

Argonne Training Program on Extreme-Scale Computing

Direct Sparse Linear Solvers

- with hands-on examples using SuperLU

ATPESC 2018

X. Sherry Li Senior Scientist, LBNL

Q Center, St. Charles, IL (USA) August 6, 2018







Tutorial Content

- Sparse matrix representations
- Algorithms
 - Gaussian elimination, sparsity and graph, ordering, symbolic factorization
- Different types of factorization
- Parallelism exploiting sparsity (trees, DAGs)
 - Task scheduling, avoiding communication
- Parallel performance
 - Numerical factorization, triangular solution
- Software interface, examples



Strategies of solving sparse linear systems

Iterative methods: (e.g., Krylov, multigrid, …)

- A is not changed (read-only)
- Key kernel: sparse matrix-vector multiply
 - Easier to optimize and parallelize
- Low algorithmic complexity, but may not converge
- Direct methods:
 - A is modified (factorized) : A = L*U
 - Harder to optimize and parallelize
 - Numerically robust, but higher algorithmic complexity
- Often use direct method to precondition iterative method
 - Solve an easy system: M⁻¹Ax = M⁻¹b



Flowchart of iterative methods

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", R. Barrett et al.





Available direct solvers

Survey of different types of factorization codes

http://crd.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf

- $LL^{T}(s.p.d.)$
- LDL^T (symmetric indefinite)
- LU (nonsymmetric)
- QR (least squares)
- Sequential, shared-memory (multicore), distributed-memory, out-of-core, few are GPU-enabled ...
- Distributed-memory codes:
 - SuperLU_DIST (Li, Demmel, Grigori, Liu, Sao, Yamazaki)
 - accessible from PETSc, Trilinos, . . .
 - MUMPS, PasTiX, WSMP, . . .



Code	Technique	Scope	Contact	
Serial platforms				
CHOLMOD	Left-looking	SPD	Davis	[8]
KLU	Left-looking	Unsym	Davis	[9]
MA57	Multifrontal	Sym	HSL	[19]
MA41	Multifrontal	Sym-pat	HSL	[1]
MA42	Frontal	Unsym	HSL	[20]
MA67	Multifrontal	Svm	HSL	[17]
MA48	Right-looking	Unsym	HSL	[18]
Oblio	Left/right/Multifr.	sym, Out-core	Dobrian	[14]
SPARSE	Right-looking	Ünsym	Kundert	[29]
SPARSPAK	Left-looking	SPD, Unsym, QR	George et al.	[22]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SuperLLT	Left-looking	SPD	Ng	[32]
SuperLU	Left-looking	Unsym	Li	[12]
UMFPACK	Multifrontal	Unsym	Davis	[10]
Shared memory	parallel machines	*		
BCSLIB-EXT	Multifrontal	Svm, Unsvm, QR	Ashcraft et al.	[6]
Cholesky	Left-looking	SPD	Rothberg	[36]
MF2	Multifrontal	Sym, Sym-pat, Out-core, GPU	Lucas	[31]
MA41	Multifrontal	Sym-pat	HSL	[4]
MA49	Multifrontal	QR	HSL	[3]
PanelLLT	Left-looking	SPD	Ng	[25]
PARASPAR	Right-looking	Unsym	Zlatev	[38]
PARDISO	Left-Right looking	Sym-pat	Schenk	[35]
SPOOLES	Left-looking	Sym, Sym-pat	Ashcraft	[5]
SuiteSparseQR	Multifrontal	Rank-revealing QR	Davis	[11]
SuperLU_MT	Left-looking	Unsym	Li	[13]
TAUCS	Left/Multifr.	Sym, Unsym, Out-core	Toledo	[7]
WSMP	Multifrontal	SPD, Unsym	Gupta	[26]
Distributed memory parallel machines				
Clique	Multifrontal	Sym (no pivoting)	Poulson	[33]
MF2	Multifrontal	Sym, Sym-pat, Out-core, GPU	Lucas	[31]
DSCPACK	Multifrontal	SPD	Raghavan	[28]
MUMPS	Multifrontal	Sym, Sym-pat	Amestoy	[2]
PaStiX	Left-Right looking	SPD, Sym, Sym-pat	Ramet	[23]
PSPASES	Multifrontal	SPD	Gupta	[24]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	َ [5] ا
SuperLU_DIST	Right-looking	Unsym, GPU	Li	[30]
symPACK	Left-Right looking	SPD	Jacquelin	[37]
S+	Right-looking [†]	Unsym	Yang	[21]
WSMP	Multifrontal	SPD, Unsym	Gupta	[26]

Table 1: Software to solve sparse linear systems using direct methods.

† Uses QR storage to statically accommodate any LU fill-in

Abbreviations used in the table:

SPD = symmetric and positive definite

Sym = symmetric and may be indefinite

Sym-pat = symmetric nonzero pattern but unsymmetric values 2

Unsym = unsymmetric

 $\mathrm{HSL} = \mathrm{Harwell} \ \mathrm{Subroutine} \ \mathrm{Library:} \ \mathrm{http://www.cse.clrc.ac.uk/Activity/HSL}$



Direct solvers can support wide range of applications

- fluid dynamics, structural mechanics, chemical process simulation, circuit simulation, electromagnetic fields, magneto-hydrodynamics, seismic-imaging, economic modeling, optimization, data analysis, statistics, ...
- (non)symmetric, indefinite, ill-conditioned ...
- Example: A of dimension 10⁶, 10~100 nonzeros per row
- Matlab: > spy(A)

Boeing/msc00726 (structural eng.)



Mallya/lhr01 (chemical eng.)





Sparse data structure: Compressed Row Storage (CRS)

- Store nonzeros row by row contiguously
- Example: N = 7, NNZ = 19
- **3** arrays:
 - Storage: NNZ reals, NNZ+N+1 integers





Many other data structures: "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", R. Barrett et al.



Distributed input interface

- **Matrices involved:**
 - A, B (turned into X) input, users manipulate them
 - L, U output, users do not need to see them
- A (sparse) and B (dense) are distributed by block rows





Distributed input interface

 Each process has a structure to store local part of A Distributed Compressed Row Storage

```
typedef struct {
```

int_t nnz_loc; // number of nonzeros in the local submatrix

int_t m_loc; // number of rows local to this processor

```
int_t fst_row; // global index of the first row
```

void *nzval; // pointer to array of nonzero values, packed by row

int_t *colind; // pointer to array of column indices of the nonzeros

int_t *rowptr; // pointer to array of beginning of rows in nzval[]and colind[]
NRformat_loc;



Distributed Compressed Row Storage SuperLU_DIST/FORTRAN/f_5x5.f90

A is distributed on 2 processors:



Processor P0 data structure:

- **nnz_loc** = 5
- m_loc = 2
- fst_row = 0 // 0-based indexing
- nzval = { s, u, u, l, u }
- colind = $\{0, 2, 4, 0, 1\}$
- rowptr = { 0, 3, 5 }

Processor P1 data structure:

- nnz_loc = 7
- $m_{loc} = 3$
- fst_row = 2 // 0-based indexing
- nzval = { l, p, e, u, l, l, r }
- colind = $\{1, 2, 3, 4, 0, 1, 4\}$
- rowptr = { 0, 2, 4, 7 }



Distributed L & U factored matrices (SuperLU Internal)

- 2D block cyclic layout specified by user.
- Rule: process grid should be as square as possible.
 - Or, set the row dimension (*nprow*) slightly smaller than the column dimension (*npcol*).
 - For example: 2x3, 2x4, 4x4, 4x8, etc.



MPI Process Grid





Review of Gaussian Elimination (GE)

• First step of GE:

$$A = \begin{bmatrix} a & w^{T} \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a & I \end{bmatrix} \cdot \begin{bmatrix} a & w^{T} \\ 0 & C \end{bmatrix}$$
$$C = B - \frac{v \cdot w^{T}}{\alpha}$$

- Repeat GE on C
- Result in LU factorization (A = LU)
 - L lower triangular with unit diagonal, U upper triangular
- Then, \boldsymbol{x} is obtained by solving two triangular systems with \boldsymbol{L} and \boldsymbol{U}





 $S^{(j)} \leftarrow ((S^{(j)} - D^{(k1)}) - D^{(k2)}) - \dots S^{(j)} \leftarrow S^{(j)} - ((D^{(k1)}) + D^{(k2)}) + \dots)$





U

Sparse LU factorization

Two algorithm variants

Algorithm phases

- 1. Reorder equations to minimize fill, maximize parallelism (~10% time)
 - Sparsity structure of L & U depends on A, which can be changed by row/column permutations (vertex re-labeling of the underlying graph)
 - Ordering (combinatorial algorithms; "NP-complete" to find optimum [Yannakis '83]; use heuristics)
- 2. Predict the fill-in positions in L & U (~10% time)
 - Symbolic factorization (combinatorial algorithms)
- 3. Design efficient data structure for storage and quick retrieval of the nonzeros
 - Compressed storage schemes
- 4. Perform factorization and triangular solutions (~80% time)
 - Numerical algorithms (F.P. operations only on nonzeros)
 - Usually dominate the total runtime

For sparse Cholesky and QR, the steps can be separate. For sparse LU with pivoting, steps 2 and 4 must be interleaved.

Numerical Pivoting

Goal of pivoting is to control element growth in L & U for stability

- For sparse factorizations, often relax the pivoting rule to trade with better sparsity and parallelism (e.g., threshold pivoting, static pivoting, ...)
- Partial pivoting used in sequential SuperLU and SuperLU_MT (GEPP)
 - Can force diagonal pivoting (controlled by diagonal
 - threshold)
 - Hard to implement scalably for sparse factorization
- Static pivoting used in SuperLU_DIST (GESP)
 - Before factor, scale and permute A to maximize diagonal: P_r D_r A D_c = A'
 - During factor A' = LU, replace tiny pivots by $\sqrt{\varepsilon} \|A\|$, without changing data structures for L & U
 - If needed, use a few steps of iterative refinement after the first solution
 - quite stable in practice

ХХ

Ordering to preserve sparsity : Minimum Degree

Local greedy: minimize upper bound on fill-in

Ordering to preserve sparsity : Nested Dissection

- Model problem: discretized system Ax = b from certain PDEs, e.g., 5-point stencil on k x k grid, N = k²
 - Factorization flops: $O(k^3) = O(N^{3/2})$
- Theorem: ND ordering gives optimal complexity in exact arithmetic [George '73, Hoffman/Martin/Rose]

ND Ordering

- Generalized nested dissection [Lipton/Rose/Tarjan '79]
 - Global graph partitioning: top-down, divide-and-conqure
 - Best for largest problems
 - Parallel codes available: ParMetis, PT-Scotch
 - First level

- Recurse on A and B
- Goal: find the smallest possible separator S at each level
 - Multilevel schemes:
 - Chaco [Hendrickson/Leland `94], Metis [Karypis/Kumar `95]
 - Spectral bisection [Simon et al. `90-`95]
 - Geometric and spectral bisection [Chan/Gilbert/Teng `94]

ND Ordering

2D mesh

A, with row-wise ordering

Ordering for LU (unsymmetric)

- Can use a symmetric ordering on a symmetrized matrix
- Case of partial pivoting (serial SuperLU, SuperLU_MT):
 - Use ordering based on A^T*A
- Case of static pivoting (SuperLU_DIST):
 - Use ordering based on AT+A
- Can find better ordering based solely on A, without symmetrization
 - Diagonal Markowitz [Amestoy/Li/Ng `06]
 - Similar to minimum degree, but without symmetrization
 - Hypergraph partition [Boman, Grigori, et al. `08]
 - Similar to ND on $A^{T}A$, but no need to compute $A^{T}A$

User-controllable options in SuperLU_DIST

For stability and efficiency, need to factorize a transformed matrix:

 $P_c(P_r(D_rAD_c))P_c^T$

"Options" fields with C enum constants:

- Equil: {NO, **YES**}
- RowPerm: {NOROWPERM, LargeDiag_MC64, LargeDiag_AWPM, MY_PERMR}
- ColPerm: {NATURAL, MMD_ATA, MMD_AT_PLUS_A, COLAMD, METIS_AT_PLUS_A, PARMETIS, ZOLTAN, MY_PERMC}

Call routine set_default_options_dist(&options) to set default values.

Synchronization-avoiding triangular solve in SuperLU_DIST (GitHub "trisolve" branch. Liu, Jacquelin, Ghysels, Li, SIAM CSC'18)

- In preconditioning, need multiple triangular solves for each factorization.
- Challenge: lower Arithmetic Intensity, higher task dependency.
 - Flops ~ nonzeros in triangular matrix L.
- Customized asynchronous tree-based broadcast/reduction communication
 - Each tree involves a subset of sqrt(P) processes.
 - Latency log(P) for P MPI ranks.
- 4096 cores Cori Haswell:
 - 4.4x faster with 1-RHS, 6x faster with 50-RHS

2 4 6

process grid

Communication-avoid 3D sparse LU in SuperLU_DIST

(P. Sao, X.S. Li, R. Vuduc, IPDPS 2018)

- For matrices from planar graph, provably asymptotic lower communication complexity:
 - Comm. volume reduced by a factor of sqrt(log(n)).
 - Latency reduced by a factor of log(n).
- Strong scale to 24,000 cores.

Compared to 2D algorithm:

- Planar graph: up to 27x faster, 30% more memory
 @ P_z = 16
- Non-planar graph: up to 3.3x faster, 2x more memory @ P_z = 16

Combining 3D algorithm with GPU acceleration

Sao, Vuduc, Li, JPDC preprint, 2018)

- Co-processor acceleration to reduce intra-node communication
 - Sao, Vuduc, Li (EuroPar'14); Sao, Liu, Vuduc, Li (IPDPS'15)
 - Offload Schur-complement update to GPU
- Empirical study on Cray XK7 (titan @ OLCF)

Each node: AMD Opteron processor (16 cores) + 1 Nvidia K20X GPU

Speedup of combined 3D-CPU-GPU over 3D-CPU:

SuperLU Installation

- Download site:
 - Tarball: http://crd.lbl.gov/~xiaoye/SuperLU
 - Github: https://github.com/xiaoyeli/superlu_dist
 - Users' Guide, HTML code documentation, papers.

Follow README at top level directory

Two ways of building:

- 1. CMake build system.
- 2. Edit make.inc (compilers, optimizations, libraries, ...)
- Dependency: BLAS, ParMetis or PT-Scotch (parallel ND ordering)
 - Link with a fast BLAS library
 - The one under CBLAS/ is functional, but not optimized
 - Vendor, OpenBLAS, ATLAS, ...

Use multicore, GPU

- Instructions in top-level README.
- To use OpenMP parallelism: Export OMP_NUM_THREADS=<##>
- To enable Nvidia GPU access, need to take the following 2 step:
 - set the following Linux environment variable: export ACC=GPU
 - Add the CUDA library location in make.inc: (see sample make.inc) ifeq "\${ACC}" "GPU"

```
CUDA_FLAGS = -DGPU_ACC
```

INCS += -I<CUDA directory>/include

LIBS += -L<CUDA directory>/lib64 -lcublas –lcudart

endif

Tips for Debugging Performance

- Check sparsity ordering
- Diagonal pivoting is preferable
 - E.g., matrix is diagonally dominant, ...
- Need good BLAS library (vendor, OpenBLAS, ATLAS)
 - May need adjust block size for each architecture
 - (Parameters modifiable in routine sp_ienv())
 - Larger blocks better for uniprocessor
 - Smaller blocks better for parallellism and load balance
 - New xSDK4ECP project: automatic tuning for parameter setting.

Evening hands-on: HandsOnLessons/superlu_mfem/superlu-dist

See README

- pddrive.c: Solve one linear system.
- pddrive1.c: Solve the systems with same A but different right-hand side at different times.
 - Reuse the factored form of A.
- pddrive2.c: Solve the systems with the same pattern as A.
 - Reuse the sparsity ordering.
- pddrive3.c: Solve the systems with the same sparsity pattern and similar values.
 - Reuse the sparsity ordering and symbolic factorization.
- pddrive4.c: Divide the processes into two subgroups (two grids) such that each subgroup solves a linear system independently from the other.

Block Jacobi preconditioner

Evening hands-on: HandsOnLessons/superlu_mfem/superlu-dist

Four input matrices:

- g4.rua
- g20.rua (400 dofs)
- big.rua (4960 dofs)
- stomach.rua (213k dofs, ~15 sec @ P=16)

STRUMPACK "inexact" direct solver

- Baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
 - O(N^{4/3} logN) flops, O(N) memory for 3D elliptic PDEs.
- Hierarchical matrix algebra generalizes Fast Multipole
 - Diagonal block ("near field") exact; off-diagonal block ("far field") approximated via low-rank compression.
 - Hierarchically semi-separable (HSS), HODLR, etc.
 - Nested bases + randomized sampling to achieve linear scaling.
- Applications: PDEs, BEM methods, integral equations, machine learning, and structured matrices such as Toeplitz, Cauchy matrices.

STRUMPACK algorithm scaling for 3D Poisson

- Theory predicts $O(n^{4/3} \log n)$ flops for compression.
- HSS ranks grow with mesh size ~ $n^{1/3} = k$
- Use as a preconditioner with aggressive compression.

STRUMPACK-sparse: strong scaling

- Matrix from SuiteSparse Matrix Collection:
 - Flan_1565 : N= 1,564,794, NNZ = 114,165,372
- Flat MPI on nodes with 2 x 12-core Intel Ivy Bridge, 64GB (NERSC Edison)
- Fill-reducing reordering (ParMetis) has poor scalability, quality decreases

Software summary

- SuperLU: conventional direct solver for general unsymmetric linear systems. (X.S. Li, J. Demmel, J. Gilbert, L. Grigori, Y. Liu, P. Sao, M. Shao, I. Yamazaki)
 - O(N²) flops, O(N^{4/3}) memory for typical 3D PDEs.
 - C, hybrid MPI+ OpenMP + CUDA; Provide Fortran interface.
 - Real, complex.
 - Componentwise error analysis and error bounds (guaranteed solution accuracy), condition number estimation.
 - http://crd-legacy.lbl.gov/~xiaoye/SuperLU/
- STRUMPACK: "inexact" direct solver, preconditioner.

(P. Ghysels, G. Chavez, C. Gorman, F.-H. Rouet, X.S. Li)

- O(N^{4/3} logN) flops, O(N) memory for 3D elliptic PDEs.
- C++, hybrid MPI + OpenMP; Provide Fortran interface.
- Real, complex.
- http://portal.nersc.gov/project/sparse/strumpack/

Final remarks

- Sparse (approximate) factorizations are important kernels for numerically difficult problems.
- Performance more sensitive to latency than dense case.
- Continuing developments funded by DOE ECP and SciDAC projects
 - Hybrid model of parallelism for multicore/vector nodes, differentiate intra-node and inter-node parallelism
 - Hybrid programming models, hybrid algorithms
 - More parallel structured matrix precondtioners:
 - HODLR, H/H², butterfly, ...

References

- Short course, "Factorization-based sparse solvers and preconditioners", 4th Gene Golub SIAM Summer School, 2013.<u>https://archive.siam.org/students/g2s3/2013/index.html</u>
 - 10 hours lectures, hands-on exercises
 - Extended summary: <u>http://crd-legacy.lbl.gov/~xiaoye/g2s3-summary.pdf</u> (in book "Matrix Functions and Matrix Equations", https://doi.org/10.1142/9590
- SuperLU: http://crd-legacy.lbl.gov/ xiaoye/SuperLU/
- STRUMPACK: portal.nersc.gov/project/sparse/strumpack/

SuperLU_DIST hands-on

xsdk-project.github.io/ATPESC2018HandsOnLessons/lessons/superlu_mfem/

- Solve steady-state convection-diffusion equations
- Get 2 compute nodes: qsub -I -n 2 -t 30 -A ATPESC2018
- run 1: ./convdiff >& run1.out
- run 2: ./convdiff --velocity 1000 >& run2.out
- run 3: ./convdiff --velocity 1000 -slu -cp 0 >& run3.out
- run 4: ./convdiff --velocity 1000 -slu -cp 2 >& run4.out
- run 5: ./convdiff --velocity 1000 -slu -cp 4 >& run5.out
- run 6: mpiexec -n 1 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 >& run6.out
- run 7: mpiexec -n 16 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 >& run7.out
- run 8: mpiexec -n 16 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 -2rhs >& run8.out
- run 9: mpiexec -n 16 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 -2mat >& run9.out

SuperLU_DIST hands-on

xsdk-project.github.io/ATPESC2018HandsOnLessons/lessons/superlu_mfem/

Convection-Diffusion equation (steady-state):

HandsOnLessons/superlu-mfem/

GMRES iterative solver with BoomerAMG preconditioner

\$./convdiff (default velocity = 100)

- \$./convdiff --velocity 1000 (no convergence)
- Switch to SuperLU direct solver
 \$./convdiff -slu --velocity 1000
- Experiment with different orderings: --slu-colperm (you see different number of nonzeros in L+U)
 - 0 natural (default)
 - 1 mmd-ata (minimum degree on graph of A^T*A)
 - 2 mmd_at_plus_a (minimum degree on graph of A^T+A)
 - 3 colamd
 - 4 metis_at_plus_a (Metis on graph of A^T+A)
 - 5 parmetis (ParMetis on graph of A^T+A)
- Lessons learned
 - Direct solver can deal with ill-conditioned problems.
 - Performance may vary greatly with different elimination orders.

Thank you!

exascaleproject.org

EXTRA SLIDES

SuperLU_DIST performance on Intel KNL

- Single node improvement
 - Aggregate large GEMM
 - OpenMP task parallel
 - Vectorize scatter
 - Cacheline- & Page-aligned malloc

nlpttk80,	n = 1.1M, nnz = 28M
Ga19As19H42,	n = 1.3M, nnz = 8.8M
RM07R,	n = 0.3M, nnz = 37.5N

Strong scaling to 32 nodes

• Current work: 3D algorithm to reduce communication, increase parallelism

STRUMPACK Installation

- Download site:
 - Tarball: http://portal.nersc.gov/project/sparse/strumpack/
 - Github: https://github.com/pghysels/STRUMPACK
 - Users' Guide, code documentation, papers
- Dependency: BLAS, ParMetis or PT-Scotch, SCALAPACK

CMake example:

- > export METISDIR=/path/to/metis
- > export PARMETISDIR=/path/to/parmetis
 > export SCOTCHDIR=/path/to/scotch_____

- > export SCOTCHDIR=/path/to/scotch
 > cmake ../strumpack-sparse -DCMAKE_BUILD_TYPE=Release \
 -DCMAKE_INSTALL_PREFIX=/path/to/install \
 -DCMAKE_CXX_COMPILER=<C++ (MPI) compiler> \
 -DCMAKE_C_COMPILER=<C (MPI) compiler> \
 -DCMAKE_Fortran_COMPILER=<Fortran77 (MPI) compiler> \
 -DCMAKE_Fortran_COMPILER=<Fortran77 (MPI) compiler> \
 -DSCALAPACK_LIBRARIES="/path/to/scalapack/libscalapack.a;/path/to/blacs/libblacs.a" \
 -DMETIS_INCLUDES=/path/to/metis/libmetis.a \
 -DPARMETIS_INCLUDES=/path/to/parmetis/include \
 -DPARMETIS_LIBRARIES=/path/to/parmetis/libparmetis.a \
 -DSCOTCH_INCLUDES=/path/to/scotch/include \
 -DSCOTCH_LIBRARIES="/path/to/ptscotch/libscotch.a;...libptscot
- > make
- > make examples #optional
 > make install

Use through PETSc

./configure \
 --with-shared-libraries=0 \
 --download-strumpack \
 --with-openmp \
 --with-cxx-dialect=C++11 \
 --download-scalapack \
 --download-parmetis \
 --download-metis \
 --download-ptscotch \

make PETSC_DIR=<petsc-dir> PETSC_ARCH=<petsc-arch-dir> all

make PETSC_DIR=<...> PETSC_ARCH=<...> test

export PETSC_DIR=<...> export PETSC_ARCH=<...> cd src/ksp/ksp/examples/tutorials make ex52

use as direct solver OMP_NUM_THREADS=1 mpirun -n 2 ./ex52 -pc_type lu -pc_factor_mat_solver_package strumpack -mat_strumpack_verbose 1

use as approximate factorization preconditioner OMP_NUM_THREADS=1 mpirun -n 2 ./ex52 -pc_type ilu -pc_factor_mat_solver_package strumpack -mat_strumpack_verbose 1

Examples in examples/

See README

- testPoisson2d:
 - A double precision C++ example, solving the 2D Poisson problem with the sequential or multithreaded solver.
- testPoisson2dMPIDist:
 - A double precision C++ example, solving the 2D Poisson problem with the fully distributed MPI solver.
- testMMdoubleMPIDist:
 - A double precision C++ example, solving a linear system with a matrix given in a file in the matrixmarket format, using the fully distributed MPI solver.
- testMMdoubleMPIDist64:
 - A double precision C++ example using 64 bit integers for the sparse matrix.
- {s,d,c,z}example:
 - examples to use C interface.

