set_zero_flag);
00324
00325 int vect2nfftblock(double *V1, int v1_size, double *V2, int
      fft_size, int nfft, int lambda);
00326
00327 int nfftblock2vect(double *V2, int fft_size, int nfft, int lambda
      , double *V1, int v1_size);
00328
00329 int get_overlapping_blocks_params(int nbloc, Block
       *tpltzblocks, int local_V_size, int64_t nrow, int64_t idp, int64_t *idpnew, int
       *local_V_size_new, int *nnew, int *ifirstBlock, int *ilastBlock);
00330
00331
00332
00333 //Wizard routines
00334 int stbmmProd( Tpltz Nm1, double *V);
00335
00336 //==========================================================================
00337 #endif      /* !TOEPLITZ_H_ */
```

15.49 toeplitz_block.c File Reference

Contains routines related to the Toeplitz blocks diagonal routine for Toeplitz algebra.

Functions

- int mpi_stbmm (double **V, int64_t nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int local_V_size, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.

15.49.1 Detailed Description

Contains routines related to the Toeplitz blocks diagonal routine for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne, Radek Stompor

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Definition in file toeplitz_block.c.

15.50 toeplitz_block.c

```
00001
00049 #include "toeplitz.h"
00050
00051
00052 //r1.1 - Frederic Dauvergne (APC)
00053 //This is the routines related to the Toeplitz blocks diagonal routine.
00054 //There is a sequential equivalent routine in the file toeplitz_seq.c
00055
00056 //todo:
00057 //- remove the nooptimize communication
00058
00059 //============================================================================
00060 #ifdef W_MPI
00061
00062
00087 int mpi_stbmm(double **V, int64_t nrow, int m, int m_rowwise, Block
       *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int
      local_V_size, Flag flag_stgy, MPI_Comm comm)
00088 {
00089 #else  //for sequential use only
00090 int mpi_stbmm(double **V, int64_t nrow, int m, int m_rowwise, Block
       *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t idp, int
      local_V_size, Flag flag_stgy)
00091 {
00092 #endif
00093
00094
00095   //MPI parameters
00096   int rank;    //process rank
00097   int size;    //process number
00098
00099 #ifdef W_MPI
00100   MPI_Status status;
00101   MPI_Comm_rank(comm, &rank);
00102   MPI_Comm_size(comm, &size);
00103
00104 #else
00105   rank=0;
00106   size=1;
00107 #endif
00108
00109   PRINT_RANK=rank;
```

```
00110
00111   FILE *file;
00112   file = stdout;
00113
00114   int i,j,k;  //some indexes
00115
00116
00117   //identification of the mpi neighbours process to communicate when there is a
       shared block
00118   int right = rank+1;
00119   int left  = rank-1;
00120
00121
00122   //Define the indices for each process
00123   int idv0, idvn;  //indice of the first and the last block of V for each
       processes
00124
00125   int *nnew;
00126   nnew = (int*) calloc(nb_blocks_local, sizeof(int));
00127   int64_t idpnew;
00128   int local_V_size_new;
00129   int n_rowwise=local_V_size;
00130
00131   int status_params = get_overlapping_blocks_params
      ( nb_blocks_local, tpltzblocks, local_V_size, nrow, idp, &idpnew, &
      local_V_size_new, nnew, &idv0, &idvn);
00132
00133
00134   if(PRINT_RANK==0 && VERBOSE>2)
00135     printf("status_params=%d\n", status_params);
00136
00137   if( status_params == 0) {
00138     free(nnew);
00139     return(0); //no work to be done
00140   }
00141
00142   if (tpltzblocks[idv0].lambda==0 || tpltzblocks[idvn].lambda==0)
00143     return print_error_message (2, __FILE__, __LINE__);
00144
00145
00146   if(PRINT_RANK==0 && VERBOSE>2) {  //print on screen news
       parameters definition if VERBOSE
00147     fprintf(file, "new parameters caracteristics:\n");
00148     fprintf(file, "[%d] idp=%ld ; idpnew=%ld\n", rank, idp, idpnew);
00149     fprintf(file, "[%d] local_V_size=%d ; local_V_size_new=%d\n", rank,
      local_V_size, local_V_size_new);
00150     for(i=0;i<nb_blocks_local;i++)
00151       fprintf(file, "[%d] n[%d]=%d ; nnew[%d]=%d\n", rank, i, (tpltzblocks[i].n
      ), i, nnew[i]);
00152     for(i=0;i<nb_blocks_local;i++)
00153       fprintf(file, "[%d] tpltzblocks[%d].idv=%ld\n", rank, i, tpltzblocks[i].
      idv);
00154 }
00155
00156
00157   int vShft=idpnew-idp;   //new first element of relevance in V
00158
00159   //Define the column indices:
00160   //index of the first and the last column of V for the current process
00161   int idvm0 = idpnew/nrow;
00162   int idvmn = (idpnew+local_V_size_new-1)/nrow;
00163   //number of columns of V for the current process
00164   int ncol_rank = idvmn-idvm0+1;
00165   //number of blocks for the current process with possibly repetitions
00166   int nb_blocks_rank;
00167
00168   if(ncol_rank == 1) // Empty process not allowed
00169     nb_blocks_rank = idvn - idv0 + 1;
00170   else
00171     nb_blocks_rank = (ncol_rank-2)*nb_blocks_local + (nb_blocks_local-idv0) + (
      idvn+1);  //in this case nb_blocks_local = nblocs_all
00172
00173   if(PRINT_RANK==0 && VERBOSE>2)
00174   fprintf(file, "[%d] nb_blocks_rank=%d, nb_blocks_local=%d\n", rank,
      nb_blocks_rank, nb_blocks_local);
00175
00176   //Define the indices for the first and the last element in each blocks
00177   int idvp0 = idpnew%nrow-tpltzblocks[idv0].idv;  //index of the first
       element of the process in the first block
00178   int idvpn;  //reverse index of the last element of the process in the last
       block
00179               //It's the number of remaining elements needed to fully complete
       the last block
00180   idvpn = tpltzblocks[idvn].idv+nnew[idvn]-1 - (idpnew+local_V_size_new-1)%
      nrow ;
00181
00182
```

```
00183   //Define the offsets for the first and last blocks of the process for V1
00184   int offset0, offsetn;
00185   int distcorrmin_idv0 = (tpltzblocks[idv0].lambda)-1;
00186   int distcorrmin_idvn = (tpltzblocks[idvn].lambda)-1;
00187
00188   //if(idvp0 != 0)
00189     offset0 = min( idvp0, distcorrmin_idv0);
00190   //if(idvpn != 0)
00191     offsetn = min(idvpn, distcorrmin_idvn);
00192
00193
00194   int toSendLeft=0;
00195   int toSendRight=0;
00196
00197 #ifdef W_MPI
00198   if(offset0!=0) {
00199     toSendLeft = min( tpltzblocks[idv0].idv+nnew[idv0]-idpnew%nrow,
      distcorrmin_idv0);
00200   }
00201   if( offsetn != 0) {
00202     toSendRight = min( (idpnew+local_V_size_new)%nrow-tpltzblocks[idvn].idv,
      distcorrmin_idvn);
00203   }
00204
00205  int flag_optimlambda=1; //to allocate only the memory place needed
00206
00207  int lambdaOut_offset;
00208  int lambdaIn_offset;
00209  double *LambdaOut;
00210  int lambdaOut_size, lambdaIn_size;
00211
00212  if (flag_optimlambda==1) {
00213   LambdaOut=(double *) calloc((toSendLeft+toSendRight)*m_rowwise, sizeof(double
      ));
00214   lambdaOut_offset = toSendLeft*m_rowwise;
00215   lambdaIn_offset = offset0*m_rowwise;
00216   lambdaOut_size = (toSendLeft+toSendRight)*m_rowwise ;
00217   lambdaIn_size = (offset0+offsetn)*m_rowwise;
00218  }
00219  else {
00220   LambdaOut=(double *) calloc((tpltzblocks[idv0].lambda+tpltzblocks[idvn].
      lambda)*m_rowwise, sizeof(double));
00221   lambdaOut_offset = tpltzblocks[idv0].lambda*m_rowwise;
00222   lambdaIn_offset = tpltzblocks[idv0].lambda*m_rowwise;
00223   lambdaOut_size = (tpltzblocks[idv0].lambda+tpltzblocks[idvn].lambda
      )*m_rowwise;
00224   lambdaIn_size = (tpltzblocks[idv0].lambda+tpltzblocks[idvn].lambda
      )*m_rowwise;
00225  }
00226
00227
00228   if(offset0!=0) {
00229     for (j=0;j<m_rowwise;j++)
00230     for (i=0;i<toSendLeft;i++)
00231       LambdaOut[i+j*toSendLeft]=(*V)[i+j*n_rowwise]; //good because
       toSendLeft=0 if it
00232   }                                                  //doesnt start on a the
       first block.
00233   if( offsetn != 0) {
00234     for (j=0;j<m_rowwise;j++)
00235     for (i=0;i<toSendRight;i++)
00236       LambdaOut[i+j*toSendRight+lambdaOut_offset]=(*V)[i+j*n_rowwise+
      local_V_size-toSendRight];
00237   }                                        //good too using same argument than
       for offset0!=0
00238                                            //if
       local_V_size!=local_V_size_new+vShft mean there is extra
00239                                            //terms a the end and so offsetn=0
00240                                            //idpnew+local_V_size_new =
       idp+local_V_size and vShft=idpnew-idp
00241                                            //so
       local_V_size=vShft+local_V_size_new
00242   if(rank==0 || offset0==0)
00243     left = MPI_PROC_NULL;
00244   if(rank==size-1 || offsetn==0)
00245     right = MPI_PROC_NULL;
00246
00247   double *LambdaIn=(double *) calloc(lambdaIn_size, sizeof(double));
00248
00249
00250  int flag_blockingcomm=0;  //to use blocking comm
00251  MPI_Request requestLeft_r, requestLeft_s;
00252  MPI_Request requestRight_r, requestRight_s;
00253
00254   if (flag_blockingcomm==1) {
00255 //send and receive data
00256   MPI_Sendrecv( LambdaOut, toSendLeft*m_rowwise, MPI_DOUBLE, left,
```

```
      MPI_USER_TAG, (LambdaIn+lambdaIn_offset), offsetn*m_rowwise, MPI_DOUBLE, right,
      MPI_USER_TAG, comm, &status);
00257   MPI_Sendrecv( (LambdaOut+lambdaOut_offset), toSendRight*m_rowwise, MPI_DOUBLE
      , right,  MPI_USER_TAG, LambdaIn, offset0*m_rowwise, MPI_DOUBLE, left,
      MPI_USER_TAG, comm, &status);
00258
00259   }
00260   else {
00261 //to the Left
00262   MPI_Irecv((LambdaIn+lambdaIn_offset), offsetn*m_rowwise, MPI_DOUBLE, right,
      MPI_USER_TAG, comm, &requestLeft_r);
00263   MPI_Isend(LambdaOut, toSendLeft*m_rowwise, MPI_DOUBLE, left, MPI_USER_TAG,
      comm, &requestLeft_s);
00264
00265 //to the Right
00266   MPI_Irecv(LambdaIn, offset0*m_rowwise, MPI_DOUBLE, left, MPI_USER_TAG, comm,
      &requestRight_r);
00267   MPI_Isend((LambdaOut+lambdaOut_offset), toSendRight*m_rowwise, MPI_DOUBLE,
      right, MPI_USER_TAG, comm, &requestRight_s);
00268
00269  }
00270
00271 #endif
00272
00273
00274 //size of the first and the last block for the current process
00275   int v0rank_size, vnrank_size;
00276   if (nb_blocks_rank == 1) {  //only one block
00277     v0rank_size = ((idpnew+local_V_size_new-1)%nrow +1) - idpnew%nrow + offset0
       + offsetn;
00278     vnrank_size = 0; //just for convenience - no really need it
00279   }
00280   else { //more than one block
00281     v0rank_size = tpltzblocks[idv0].idv + nnew[idv0] - idpnew%nrow + offset0
      ;
00282     vnrank_size = ((idpnew+local_V_size_new-1)%nrow +1) - tpltzblocks[idvn].idv
       + offsetn;
00283   }
00284
00285
00286 #ifdef W_MPI
00287
00288 if (flag_blockingcomm!=1) {
00289   //MPI_Wait for lambda comm
00290   MPI_Wait(&requestLeft_r, &status);
00291   MPI_Wait(&requestLeft_s, &status);
00292   MPI_Wait(&requestRight_r, &status);
00293   MPI_Wait(&requestRight_s, &status);
00294
00295 }
00296
00297
00298   free(LambdaOut);
00299
00300 #endif
00301
00302
00303 //---------------------------------------
00304 //initialization for the blocks loop
00305
00306   int idv1=0;     //old index of *V1
00307   int idv2=0;     //index
00308
00309
00310   int mid;  //local number of column for the current block
00311   //index of the first element of the process inside the first block
00312   int offset_id0;
00313   offset_id0 = idvp0;
00314
00315 //fftw variables
00316   fftw_complex *V_fft, *T_fft;
00317   double *V_rfft;
00318   fftw_plan plan_f, plan_b;
00319 //init local block vector
00320   double *V1block;
00321 //  int lambdaShft;
00322
00323
00324 //=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
00325 //loop on the blocks inside the process
00326   int nfft, blocksize;
00327   int iblock;  //index for the loop on the blocks
00328 //  int loopindex;
00329   int id; //indice of the current block
00330
00331   int vblock_size;
00332   int id0block;
```

```
  int jj;


  for(iblock=idv0;iblock<idv0+nb_blocks_rank;iblock++) {
    id = iblock%nb_blocks_local;  //index of current block


  if(nnew[id]>0) { //the block is ok

#ifdef W_MPI
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//first case : First block of the process
  if(iblock==idv0) {
  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] First block...\n", rank);

  vblock_size=v0rank_size;
  id0block=(offset_id0-offset0);

  V1block = (double *) calloc(vblock_size*m_rowwise, sizeof(double));

  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<offset0;i++)
    V1block[i+j*vblock_size] = LambdaIn[i+j*offset0];
  }
//note: check if copyblock could be used instead.


//if (nb_blocks_rank == 1) currentsize_middlepart=vblock_size-offset0-offsetn = local_V_size_new
//else currentsize_middlepart=vblock_size-offset0
  int currentsize_middlepart=min(vblock_size-offset0, local_V_size_new);

  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<currentsize_middlepart;i++)
    V1block[offset0+i+j*vblock_size] = (*V)[i+vShft+j*n_rowwise];
  }

if (nb_blocks_rank == 1) {
  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<offsetn;i++) {
    V1block[vblock_size-offsetn+i+j*vblock_size] = LambdaIn[i+lambdaIn_offset+j
*offsetn];
  }}
}


  //init Toeplitz arrays
  tpltz_init(vblock_size, tpltzblocks[id].lambda, &nfft, &blocksize,
&T_fft, (tpltzblocks[id].T_block), &V_fft, &V_rfft, &plan_f, &plan_b, flag_stgy)
;

  //Toeplitz computation
  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] Before stmm_main call : nfft = %d, blocksize = %d\n",
rank, nfft, blocksize);
  stmm_main(&V1block, vblock_size, m_rowwise, 0, m_rowwise*vblock_size
, (tpltzblocks[id].T_block), T_fft, tpltzblocks[id].lambda, V_fft, V_rfft,
plan_f, plan_b, blocksize, nfft, flag_stgy);

  tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);


  int currentsize=min(vblock_size-offset0, local_V_size_new);
  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<currentsize;i++)
    (*V)[vShft+i+j*n_rowwise] = V1block[offset0+i+j*vblock_size];
  }

  free(V1block);

  }//end (First case)


//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//Generic case : Generic block of the process
  else if(iblock!=idv0 && iblock!=idv0+nb_blocks_rank-1) {
#endif

  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] generic block...\n");

```

```
  vblock_size=nnew[id];
  id0block=0;

  V1block = (double *) calloc(vblock_size*m_rowwise, sizeof(double));

  idv1 = (tpltzblocks[id].idv)-idp%nrow - vShft + offset0 +nrow*( (iblock/nb_blocks_local) );  //no need
//  idv2 = idv[id]-idp%nrow + nrow*( (iblock/nb_blocks_local) );
  idv2 = (tpltzblocks[id].idv)-(idpnew)%nrow+vShft + nrow*( (iblock/nb_blocks_local) );

  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<vblock_size;i++)
    V1block[i+j*vblock_size] = (*V)[i+idv2+j*n_rowwise];
//    V1block[i] = (*V)[i+idv1-offset0+vShft];
  }

  //init Toeplitz arrays
  tpltz_init(nnew[id], tpltzblocks[id].lambda, &nfft, &blocksize, &T_fft, (tpltzblocks[id].T_block), &V_fft, &V_rfft, &plan_f, &plan_b, flag_stgy);

  //Toeplitz computation
  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] Before stmm_main call : nfft = %d, blocksize = %d\n", rank, nfft, blocksize);
  stmm_main(&V1block, vblock_size, m_rowwise, 0, m_rowwise*vblock_size, (tpltzblocks[id].T_block), T_fft, tpltzblocks[id].lambda, V_fft, V_rfft, plan_f, plan_b, blocksize, nfft, flag_stgy);


  tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);


  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<vblock_size;i++) {
    (*V)[i+idv2+j*n_rowwise] = V1block[i+j*vblock_size];
  }}


  free(V1block);

#ifdef W_MPI
  }  //end (Generic case)

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// Last case : Last block of the process
  else if(iblock==idv0+nb_blocks_rank-1 && iblock!= idv0) {
  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] last block...\n");

  vblock_size=vnrank_size;
  id0block=0;

  V1block = (double *) calloc(vblock_size*m_rowwise, sizeof(double));

  idv1 = (tpltzblocks[id].idv) - idp%nrow - vShft + offset0  + nrow*( (iblock/nb_blocks_local) );
  idv2 = (tpltzblocks[id].idv)-(idpnew)%nrow+vShft + nrow*( (iblock/nb_blocks_local) );


  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<vblock_size-offsetn;i++)
    V1block[i+j*vblock_size] = (*V)[i+idv2+j*n_rowwise];
//    V1block[i] = (*V)[i+idv1-offset0+vShft];
  }

  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<offsetn;i++)
    V1block[vblock_size-offsetn+i+j*vblock_size] = LambdaIn[i+lambdaIn_offset+j*offsetn];
  }


  //init Toeplitz arrays
  tpltz_init(vblock_size, tpltzblocks[id].lambda, &nfft, &blocksize, &T_fft, (tpltzblocks[id].T_block), &V_fft, &V_rfft, &plan_f, &plan_b, flag_stgy);

  //Toeplitz computation
  if(PRINT_RANK==0 && VERBOSE>2)
    fprintf(file, "[%d] Before stmm_main call : nfft = %d, blocksize = %d\n", rank, nfft, blocksize);
```

```
  stmm_main(&V1block, vblock_size, m_rowwise, 0, vblock_size*m_rowwise, (tpltzblocks[id].T_block), T_fft, tpltzblocks[id].lambda, V_fft, V_rfft, plan_f, plan_b, blocksize, nfft, flag_stgy);

  tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);

  for (j=0;j<m_rowwise;j++) {
#pragma omp parallel for //num_threads(NB_OMPTHREADS_STBMM)
  for (i=0;i<vnrank_size-offsetn;i++) {
    (*V)[idv2+i+j*n_rowwise] = V1block[i+j*vblock_size];
  }}


  free(V1block);

  }//end of last block
  else { break; }//error  //we can put the generic case here instead of between first and last cases
#endif
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
  }//end of if(nnew[id]>0)
  }//end of loop over the blocks


  free(LambdaIn);


  return 0;
}

//#endif

//====================================================================


int get_overlapping_blocks_params(int nbloc, Block *tpltzblocks, int local_V_size, int64_t nrow, int64_t idp, int64_t *idpnew, int *local_V_size_new, int *nnew, int *ifirstBlock, int *ilastBlock)
{ int ib, nblockOK=0, nfullcol_data;
  int64_t firstrow, lastrow;
  int64_t idptmp;


//check how many full columns input data have
  nfullcol_data = max(0, (local_V_size-(nrow-idp%nrow)%nrow-(idp+local_V_size)%nrow)/nrow );

  if( nfullcol_data > 0) {

  for( ib=0; ib<nbloc; ib++) {
    if( tpltzblocks[ib].idv < nrow) {
      nnew[ib] = min( tpltzblocks[ib].n, nrow-tpltzblocks[ib].idv);  //block used for the product
      nblockOK++;
    }
  }

  }
  else {  //no full column observed

    firstrow = idp%nrow;
    lastrow = (idp+local_V_size-1)%nrow;

    if( firstrow < lastrow) {  //just one column partially observed

    for( ib=0; ib<nbloc; ib++) {
    if( (tpltzblocks[ib].idv+tpltzblocks[ib].n > firstrow) && (tpltzblocks[ib].idv < lastrow+1)) {
      nnew[ib] = min( tpltzblocks[ib].n, nrow-tpltzblocks[ib].idv);  //block used for the product
      nblockOK++;
    }
    }

    }
    else {  //two columns partially observed

      for( ib=0; ib<nbloc; ib++) {
        if( (tpltzblocks[ib].idv + tpltzblocks[ib].n > firstrow) && (tpltzblocks[ib].idv < nrow)) {  //intersects first partial column
          nnew[ib] = min( tpltzblocks[ib].n, nrow-tpltzblocks[ib].idv);  //block used for the product
          nblockOK++;
        }

        if( (tpltzblocks[ib].idv < lastrow+1) && (tpltzblocks[ib].idv+
```

```
      tpltzblocks[ib].n > 0)) {  //intersects second partial column
00572           nnew[ib] = min( tpltzblocks[ib].n, nrow-tpltzblocks[ib].idv);  //
      block used for the product
00573           nblockOK++;  //may overcount but we do not care
00574         }  //could use else insteed!
00575       }
00576      }
00577   }
00578   if(PRINT_RANK==0 && VERBOSE>2)
00579     printf("nblockOK=%d\n", nblockOK);
00580
00581
00582   if( nblockOK == 0) return(0);  //no blocks overlapping with the data
00583
00584   //find the first and last relevant blocks for the begining and end of the
       local data  V
00585
00586  //first block
00587   idptmp = idp;
00588
00589   for( *ifirstBlock = -1; *ifirstBlock == -1;     ) {
00590     for(ib=0;ib<nbloc;ib++) {
00591       if(nnew[ib] != 0 && idptmp%nrow < tpltzblocks[ib].idv+nnew[ib]) break;
00592     }
00593
00594     if (ib<nbloc && tpltzblocks[ib].idv <= idptmp%nrow) {
00595       *ifirstBlock = ib;
00596       *idpnew = idptmp;
00597     }
00598     else if (ib<nbloc && tpltzblocks[ib].idv > idptmp%nrow) {
00599       *ifirstBlock = ib;
00600   //   int64_t extrabegining = tpltzblocks[ib].idv-idp%nrow;  //note I put
       int64 just to be sure. Never used
00601 //      *idpnew = idp+extrabegining;//tpltzblocks[ib].idv;
00602       int idvfirstcolumn = idptmp/nrow;
00603       *idpnew = tpltzblocks[ib].idv+idvfirstcolumn*nrow;
00604     }
00605     else { //ib=nb_blocs
00606       idptmp += nrow-idptmp%nrow; //(int) (nrow-idptmp%nrow);
00607 //          idtmp = (int) ceil((1.0*idpnew)/(1.0*nrow))*nrow; // go to the
       first element of the next column
00608     }}
00609
00610
00611  //last block
00612   idptmp = idp+local_V_size-1;
00613
00614   for( *ilastBlock = -1; *ilastBlock == -1; ) {
00615     for(ib=nbloc-1;ib>=0;ib--) {
00616       if(nnew[ib] != 0 && tpltzblocks[ib].idv <= idptmp%nrow) break;
00617     }
00618
00619
00620     if (ib>=0 && idptmp%nrow < tpltzblocks[ib].idv+nnew[ib]) {
00621       *ilastBlock = ib;
00622       *local_V_size_new = local_V_size-(*idpnew)+idp;
00623     }
00624     else if (ib>=0 && tpltzblocks[ib].idv+nnew[ib] <= idptmp%nrow) {
00625       *ilastBlock = ib;
00626       //int64_t extraend =
       (local_V_size-1+idp)%nrow+1-(tpltzblocks[ib].idv+nnew[ib]);  //note I put int64 just to be sure
00627       //*local_V_size_new =
       (local_V_size+idp)%nrow-(idv[*ilastBlock]+nnew[*ilastBlock]);
00628      //idv[*ilastBlock]+nnew[*ilastBlock]-(*idpnew);
00629       //*local_V_size_new = local_V_size-(*idpnew)+idp-extraend;  //compute
       twice ... ? remove this one
00630
00631       int idvlastcolumn = idptmp/nrow;
00632       *local_V_size_new = tpltzblocks[ib].idv+nnew[ib]+idvlastcolumn*nrow -
      (*idpnew);
00633
00634     }
00635     else {
00636       idptmp = idptmp - (idptmp%nrow)-1;//(int) idptmp - (idptmp%nrow)-1;
00637 //        idtmp = (int) floor( (1.0*idpnew)/(1.0*nrow))*nrow-1; // go to the
       last element of the previous column
00638     }}
00639
00640
00641     return(1);
00642 }
```

15.51 toeplitz_devtools.c File Reference

Contains developpement tools routines for Toeplitz algebra.

Functions

- int stmm_cblas (int n_loc, int m_loc, double *T2_loc, double *V, double *TV2)
- int build_full_Toeplitz (int n_loc, double *T_loc, int lambda_loc, double *T2_loc)
- int print_full_Toeplitz (int n_loc, double *T2_loc)
- int print_full_matrix (int n_loc, int m_loc, double *Mat)

15.51.1 Detailed Description

Contains developpement tools routines for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz*.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Definition in file toeplitz_devtools.c.

15.51.2 Function Documentation

15.51.2.1 int stmm_cblas (int *n_loc*, int *m_loc*, double * *T2_loc*, double * *V*, double * *TV2*)

Definition at line 53 of file toeplitz_devtools.c.

15.51.2.2 int build_full_Toeplitz (int *n_loc*, double * *T_loc*, int *lambda_loc*, double * *T2_loc*)

Definition at line 63 of file toeplitz_devtools.c.

15.51.2.3 int print_full_Toeplitz (int *n_loc*, double * *T2_loc*)

Definition at line 85 of file toeplitz_devtools.c.

15.51.2.4 int print_full_matrix (int *n_loc*, int *m_loc*, double * *Mat*)

Definition at line 105 of file toeplitz_devtools.c.

15.52 toeplitz_devtools.c

```
00001
00043 #include <stdlib.h>
00044 #include <stdio.h>
00045 #include <math.h>
00046 #include "toeplitz.h"
00047 #include <cblas.h>
00048 #include <time.h>
00049
00050 //
00051 //dev tools for cblas and print - fd@apc
00052
00053 int stmm_cblas(int n_loc, int m_loc, double *T2_loc, double *V,
      double *TV2) {
00054
00055   cblas_dgemm (CblasColMajor, CblasNoTrans,CblasNoTrans, n_loc, m_loc, n_loc, 1
      , T2_loc, n_loc, (V), n_loc, 1, TV2, n_loc);
00056
00057   return 0;
00058 }
00059
00060
00061
00062 // Build full Toeplitz matrix needed for cblas computation
00063 int build_full_Toeplitz(int n_loc, double *T_loc, int
      lambda_loc, double *T2_loc) {
00064
00065   int i,j;
00066
00067       for (j=0;j<n_loc;j++) {  //reset all the matrix to zeros
00068         for(i=0;i<n_loc;i++) {
00069           T2_loc[j*n_loc+i] = 0;
00070       }}
00071
00072       for (j=0;j<n_loc;j++){ // Full Toeplitz matrix needed for cblas
       computation
00073         for (i=0;i<lambda_loc;i++){
00074           if (j-i>=0)
00075             T2_loc[j*n_loc+j-i] = T_loc[i];
00076           if (j+i<n_loc)
00077             T2_loc[j*n_loc+j+i] = T_loc[i]; }}
00078
00079
00080   return 0;
00081 }
00082
00083
00084
00085 int print_full_Toeplitz(int n_loc, double *T2_loc) {
00086
00087   int i,j;
00088
```

```
00089   FILE *file;
00090   file = stdout;
00091
00092     for(i=0;i<n_loc;i++) {
00093     for(j=0;j<n_loc;j++) {
00094       fprintf(file, "%.1f\t", T2_loc[i+j*n_loc]);
00095     }
00096       fprintf(file, "\n");
00097     }
00098
00099
00100   return 0;
00101 }
00102
00103
00104
00105 int print_full_matrix(int n_loc, int m_loc, double *Mat) {
00106
00107   int i,j;
00108
00109   FILE *file;
00110   file = stdout;
00111
00112     for(i=0;i<n_loc;i++) {
00113     for(j=0;j<m_loc;j++) {
00114       fprintf(file, "%.1f\t", Mat[i+j*n_loc]);
00115     }
00116       fprintf(file, "\n");
00117     }
00118
00119
00120   return 0;
00121 }
00122
00123
```

15.53 toeplitz_gappy.c File Reference

Functions

- int mpi_gstbmm (double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy, MPI_Comm comm)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way. This matrix V contains defined gaps which represents the useless data for the comutation. The gaps indexes are defined in the global time space as the generized toeplitz matrix, meaning the row dimension. Each of his diagonal blocks is a symmetric, band-diagonal Toeplitz matrix, which can be different for each block.

- int reset_gaps (double **V, int id0, int local_V_size, int m, int nrow, int m_rowwise, int64_t *id0gap, int *lgap, int ngap)

  Set the data to zeros at the gaps location.

- int build_gappy_blocks (int nrow, int m, Block *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t *id0gap, int *lgap, int ngap, Block *tpltzblocks_gappy, int *nb_blocks_gappy_final, int flag_param_distmin_fixed)

  Build the gappy Toeplitz block structure to optimise the product computation at gaps location.

15.54 toeplitz_gappy.c

```
00001 /*
00002 @file toeplitz_gappy.c version 1.1b, July 2012
00003 @brief Gappy routines used to compute the Toeplitz product when gaps are
       defined
00004 @author  Frederic Dauvergne
00005 **
00006 ** Project:  Midapack library, ANR MIDAS'09 - Toeplitz Algebra module
00007 ** Purpose:  Provide Toeplitz algebra tools suitable for Cosmic Microwave
       Background (CMB)
00008 **           data analysis.
00009 **
00010 ******************************************************************************
00011 @note Copyright (c) 2010-2012 APC CNRS Université Paris Diderot
```

```
00012 @note
00013 @note This program is free software; you can redistribute it and/or modify it
       under the terms
00014 @note of the GNU Lesser General Public License as published by the Free
       Software Foundation;
00015 @note either version 3 of the License, or (at your option) any later version.
       This program is
00016 @note distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
       without even
00017 @note the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE. See the GNU
00018 @note Lesser General Public License for more details.
00019 @note
00020 @note You should have received a copy of the GNU Lesser General Public License
       along with this
00021 @note program; if not, see http://www.gnu.org/licenses/lgpl.html
00022 @note
00023 @note For more information about ANR MIDAS'09 project see :
00024 @note http://www.apc.univ-paris7.fr/APC_CS/Recherche/Adamis/MIDAS09/index.html
00025 @note
00026 @note ACKNOWLEDGMENT: This work has been supported in part by the French
       National Research
00027 @note Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).
00028 ********************************************************************************
00029 ** Log: toeplitz*.c
00030 **
00031 ** Revision 1.0b  2012/05/07  Frederic Dauvergne (APC)
00032 ** Official release 1.0beta. The first installement of the library is the
       Toeplitz algebra
00033 ** module.
00034 **
00035 ** Revision 1.1b  2012/07/-  Frederic Dauvergne (APC)
00036 ** - mpi_stbmm allows now rowi-wise order per process datas and no-blocking
       communications.
00037 ** - OMP improvment for optimal cpu time.
00038 ** - bug fixed for OMP in the stmm_basic routine.
00039 ** - distcorrmin is used to communicate only lambda-1 datas when it is needed.
00040 ** - new reshaping routines using transformation functions in stmm. Thus, only
       one copy
00041 **   at most is needed.
00042 ** - tpltz_init improvement using define_nfft and define_blocksize routines.
00043 ** - add Block struture to define each Toeplitz block.
00044 ** - add Flag structure and preprocessing parameters to define the
       computational strategy.
00045 **
00046 **
00047 ********************************************************************************
00048 **
00049 */
00050
00051 #include "toeplitz.h"
00052
00053 //r1.1 - Frederic Dauvergne (APC)
00054 //this is the gappy routines used when gaps are defined
00055
00056
00057 //=====================================================================
00058 #ifdef W_MPI
00059
00060
00088 int mpi_gstbmm(double **V, int nrow, int m, int m_rowwise, Block
       *tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int
      local_V_size, int64_t *id0gap, int *lgap, int ngap,Flag flag_stgy, MPI_Comm comm)
00089 {
00090
00091   //MPI parameters
00092   int rank;  //process rank
00093   int size;  //process number
00094
00095   MPI_Status status;
00096   MPI_Comm_rank(comm, &rank);
00097   MPI_Comm_size(comm, &size);
00098
00099   int i,j,k;   //some indexes
00100
00101   int flag_skip_build_gappy_blocks = flag_stgy.flag_skip_build_gappy_blocks
      ;
00102
00103   FILE *file;
00104   file = stdout;
00105   PRINT_RANK=rank ;
00106
00107 //put zeros at the gaps location
00108   reset_gaps( V, id0p, local_V_size, m, nrow, m_rowwise, id0gap, lgap
      , ngap);
00109
00110
```

```
00111 //allocation for the gappy structure of the diagonal block Toeplitz matrix
00112   int nb_blocks_gappy;
00113
00114   int nb_blockgappy_max;
00115   int Tgappysize_max;
00116
00117   Block *tpltzblocks_gappy;
00118
00119 //some computation usefull to determine the max size possible for the gappy
       variables
00120   int Tsize=0;
00121   int lambdamax=0;
00122
00123 if (VERBOSE)
00124   fprintf(file, "[%d] flag_skip_build_gappy_blocks=%d\n", rank,
      flag_skip_build_gappy_blocks);
00125
00126   if (flag_skip_build_gappy_blocks==1) {  //no build gappy blocks strategy,
       just put zeros at gaps location
00127
00128   //compute the product using only the input Toeplitz blocks structure with
       zeros at the gaps location
00129   mpi_stbmm(V, nrow, m, m_rowwise, tpltzblocks, nb_blocks_local,
      nb_blocks_all, id0p, local_V_size, flag_stgy, MPI_COMM_WORLD);
00130
00131   }
00132   else { //build gappy blocks strategy
00133
00134   for(Tsize=i=0;i<nb_blocks_local;i++)
00135     Tsize += tpltzblocks[i].lambda;
00136
00137   for(i=0;i<nb_blocks_local;i++) {
00138     if (tpltzblocks[i].lambda>lambdamax)
00139       lambdamax = tpltzblocks[i].lambda;
00140   }
00141
00142 //compute max size possible for the gappy variables
00143   nb_blockgappy_max = nb_blocks_local+ngap;
00144   Tgappysize_max = Tsize + lambdamax*ngap;
00145
00146 //allocation of the gappy variables with max size possible
00147   tpltzblocks_gappy = (Block *) calloc(nb_blockgappy_max,sizeof(Block
      ));
00148
00149
00150 //build gappy Toeplitz block structure considering significant gaps locations,
       meaning we skip
00151 //the gaps lower than the minimum correlation distance. You can also use the
       flag_param_distmin_fixed
00152 //parameter which allows you to skip the gap lower than these value. Indeed,
       sometimes it's
00153 //better to just put somes zeros than to consider two separates blocks.
00154 //ps: This criteria could be dependant of the local lambda in futur
       impovements.
00155   int flag_param_distmin_fixed = flag_stgy.flag_param_distmin_fixed
      ;
00156   build_gappy_blocks(nrow, m, tpltzblocks, nb_blocks_local,
      nb_blocks_all, id0gap, lgap, ngap, tpltzblocks_gappy, &nb_blocks_gappy,
      flag_param_distmin_fixed);
00157
00158
00159 if (VERBOSE) {
00160     fprintf(file, "[%d] nb_blocks_gappy=%d\n", rank, nb_blocks_gappy);
00161     for(i=0;i<nb_blocks_gappy;i++)
00162       fprintf(file, "[%d] idvgappy[%d]=%d ; ngappy[%d]=%d\n", rank, i,
      tpltzblocks_gappy[i].idv, i, tpltzblocks_gappy[i].n );
00163 }
00164 //ps: we could reallocate the gappy variables to their real size. Not sure it's
       worth it.
00165
00166 //compute the product using the freshly created gappy Toeplitz blocks structure
00167   mpi_stbmm(V, nrow, m, m_rowwise, tpltzblocks_gappy, nb_blocks_local,
       nb_blocks_all, id0p, local_V_size, flag_stgy, MPI_COMM_WORLD);
00168
00169   } //end flag_skip_build_gappy_blocks==1
00170
00171
00172 //put zeros on V for the gaps location again. Indeed, some gaps are just
       replaced by zeros
00173 //in input, it's created some fakes results we need to clear after the
       computation.
00174   reset_gaps( V, id0p, local_V_size, m, nrow, m_rowwise, id0gap, lgap
      , ngap);
00175
00176
00177   return 0;
00178 }
```

```
00179
00180
00181 //========================================================================
00183
00188 //put zeros on V for the gaps location
00189 int reset_gaps(double **V, int id0, int local_V_size, int m, int nrow
      , int m_rowwise, int64_t *id0gap, int *lgap, int ngap)
00190 {
00191   int i,j,k,l;
00192
00193   for (j=0 ; j<m; j++) {
00194
00195 #pragma omp parallel for private(i) schedule(dynamic,1)
00196   for (k=0 ; k<ngap; k++)
00197    for (i=0 ; i<lgap[k]; i++)
00198    if (id0gap[k]+i+j*nrow>=id0 && id0gap[k]+i+j*nrow <id0+local_V_size) {
00199      for (l=0 ; l<m_rowwise; l++)
00200        (*V)[id0gap[k]+i+j*nrow-id0+l*local_V_size] = 0.;
00201    }
00202   }
00203
00204   return 0;
00205 }
00206 #endif
00207
00208 //========================================================================
00210
00231 int build_gappy_blocks(int nrow, int m, Block *
      tpltzblocks, int nb_blocks_local, int nb_blocks_all, int64_t *id0gap, int *lgap, int ngap
      , Block *tpltzblocks_gappy, int *nb_blocks_gappy_final, int
      flag_param_distmin_fixed)
00232 {
00233
00234   int i,j,k;
00235   int id,ib;
00236   int idtmp;
00237   int igapfirstblock, igaplastblock;
00238
00239   int param_distmin=0;
00240   if (flag_param_distmin_fixed!=0)
00241     param_distmin = flag_param_distmin_fixed;
00242
00243   int lambdaShft;
00244
00245   int igaplastblock_prev=-1;
00246   int lambdaShftgappy=0;
00247   int offset_id = 0;
00248  // int offset_id_gappy=0;
00249
00250   int flag_igapfirstinside, flag_igaplastinside;
00251   int nbloc = nb_blocks_local;
00252   int nblocks_gappy=0;
00253
00254   int idvtmp_firstblock;
00255
00256
00257   int nb_blockgappy_max = nb_blocks_local+ngap;
00258   int Tgappysize_max;
00259
00260   int ngappy_tmp;
00261   int lgap_tmp;
00262
00263   int flag_gapok=0;
00264
00265   int distcorr_min;
00266
00267   int Tgappysize=0;
00268   int k_prev=-1;
00269
00270   for (k=0;k<ngap;k++) {
00271
00272   //find block for the gap begining
00273   for( igapfirstblock = -1; igapfirstblock == -1; ) {
00274     idtmp = id0gap[k];
00275
00276     for(ib=0;ib<nbloc;ib++) {
00277       if(tpltzblocks[ib].n != 0 && idtmp%nrow < tpltzblocks[ib].idv+tpltzblocks
      [ib].n) break;  }
00278
00279     if (ib<nbloc && tpltzblocks[ib].idv <= idtmp) {
00280       igapfirstblock = ib;  //the block contained the id0gap
00281       flag_igapfirstinside = 1;
00282     }
00283     else if (ib<nbloc && tpltzblocks[ib].idv > idtmp) {
00284       igapfirstblock = ib;  //first block after the id0gap
00285       flag_igapfirstinside = 0;
00286     }
```

```
00287     else { //ib=nbloc
00288       igapfirstblock = -2;  //no block defined
00289       flag_igapfirstinside = 0;
00290     }}
00291
00292   //find block for the end of the gap - reverse way
00293   for( igaplastblock = -1; igaplastblock == -1; ) {
00294     idtmp = id0gap[k]+lgap[k]-1;
00295
00296     for(ib=nbloc-1;ib>=0;ib--) {
00297       if(tpltzblocks[ib].n != 0 && tpltzblocks[ib].idv <= idtmp) break;     }
00298
00299     if (ib>=0 && idtmp < tpltzblocks[ib].idv+tpltzblocks[ib].n) {
00300       igaplastblock = ib;
00301       flag_igaplastinside = 1;
00302     }
00303     else if (ib>=0 && tpltzblocks[ib].idv+tpltzblocks[ib].n <= idtmp) {
00304       igaplastblock = ib;
00305       flag_igaplastinside = 0;
00306     }
00307     else { //ib=-1
00308       igaplastblock = -2;  //no block defined.
00309       flag_igaplastinside = 0;
00310     }}
00311
00312
00313     if (igapfirstblock==igaplastblock)
00314       distcorr_min = tpltzblocks[igapfirstblock].lambda-1; //update for
       lambda-1
00315     else
00316       distcorr_min = 0;
00317
00318
00319 //igapfirstblock != -2 && igaplastblock != -2 not really need but it's a
       shortcut
00320   if (lgap[k]> max(distcorr_min, param_distmin) && igapfirstblock != -2 &&
      igaplastblock != -2) {
00321
00322   idvtmp_firstblock = max( tpltzblocks[igapfirstblock].idv, id0gap[k_prev]+lgap
      [k_prev]);
00323
00324 //test if the gap is ok for block reduce/split
00325   if (igapfirstblock!=igaplastblock) {
00326
00327     flag_gapok = 1;  //reduce the gap in each block. no pb if we add max()
       inside the ifs.
00328   }
00329   else if (id0gap[k]-idvtmp_firstblock>=tpltzblocks[igapfirstblock].lambda &&
      tpltzblocks[igaplastblock].idv + tpltzblocks[igaplastblock].n - (id0gap[k]+lgap[k
      ])>=tpltzblocks[igaplastblock].lambda) {
00330
00331     flag_gapok = 1;
00332   }
00333   else if (igapfirstblock==igaplastblock){
00334
00335   int ngappyleft_tmp = id0gap[k]-idvtmp_firstblock;
00336   int leftadd = max(0, tpltzblocks[igapfirstblock].lambda - ngappyleft_tmp);
00337   int ngappyright_tmp = tpltzblocks[igaplastblock].idv + tpltzblocks[
      igaplastblock].n - (id0gap[k]+lgap[k]);
00338   int rightadd = max(0,tpltzblocks[igapfirstblock].lambda - ngappyright_tmp);
00339   int restgap = lgap[k] - (leftadd+rightadd);
00340
00341 //  flag_gapok = (restgap>=0);
00342   flag_gapok = (restgap >= max(0, param_distmin));
00343
00344   }
00345   else {
00346   flag_gapok = 0;
00347   }
00348
00349
00350  //create gappy blocks if criteria is fullfill
00351   if (flag_gapok==1) {
00352
00353   //copy the begining blocks
00354     for (id=igaplastblock_prev+1;id<igapfirstblock;id++) {
00355
00356       tpltzblocks_gappy[nblocks_gappy].T_block = tpltzblocks[id].T_block
      ;
00357       tpltzblocks_gappy[nblocks_gappy].lambda = tpltzblocks[id].lambda
      ;
00358       tpltzblocks_gappy[nblocks_gappy].n = tpltzblocks[id].n;
00359       tpltzblocks_gappy[nblocks_gappy].idv = tpltzblocks[id].idv;
00360
00361       nblocks_gappy = nblocks_gappy + 1;
00362
00363     }
```

```
00364
00365   //clear last blockgappy if same block again - outside the "if" for border
        cases with n[]==0
00366     if (igaplastblock_prev==igapfirstblock && k!= 0) {
00367       nblocks_gappy = nblocks_gappy - 1;
00368 //      idvtmp_firstblock = id0gap[k-1]+lgap[k-1]; //always exist because
        igaplastblock_prev!=-1
00369                                     //so not first turn - it's replace
        "idv[igapfirstblock]"
00370     }
00371
00372   //reduce first block if defined
00373     if (flag_igapfirstinside==1 && (id0gap[k]-idvtmp_firstblock)>0) {  //check
        if inside and not on the border - meaning n[] not zero
00374
00375       tpltzblocks_gappy[nblocks_gappy].T_block = tpltzblocks[
      igapfirstblock].T_block;
00376       tpltzblocks_gappy[nblocks_gappy].lambda = tpltzblocks[id].lambda
      ;
00377       tpltzblocks_gappy[nblocks_gappy].n = id0gap[k]-idvtmp_firstblock;
00378       tpltzblocks_gappy[nblocks_gappy].n = max( tpltzblocks_gappy[
      nblocks_gappy].n, tpltzblocks[igapfirstblock].lambda);
00379
00380       tpltzblocks_gappy[nblocks_gappy].idv = idvtmp_firstblock;
00381       nblocks_gappy = nblocks_gappy + 1;
00382
00383     }
00384
00385   //reduce last block if defined
00386     if (flag_igaplastinside==1  && (tpltzblocks[igaplastblock].idv+tpltzblocks[
      igaplastblock].n -(id0gap[k]+lgap[k]))>0 ) {  //check if inside and not on the
        border - meaning n[] not zero
00387
00388       tpltzblocks_gappy[nblocks_gappy].T_block = tpltzblocks[
      igaplastblock].T_block;
00389       tpltzblocks_gappy[nblocks_gappy].lambda = tpltzblocks[id].lambda
      ;
00390       tpltzblocks_gappy[nblocks_gappy].n = tpltzblocks[igaplastblock].idv+
      tpltzblocks[igaplastblock].n-(id0gap[k]+lgap[k]);
00391       int rightadd0 = max(0, tpltzblocks[igapfirstblock].lambda -
      tpltzblocks_gappy[nblocks_gappy].n);
00392
00393       tpltzblocks_gappy[nblocks_gappy].n = max( tpltzblocks_gappy[
      nblocks_gappy].n , tpltzblocks[igaplastblock].lambda);
00394
00395       tpltzblocks_gappy[nblocks_gappy].idv = id0gap[k]+lgap[k]-rightadd0;
00396
00397       nblocks_gappy = nblocks_gappy + 1;
00398       lambdaShftgappy = lambdaShftgappy + tpltzblocks[igaplastblock].lambda
      ;
00399
00400     }
00401
00402   igaplastblock_prev = igaplastblock;
00403   k_prev = k;
00404
00405   }//end if (flag_gapok)
00406   }//end if (lgap[k]>param_distmin)
00407   }//end gap loop
00408
00409
00410  //now continu to copy the rest of the block left
00411   for (id=igaplastblock_prev+1;id<nb_blocks_local;id++) {
00412
00413     tpltzblocks_gappy[nblocks_gappy].T_block = tpltzblocks[id].T_block
      ;
00414     tpltzblocks_gappy[nblocks_gappy].lambda = tpltzblocks[id].lambda
      ;
00415     tpltzblocks_gappy[nblocks_gappy].n = tpltzblocks[id].n;
00416     tpltzblocks_gappy[nblocks_gappy].idv = tpltzblocks[id].idv;
00417     nblocks_gappy = nblocks_gappy + 1;
00418     lambdaShftgappy = lambdaShftgappy + tpltzblocks[id].lambda;
00419
00420   }
00421
00422   *nb_blocks_gappy_final = nblocks_gappy;  //just for output
00423
00424
00425   return 0;
00426 }
00427
00428
```

15.55 toeplitz_gappy_seq.dev.c File Reference

Functions

- int gstmm (double **V, int n, int m, int id0, int l, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, int *id0gap, int *lgap, int ngap)
- int gap_reduce (double **V, int id0, int l, int lambda, int *id0gap, int *lgap, int ngap, int *newl, int id0out)

  ...convert the data vector structure into a matrix structure optimized for nfft
- int gap_masking (double **V, int l, int *id0gap, int *lgap, int ngap)

  Reduce the vector and mask the defined gaps.
- int gap_filling (double **V, int l, int *id0gap, int *lgap, int ngap)

  Extend the vector and add zeros on the gaps locations.
- int gap_masking_naive (double **V, int id0, int local_V_size, int m, int nrow, int *id0gap, int *lgap, int ngap)

15.55.1 Function Documentation

15.55.1.1 int gstmm (double ** *V*, int *n*, int *m*, int *id0*, int *l*, fftw_complex * *T_fft*, int *lambda*, fftw_complex * *V_fft*, double * *V_rfft*, fftw_plan *plan_f*, fftw_plan *plan_b*, int *blocksize*, int *nfft*, int * *id0gap*, int * *lgap*, int *ngap*)

Definition at line 7 of file toeplitz_gappy_seq.dev.c.

15.55.1.2 int gap_masking_naive (double ** *V*, int *id0*, int *local_V_size*, int *m*, int *nrow*, int * *id0gap*, int * *lgap*, int *ngap*)

Definition at line 330 of file toeplitz_gappy_seq.dev.c.

15.56 toeplitz_gappy_seq.dev.c

```
00001
00002
00003 //============================================================================
00004 //Alternave version of the sequential routine for the gaps - in dev
00005 //Need to change the name to gap_reduce_matrix for more explicit purpose
00006 //fd@apc
00007 int gstmm(double **V, int n, int m, int id0, int l, fftw_complex *T_fft,
      int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan
      plan_b, int blocksize, int nfft, int *id0gap, int *lgap, int ngap)
00008 {
00009
00010   //routine variable
00011   int i,j,k,p;  //loop index
00012   int idf       = id0+l-1;
00013   int cfirst    = id0/n;  //first column index
00014   int clast     = idf/n;  //last column index
00015   int clast_1   = (idf+1)/n;
00016   int m_eff     = clast - cfirst + 1 ;  //number of columns
00017   int rfirst    = id0%n;
00018   int rlast     = idf%n;
00019
00020   if (l<lambda) // test to avoid communications errors
00021     return print_error_message (1, __FILE__, __LINE__);
00022
00023   int nnew=0;
00024   int lcolnew=0;
00025   int lcol;
00026   double *Vcol;
00027   int icol, igap;
00028 ;
00029   int id0col;
00030   int id0out=0;
00031   int lnew;
00032
00033   int nfullcol;
00034   nfullcol = (l-(n-id0%n)-(id0+l)%n)/n;  //check how many full columns input
      data have
00035
00036   int ifirstgap, ilastgap;
00037
```

```
00038 if (id0%n != 0) {
00039 //first column if not full
00040   id0out=0;
00041   id0col=id0%n;
00042   Vcol = (*V)+id0col;
00043   lcol= n-id0col;
00044
00045   //find the first and last gap on the column
00046   for(k=0;k<ngap;k++) {
00047     if(lgap[k] != 0 && id0col < id0gap[k]+lgap[k]) break;
00048   }
00049   ifirstgap = k;
00050
00051   for(k=ngap-1;k>=0;k--) {
00052     if(lgap[k] != 0 && id0gap[k] <= id0col) break;
00053   }
00054   ilastgap = k;
00055
00056
00057   for (i=0 ; i<ngap; i++)
00058     printf("id0gap[%d]=%d\n", i, id0gap[i]);
00059   for (i=0 ; i<ngap; i++)
00060     printf("lgap[%d]=%d\n", i, lgap[i]);
00061
00062     printf("---\n");
00063
00064     printf("lcol=%d\n", lcol);
00065   for (i=0 ; i<lcol; i++)
00066     printf("Vcol[%d]=%f\n", i, Vcol[i]);
00067
00068 //reduce the gap to lambda zeros on the column
00069   gap_reduce(&Vcol, id0col, lcol, lambda, id0gap, lgap, ngap, &
      lcolnew, id0out);
00070
00071   id0out += lcolnew;
00072
00073     printf("---\n");
00074
00075   for (i=0 ; i<lcol; i++)
00076     printf("Vcolout[%d]=%f\n", i, Vcol[i]);
00077     printf("===\n");
00078
00079     printf("---\n");
00080
00081   for (i=0 ; i<l; i++)
00082     printf("(*V)[%d]=%f\n", i, (*V)[i]);
00083
00084 }
00085
00086 //generic loop on the full column
00087   for (icol=0 ; icol<nfullcol; icol++) {
00088
00089     id0out= lcolnew;
00090     id0col=0;
00091     Vcol = (*V)+id0col;
00092     lcol= n-id0col;
00093
00094     gap_reduce(&Vcol, id0col, lcol, lambda, id0gap, lgap, ngap, &
      lcolnew, id0out);
00095
00096     id0out += lcolnew;
00097
00098     if (icol==0)  //take the nnew directly if they is a full column
00099       nnew=lcolnew;
00100
00101   }
00102
00103
00104 //last column if not full and more than one column
00105   if (idf%n != n && m > 1) {
00106
00107   id0out=0;
00108   id0col=0;
00109   Vcol = (*V)+id0col;
00110   lcol= idf%n;
00111
00112   //find the first and last gap on the column
00113   ifirstgap = 0;  //we already know it, no need to compute
00114
00115   for(k=ngap-1;k>=0;k--) {
00116     if(lgap[k] != 0 && id0gap[k] <= id0col) break;
00117   }
00118   ilastgap = k;
00119
00120 //reduce the gap to lambda zeros on the column
00121   gap_reduce(&Vcol, id0col, lcol, lambda, id0gap, lgap, ngap, &
      lcolnew, id0out);
```

```
00122
00123   }
00124
00125
00126 //cleaning the extra terms
00127   if (nnew==0) {  //compute the nnew if there is no full column
00128     for (igap=0 ; igap<ngap; igap++)
00129       nnew += id0gap[igap] - min(id0gap[igap], lambda);
00130   }
00131
00132   for (i=nnew*m ; i<n*m; i++)
00133     (*V)[i] = 0;
00134
00135     printf("===\n");
00136   for (i=0 ; i<l; i++)
00137     printf("(*V)[%d]=%f\n", i, (*V)[i]);
00138
00139
00140 //now do the product
00141 //  int id0new = ..;
00142
00143 //  stmm( V, nnew, m, id0new, lnew, T_fft, lambda, V_fft, V_rfft, plan_f,
       plan_b, blocksize, nfft)
00144
00145
00146 //gap extend - same thing as above and reverse gap_reduce routine
00147 // gap_extend()
00148
00149 //maybe create gap_blockreduce to clarify this routine.
00150
00151   return 0;
00152 }
00153
00154 //========================================================================
00155
00157
00163 int gap_reduce(double **V, int id0, int l, int lambda, int *id0gap,
      int *lgap, int ngap, int *newl, int id0out)
00164 {
00165   //routine variables
00166   int i,k, ii;
00167   int lg;
00168   int newid0gap_ip1;
00169   int newlgap_ip1;
00170
00171
00172   double *Vout;
00173   Vout =(*V)-(id0-id0out);
00174
00175
00176   if (id0gap[0]>id0) {
00177     lg = id0gap[0]-id0;
00178     printf("lg=%d\n", lg);
00179     memmove(&(Vout)[0], &(*V)[0], lg*sizeof(double));
00180   //to do if the first element isn't a gap
00181   }
00182
00183   int offset_first=id0gap[0]-id0+min(lambda,lgap[0]);
00184   offset_first=max(0, offset_first);
00185
00186
00187 //  for (k=0; k<offset_first; k++)
00188 //    (Vout)[k] = 0;
00189
00190   printf("offset_first=%d\n", offset_first);
00191
00192   for (i=0 ; i<l; i++)
00193     printf("(Vout)[%d]=%f\n", i, (Vout)[i]);
00194     printf("---\n");
00195
00196     printf("l=%d\n", l);
00197
00198
00199   for (i=0 ; i<(ngap-1); i++) {
00200     lg = id0gap[i+1]-id0-(id0gap[i]-id0+lgap[i]);
00201     printf("lg=%d\n", lg);
00202     memmove(&(Vout)[offset_first], &(*V)[id0gap[i]-id0+lgap[i]], lg*sizeof(
      double));
00203
00204   for (ii=0 ; ii<l; ii++)
00205     printf("(Vout)[%d]=%f\n", ii, (Vout)[ii]);
00206
00207     newid0gap_ip1=offset_first+lg;
00208     newlgap_ip1=min(lambda,lgap[i+1]);
00209     for (k=newid0gap_ip1; k<newid0gap_ip1+newlgap_ip1; k++)
00210       (Vout)[k] = 0;
00211
```

```
00212     printf("lg=%d ; lambda=%d ; lgap[i+1]=%d\n", lg, lambda, lgap[i+1]);
00213     offset_first += lg+min(lambda,lgap[i+1]);
00214   }  //end gaps loop
00215
00216     printf("---\n");
00217   printf("offset_first=%d\n", offset_first);
00218   for (i=0 ; i<l; i++)
00219     printf("(Vout)[%d]=%f\n", i, (Vout)[i]);
00220
00221   i=(ngap-1);
00222
00223   if (id0gap[i]-id0+lgap[i]<l) {
00224     printf("toc\n");
00225     lg = l-(id0gap[i]-id0+lgap[i]);
00226     memmove(&(Vout)[offset_first], &(*V)[id0gap[i]-id0+lgap[i]], lg*sizeof(
      double));
00227     offset_first += lg;
00228     *newl = offset_first;
00229   }
00230   else {
00231     *newl = offset_first-min(lambda,lgap[ngap-1])+min( lambda, (id0+l-id0gap[
      ngap-1]) );
00232   }
00233
00234     printf("---\n");
00235   printf("l=%d, *newl=%d, offset_first=%d\n", l, *newl, offset_first);
00236
00237   for (i=offset_first ; i<l; i++)
00238     (Vout)[i] = 0;
00239
00240     printf("*newl=%d\n", *newl);
00241
00242
00243  return 0;
00244 }
00245 //=========================================================================
00246
00248
00255 int gap_masking(double **V, int l, int *id0gap, int *lgap, int ngap)
00256 {
00257
00258   int i,k;
00259   int lg;
00260
00261   int offset_first=id0gap[0];
00262   for (i=0 ; i<(ngap-1); i++) {
00263     lg = id0gap[i+1]-(id0gap[i]+lgap[i]);
00264     memmove(&(*V)[offset_first], &(*V)[id0gap[i]+lgap[i]], lg*sizeof(double));
00265     offset_first += lg;
00266   }
00267
00268   i=(ngap-1);
00269   lg = l-(id0gap[i]+lgap[i]);
00270
00271     memmove(&(*V)[offset_first], &(*V)[id0gap[i]+lgap[i]], lg*sizeof(double));
00272     offset_first += lg;
00273
00274
00275   for (i=offset_first ; i<l; i++)
00276     (*V)[i] = 0;
00277
00278
00279  return 0;
00280 }
00281
00282
00283 //=========================================================================
00284
00286
00293 int gap_filling(double **V, int l, int *id0gap, int *lgap, int ngap)
00294 {
00295
00296   int i,k;
00297   int lg;
00298
00299   int lgaptot = 0;
00300   for (i=0 ; i<ngap; i++)
00301     lgaptot += lgap[i];
00302
00303   int id0_Vmv;
00304   int offset_last;
00305
00306   id0_Vmv = id0gap[ngap-1]+lgap[ngap-1];
00307   offset_last = id0_Vmv - lgaptot;
00308   lg = l-id0_Vmv;
00309
00310   if (lg>0)
```

```
00311     memmove(&(*V)[id0_Vmv], &(*V)[offset_last], lg*sizeof(double));
00312
00313   for (i=ngap-2 ; i>=0; i--) {
00314     id0_Vmv = id0gap[i]+lgap[i];
00315     lg = id0gap[i+1]-id0_Vmv;
00316     offset_last -= lg;
00317
00318     memmove(&(*V)[id0_Vmv], &(*V)[offset_last], lg*sizeof(double));
00319   }
00320
00321   for (i=0 ; i<ngap; i++)
00322    for (k=0 ; k<lgap[i]; k++)
00323     (*V)[id0gap[i]+k] = 0;
00324
00325  return 0;
00326 }
00327
00328
00329 //a naive version - in dev
00330 int gap_masking_naive(double **V, int id0,int local_V_size,
      int m, int nrow, int *id0gap, int *lgap, int ngap)
00331 {
00332   int i,j,k;
00333   int icol;
00334
00335   int offsetV=id0;
00336   k=0;
00337
00338   for (i=0 ; i<local_V_size; i++) {
00339   icol=(i+id0)%nrow;
00340
00341   if ( icol<id0gap[k] ) {
00342     (*V)[offsetV]=(*V)[i];
00343     offsetV=offsetV+1;
00344   }
00345   else if (icol>=id0gap[k]+lgap[k]) {
00346     k=k+1;
00347     i=i-1;
00348 //    (*V)[offsetV]=(*V)[i];
00349 //    offsetV=offsetV+1;
00350   }
00351   else {//do not copy, just skip
00352   }
00353
00354
00355   return 0;
00356 }
00357
00358
```

15.57 toeplitz_gappy_seq.dev.h File Reference

Functions

- int tpltz_init (int n, int lambda, int *nfft, int *blocksize, fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b)
- int tpltz_cleanup (fftw_complex **T_fft, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b)

  Cleans fftw workspace used in the Toeplitz matrix matrix product's computation.
- int stmm_core (double **V, int n, int m, fftw_complex *T_fft, int blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f, fftw_plan plan_b, int flag_offset)
- int stmm (double **V, int n, int m, int id0, int l, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft)
- int gstmm (double **V, int n, int m, int id0, int l, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft, int *id0gap, int *lgap, int ngap)
- int gap_masking (double **V, int l, int *id0gap, int *lgap, int ngap)

  Reduce the vector and mask the defined gaps.
- int gap_filling (double **V, int l, int *id0gap, int *lgap, int ngap)

  Extend the vector and add zeros on the gaps locations.
- int reset_gaps (double **V, int id0, int local_V_size, int m, int nrow, int *id0gap, int *lgap, int ngap)
- int mpi_stmm (double **V, int n, int m, int id0, int l, double *T, int lambda, MPI_Comm comm)

- int mpi_stbmm (double **V, int *n, int m, int nrow, double *T, int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int idp, int local_V_size, MPI_Comm comm)
- int mpi_gstbmm (double **V, int *n, int m, int nrow, double *T, int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int id0p, int local_V_size, int *id0gap, int *lgap, int ngap, MPI_Comm comm)
- int optimal_blocksize (int n, int lambda, int bs_flag)
- int fftw_init_omp_threads ()
- int rhs_init_fftw (int *nfft, int fft_size, fftw_complex **V_fft, double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag)

  Initializes fftw array and plan for the right hand side, general matrix V.
- int circ_init_fftw (double *T, int fft_size, int lambda, fftw_complex **T_fft)

  Initializes fftw array and plan for the circulant matrix T_circ obtained from T.
- int scmm_direct (int fft_size, fftw_complex *C_fft, int ncol, double *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V, fftw_plan plan_b_CV)
- int scmm_basic (double **V, int blocksize, int m, fftw_complex *C_fft, int lambda, double **CV, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan plan_f_V, fftw_plan plan_b_CV)
- int stmm_reshape (double **V, int n, int m, int id0, int l, fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b, int blocksize, int nfft)
- int build_gappy_blocks (int *n, int m, int nrow, double *T, int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int *id0gap, int *lgap, int ngap, int *nb_blocks_gappy_final, double *Tgappy, int *idvgappy, int *ngappy, int *lambdagappy, int flag_param_distmin_fixed)
- int print_error_message (int error_number, char const *file, int line)

  Prints error message corresponding to an error number.
- int copy_block (int ninrow, int nincol, double *Vin, int noutrow, int noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int outrow, int outcol, double norm, int set_zero_flag)

  Copies (and potentially reshapes) a selected block of the input matrix to a specified position of the output matrix.
- int vect2nfftblock (double *V1, int v1_size, double *V2, int fft_size, int nfft, int lambda)
- int nfftblock2vect (double *V2, int fft_size, int nfft, int lambda, double *V1, int v1_size)
- int gap_reduce (double **V, int id0, int l, int lambda, int *id0gap, int *lgap, int ngap, int *newl, int id0out)

  ...convert the data vector structure into a matrix structure optimized for nfft
- int get_overlapping_blocks_params (int nbloc, int *idv, int *n, int local_V_size, int nrow, int idp, int *idpnew, int *local_V_size_new, int *nnew, int *ifirstBlock, int *ilastBlock)

15.57.1 Function Documentation

15.57.1.1 **int tpltz_init (int *n,* int *lambda,* int * *nfft,* int * *blocksize,* fftw_complex ** *T_fft,* double * *T,* fftw_complex ** *V_fft,* double ** *V_rfft,* fftw_plan * *plan_f,* fftw_plan * *plan_b*)**

15.57.1.2 **int stmm_core (double ** *V,* int *n,* int *m,* fftw_complex * *T_fft,* int *blocksize,* int *lambda,* fftw_complex * *V_fft,* double * *V_rfft,* int *nfft,* fftw_plan *plan_f,* fftw_plan *plan_b,* int *flag_offset*)**

15.57.1.3 **int stmm (double ** *V,* int *n,* int *m,* int *id0,* int *l,* fftw_complex * *T_fft,* int *lambda,* fftw_complex * *V_fft,* double * *V_rfft,* fftw_plan *plan_f,* fftw_plan *plan_b,* int *blocksize,* int *nfft*)**

15.57.1.4 **int gstmm (double ** *V,* int *n,* int *m,* int *id0,* int *l,* fftw_complex * *T_fft,* int *lambda,* fftw_complex * *V_fft,* double * *V_rfft,* fftw_plan *plan_f,* fftw_plan *plan_b,* int *blocksize,* int *nfft,* int * *id0gap,* int * *lgap,* int *ngap*)**

Definition at line 7 of file toeplitz_gappy_seq.dev.c.

15.57.1.5 **int reset_gaps (double ** *V,* int *id0,* int *local_V_size,* int *m,* int *nrow,* int * *id0gap,* int * *lgap,* int *ngap*)**

15.57.1.6 **int mpi_stmm (double ** *V,* int *n,* int *m,* int *id0,* int *l,* double * *T,* int *lambda,* MPI_Comm *comm*)**

15.57.1.7 **int mpi_stbmm (double ** *V,* int * *n,* int *m,* int *nrow,* double * *T,* int *nb_blocks_local,* int *nb_blocks_all,* int * *lambda,* int * *idv,* int *idp,* int *local_V_size,* MPI_Comm *comm*)**

15.57.1.8 int mpi_gstbmm (double ** *V,* int * *n,* int *m,* int *nrow,* double * *T,* int *nb_blocks_local,* int *nb_blocks_all,* int * *lambda,* int * *idv,* int *id0p,* int *local_V_size,* int * *id0gap,* int * *lgap,* int *ngap,* MPI_Comm *comm*)

15.57.1.9 int optimal_blocksize (int *n,* int *lambda,* int *bs_flag*)

15.57.1.10 int fftw_init_omp_threads ( )

15.57.1.11 int scmm_direct (int *fft_size,* fftw_complex * *C_fft,* int *ncol,* double * *V_rfft,* double ** *CV,* fftw_complex * *V_fft,* fftw_plan *plan_f_V,* fftw_plan *plan_b_CV*)

15.57.1.12 int scmm_basic (double ** *V,* int *blocksize,* int *m,* fftw_complex * *C_fft,* int *lambda,* double ** *CV,* fftw_complex * *V_fft,* double * *V_rfft,* int *nfft,* fftw_plan *plan_f_V,* fftw_plan *plan_b_CV*)

15.57.1.13 int stmm_reshape (double ** *V,* int *n,* int *m,* int *id0,* int *l,* fftw_complex * *T_fft,* int *lambda,* fftw_complex * *V_fft,* double * *V_rfft,* fftw_plan *plan_f,* fftw_plan *plan_b,* int *blocksize,* int *nfft*)

15.57.1.14 int build_gappy_blocks (int * *n,* int *m,* int *nrow,* double * *T,* int *nb_blocks_local,* int *nb_blocks_all,* int * *lambda,* int * *idv,* int * *id0gap,* int * *lgap,* int *ngap,* int * *nb_blocks_gappy_final,* double * *Tgappy,* int * *idvgappy,* int * *ngappy,* int * *lambdagappy,* int *flag_param_distmin_fixed*)

15.57.1.15 int vect2nfftblock (double * *V1,* int *v1_size,* double * *V2,* int *fft_size,* int *nfft,* int *lambda*)

15.57.1.16 int nfftblock2vect (double * *V2,* int *fft_size,* int *nfft,* int *lambda,* double * *V1,* int *v1_size*)

15.57.1.17 int get_overlapping_blocks_params (int *nbloc,* int * *idv,* int * *n,* int *local_V_size,* int *nrow,* int *idp,* int * *idpnew,* int * *local_V_size_new,* int * *nnew,* int * *ifirstBlock,* int * *ilastBlock*)

15.58 toeplitz_gappy_seq.dev.h

```
00001
00002 #ifndef         TOEPLITZ_H_
00003 #define         TOEPLITZ_H_
00004
00005 #ifdef MPI
00006 #include <mpi.h>
00007 #endif
00008
00009 #include <fftw3.h>
00010 #include <omp.h>
00011 #include <stdlib.h>
00012 #include <stdio.h>
00013 #include <math.h>
00014 #include <string.h>
00015
00016 //============================================================================
00017 //Basic functions definition
00018 #define max(a,b) (((a) > (b)) ? (a) : (b))
00019 #define min(a,b) (((a) < (b)) ? (a) : (b))
00020
00021 //============================================================================
00022 //Fixed parameters
00023
00025
00027 #ifndef VERBOSE
00028 #define VERBOSE 0
00029 #endif
00030
00032
00034 #ifndef MPI_USER_TAG
00035 #define MPI_USER_TAG 123
00036 #endif
00037
00038
00039 //============================================================================
00040 // User routines definition
00041
00042 //Sequential routines (11)
00043 int tpltz_init(int n, int lambda, int *nfft, int *blocksize,
      fftw_complex **T_fft, double *T, fftw_complex **V_fft, double **V_rfft, fftw_plan *
      plan_f, fftw_plan *plan_b);
00044
```

```
00045 int tpltz_cleanup(fftw_complex **T_fft, fftw_complex **V_fft,
      double **V_rfft,fftw_plan *plan_f, fftw_plan *plan_b);
00046
00047 int stmm_core(double **V, int n, int m, fftw_complex *T_fft, int
      blocksize, int lambda, fftw_complex *V_fft, double *V_rfft, int nfft, fftw_plan
      plan_f, fftw_plan plan_b, int flag_offset);
00048
00049 int stmm(double **V, int n, int m, int id0, int l, fftw_complex *T_fft, int
       lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan plan_b
      , int blocksize, int nfft);
00050
00051 int gstmm(double **V, int n, int m, int id0, int l, fftw_complex *T_fft,
      int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f, fftw_plan
      plan_b, int blocksize, int nfft, int *id0gap, int *lgap, int ngap);
00052
00053 //int stbmm(double **V, int *n, int m, int nrow, double *T, int
       nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int idp, int local_V_size);
00054
00055 //int gstbmm(double **V, int *n, int m, int nrow, double *T, int
       nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int id0p, int local_V_size, int
       *id0gap, int *lgap, int ngap);
00056
00057 int gap_masking(double **V, int l, int *id0gap, int *lgap, int ngap)
      ;
00058
00059 int gap_filling(double **V, int l, int *id0gap, int *lgap, int ngap)
      ;
00060
00061 int reset_gaps(double **V, int id0,int local_V_size, int m, int nrow,
       int *id0gap, int *lgap, int ngap);
00062
00063
00064 //Mpi routines (12)
00065 #ifdef MPI
00066 int mpi_stmm(double **V, int n, int m, int id0, int l, double *T, int
      lambda, MPI_Comm comm);
00067
00068 int mpi_stbmm(double **V, int *n, int m, int nrow, double *T, int
      nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int idp, int local_V_size
      , MPI_Comm comm);
00069
00070 int mpi_gstbmm(double **V, int *n, int m, int nrow, double *T, int
      nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int id0p, int
      local_V_size, int *id0gap, int *lgap, int ngap, MPI_Comm comm);
00071
00072 #endif
00073
00074
00075 //========================================================================
00076 // User routines definition
00077
00078 //Low level routines (21)
00079 int optimal_blocksize(int n, int lambda, int bs_flag);
00080
00081 int fftw_init_omp_threads();
00082
00083 int rhs_init_fftw(int *nfft, int fft_size, fftw_complex **V_fft,
      double **V_rfft, fftw_plan *plan_f, fftw_plan *plan_b, int fftw_flag);
00084
00085 int circ_init_fftw(double *T, int fft_size, int lambda,
      fftw_complex **T_fft);
00086
00087 int scmm_direct(int fft_size, fftw_complex *C_fft, int ncol, double
      *V_rfft, double **CV, fftw_complex *V_fft, fftw_plan plan_f_V, fftw_plan
      plan_b_CV);
00088
00089 int scmm_basic(double **V, int blocksize, int m, fftw_complex *C_fft,
       int lambda, double **CV, fftw_complex *V_fft, double *V_rfft, int nfft,
      fftw_plan plan_f_V, fftw_plan plan_b_CV);
00090
00091 int stmm_reshape(double **V, int n, int m, int id0, int l,
      fftw_complex *T_fft, int lambda, fftw_complex *V_fft, double *V_rfft, fftw_plan plan_f,
      fftw_plan plan_b, int blocksize, int nfft);
00092
00093 int build_gappy_blocks(int *n, int m, int nrow, double *T,
      int nb_blocks_local, int nb_blocks_all, int *lambda, int *idv, int *id0gap, int *
      lgap, int ngap, int *nb_blocks_gappy_final, double *Tgappy, int *idvgappy, int *
      ngappy, int *lambdagappy, int flag_param_distmin_fixed);
00094
00095
00096 //Internal routines (22)
00097 int print_error_message(int error_number, char const *file
      , int line);
00098
00099 int copy_block(int ninrow, int nincol, double *Vin, int noutrow, int
      noutcol, double *Vout, int inrow, int incol, int nblockrow, int nblockcol, int
      outrow, int outcol, double norm, int set_zero_flag);
```

```
00100
00101 int vect2nfftblock(double *V1, int v1_size, double *V2, int
      fft_size, int nfft, int lambda);
00102
00103 int nfftblock2vect(double *V2, int fft_size, int nfft, int lambda
      , double *V1, int v1_size);
00104
00105 int gap_reduce(double **V, int id0, int l, int lambda, int *id0gap,
      int *lgap, int ngap, int *newl, int id0out);
00106
00107 int get_overlapping_blocks_params(int nbloc, int *
      idv, int *n, int local_V_size, int nrow, int idp, int *idpnew, int *
      local_V_size_new, int *nnew, int *ifirstBlock, int *ilastBlock);
00108
00109
00110 //============================================================================
00111 #endif      /* !TOEPLITZ_H_ */
00112
00113
```

15.59 toeplitz_nofft.c File Reference

Contains basic product without using ffts for Toeplitz algebra.

Functions

- int stmm_simple_basic (double **V, int n, int m, double *T, int lambda, double **TV)

  Perform the product of a Toeplitz matrix by a matrix without using FFT's.
- int stmm_simple_core (double **V, int n, int m, double *T, int blocksize, int lambda, int nfft, int flag_offset)

  Perform the stand alone product of a Toeplitz matrix by a matrix using the sliding window algorithm.

Variables

- int PRINT_RANK

15.59.1 Detailed Description

Contains basic product without using ffts for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research
Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz_nofft.c.

15.59.2 Variable Documentation

15.59.2.1 int PRINT_RANK

Definition at line 82 of file toeplitz.c.

15.60 toeplitz_nofft.c

```
00001
00057 #include "toeplitz.h"
00058 extern int PRINT_RANK;
00059
00060 //r1.1 - Frederic Dauvergne (APC)
00061 //basic product without fft use.
00062 //stmm_simple_core is not used by the API. This is similar to stmm_core by
       using a sliding
00063 //windows algorithm with differents parameters.
00064
00065
00066 //============================================================================
00068
00073 int stmm_simple_basic(double **V, int n, int m, double *T, int
       lambda, double **TV)
00074 {
00075
00076   int j_first, j_last;
00077   int i,j,k,Tid;
00078   int n_thread;
00079   int idx;
00080
```

```
00081   int flag_nocomputeedges=1;
00082   int offset_edges=0;
00083
00084   int distcorrmin= lambda-1;
00085
00086   if (flag_nocomputeedges==1)
00087     offset_edges=distcorrmin;
00088
00089
00090  for (k=0;k<m;k++) {
00091
00092 #pragma omp parallel for shared(k,lambda,n) private(i,j,j_first,j_last,Tid)
00093   for(i=0+offset_edges;i<n-offset_edges;i++) {
00094
00095     (*TV)[i+k*n]=0;
00096     j_first=max( i-(lambda-1) , 0);
00097     j_last =min( i+lambda , n);
00098
00099   for(j=j_first;j<j_last;j++) {
00100     Tid=abs(j-i);
00101     (*TV)[i+k*n] += T[Tid] * (*V)[j+k*n];
00102   } //End j loop
00103
00104   } //End i loop
00105  } //End k loop
00106
00107   return 0;
00108 }
00109
00110
00111
00112 //========================================================================
00113
00115
00128 int stmm_simple_core(double **V, int n, int m, double *T, int
      blocksize, int lambda, int nfft, int flag_offset)
00129 {
00130
00131   //routine variable
00132   int status;
00133   int i,j,k,p;  //loop index
00134   int currentsize;
00135   int distcorrmin= lambda-1;
00136   int blocksize_eff = blocksize-2*distcorrmin;  //just a good part after
       removing the overlaps
00137   int nbloc;  //a number of subblock of slide/overlap algorithm
00138
00139   if (flag_offset==1)
00140     nbloc = ceil((1.0*(n-2*distcorrmin))/blocksize_eff);
00141   else
00142     nbloc = ceil( (1.0*n)/blocksize_eff);
00143
00144
00145   double *V_bloc, *TV_bloc;
00146   V_bloc  = (double *) calloc(blocksize*m, sizeof(double));
00147   TV_bloc = (double *) calloc(blocksize*m, sizeof(double));
00148   if((V_bloc==0)||(TV_bloc==0))
00149     return print_error_message(2, __FILE__, __LINE__);
00150
00151   int offset=0;
00152   if (flag_offset==1)
00153     offset=distcorrmin;
00154
00155   int iV = 0;  //"-distcorrmin+offset";  //first index in V
00156   int iTV = offset;  //first index in TV
00157
00158   //"k=0";
00159   //first subblock separately as it requires some padding. prepare the block of
       the data vector
00160   //with the overlaps on both sides
00161   currentsize = min( blocksize-distcorrmin+offset, n-iV);
00162   //note: if flag_offset=0, pad first distcorrmin elements with zeros (for the
       first subblock only)
00163   // and if flag_offset=1 there is no padding with zeros.
00164   copy_block( n, m, *V, blocksize, m, V_bloc, 0, 0, currentsize, m,
      distcorrmin-offset, 0, 1.0, 0);
00165
00166   //do block computation
00167   status = stmm_simple_basic(&V_bloc, blocksize, m, T, lambda,
       &TV_bloc);
00168
00169   if (status!=0) {
00170     printf("Error in stmm_core.");
00171     return print_error_message(7, __FILE__, __LINE__);  }
00172
00173   //now copy first the new chunk of the data matrix **before** overwriting the
       input due to overlaps !
```

```
00174   iV = blocksize_eff-distcorrmin+offset;
00175
00176   if(nbloc > 1) {
00177     currentsize  = min( blocksize, n-iV);  //not to overshoot
00178
00179     int flag_reset = (currentsize!=blocksize);  //with flag_reset=1, always
      "memset" the block.
00180     copy_block( n, m, *V, blocksize, m, V_bloc, iV, 0, currentsize, m
     , 0, 0, 1.0, flag_reset);
00181   }
00182
00183   //and now store the ouput back in V
00184   currentsize  = min( blocksize_eff, n-iTV);       // to trim the extra rows
00185   copy_block( blocksize, m, TV_bloc, n, m, *V, distcorrmin, 0,
      currentsize, m, iTV, 0, 1.0, 0);
00186
00187
00188   iTV += blocksize_eff;
00189   //now continue with all the other subblocks
00190   for(k=1;k<nbloc;k++) {
00191
00192     //do bloc computation
00193     status = stmm_simple_basic(&V_bloc, blocksize, m, T,
     lambda, &TV_bloc);
00194     if (status!=0) break;
00195
00196     iV += blocksize_eff;
00197     //copy first the next subblock to process
00198     if(k != nbloc-1) {
00199       currentsize = min(blocksize, n-iV);  //not to overshoot
00200
00201       int flag_resetk = (currentsize!=blocksize);  //with flag_reset=1, always
      "memset" the block.
00202       copy_block( n, m, *V, blocksize, m, V_bloc, iV, 0, currentsize,
      m, 0, 0, 1.0, flag_resetk);
00203     }
00204
00205     //and then store the output in V
00206     currentsize  = min( blocksize_eff, n-iTV);  //not to overshoot

00207     copy_block( blocksize, m, TV_bloc, n, m, *V, distcorrmin, 0,
      currentsize, m, iTV, 0, 1.0, 0);
00208     iTV += blocksize_eff;
00209
00210   }//end bloc computation
00211
00212
00213   free(V_bloc);
00214   free(TV_bloc);
00215
00216   return status;
00217 }
00218
00219
00220
```

15.61 toeplitz_params.c File Reference

Routines to set the flag strategy parameters for Toeplitz algebra.

Functions

- int flag_stgy_init_auto (Flag *flag_stgy)

  Set the flag to automatic paramaters.

- int flag_stgy_init_zeros (Flag *flag_stgy)

  Set the flag parameters to zeros. This is almost the same as automatic.

- int flag_stgy_init_defined (Flag *flag_stgy)

  Set the parameters flag to the defined ones.

- int print_flag_stgy_init (Flag flag_stgy)

  Print the flag parameters values.

15.61.1 Detailed Description

Routines to set the flag strategy parameters for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz_params.c.

15.62 toeplitz_params.c

```
00001
00057 #include "toeplitz.h"
00058
00059
00060 //r1.1 - Frederic Dauvergne (APC)
00061 //This is some routines to set the flag strategy parameters
00062
00063 //todo:
00064 //- add some routines to estimate the best choice strategy when automatic
00065 //parameters are choosen.
00066
00067 //============================================================================
00068
00070
00073 int flag_stgy_init_auto(Flag *flag_stgy)
00074 {
00075   static const Flag z = {0};
00076    *flag_stgy = z;
00077
00078   flag_stgy->flag_fftw=FLAG_FFTW;
00079 //  flag_stgy->flag_verbose=FLAG_VERBOSE;
00080
00081   return 0;
00082 }
00083
00084
00085 //============================================================================
00086
00088
00091 int flag_stgy_init_zeros(Flag *flag_stgy)
00092 {
00093   static const Flag z = {0};
00094    *flag_stgy = z;
00095
00096   return 0;
00097 }
00098
00099
00100 //============================================================================
00101
00103
00106 int flag_stgy_init_defined(Flag *flag_stgy)
00107 {
00108
00109   flag_stgy->flag_bs=FLAG_BS;  //0:auto 1:fixed 2:zero 3:3lambda
       4:4lambda 5:formula2
00110   flag_stgy->flag_nfft=FLAG_NFFT;  //0:auto  1:fixed  2:numthreads
       3:fftwthreads
00111   flag_stgy->flag_fftw=FLAG_FFTW;
00112   flag_stgy->flag_no_rshp=FLAG_NO_RSHP;  //0:auto  1:yes  1:no
00113   flag_stgy->flag_nofft=FLAG_NOFFT; //0:auto  1:yes  1:no
00114   flag_stgy->flag_blockingcomm=FLAG_BLOCKINGCOMM;  //0:auto
       1:noblocking 2:blocking 3:blocking_nooptim
00115   flag_stgy->fixed_nfft=FIXED_NFFT;  //fixed init value for nfft
00116   flag_stgy->fixed_bs=FIXED_BS;    //fixed init value for blockside
00117   flag_stgy->flag_verbose=FLAG_VERBOSE;
00118   flag_stgy->flag_skip_build_gappy_blocks=
      FLAG_SKIP_BUILD_GAPPY_BLOCKS;
00119   flag_stgy->flag_param_distmin_fixed=
      FLAG_PARAM_DISTMIN_FIXED;
00120   flag_stgy->flag_precompute_lvl=FLAG_PRECOMPUTE_LVL;
00121   return 0;
00122 }
00123
00124
00125 //============================================================================
00126
00128
00131 int print_flag_stgy_init(Flag flag_stgy)
00132 {
00133
00134   FILE *file;
00135   file = stdout;
00136
00137   fprintf(file, "flag_bs=%d\n", flag_stgy.flag_bs);
00138   fprintf(file, "flag_nfft=%d\n", flag_stgy.flag_nfft);
00139   fprintf(file, "flag_fftw=%d\n", flag_stgy.flag_fftw);
00140   fprintf(file, "flag_no_rshp=%d\n", flag_stgy.flag_no_rshp);
00141   fprintf(file, "flag_nofft=%d\n", flag_stgy.flag_nofft);
00142   fprintf(file, "flag_blockingcomm=%d\n", flag_stgy.flag_blockingcomm
      );
00143   fprintf(file, "fixed_nfft=%d\n", flag_stgy.fixed_nfft);
00144   fprintf(file, "fixed_bs=%d\n", flag_stgy.fixed_bs);
00145   fprintf(file, "flag_verbose=%d\n", flag_stgy.flag_verbose);
```

```
00146   fprintf(file, "flag_skip_build_gappy_blocks=%d\n", flag_stgy.
      flag_skip_build_gappy_blocks);
00147   fprintf(file, "flag_param_distmin_fixed=%d\n", flag_stgy.
      flag_param_distmin_fixed);
00148   fprintf(file, "flag_precompute_lvl=%d\n", flag_stgy.flag_precompute_lvl
      );
00149
00150   return 0;
00151 }
00152
00153
00154
```

15.63 toeplitz_rshp.c File Reference

Contains reshaping routines to build the optimal data structure when needed for Toeplitz algebra.

Functions

- int fctid_mat2vect (int i, int id0, int n, int lambda)
- int fctid_mat2vect_inv (int i, int id0, int n, int lambda)
- int fctid_concatcol (int i, int id0, int n, int m, int l, int lconc, int lambda, int ∗nocol, int nbcol)
- int fctid_concatcol_inv (int i, int id0, int n, int m, int l, int lconc, int lambda, int ∗nocol_inv, int nbcol)
- int fctid_vect2nfftblock (int i, int v1_size, int fft_size, int nfft, int lambda)
- int is_needconcat (int ∗nocol, int nbcol)
- int fctid_vect2nfftblock_inv (int i, int v1_size, int fft_size, int nfft, int lambda)
- int define_rshp_size (int flag_format_rshp, int fft_size, int nfft, int v1_size, int vedge_size, int ∗nrshp, int ∗mrshp, int ∗lrshp)
- int build_nocol_inv (int ∗nocol, int nbcol, int m)
- int build_reshape (double ∗Vin, int ∗nocol, int nbcol, int lconc, int n, int m, int id0, int l, int lambda, int nfft, double ∗Vrshp, int nrshp, int mrshp, int lrshp, int flag_format_rshp)
- int extract_result (double ∗Vout, int ∗nocol, int nbcol, int lconc, int n, int m, int id0, int l, int lambda, int nfft, double ∗Vrshp, int nrshp, int mrshp, int lrshp, int flag_format_rshp)

15.63.1 Detailed Description

Contains reshaping routines to build the optimal data structure when needed for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz_rshp.c.

15.63.2 Function Documentation

15.63.2.1 int fctid_mat2vect (int *i,* int *id0,* int *n,* int *lambda*)

Definition at line 65 of file toeplitz_rshp.c.

15.63.2.2 int fctid_mat2vect_inv (int *i,* int *id0,* int *n,* int *lambda*)

Definition at line 88 of file toeplitz_rshp.c.

15.63.2.3 int fctid_concatcol (int *i,* int *id0,* int *n,* int *m,* int *l,* int *lconc,* int *lambda,* int ∗ *nocol,* int *nbcol*)

Definition at line 106 of file toeplitz_rshp.c.

15.63.2.4 int fctid_concatcol_inv (int *i,* int *id0,* int *n,* int *m,* int *l,* int *lconc,* int *lambda,* int ∗ *nocol_inv,* int *nbcol*)

Definition at line 128 of file toeplitz_rshp.c.

15.63.2.5 int fctid_vect2nfftblock (int *i*, int *v1_size*, int *fft_size*, int *nfft*, int *lambda*)

Definition at line 154 of file toeplitz_rshp.c.

15.63.2.6 int is_needconcat (int * *nocol*, int *nbcol*)

Definition at line 176 of file toeplitz_rshp.c.

15.63.2.7 int fctid_vect2nfftblock_inv (int *i*, int *v1_size*, int *fft_size*, int *nfft*, int *lambda*)

Definition at line 190 of file toeplitz_rshp.c.

15.63.2.8 int define_rshp_size (int *flag_format_rshp*, int *fft_size*, int *nfft*, int *v1_size*, int *vedge_size*, int * *nrshp*, int * *mrshp*, int * *lrshp*)

Definition at line 208 of file toeplitz_rshp.c.

15.63.2.9 int build_nocol_inv (int * *nocol*, int *nbcol*, int *m*)

Definition at line 233 of file toeplitz_rshp.c.

15.63.2.10 int build_reshape (double * *Vin*, int * *nocol*, int *nbcol*, int *lconc*, int *n*, int *m*, int *id0*, int *l*, int *lambda*, int *nfft*, double * *Vrshp*, int *nrshp*, int *mrshp*, int *lrshp*, int *flag_format_rshp*)

Definition at line 249 of file toeplitz_rshp.c.

15.63.2.11 int extract_result (double * *Vout*, int * *nocol*, int *nbcol*, int *lconc*, int *n*, int *m*, int *id0*, int *l*, int *lambda*, int *nfft*, double * *Vrshp*, int *nrshp*, int *mrshp*, int *lrshp*, int *flag_format_rshp*)

Definition at line 307 of file toeplitz_rshp.c.

15.64 toeplitz_rshp.c

```
00001
00057 #include "toeplitz.h"
00058
00059 //r1.1 - Frederic Dauvergne (APC)
00060 //This is the reshaping routines to build the optimal data structure when
       needed.
00061 //The index functions find the right index number of the data location for a
       choosen
00062 //transformation.
00063
00064
00065 int fctid_mat2vect(int i, int id0, int n, int lambda)
00066 {
00067   int I,J, i_out;
00068   int distcorrmin= lambda-1;
00069   int rfirst=id0%n;
00070
00071   if (i==-1)
00072     return (-1);
00073
00074
00075   I = (i+rfirst)%(n+distcorrmin);
00076   J = (i+rfirst)/(n+distcorrmin);
00077
00078   if (I<n)
00079     i_out = I-rfirst+J*n;
00080   else
00081     i_out = -1; //not defined. value is zero.
```

```
00082
00083
00084   return i_out;
00085 }
00086
00087
00088 int fctid_mat2vect_inv(int i, int id0, int n, int lambda)
00089 {
00090   int I,J, i_out;
00091   int distcorrmin= lambda-1;
00092   int rfirst=id0%n;
00093
00094   if (i==-1)
00095     i_out = -1; //not defined. value is zero.
00096
00097   I = (i+rfirst)%(n);
00098   J = (i+rfirst)/(n);
00099
00100   i_out = I-rfirst+J*(n+distcorrmin);
00101
00102   return i_out;
00103 }
00104
00105
00106 int fctid_concatcol(int i, int id0, int n, int m, int l, int
      lconc, int lambda, int *nocol, int nbcol)
00107 {
00108   int I,J, i_out;
00109   int distcorrmin= lambda-1;
00110   int rfirst=id0%n;
00111
00112   if (i==-1)
00113     return (-1);
00114
00115   if (i>=lconc)
00116     return (-2);//this indice not define. It shouldn't be used
00117
00118   I = (i+rfirst)%(n);
00119   J = (i+rfirst)/(n);
00120
00121     i_out = I-rfirst+nocol[J]*(n);
00122
00123
00124   return i_out;
00125 }
00126
00127
00128 int fctid_concatcol_inv(int i, int id0, int n, int m, int l,
       int lconc, int lambda, int *nocol_inv, int nbcol)
00129 {
00130   int I,J, i_out;
00131   int distcorrmin= lambda-1;
00132   int rfirst=id0%n;
00133
00134   if (i==-1)
00135     return (-1);
00136
00137   if (i>=l)
00138     return (-2);//this indice not define. It shouldn't be used
00139
00140   I = (i+rfirst)%(n);
00141   J = (i+rfirst)/(n);
00142
00143   if (nocol_inv[J]==(-1))
00144     i_out = -1;
00145   else
00146     i_out = I-rfirst+nocol_inv[J]*(n);
00147
00148
00149   return i_out;
00150 }
00151
00152
00153
00154 int fctid_vect2nfftblock(int i, int v1_size, int fft_size,
      int nfft, int lambda)
00155 {
00156
00157   int I,J, i_out;
00158   int distcorrmin= lambda-1;
00159
00160   if (i==-1)
00161     return (-1);
00162
00163   I = (i)%(fft_size);
00164   J = (i)/(fft_size);
00165
```

```
00166   i_out = (I-distcorrmin) + J*(fft_size-2*distcorrmin) ;
00167
00168   if (i_out<0 || i_out>=v1_size)
00169     i_out = -1;
00170
00171
00172   return i_out;
00173 }
00174
00175
00176 int is_needconcat(int *nocol, int nbcol)
00177 {
00178   int i;
00179   int ip=nocol[0];
00180   for(i=1;i<nbcol;i++) {
00181     if (nocol[i]!=(ip+i))
00182       return 1;
00183   }
00184
00185
00186   return 0;
00187 }
00188
00189
00190 int fctid_vect2nfftblock_inv(int i, int v1_size, int
      fft_size, int nfft, int lambda)
00191 {
00192
00193   int I,J, i_out;
00194   int distcorrmin= lambda-1;
00195
00196   if (i<0 || i>=v1_size)
00197     return (-2);
00198
00199   I = (i)%(fft_size-2*distcorrmin);
00200   J = (i)/(fft_size-2*distcorrmin);
00201
00202   i_out = (I+distcorrmin) + J*(fft_size) ;
00203
00204   return i_out;
00205 }
00206
00207
00208 int define_rshp_size(int flag_format_rshp, int fft_size, int
      nfft, int v1_size, int vedge_size, int *nrshp, int *mrshp, int *lrshp)
00209 {
00210
00211   if (flag_format_rshp==2) {
00212     *nrshp=fft_size;
00213     *mrshp=nfft;
00214     *lrshp=(*nrshp)*(*mrshp);
00215   }
00216   else if (flag_format_rshp==1) {
00217     *nrshp=v1_size;
00218     *mrshp=1;
00219     *lrshp=(*nrshp)*(*mrshp);
00220   }
00221   else if (flag_format_rshp==0) { //this case appear only if
       flag_shortcut_nbcol_eq_1==0
00222     *nrshp=vedge_size;
00223     *mrshp=1;
00224     *lrshp=vedge_size;
00225   }
00226   else {//error not a good flag_format_rshp
00227   }
00228
00229   return 0;
00230 }
00231
00232
00233 int build_nocol_inv(int *nocol, int nbcol, int m)  //ncol_inv to
       define as parameters
00234 {
00235  int i;
00236  int *nocol_inv;
00237   nocol_inv = (int *) calloc(m, sizeof(double));
00238
00239   for(i=0;i<m;i++)
00240     nocol_inv[i]=-1;
00241   for(i=0;i<nbcol;i++)
00242     nocol_inv[nocol[i]]=i;
00243
00244
00245   return 0;
00246 }
00247
00248
```

```
00249 int build_reshape(double *Vin, int *nocol, int nbcol, int lconc,
      int n, int m, int id0, int l, int lambda, int nfft, double *Vrshp, int nrshp, int
       mrshp, int lrshp, int flag_format_rshp)
00250 {
00251
00252   int i;
00253   int rfirst=id0%n;
00254   int i_out1, i_out2, i_out3;
00255   int distcorrmin=lambda-1;
00256
00257   int v1_size;
00258   int fft_size;
00259
00260   int idf = id0+l-1;
00261   int lconc0;
00262
00263   FILE *file;
00264   file = stdout;
00265
00266   v1_size=lconc+(distcorrmin)*(nbcol-1);
00267   fft_size = ceil(1.0*v1_size/nfft)+2*distcorrmin;
00268
00269  //used transformation
00270 if (VERBOSE) {
00271   fprintf(file, "fctid_concatcol: \t %d\n", (is_needconcat(nocol,
      nbcol)==1));
00272   fprintf(file, "fctid_mat2vect: \t %d\n", (nbcol>1));
00273   fprintf(file, "fctid_vect2nfftblock \t %d\n", (nfft>1));
00274 }
00275
00276
00277   for(i=0;i<lrshp;i++) {
00278
00279   if (nfft>1)
00280     i_out1 = fctid_vect2nfftblock( i, v1_size, fft_size,
      nfft, lambda);
00281   else
00282     i_out1 = i;
00283
00284   if (nbcol>1)
00285     i_out2 = fctid_mat2vect(i_out1 , rfirst, n, lambda);
00286   else
00287     i_out2 = i_out1;
00288
00289   if (is_needconcat(nocol, nbcol)==1)
00290     i_out3 = fctid_concatcol(i_out2, id0, n, m, l, lconc, lambda
      , nocol, nbcol);
00291   else
00292     i_out3 = i_out2;
00293
00294
00295   if (i_out3==-1)
00296     Vrshp[i]=0;
00297   else
00298     Vrshp[i]=Vin[i_out3];
00299
00300   }//end for
00301
00302
00303   return 0;
00304 }
00305
00306
00307 int extract_result(double *Vout, int *nocol, int nbcol, int lconc
      , int n, int m, int id0, int l, int lambda, int nfft, double *Vrshp, int nrshp,
      int mrshp, int lrshp, int flag_format_rshp)
00308 {
00309
00310   int i;
00311   int rfirst=id0%n;
00312   int i_out1, i_out2, i_out3;
00313   int i_in1;
00314   int distcorrmin=lambda-1;
00315
00316   int v1_size;
00317   int fft_size;
00318
00319   FILE *file;
00320   file = stdout;
00321
00322   v1_size=lconc+(distcorrmin)*(nbcol-1);
00323   fft_size = ceil(1.0*v1_size/nfft)+2*distcorrmin;
00324
00325  //used transformation
00326 if (VERBOSE) {
00327   fprintf(file, "fctid_concatcol: \t %d\n", (is_needconcat(nocol,
      nbcol)==1));
```

```
00328   fprintf(file, "fctid_mat2vect: \t %d\n", (nbcol>1));
00329   fprintf(file, "fctid_vect2nfftblock \t %d\n", (nfft>1));
00330 }
00331
00332   int lcol;
00333   int j,k;
00334
00335   for(i=0;i<lconc;i++) {
00336
00337   if (is_needconcat(nocol, nbcol)==1)
00338     i_in1=fctid_concatcol(i, id0, n, m, l, lconc, lambda, nocol,
       nbcol);
00339   else
00340     i_in1 = i;
00341
00342   if (nbcol>1)
00343     i_out2 = fctid_mat2vect_inv(i , rfirst, n, lambda);
00344   else
00345     i_out2 = i_out1;
00346
00347   if (nfft>1)
00348     i_out3 = fctid_vect2nfftblock_inv(i_out2, v1_size,
      fft_size, nfft, lambda);
00349   else
00350     i_out3 = i_out2;
00351
00352   if (i_out3==-1)
00353     Vout[i]=-1;
00354   else if (i_out3==-2)
00355     Vout[i]=-2;
00356   else
00357       Vout[i_in1] = Vrshp[i_out3];
00358   }
00359
00360
00361   return 0;
00362 }
00363
```

15.65 toeplitz_seq.c File Reference

Contains sequential/openMP routines for Toeplitz algebra.

Functions

- int stmm (double **V, int n, int m, double *T, int lambda, Flag flag_stgy)

  Perform the product of a Toeplitz matrix by a general matrix using the sliding window algorithm.
- int stbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_V-_size, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix, T, by an arbitrary matrix, V, distributed over processes in the generalized column-wise way.
- int gstbmm (double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks, int64_t idp, int local_-V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy)

  Performs the multiplication of a symmetric, Toeplitz block-diagonal matrix with gaps, T, by an arbitrary matrix, V, distributed over processes.
- int gstbmm0 (double **V, int nrow, int m, int m_rowwise, Block *tpltzblocks, int nb_blocks_local, int nb_-blocks_all, int id0p, int local_V_size, int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy)

15.65.1 Detailed Description

Contains sequential/openMP routines for Toeplitz algebra. version 1.1b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Definition in file toeplitz_seq.c.

15.65.2 Function Documentation

15.65.2.1 int gstbmm0 (double ∗∗ *V,* int *nrow,* int *m,* int *m_rowwise,* Block ∗ *tpltzblocks,* int *nb_blocks_local,* int *nb_blocks_all,* int *id0p,* int *local_V_size,* int64_t ∗ *id0gap,* int ∗ *lgap,* int *ngap,* Flag *flag_stgy*)

Definition at line 186 of file toeplitz_seq.c.

15.66 toeplitz_seq.c

```
00001
00049 #include "toeplitz.h"
00050
00051 //r1.1 - Frederic Dauvergne (APC)
00052 //This is the sequential version of the mpi routines for the API.
00053 //The stbmm and gstbmm are the same as the mpi_stbmm and mpi_gstbmm but without
00054 //any communication. stmm is a simplifed call of the sequential product
       including
00055 //initialization and cleaning.
00056
00057
00058 //=========================================================================
00060
```

```
00072 int stmm(double **V, int n, int m, double *T, int lambda, Flag
      flag_stgy)
00073 {
00074
00075 //fftw variables
00076   fftw_complex *V_fft, *T_fft;
00077   double *V_rfft;
00078   fftw_plan plan_f, plan_b;
00079
00080 //product parameters
00081   int nfft, blocksize;
00082
00083   FILE *file;
00084   file = stdout;
00085
00086
00087   tpltz_init(n, lambda, &nfft, &blocksize, &T_fft, T, &V_fft, &V_rfft
      , &plan_f, &plan_b, flag_stgy);
00088
00089   //Toeplitz computation
00090   if(VERBOSE)
00091     fprintf(file, "Before stmm_main call : nfft = %d, blocksize = %d\n", nfft,
      blocksize);
00092   stmm_main(V, n, m, 0, n*m, T, T_fft, lambda, V_fft, V_rfft, plan_f,
      plan_b, blocksize, nfft, flag_stgy);
00093
00094   tpltz_cleanup(&T_fft, &V_fft, &V_rfft, &plan_f, &plan_b);
00095
00096
00097   return 0;
00098 }
00099
00100
00101 //========================================================================
00103
00121 int stbmm(double **V, int nrow, int m_cw, int m_rw, Block *
      tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, Flag flag_stgy)
00122 {
00123
00124
00125   int nb_blocks_local=nb_blocks;
00126   int nb_blocks_all=nb_blocks;
00127  // int idp=0;
00128  // int local_V_size=nrow;
00129  //MPI_Comm comm=NULL;
00130
00131   mpi_stbmm(V, nrow, m_cw, m_rw, tpltzblocks, nb_blocks, nb_blocks,
      idp, local_V_size, flag_stgy, NULL);
00132
00133
00134
00135   return 0;
00136 }
00137
00138
00139 //========================================================================
00140
00142 // This matrix V contains defined gaps which represents the useless data for
       the comutation. The gaps indexes are defined in the global time space as the
       generized toeplitz matrix,
00143 // meaning the row dimension. Each of his diagonal blocks is a symmetric,
       band-diagonal Toeplitz matrix, which can be different for each block.
00170 int gstbmm(double **V, int nrow, int m_cw, int m_rw, Block *
      tpltzblocks, int nb_blocks, int64_t idp, int local_V_size, int64_t *id0gap, int *lgap,
      int ngap, Flag flag_stgy)
00171 {
00172   int nb_blocks_local=nb_blocks;
00173   int nb_blocks_all=nb_blocks;
00174  // int idp=0;
00175  // int local_V_size=nrow;
00176  //MPI_Comm comm=NULL;
00177
00178   mpi_gstbmm(V, nrow, m_cw, m_rw, tpltzblocks, nb_blocks, nb_blocks,
      idp, local_V_size, id0gap, lgap, ngap, flag_stgy, NULL);
00179
00180
00181
00182   return 0;
00183 }
00184
00185
00186 int gstbmm0(double **V, int nrow, int m, int m_rowwise, Block *
      tpltzblocks, int nb_blocks_local, int nb_blocks_all, int id0p, int local_V_size,
      int64_t *id0gap, int *lgap, int ngap, Flag flag_stgy)
00187 {
00188
00189   int rank=0;
```

```
00190   int i,j,k;   //some indexes
00191
00192   int flag_skip_build_gappy_blocks = flag_stgy.flag_skip_build_gappy_blocks
      ;
00193
00194   FILE *file;
00195   file = stdout;
00196   PRINT_RANK=rank ;
00197
00198 //put zeros at the gaps location
00199   reset_gaps( V, id0p, local_V_size, m, nrow, m_rowwise, id0gap, lgap
      , ngap);
00200
00201
00202 //allocation for the gappy structure of the diagonal block Toeplitz matrix
00203   int nb_blocks_gappy;
00204   int nb_blockgappy_max;
00205   int Tgappysize_max;
00206
00207   Block *tpltzblocks_gappy;
00208
00209 //some computation usefull to determine the max size possible for the gappy
       variables
00210   int Tsize=0;
00211   int lambdamax=0;
00212
00213 if (VERBOSE)
00214   fprintf(file, "[%d] flag_skip_build_gappy_blocks=%d\n", rank,
      flag_skip_build_gappy_blocks);
00215
00216   if (flag_skip_build_gappy_blocks==1) {  //no build gappy blocks strategy,
       just put zeros at gaps location
00217
00218   //compute the product using only the input Toeplitz blocks structure with
       zeros at the gaps location
00219 //to remake  stbmm(V, nrow, m, m_rowwise, tpltzblocks, nb_blocks_local,
       nb_blocks_all, id0p, local_V_size, flag_stgy);
00220
00221   }
00222   else { //build gappy blocks strategy
00223
00224   for(Tsize=i=0;i<nb_blocks_local;i++)
00225     Tsize += tpltzblocks[i].lambda;
00226
00227   for(i=0;i<nb_blocks_local;i++) {
00228     if (tpltzblocks[i].lambda>lambdamax)
00229       lambdamax = tpltzblocks[i].lambda;
00230   }
00231
00232 //compute max size possible for the gappy variables
00233   nb_blockgappy_max = nb_blocks_local+ngap;
00234   Tgappysize_max = Tsize + lambdamax*ngap;
00235
00236 //allocation of the gappy variables with max size possible
00237   tpltzblocks_gappy = (Block *) calloc(nb_blockgappy_max,sizeof(Block
      ));
00238
00239
00240 //build gappy Toeplitz block structure considering significant gaps locations,
       meaning we skip
00241 //the gaps lower than the minimum correlation distance. You can also use the
       flag_param_distmin_fixed
00242 //parameter which allows you to skip the gap lower than these value. Indeed,
       sometimes it's
00243 //better to just put somes zeros than to consider two separates blocks.
00244 //ps: This criteria could be dependant of the local lambda in futur
       impovements.
00245   int flag_param_distmin_fixed = flag_stgy.flag_param_distmin_fixed
      ;
00246   build_gappy_blocks(nrow, m, tpltzblocks, nb_blocks_local,
      nb_blocks_all, id0gap, lgap, ngap, tpltzblocks_gappy, &nb_blocks_gappy,
      flag_param_distmin_fixed);
00247
00248
00249 if (VERBOSE) {
00250     fprintf(file, "[%d] nb_blocks_gappy=%d\n", rank, nb_blocks_gappy);
00251     for(i=0;i<nb_blocks_gappy;i++)
00252       fprintf(file, "[%d] idvgappy[%d]=%d ; ngappy[%d]=%d\n", rank, i,
      tpltzblocks_gappy[i].idv, i, tpltzblocks_gappy[i].n );
00253 }
00254 //ps: we could reallocate the gappy variables to their real size. Not sure it's
       worth it.
00255
00256 //compute the product using the freshly created gappy Toeplitz blocks structure
00257 //to remake  stbmm(V, nrow, m, m_rowwise, tpltzblocks_gappy, nb_blocks_local,
       nb_blocks_all, id0p, local_V_size, flag_stgy);
00258
```

```
00259   } //end flag_skip_build_gappy_blocks==1
00260
00261
00262 //put zeros on V for the gaps location again. Indeed, some gaps are just
       replaced by zeros
00263 //in input, it's created some fakes results we need to clear after the
       computation.
00264   reset_gaps( V, id0p, local_V_size, m, nrow, m_rowwise, id0gap, lgap
      , ngap);
00265
00266
00267   return 0;
00268 }
00269
```

15.67 toeplitz_utils.c File Reference

Contains a set of utilitaries routines for Toeplitz algebra.

Functions

- int defineTpltz (Tpltz *Nm1, int64_t nrow, int m_cw, int m_rw, Block *tpltzblocks, int nb_blocks_loc, int nb_blocks_tot, int64_t idp, int local_V_size, Flag flag_stgy, MPI_Comm comm)
- int defineBlocks_avg (Block *tpltzblocks, double *T, int nb_blocks_loc, int n_block_avg, int lambda_block_avg, int64_t id0)
- int createRandomT (double *T, int Tsize)
- int createTbasic1 (double *T, int Tsize)
- int createTbasic2 (double *T, int Tsize)
- int createTbasic3 (double *T, int Tsize)
- int createTfrominvtt (double *T, int Tsize)

Variables

- int PRINT_RANK

15.67.1 Detailed Description

Contains a set of utilitaries routines for Toeplitz algebra. version 1.2b, July 2012

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Log: toeplitz∗.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.
- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Definition in file toeplitz_utils.c.

15.67.2 Function Documentation

15.67.2.1 int defineTpltz (Tpltz ∗ *Nm1,* int64_t *nrow,* int *m_cw,* int *m_rw,* Block ∗ *tpltzblocks,* int *nb_blocks_loc,* int *nb_blocks_tot,* int64_t *idp,* int *local_V_size,* Flag *flag_stgy,* MPI_Comm *comm*)

Definition at line 55 of file toeplitz_utils.c.

15.67.2.2 int defineBlocks_avg (Block ∗ *tpltzblocks,* double ∗ *T,* int *nb_blocks_loc,* int *n_block_avg,* int *lambda_block_avg,* int64_t *id0*)

Definition at line 75 of file toeplitz_utils.c.

15.67.2.3 int createRandomT (double ∗ *T,* int *Tsize*)

Definition at line 103 of file toeplitz_utils.c.

15.67.2.4 int createTbasic1 (double ∗ *T,* int *Tsize*)

Definition at line 118 of file toeplitz_utils.c.

15.67.2.5 int createTbasic2 (double ∗ *T,* int *Tsize*)

Definition at line 133 of file toeplitz_utils.c.

15.67.2.6 int createTbasic3 (double ∗ *T,* int *Tsize*)

Definition at line 155 of file toeplitz_utils.c.

15.67.2.7 int createTfrominvtt (double * T, int Tsize)

Definition at line 178 of file toeplitz_utils.c.

15.67.3 Variable Documentation

15.67.3.1 int PRINT_RANK

Definition at line 82 of file toeplitz.c.

15.68 toeplitz_utils.c

```
00001
00049 #include "toeplitz.h"
00050 extern int PRINT_RANK;
00051
00052 //set of utilitaries routines - fd@apc
00053
00054
00055 int defineTpltz( Tpltz *Nm1, int64_t nrow, int m_cw, int m_rw,
      Block *tpltzblocks, int nb_blocks_loc, int nb_blocks_tot, int64_t idp, int
       local_V_size, Flag flag_stgy, MPI_Comm comm)
00056 {
00057
00058   Nm1->nrow = nrow; //glob //recup du fichier params apres (en variables
       globales)
00059   Nm1->m_cw = m_cw; //glob
00060   Nm1->m_rw = m_rw; //glob
00061   Nm1->tpltzblocks = tpltzblocks; //toep
00062   Nm1->nb_blocks_loc = nb_blocks_loc; //toep
00063   Nm1->nb_blocks_tot = nb_blocks_tot;  //toep
00064   Nm1->idp = idp; //comput
00065   Nm1->local_V_size = local_V_size; //comput
00066   Nm1->flag_stgy = flag_stgy; //param
00067   Nm1->comm = comm; //param
00068
00069
00070   return 0;
00071 }
00072
00073
00074
00075 int defineBlocks_avg(Block *tpltzblocks, double *T, int
      nb_blocks_loc, int n_block_avg, int lambda_block_avg, int64_t id0 )
00076 {
00077
00078 int i;
00079
00080
00081   for ( i=0; i<nb_blocks_loc; i++)
00082     tpltzblocks[i].n = n_block_avg;
00083
00084   for ( i=0; i<nb_blocks_loc; i++)
00085     tpltzblocks[i].lambda = lambda_block_avg;
00086
00087   tpltzblocks[0].idv = (int64_t) (id0/n_block_avg) * n_block_avg ;
00088   for(i=1;i<nb_blocks_loc;i++)
00089     tpltzblocks[i].idv = (int64_t) tpltzblocks[i-1].idv + tpltzblocks[i-1].n
      ;
00090
00091   for( i=0; i<nb_blocks_loc; i++) {
00092     tpltzblocks[i].T_block = (T);
00093   }
00094
00095
00096   return 0;
00097 }
00098
00099
00100 //============================================
00101
00102
00103 int createRandomT(double *T, int Tsize)
00104 {
00105
00106   int i;
00107   srand (time (NULL));  //init seed
```

```
00108
00109   //input matrix definition of T
00110     for(i=0;i<Tsize;i++)
00111       T[i]= rand()/((double) RAND_MAX);
00112
00113   return 0;
00114 }
00115
00116
00117
00118 int createTbasic1(double *T, int Tsize)
00119 {
00120
00121   int i;
00122   srand (Tsize);
00123
00124   //input matrix definition of T
00125     for(i=0;i<Tsize;i++)
00126       T[i]= 1.0 + rand()/((double) RAND_MAX);
00127
00128   return 0;
00129 }
00130
00131
00132
00133 int createTbasic2(double *T, int Tsize)
00134 {
00135
00136   int i;
00137   srand (Tsize);
00138
00139   //input matrix definition of T
00140     for(i=0;i<Tsize;i++) {
00141       if (i == 0) {
00142         T[i]=10.;}
00143       else if (i == 1) {
00144         T[i]=2.;}
00145       else if (i == 2) {
00146         T[i]=3.;}
00147       else {
00148         T[i]=rand()/((double) RAND_MAX);
00149      }}
00150
00151   return 0;
00152 }
00153
00154
00155 int createTbasic3(double *T, int Tsize)
00156 {
00157
00158   int i;
00159   srand (Tsize);
00160
00161   //input matrix definition of T
00162     for(i=0;i<Tsize;i++) {
00163       if (i == 0) {
00164         T[i]=2.;}
00165       else if (i == 1) {
00166         T[i]=-1.;}
00167       else if (i == 2) {
00168         T[i]=0.;}
00169       else {
00170         T[i]=0.;//rand()/((double) RAND_MAX);
00171      }}
00172
00173
00174   return 0;
00175 }
00176
00177
00178 int createTfrominvtt(double *T, int Tsize)
00179 {
00180
00181   int i;
00182
00183 //#include "invtt_params.h"
00184
00185   double *invtt;
00186
00187   T = invtt;
00188 //  createinvtt(invtt);
00189
00190
00191   return 0;
00192 }
```

15.69 toeplitz_wizard.c File Reference

easy-to-use and "all-in-one" wizard routines for the Toeplitz module

Functions

- int stbmmProd (Tpltz Nm1, double *V)

  Performs the product of a Toeplitz matrix by a general matrix either sequentially or using MPI. The complexity is hidden in the input structure, which needs to be defined by a user.
- int gstbmmProd (Tpltz Nm1, double *V, Gap Gaps)

15.69.1 Detailed Description

easy-to-use and "all-in-one" wizard routines for the Toeplitz module

Author

Frederic Dauvergne

Project: Midapack library, ANR MIDAS'09 - Toeplitz Algebra module Purpose: Provide Toeplitz algebra tools suitable for Cosmic Microwave Background (CMB) data analysis.

Note

Log: toeplitz*.c

Revision 1.0b 2012/05/07 Frederic Dauvergne (APC) Official release 1.0beta. The first installement of the library is the Toeplitz algebra module.

Revision 1.1b 2012/07/- Frederic Dauvergne (APC)

- mpi_stbmm allows now rowi-wise order per process datas and no-blocking communications.
- OMP improvment for optimal cpu time.
- bug fixed for OMP in the stmm_basic routine.
- distcorrmin is used to communicate only lambda-1 datas when it is needed.
- new reshaping routines using transformation functions in stmm. Thus, only one copy at most is needed.
- tpltz_init improvement using define_nfft and define_blocksize routines.
- add Block struture to define each Toeplitz block.

- add Flag structure and preprocessing parameters to define the computational strategy. All the flag parameters are then available directly from the API.

Revision 1.2b 2012/11/30 Frederic Dauvergne (APC)

- extend the mpi product routine to rowwise order data distribution. This is now allowing tree kinds of distribution.
- add int64 for some variables to extend the global volume of data you can use.
- Openmp improvments.
- Add toeplitz_wizard.c, which contains a set of easy to use routines with defined structures.

Definition in file toeplitz_wizard.c.

15.69.2 Function Documentation

15.69.2.1 int gstbmmProd (Tpltz *Nm1*, double * *V*, Gap *Gaps*)

Definition at line 83 of file toeplitz_wizard.c.

15.70 toeplitz_wizard.c

```
00001
00057 #include "toeplitz.h"
00058
00060
00063 int stbmmProd( Tpltz Nm1, double *V)
00064 {
00065
00066 #ifdef W_MPI
00067
00068   mpi_stbmm(&V, Nm1.nrow, Nm1.m_cw, Nm1.m_rw, Nm1.
      tpltzblocks, Nm1.nb_blocks_loc, Nm1.nb_blocks_tot
      , Nm1.idp, Nm1.local_V_size, Nm1.flag_stgy, Nm1.comm
      );
00069
00070 #else
00071
00072 //int stbmm(double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int
       nb_blocks, int64_t idp, int local_V_size, Flag flag_stgy)
00073
00074   stbmm(&V, Nm1.nrow, Nm1.m_cw, Nm1.m_rw, Nm1.tpltzblocks
      , Nm1.nb_blocks_loc, Nm1.idp, Nm1.local_V_size, Nm1.
      flag_stgy);
00075
00076 #endif
00077
00078   return 0;
00079 }
00080
00081
00082
00083 int gstbmmProd( Tpltz Nm1, double *V, Gap Gaps)
00084 {
00085
00086 #ifdef W_MPI
00087
00088   mpi_gstbmm(&V, Nm1.nrow, Nm1.m_cw, Nm1.m_rw, Nm1.
      tpltzblocks, Nm1.nb_blocks_loc, Nm1.nb_blocks_tot
      , Nm1.idp, Nm1.local_V_size, Gaps.id0gap, Gaps.lgap,
      Gaps.ngap, Nm1.flag_stgy, Nm1.comm);
00089
00090 #else
00091
00092 //int gstbmm0(double **V, int nrow, int m_cw, int m_rw, Block *tpltzblocks, int
       nb_blocks, int64_t idp, int local_V_size, int *id0gap, int *lgap, int ngap,
       Flag flag_stgy)
00093
00094   gstbmm(&V, Nm1.nrow, Nm1.m_cw, Nm1.m_rw, Nm1.tpltzblocks
      , Nm1.nb_blocks_loc, Nm1.idp, Nm1.local_V_size, Gaps
      .id0gap, Gaps.lgap, Gaps.ngap, Nm1.flag_stgy);
```

```
00095
00096 #endif
00097
00098   return 0;
00099 }
00100
00101
```

15.71 truebutterfly.c File Reference

Implementation of routines for butterfly-like communication scheme, with classic pair wise butterfly scheme.

Functions

- int truebutterfly_init (int *indices, int count, int **R, int *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm comm)

  Initialize tables for butterfly-like communication scheme (true means pair wise) This routine set up needed tables for the butterfly communication scheme. Sending and receiving tabs should be well allocated(at least size of number of steps in butterfly scheme). Double pointer are partially allocated, the last allocation is performed inside the routine. com_indices and com_count are also allocated inside the routine, thus they are passing by reference. They represent indices which have to be communicated an their number. Algotithm is based 2 parts. The first one identify intersection between processors indices, using 3 successives butterfly communication schemes : bottom up, top down, and top down again. The second part works locally to build sets of indices to communicate.
- double truebutterfly_reduce (int **R, int *nR, int nRmax, int **S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm)

  Perform a sparse sum reduction (or mapped reduction) using a butterfly-like communication scheme.

15.71.1 Detailed Description

Implementation of routines for butterfly-like communication scheme, with classic pair wise butterfly scheme.

Note

ACKNOWLEDGMENT: This work has been supported in part by the French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009).

Author

Radek Stompor

Date

October 2012

Definition in file truebutterfly.c.

15.72 truebutterfly.c

```
00001
00009 #ifdef W_MPI
00010 #include <mpi.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014
00037 int truebutterfly_init(int *indices, int count, int **R, int
      *nR, int **S, int *nS, int **com_indices, int *com_count, int steps, MPI_Comm
      comm){
00038
00039   int i, k, p2k, p2k1;
00040   int rank, size, rk, sk;
00041   int tag;
00042   MPI_Request s_request, r_request;
00043   int nbuf, *buf;
00044   int **I, *nI;
00045   int **J, *nJ;
00046
00047   MPI_Comm_size(comm, &size);
00048   MPI_Comm_rank(comm, &rank);
00049
00050   I = (int **) malloc(steps * sizeof(int*));
00051   nI = (int *) malloc(steps * sizeof(int));
00052   tag=0;
00053   p2k=size/2;
00054   p2k1=2*p2k;
00055
00056   for(k=0; k<steps; k++){               //butterfly first pass : bottom up
       (fill tabs nI and I)
00057
00058     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00059
00060     if(k==0){                                          //S^0 := A
00061       nS[k] = count;
00062       S[k] = (int *) malloc(nS[k] * sizeof(int));
00063       memcpy( S[k], indices, nS[k]*sizeof(int));
00064     }
00065     else{                                              //S^k := S^{k-1} \cup
       R^{k-1}
00066       nS[k] = card_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k]);
00067       S[k] = (int *) malloc(nS[k] * sizeof(int));
00068       set_or(S[k-1], nS[k-1], I[steps-k], nI[steps-k], S[k]);
00069     }
00070
00071     MPI_Irecv(&nI[steps-k-1], 1, MPI_INT, rk, tag, comm, &r_request);   //
      receive number of indices
00072     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);           //send
       number of indices
00073     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00074     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00075
00076     I[steps-k-1]= (int *) malloc(nI[steps-k-1] * sizeof(int));
00077
00078     tag++;
00079     MPI_Irecv(I[steps-k-1], nI[steps-k-1], MPI_INT, rk, tag, comm, &r_request);
      //receive indices
00080     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request);
      //send indices
00081     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00082     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00083
00084     p2k/=2;
00085     p2k1/=2;
00086     tag++;
00087   }
00088
00089   J = (int **) malloc(steps * sizeof(int*));
00090   nJ = (int *) malloc(steps * sizeof(int));
00091
00092   tag=0;
00093   p2k=1;
00094   p2k1=p2k*2;
00095   for(k=0; k<steps; k++){               //buuterfly second pass : top down
       (fill tabs nJ and J)
00096     free(S[k]);
00097
00098     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00099
00100     if(k==0){
00101       nJ[k] = count;
00102       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00103       memcpy( J[k], indices, nJ[k]*sizeof(int));
00104     }
```

```
00105     else{
00106       nJ[k] = card_or(J[k-1], nJ[k-1], R[k-1], nR[k-1]);
00107       J[k] = (int *) malloc(nJ[k] * sizeof(int));
00108       set_or(J[k-1], nJ[k-1], R[k-1], nR[k-1], J[k]);  //J^k=R^k-1 \cup
      J^k-1
00109       free(R[k-1]);
00110     }
00111     if(k!=steps-1){
00112     MPI_Irecv(&nR[k], 1, MPI_INT, rk, tag, comm, &r_request);
00113     MPI_Isend(&nJ[k], 1, MPI_INT, sk, tag, comm, &s_request);
00114     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00115     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00116
00117     R[k]= (int *) malloc( nR[k] * sizeof(int));
00118     tag++;
00119
00120     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request);
00121     MPI_Isend(J[k], nJ[k], MPI_INT, sk, tag, comm, &s_request);
00122     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00123     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00124     }
00125     p2k*=2;
00126     p2k1*=2;
00127     tag++;
00128   }
00129
00130
00131   tag=0;
00132   p2k=1;
00133   p2k1=p2k*2;
00134   for(k=0; k<steps; k++){               //butterfly last pass : know that
      Sending tab is S = I \cap J, so send S and we'll get R
00135
00136     if( rank%p2k1 < p2k) sk=rk=rank+p2k; else sk=rk=rank-p2k;
00137
00138     nS[k] = card_and(I[k], nI[k], J[k], nJ[k]);
00139     S[k] = (int *) malloc(nJ[k] *sizeof(int));
00140     set_and( I[k], nI[k], J[k], nJ[k], S[k]);    //S^k=I^k \cap J^k
00141
00142     free(I[k]);
00143     free(J[k]);
00144
00145     MPI_Irecv(&nR[k],1, MPI_INT, rk, tag, comm, &r_request);    //receive size
00146     MPI_Isend(&nS[k], 1, MPI_INT, sk, tag, comm, &s_request);   //send size
00147     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00148     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00149
00150     R[k]= (int *) malloc( nR[k] * sizeof(int));
00151     tag++;
00152
00153     MPI_Irecv(R[k], nR[k], MPI_INT, rk, tag, comm, &r_request); //receive
      indices
00154     MPI_Isend(S[k], nS[k], MPI_INT, sk, tag, comm, &s_request); //send indices
00155     MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00156     MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00157
00158     p2k*=2;
00159     p2k1*=2;
00160     tag++;
00161   }
00162
00163   //Now we work locally
00164   int **USR, *nUSR, **U, *nU;
00165
00166   USR = (int **) malloc(steps*sizeof(int *));
00167   nUSR = (int *) malloc(steps*sizeof(int));
00168   U = (int **) malloc(steps*sizeof(int *));
00169   nU = (int *) malloc(steps*sizeof(int));
00170
00171   for(k=0; k<steps; k++){
00172     nUSR[k] = card_or(S[k], nS[k], R[k], nR[k]);
00173     USR[k] = (int *) malloc(nUSR[k]*sizeof(int));
00174     set_or(S[k], nS[k], R[k], nR[k], USR[k]);
00175   }
00176   for(k=0; k<steps; k++){
00177     if(k==0){
00178       nU[k]=nUSR[k];
00179       U[k] = (int *) malloc(nU[k] * sizeof(int));
00180       memcpy( U[k], USR[k], nU[k]*sizeof(int));
00181     }
00182     else{
00183       nU[k] = card_or(U[k-1], nU[k-1], USR[k], nUSR[k]);
00184       U[k] = (int *) malloc(nU[k]*sizeof(int *));
00185       set_or(U[k-1], nU[k-1], USR[k], nUSR[k], U[k]);
00186     }
00187   }
00188   *com_count=nU[steps-1];
```

```
00189   *com_indices = (int *) malloc(*com_count * sizeof(int));
00190   memcpy(*com_indices, U[steps-1], *com_count * sizeof(int));
00191   //====================================================================
00192
00193   for(k=0; k<steps; k++){
00194     subset2map(*com_indices, *com_count, S[k], nS[k]);
00195     subset2map(*com_indices, *com_count, R[k], nR[k]);
00196   }
00197   free(USR);
00198   free(U);
00199
00200  return 0;
00201 }
00202
00203
00216 int truebutterfly_reduce(int **R, int *nR, int nRmax, int *
      *S, int *nS, int nSmax, double *val, int steps, MPI_Comm comm){
00217 //  double st, t;
00218 //  t=0.0;
00219   int k, p2k, p2k1, tag;
00220   int rank, size, rk, sk;
00221   MPI_Status status;
00222   MPI_Request s_request, r_request;
00223   double *sbuf, *rbuf;
00224
00225   MPI_Comm_size(comm, &size);
00226   MPI_Comm_rank(comm, &rank);
00227
00228   sbuf = (double *) malloc(nSmax * sizeof(double));
00229   rbuf = (double *) malloc(nRmax * sizeof(double));
00230   tag=0;
00231   p2k=1;
00232   p2k1=p2k*2;
00233
00234   for(k=0; k<steps; k++){
00235
00236     if( rank%p2k1 < p2k){
00237
00238       sk=rk=rank+p2k;
00239
00240    //   st=MPI_Wtime();
00241
00242       //      MPI_Sendrecv(sbuf, nS[k], MPI_DOUBLE, sk, tag, rbuf, nR[k],
      MPI_DOUBLE, rk, tag, comm, &status);
00243
00244       m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00245       MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00246       MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00247
00248       MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00249       MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00250       s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
00251
00252
00253      // t=t+MPI_Wtime()-st;
00254
00255     } else {
00256
00257       sk=rk=rank-p2k;
00258
00259      // st=MPI_Wtime();
00260
00261       MPI_Irecv(rbuf, nR[k], MPI_DOUBLE, rk, tag, comm, &r_request);
00262       m2s(val, sbuf, S[k], nS[k]); //fill the sending buffer
00263       MPI_Isend(sbuf, nS[k], MPI_DOUBLE, sk, tag, comm, &s_request);
00264
00265       MPI_Wait(&r_request, MPI_STATUS_IGNORE);
00266       s2m_sum(val, rbuf, R[k], nR[k]); //sum receive buffer into values
00267
00268       MPI_Wait(&s_request, MPI_STATUS_IGNORE);
00269
00270       // MPI_Sendrecv(sbuf, nS[k], MPI_DOUBLE, sk, tag, rbuf, nR[k],
      MPI_DOUBLE, rk, tag, comm, &status);
00271
00272    //   t=t+MPI_Wtime()-st;
00273
00274     }
00275
00276     p2k*=2;
00277     p2k1*=2;
00278     tag++;
00279
00280   }
00281   free(sbuf);
00282   free(rbuf);
00283   return 0;
00284 }
```

```
00285
00286 #endif
00287
00288
```