PanguLU Users' Guide v4.2.0

December 13, 2024

PanguLU Developer Team

Contents

1 Introduction

This chapter includes an introduction to PanguLU and its code structure.

1.1 Introduction to PanguLU

PanguLU [1] is an open source software package for solving a linear system $Ax = b$ on heterogeneous distributed platforms. The library is written in pure C, and exploits parallelism from MPI, OpenMP and CUDA. The sparse LU factorisation algorithm used in PanguLU splits the sparse matrix into multiple equally-sized sparse matrix blocks and computes them by using sparse basic linear algebra subprograms (sparse BLAS). PanguLU also uses a synchronisation-free communication strategy to reduce the overall latency overhead, and a variety of block-wise sparse kernels have been adaptively called to improve efficiency on CPUs and GPUs. Currently, PanguLU supports data types of single, double, and complex floating points.

Pangu [2] is a primordial being and creation figure in Chinese mythology who separated heaven and earth. This is very like that an LU factorisation method decomposes a matrix A into two triangular matrices L and U. This is why we named our work PanguLU and designed the logo shown in the cover page of this guide.

Our team at the Super Scientific Software Laboratory (SSSLab, https://www.ssslab.cn/), China University of Petroleum-Beijing, is constantly optimising and updating PanguLU.

1.2 Code Structure

Field	**Description**
`PanguLU/README.md`	Instructions on installation
`PanguLU/src`	C and CUDA source code, to be compiled into libpangulu.a and libpangulu.so
`PanguLU/examples`	Example code
`PanguLU/include`	Contains headers archive libpangulu.a and libpangulu.so
`PanguLU/lib`	Contains library archive libpangulu.a and libpangulu.so
`PanguLU/Makefile`	Top-level Makefile that does installation and testing
`PanguLU/make.inc`	Compiler, compiler flags included in all Makefiles

2 Installation

This chapter shows you how to install and configure PanguLU.

2.1 Compilers

Before you install the PanguLU software package by using the ‘make’ command, you need to have the following compilers installed on your system:

- **MPI** (Tested version: OpenMPI-4.1.2)
  Download link: https://www.open-mpi.org/software/ompi
- **CUDA** (Tested version: CUDA-12.2)
  Download link: https://developer.nvidia.com/cuda-downloads

2.2 Paths to Dependent Packages

Before compiling, you need to specify the library and header file paths of OpenBLAS and METIS [3] in the make.inc file. If the METIS version you are using depends on GKlib, please add the path of gklib.h in METISFLAGS.

You can refer to the following content for configuration:

```
COMPILE_LEVEL = -O3

#0201000,GPU_CUDA
CUDA_PATH = /path/to/cuda
CUDA_INC = -I$(CUDA_PATH)/include
CUDA_LIB = -L$(CUDA_PATH)/lib64 -lcudart -lcusparse
NVCC = nvcc $(COMPILE_LEVEL)
NVCCFLAGS = $(PANGULU_FLAGS) -w -Xptxas -dlcm=cg -gencode=arch=compute_61,co
de=sm_61 -gencode=arch=compute_61,code=compute_61 $(CUDA_INC) $(CUDA_LIB)

#general
CC = gcc $(COMPILE_LEVEL)
MPICC = mpicc $(COMPILE_LEVEL)
OPENBLAS_INC = -I/path/to/OpenBLAS/include
OPENBLAS_LIB = -L/path/to/OpenBLAS/lib -lopenblas
MPICCFLAGS = $(OPENBLAS_INC) $(CUDA_INC) $(OPENBLAS_LIB) -fopenmp -lpthread -
lm
MPICCLINK = $(OPENBLAS_LIB)
METISFLAGS = -I/path/to/GKlib/include -I/path/to/METIS/include
PANGULU_FLAGS = -DPANGULU_LOG_INFO -DCALCULATE_TYPE_R64 -DPANGULU_MC64 -DMET
IS #-DGPU_OPEN -DHT_IS_OPEN
```

2.3 Compute Parameters and Options

PanguLU supports matrix calculations using real and complex types, and can be executed on CPU or GPU, while supporting the hyperthreading function of hardware devices.

The following are the specific configuration steps:

- **Configuration options for real and complex types and precisions**
  To configure real and complex types, as well as single and double precisions, adjust the PANGULU_FLAGS in the make.inc file by including one of the following options:

	Real	Complex
Single Precision	-DCALCULATE_TYPE_R32	-DCALCULATE_TYPE_CR32
Double Precision	-DCALCULATE_TYPE_R64	-DCALCULATE_TYPE_CR64

- **Enable hyperthreading function**
  If your machine has enabled the hyperthreading feature, adding the following option to PANGULU_FLAGS in the make.inc file may improve performance:

```
-DHT_IS_OPEN
```

  Note: If the option is enabled, in the numeric decomposition phase, only even numbered cores are bound when the thread is bound to the CPU. If user can disable the hyperthreading, please try to disable it and not use this option.
- **Use GPU for calculation**
  To use GPU, please modify PANGULU_FLAGS in the make.inc file and add the following option:

```
-DGPU_OPEN
```

  Note: Current GPU version does not support the calculation of complex numbers.
- **Call METIS to reorder the matrix**
  If you want to use METIS, modify PANGULU_FLAGS in the make.inc file and add the following option:

```
-DMETIS
```

- **Call MC64 [4] to reorder the matrix**
  If you want to use MC64 to improve numerical stability, modify PANGULU_FLAGS in the make.inc file and add the following option:

```
-DPANGULU_MC64
```

  Note: Our implementation of MC64 currently does not support the calculation of complex numbers.
- **Output information selection**
  If you want to output execution information to stdout, modify PANGULU_FLAGS in the make.inc file and add the following options:

```
-DPANGULU_LOG_INFO
```

If you just want to output warning and error messages, modify PANGULU_FLAGS in the make.inc file and add the following option:

```
-DPANGULU_LOG_WARNING
```

If you only want to output error messages that cause PanguLU to fail to run, modify PANGULU_FLAGS in the make.inc file and add the following option:

```
-DPANGULU_LOG_ERROR
```

After you have completed the desired configuration in the make.inc file, you can use the ‘make’ command to automatically install and compile PanguLU. If you need to change the configuration options later, use the ‘make update’ command to recompile after the change.

3 Interface

This chapter describes the main function interfaces of the PanguLU library, covering the steps from matrix initialization to the solution process. The parameters and return values of each function are described.

3.1 pangulu_init()

- **Description**
  pangulu_init() is used to initialize a handle of PanguLU, and to distribute the matrix to all MPI ranks. Please note that if you need to factorise more than one different matrices, call pangulu_finalize() after completing the solution of one matrix, and then use pangulu_init() to to initialize the next matrix.
- **Function**

```
void pangulu_init(
  int pangulu_n,
  long long pangulu_nnz,
  long *csr_rowptr,
  int *csr_colidx,
  pangulu_calculate_type *csr_value,
  pangulu_init_options *init_options,
  void **pangulu_handle);
```

- **Parameters**
  pangulu_n: Specifies the number of rows in the CSR matrix.
  pangulu_nnz: Specifies the total number of non-zero elements in the CSR matrix.
  csr_rowptr: Points to an array that stores pointers to rows of the CSR matrix.
  csr_colidx: Points to an array that stores indices to columns of the CSR matrix.
  csr_value: Points to an array that stores the values of non-zero elements of the CSR matrix.

init_options: Pointer to a pangulu_init_options structure containing initialization options: nthread (number of threads to use) and nb (block size).
pangulu_handle: On return, contains a handle pointer to the library's internal state.

3.2 pangulu_gstrf()

- **Description**
  pangulu_gstrf() performs distribute sparse LU factorisation. Note that you should call pangulu_init() before calling pangulu_gstrf() to create a handle of PanguLU.
- **Function**

```
void pangulu_gstrf(
  pangulu_gstrf_options *gstrf_options,
  void **pangulu_handle);
```

- **Parameters**
  gstrf_options: Pointer to pangulu_gstrf_options structure.
  pangulu_handle: Pointer to the library internal status handle returned from initialization.

3.3 pangulu_gstrs()

- **Description**
  pangulu_gstrs() solves linear equation with factorised L and U, and right-hand side vector b. Note that you should call pangulu_gstrf() before calling pangulu_gstrs() to ensure that L and U are available.
- **Function**

```
void pangulu_gstrs(
  pangulu_calculate_type *rhs,
  pangulu_gstrs_options *gstrs_options,
  void** pangulu_handle);
```

- **Parameters**
  rhs: Pointer to the right-hand side vector.
  gstrs_options: Pointer to the pangulu_gstrs_options structure.
  pangulu_handle: Pointer to the library internal state handle returned from initialization.

3.4 pangulu_gssv()

- **Description**
  pangulu_gssv() solves the linear equation with *A* and right-hand size *b*. This function is equivalent to calling pangulu_gstrs() after pangulu_gstrf().
- **Function**

```
void pangulu_gssv(
  pangulu_calculate_type *rhs,
  pangulu_gstrf_options *gstrf_options,
  pangulu_gstrs_options *gstrs_options,
  void **pangulu_handle);
```

- **Parameters**
  rhs: Pointer to the right-hand side vector.
  gstrf_options: Pointer to a pangulu_gstrf_options structure.
  gstrs_options: Pointer to a pangulu_gstrs_options structure.
  pangulu_handle: Pointer to the library internal status handle returned from initialization.

3.5 pangulu_finalize()

- **Description**
  pangulu_finalize() is used to destroy a handle of PanguLU.
- **Function**

```
void pangulu_finalize(
  void **pangulu_handle);
```

- **Parameters**
  pangulu_handle: Pointer to the library internal state handle returned on initialization.

4 Execution of an Example

4.1 example.c

The following is the content of example.c in the examples folder calling the PanguLU interfaces:

```
typedef unsigned long long int sparse_pointer_t;
#define MPI_SPARSE_POINTER_T MPI_UNSIGNED_LONG_LONG
#define FMT_SPARSE_POINTER_T "%llu"

typedef unsigned int sparse_index_t;
#define MPI_SPARSE_INDEX_T MPI_UNSIGNED
#define FMT_SPARSE_INDEX_T "%u"

#if defined(CALCULATE_TYPE_R64)
typedef double sparse_value_t;
#elif defined(CALCULATE_TYPE_R32)
typedef float sparse_value_t;
#elif defined(CALCULATE_TYPE_CR64)
typedef double _Complex sparse_value_t;
typedef double sparse_value_real_t;
#define COMPLEX_MTX
#elif defined(CALCULATE_TYPE_CR32)
typedef float _Complex sparse_value_t;
typedef float sparse_value_real_t;
#define COMPLEX_MTX
#else
#error[PanguLU Compile Error] Unknown value type. Set -DCALCULATE_TYPE_CR64
or -DCALCULATE_TYPE_R64 or -DCALCULATE_TYPE_CR32 or -DCALCULATE_TYPE_R32 in
compile command line.
#endif
```

```
#include "../include/pangulu.h"
#include <sys/resource.h>
#include <getopt.h>
#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include "mmio_highlevel.h"

#ifdef COMPLEX_MTX
sparse_value_real_t complex_fabs(sparse_value_t x)
{
    return sqrt(__real__(x) * __real__(x) + __imag__(x) * __imag__(x));
}

sparse_value_t complex_sqrt(sparse_value_t x)
{
    sparse_value_t y;
    __real__(y) = sqrt(complex_fabs(x) + __real__(x)) / sqrt(2);
    __imag__(y) = (sqrt(complex_fabs(x) - __real__(x)) / sqrt(2)) * (__imag
    __(x) > 0 ? 1 : __imag__(x) == 0 ? 0
                                     : -1);
    return y;
}
#endif

void read_command_params(int argc, char **argv, char *mtx_name,
                         char *rhs_name, int *nb)
{
    int c;
    extern char *optarg;
    while ((c = getopt(argc, argv, "nb:f:r:")) != EOF)
    {
        switch (c)
        {
        case 'b':
            *nb = atoi(optarg);
            continue;
        case 'f':
            strcpy(mtx_name, optarg);
            continue;
        case 'r':
            strcpy(rhs_name, optarg);
            continue;
        }
    }

    if ((nb) == 0)
    {
        printf("Error : nb is 0\n");
        exit(1);
    }
}

int main(int ARGC, char **ARGV)
{
    // Step 1: Create varibles, initialize MPI environment.
    int provided = 0;
    int rank = 0, size = 0;
    int nb = 0;
    MPI_Init_thread(&ARGC, &ARGV, MPI_THREAD_MULTIPLE, &provided);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    sparse_index_t m = 0, n = 0, is_sym = 0;
    sparse_pointer_t nnz;
    sparse_pointer_t *rowptr = NULL;
    sparse_index_t *colidx = NULL;
    sparse_value_t *value = NULL;
    sparse_value_t *sol = NULL;
    sparse_value_t *rhs = NULL;

    // Step 2: Read matrix and rhs vectors.
    if (rank == 0)
    {
        char mtx_name[200] = {'\0'};
        char rhs_name[200] = {'\0'};
        read_command_params(ARGC, ARGV, mtx_name, rhs_name, &nb);

        printf("Reading matrix %s\n", mtx_name);
        mmio_info(&m, &n, &nnz, &is_sym, mtx_name);
        rowptr = (sparse_pointer_t *)malloc(sizeof(sparse_pointer_t) * (n + 1));
        colidx = (sparse_index_t *)malloc(sizeof(sparse_index_t) * nnz);
        value = (sparse_value_t *)malloc(sizeof(sparse_value_t) * nnz);
        mmio_data_csr(rowptr, colidx, value, mtx_name);
        printf("Read mtx done.\n");

        sol = (sparse_value_t *)malloc(sizeof(sparse_value_t) * n);
        rhs = (sparse_value_t *)malloc(sizeof(sparse_value_t) * n);
        for (int i = 0; i < n; i++)
        {
            rhs[i] = 0;
            for (sparse_pointer_t j = rowptr[i]; j < rowptr[i + 1]; j++)
            {
                rhs[i] += value[j];
            }
            sol[i] = rhs[i];
        }
        printf("Generate rhs done.\n");
    }
    MPI_Bcast(&n, 1, MPI_SPARSE_INDEX_T, 0, MPI_COMM_WORLD);
    MPI_Bcast(&nb, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    // Step 3: Initialize PanguLU solver.
    pangulu_init_options init_options;
    init_options.nb = nb;
    init_options.nthread = 20;
    void *pangulu_handle;
    pangulu_init(n, nnz, rowptr, colidx, value, &init_options, &pangulu_handle);

    // Step 4: Execute LU factorisation.
    pangulu_gstrf_options gstrf_options;
    pangulu_gstrf(&gstrf_options, &pangulu_handle);

    // Step 5: Execute triangle solve using factorise results.
    pangulu_gstrs_options gstrs_options;
    pangulu_gstrs(sol, &gstrs_options, &pangulu_handle);
    MPI_Barrier(MPI_COMM_WORLD);

    // Step 6: Check the answer.
    sparse_value_t *rhs_computed;
    if (rank == 0)
    {
        // Step 6.1: Calculate rhs_computed = A * x.
        rhs_computed = (sparse_value_t *)malloc(sizeof(sparse_value_t) * n);
        for (int i = 0; i < n; i++)
        {
            rhs_computed[i] = 0.0;
            sparse_value_t c = 0.0;
            for (sparse_pointer_t j = rowptr[i]; j < rowptr[i + 1]; j++)
            {
                sparse_value_t num = value[j] * sol[colidx[j]];
                sparse_value_t z = num - c;
                sparse_value_t t = rhs_computed[i] + z;
                c = (t - rhs_computed[i]) - z;
                rhs_computed[i] = t;
            }
        }

        // Step 6.2: Calculate residual residual = rhs_comuted - rhs.
        sparse_value_t *residual = rhs_computed;
        for (int i = 0; i < n; i++)
        {
            residual[i] = rhs_computed[i] - rhs[i];
        }

        sparse_value_t sum, c;
        // Step 6.3: Calculte norm2 of residual.
        sum = 0.0;
        c = 0.0;
        for (int i = 0; i < n; i++)
        {
            sparse_value_t num = residual[i] * residual[i];
            sparse_value_t z = num - c;
            sparse_value_t t = sum + z;
            c = (t - sum) - z;
            sum = t;
        }
#ifdef COMPLEX_MTX
        sparse_value_real_t residual_norm2 = complex_fabs(complex_sqrt(sum));
#else
```

```
        sparse_value_t residual_norm2 = sqrt(sum);
#endif

        // Step 6.4: Calculte norm2 of original rhs.
        sum = 0.0;
        c = 0.0;
        for (int i = 0; i < n; i++)
        {
            sparse_value_t num = rhs[i] * rhs[i];
            sparse_value_t z = num - c;
            sparse_value_t t = sum + z;
            c = (t - sum) - z;
            sum = t;
        }
#ifdef COMPLEX_MTX
        sparse_value_real_t rhs_norm2 = complex_fabs(complex_sqrt(sum));
#else
        sparse_value_t rhs_norm2 = sqrt(sum);
#endif

        // Step 6.5: Calculate relative residual.
        double relative_residual = residual_norm2 / rhs_norm2;
        printf("|| Ax - b || / || b || = %le\n", relative_residual);
    }

    // Step 7: Clean and finalize.
    pangulu_finalize(&pangulu_handle);
    if (rank == 0)
    {
        free(rowptr);
        free(colidx);
        free(value);
        free(sol);
        free(rhs);
        free(rhs_computed);
    }
    MPI_Finalize();
}
```

4.2 Execution

After installing PanguLU, the example.c file is automatically compiled as part of the library. Subsequently, the following instructions can be referenced for execution:

```
mpirun -np process_count ./pangulu_example.elf -nb block_size -f path_to_mtx
```

process_count: MPI process number to launch PanguLU;
block_size: Rank of each non-zero block;
path_to_mtx : The matrix name in mtx format.
You can also use the run.sh command:

```
bash ./run.sh path_to_mtx block_size process_count
```

In the following example, four MPI processes are used, the NB_size is 4, the matrix file is Trefethen_20b.mtx with the id 2203 in the SuiteSparse Matrix Collection [5]:

```
mpirun -np 4 ./pangulu_example.elf -nb 4 -f Trefethen_20b.mtx
```

or use the run.sh in this way:

```
./run.sh Trefethen_20b.mtx 4 4
```

4.3 Output

If the execution above runs successfully, you will get an output similar to the following:

```
Reading matrix Trefethen_20b.mtx
Read mtx done.
Generate rhs done.
[PanguLU Info] n=19 nnz=147 nb=5 mpi_process=8 preprocessing_thread=20 METIS:i64
[PanguLU Info] 1.1 PanguLU MC64 generate the perm time is 0.000038 s.
[PanguLU Info] 1.2 PanguLU MC64 reordering time is 0.001803 s.
[PanguLU Info] 2.1 PanguLU METIS add_diagonal_element time is 0.000003 s.
[PanguLU Info] 2.2 PanguLU METIS transpose time is 0.000031 s.
[PanguLU Info] 2.3 PanguLU METIS get_graph_struct.other_code time is 0.000007 s.
[PanguLU Info] 2 PanguLU METIS get_graph_struct time is 0.000062 s.
[PanguLU Info] 3.1 PanguLU METIS generates the perm time is 0.000330 s.
[PanguLU Info] 3.2 PanguLU METIS reordering time is 0.000048 s.
[PanguLU Info] Reordering time is 0.002359 s.
[PanguLU Info] 4 PanguLU A+AT (before symbolic) time is 0.000030 s.
[PanguLU Info] Symbolic nonzero count is 261.
[PanguLU Info] 5 PanguLU symbolic time is 0.000017 s.
[PanguLU Info] Symbolic factorization time is 0.000063 s.
[PanguLU Info] 6 PanguLU transpose reordered matrix time is 0.000005 s.
[PanguLU Info] 7 PanguLU generate full symbolic matrix time is 0.000026 s.
[PanguLU Info] Preprocessing time is 0.182785 s.
[PanguLU Info] Numeric factorization time is 0.011918 s.
[PanguLU Info] Solving time is 0.001291 s.
|| Ax - b || / || b || = 1.919634e-16
```

5 History Versions

Version 4.2.0 (Dec. 13, 2024)

- Updated preprocessing phase to distributed data structure.

Version 4.1.0 (Sep. 01, 2024)

- Optimized memory usage of numeric factorisation and solving.
- Added parallel building support.

Version 4.0.0 (Jul. 24, 2024)

- Optimized user interfaces of solver routines.
- Optimized performance of numeric factorisation phase on CPU platforms.
- Added support on complex matrix solving.
- Optimized pre-processing performance.

Version 3.5.0 (Aug. 06, 2023)

- Updated the pre-processing phase with OpenMP.
- Updated the compilation method, compilling libpangulu.so and libpangulu.a at the same time.
- Updated timing for the reordering phase, the symbolic factorisation phase, and the pre-processing phase.
- Computed GFLOPS for the numeric factorisation phase.

Version 3.0.0 (Apr. 02, 2023)

- Used an adaptive method for selecting sparse BLAS in the numeric factorisation phase.
- Added the reordering phase.
- Added the symbolic factorisation phase.
- Added the MC64 algorithm in the reordering phase.
- Added an interface for 64-bit METIS package in the reordering phase.

Version 2.0.0 (Jul. 22, 2022)

- Used a synchronisation-free scheduling strategy in the numeric factorisation

phase.
- Updated the MPI communication method in the numeric factorisation phase.
- Added single precision in the numeric factorisation phase.

Version 1.0.0 (Oct. 19, 2021)

- Used a rule-based 2D LU factorisation scheduling strategy.
- Used sparse BLAS for floating point calculations on GPUs.
- Added the pre-processing phase.
- Added the numeric factorisation phase.
- Added the triangular solve phase.

6 License

PanguLU uses the GNU Affero General Public License 3.0, details of which can be found at the following link:
https://github.com/SuperScientificSoftwareLaboratory/PanguLU?tab=AGPL-3.0-1-ov-file

7 Contributors

The project is leaded by Prof. Weifeng Liu and Associate Prof. Zhou Jin. Active contributors are Yida Li, Yiduo Niu, Siwei Zhang, Zhengye Ye, and Yang Du.

If you have any questions on PanguLU, please contact the major developer Yida Li by emailing yida.li@student.cup.edu.cn.

Before the year 2024, the contributors include Xu Fu, Bingbin Zhang, Tengcheng Wang, Wenhao Li, Yuechen Lu, Enxin Yi, Jianqi Zhao, Xiaohan Geng, Fangying Li, and Jingwen Zhang.

8 Citing PanguLU

```
@inproceedings{10.1145/3581784.3607050,
author = {Fu, Xu and Zhang, Bingbin and Wang, Tengcheng and Li,
Wenhao and Lu, Yuechen and Yi, Enxin and Zhao, Jianqi and Geng, Xiaohan
and Li, Fangying and Zhang, Jingwen and Jin, Zhou and Liu, Weifeng},
title = {PanguLU: A Scalable Regular Two-Dimensional Block-Cyclic
Sparse Direct Solver on Distributed Heterogeneous Systems},
year = {2023},
isbn = {9798400701092},
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
url = {https://doi.org/10.1145/3581784.3607050},
doi = {10.1145/3581784.3607050},
booktitle = {Proceedings of the International Conference for High
Performance Computing, Networking, Storage and Analysis},
articleno = {51},
numpages = {14},
keywords = {sparse LU, regular 2D block, distributed heterogeneous systems},
location = {Denver, CO, USA},
series = {SC '23}
}
```

Bibliography

[1] Xu Fu, Bingbin Zhang, Tengcheng Wang, Wenhao Li, Yuechen Lu, Enxin Yi, Jianqi Zhao, Xiaohan Geng, Fangying Li, Jingwen Zhang, Zhou Jin, and Weifeng Liu: “PanguLU: A Scalable Regular Two-Dimensional Block-Cyclic Sparse Direct Solver on Distributed Heterogeneous Systems”, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’23 (2023) doi:10.1145/3581784.3607050

[2] Wikipedia: *Pangu*, https://en.wikipedia.org/wiki/Pangu, 2007

[3] Karypis, George and Kumar, Vipin: “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”, SIAM Journal on Scientific Computing **20**, 359–392 (1998) doi:10.1137/S1064827595287997

[4] Iain S. Duff and Jacko Koster: “On Algorithms For Permuting Large Entries to the Diagonal of a Sparse Matrix”, SIAM Journal on Matrix Analysis and Applications **22**, 973–996 (2000) doi:10.1137/S0895479899358443

[5] Timothy A. Davis and Yifan Hu: “The University of Florida sparse matrix collection”, ACM Transactions on Mathematical Software **38**, 1–25 (2011) doi:10.1145/2049662.2049663