

Sparse Direct Solvers

Presented to
ATPESC 2017 Participants

X. Sherry Li
Senior Scientist, LBNL

Q Center, St. Charles, IL (USA)
08/07/2017



ATPESC Numerical Software Track



Outline of Tutorial

- **Why direct solvers?**
- **Sparse matrix distributed data structure**
- **Algorithms**
- **Software, user interface**

- **Examples, Fortran 90 interface**
- **Hands-on exercises**

Strategies of sparse linear solvers

- **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: $M^{-1}Ax = M^{-1}b$

- **SuperLU: conventional direct solver for general unsymmetric linear systems.**

X.S. Li, J. Demmel, J. Gilbert, L. Grigori, P. Sao, M. Shao, I. Yamazaki

- $O(N^2)$ 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, C. Gorman, F.-H. Rouet, X.S. Li

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

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, ...

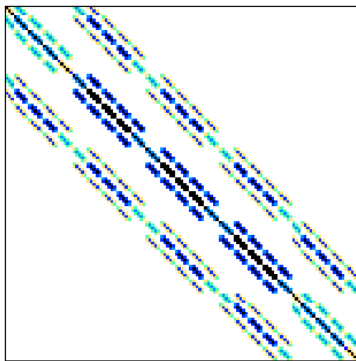
Use multicore, GPU

- **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:**
 1. set the following Linux environment variable:
`export ACC=GPU`
 2. 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`

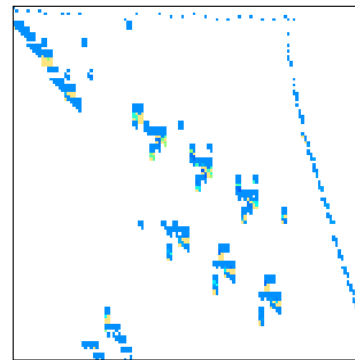
Direct solvers can support wide range of applications

- fluid dynamics, structural mechanics, chemical process simulation, circuit simulation, electromagnetic fields, magneto-hydrodynamics, seismic-imaging, economic modeling, optimization, data analysis, statistics, . . .
- Symmetric, nonsymmetric, indefinite, ill-conditioned ...
- Example: A of dimension 10^6 , 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

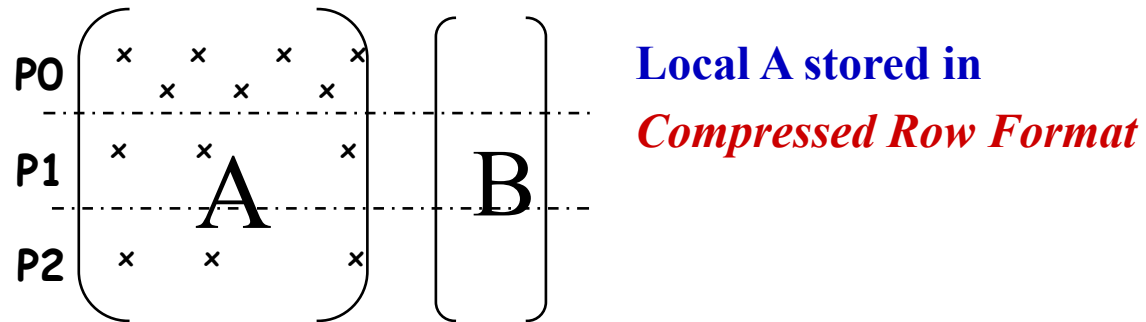
$$\begin{pmatrix} 1 & & a & & & & \\ & 2 & & b & & & \\ c & d & 3 & & & & \\ & e & & 4 & f & & \\ & & & & 5 & & g \\ & & & h & i & 6 & j \\ & k & & l & & & 7 \end{pmatrix}$$

	1	3	5	8	11	13	17	20
nzval	1 a	2 b	c d 3	e 4 f	5 g	h i 6 j	k l 7	
colind	1 4	2 5	1 2 3	2 4 5	5 7	4 5 6 7	3 5 7	
rowptr	1 3 5 8 11 13 17 20							

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

A is distributed on 2 processors:

P0	s		u		u
	l	u			
<hr/>					
P1		l	p		
				e	u
	l	l			r

▪ Processor P0 data structure:

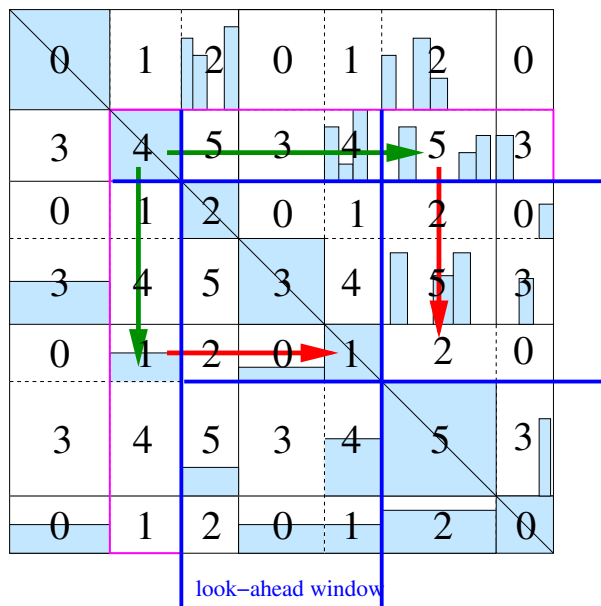
- $nnz_loc = 5$
- $m_loc = 2$
- $fst_row = 0$ // 0-based indexing
- $nzval = \{s, u, u, l, u\}$
- $colind = \{0, 2, 4, 0, 1\}$
- $rowptr = \{0, 3, 5\}$

▪ Processor P1 data structure:

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

Internal : distributed L & U factored matrices

- 2D block cyclic layout – input by user
- 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.



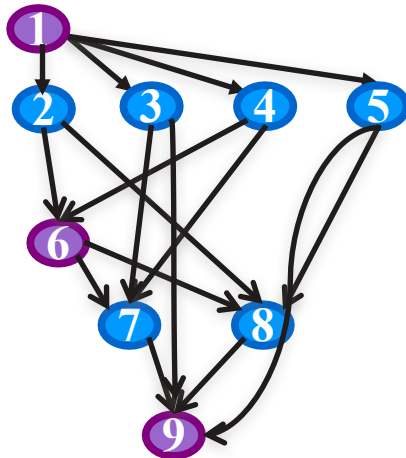
MPI Process Grid

0	1	2
3	4	5

Sparse LU factorization

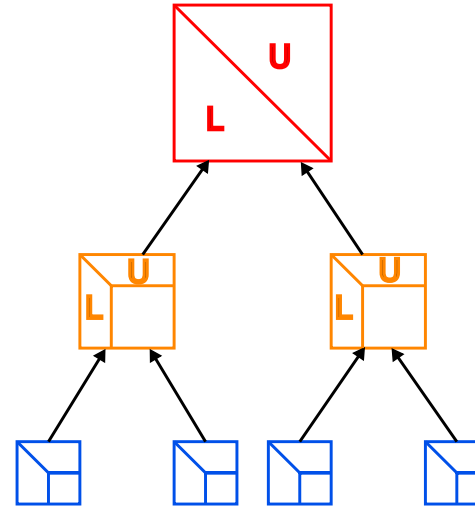
Two algorithm variants

DAG based
Supernodal: SuperLU

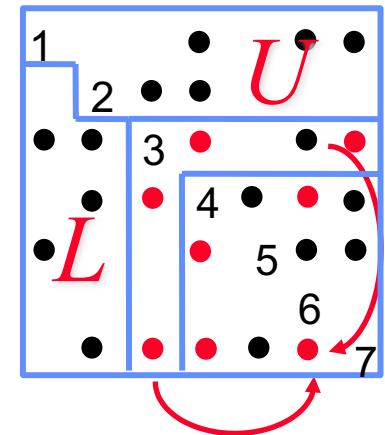


$$S^{(j)} \leftarrow ((S^{(j)} - D^{(k1)}) - D^{(k2)}) - \dots$$

Tree based
Multifrontal: STRUMPACK



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



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 may be interleaved.

User-controllable options

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

$$P_c (P_r (D_r A D_c)) P_c^T$$

“Options” fields with C enum constants:

- Equil: {NO, YES}
- RowPerm: {NOROWPERM, LargeDiag, 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.

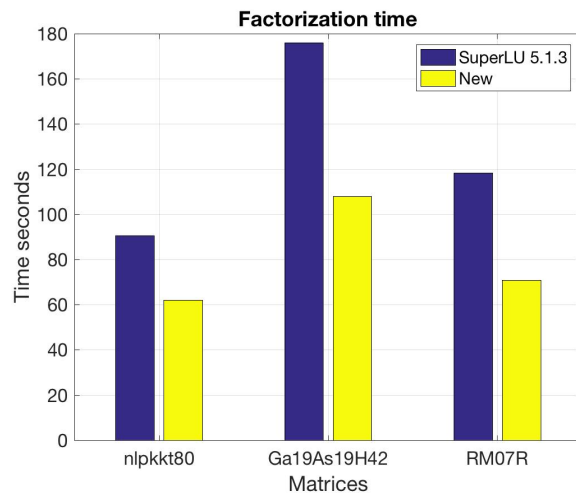
Tips for Debugging Performance

- 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 parallelism and load balance
 - New xSDK4ECP project: automatic tuning for block size.

SuperLU_DIST performance on Intel KNL

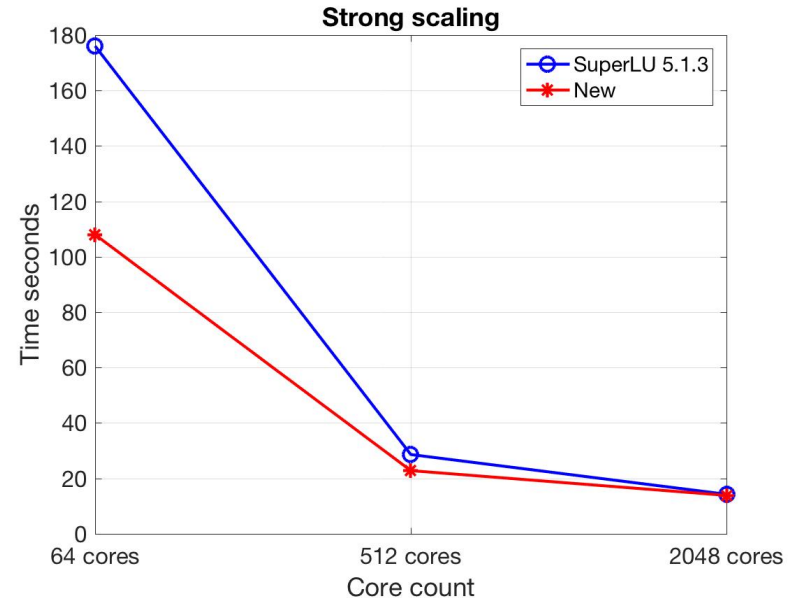
- **Single node improvement**

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



nlpkkt80, $n = 1.1\text{M}$, $\text{nnz} = 28\text{M}$
Ga19As19H42, $n = 1.3\text{M}$, $\text{nnz} = 8.8\text{M}$
RM07R, $n = 0.3\text{M}$, $\text{nnz} = 37.5\text{M}$


- **Strong scaling to 32 nodes**



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

Examples in EXAMPLE/

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.
- 

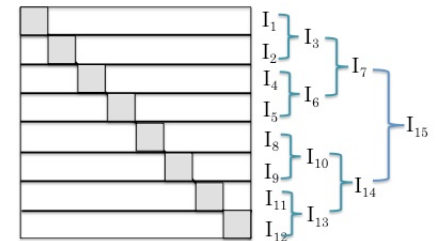
Block Jacobi preconditioner

0	1		
2	3		
		4	5
		6	7
			8
			9
			10
			11

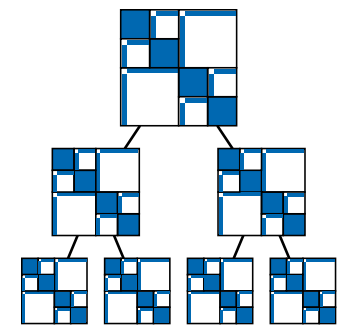
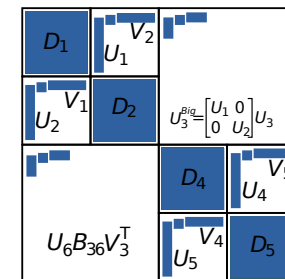
STRUMPACK “inexact” direct solver

- Baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
 - $O(N \log N)$ 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.

Cluster tree



$$A \approx \begin{bmatrix} \begin{bmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{bmatrix} & U_3 B_3 V_6^T \\ U_6 B_6 V_3^T & \begin{bmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{bmatrix} \end{bmatrix}$$



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
> 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> \
  -DSCALAPACK_LIBRARIES="/path/to/scalapack/libscalapack.a;/path/to/blacs/libblacs.a" \
  -DMETIS_INCLUDES=/path/to/metis/include \
  -DMETIS_LIBRARIES=/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;...libscotcherr.a;...libptscotch.a;...libptscotcherr.a"

> 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
```

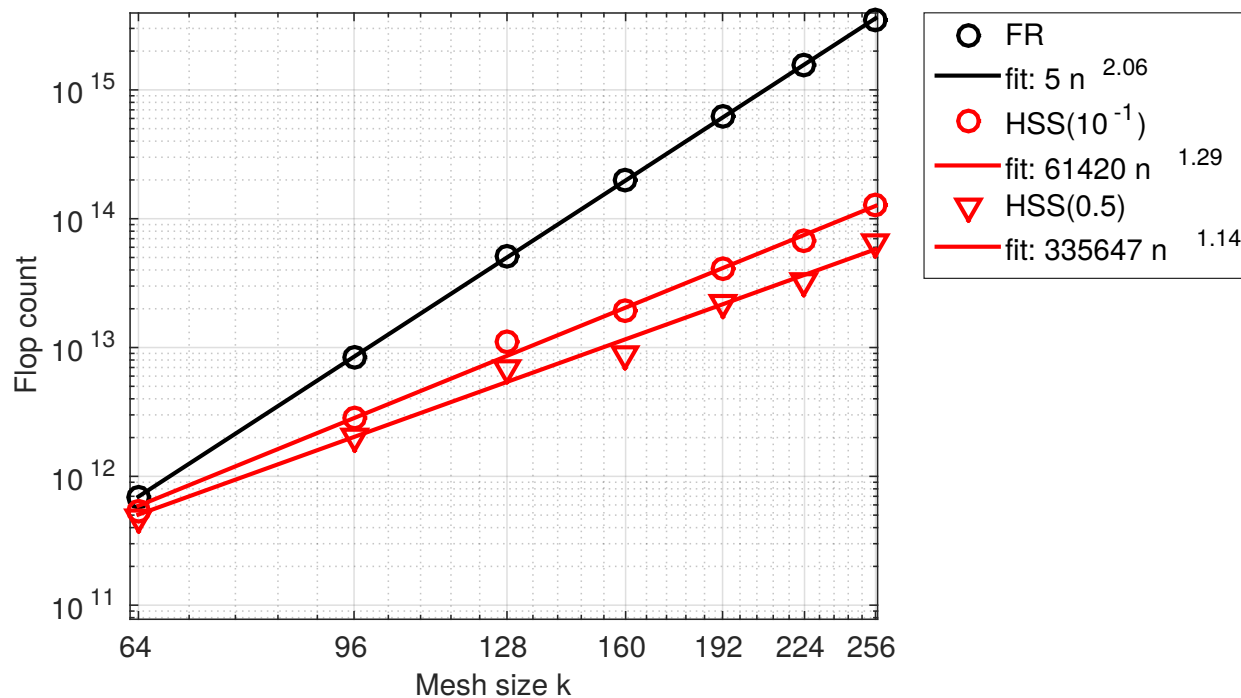
```
OMP_NUM_THREADS=1 mpirun -n 2 ./ex52 -pc_type lu -pc_factor_mat_solver_package strumpack -  
mat_strumpack_verbose 1
```

```
## 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
```

STRUMPACK algorithm scaling for 3D Poisson

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



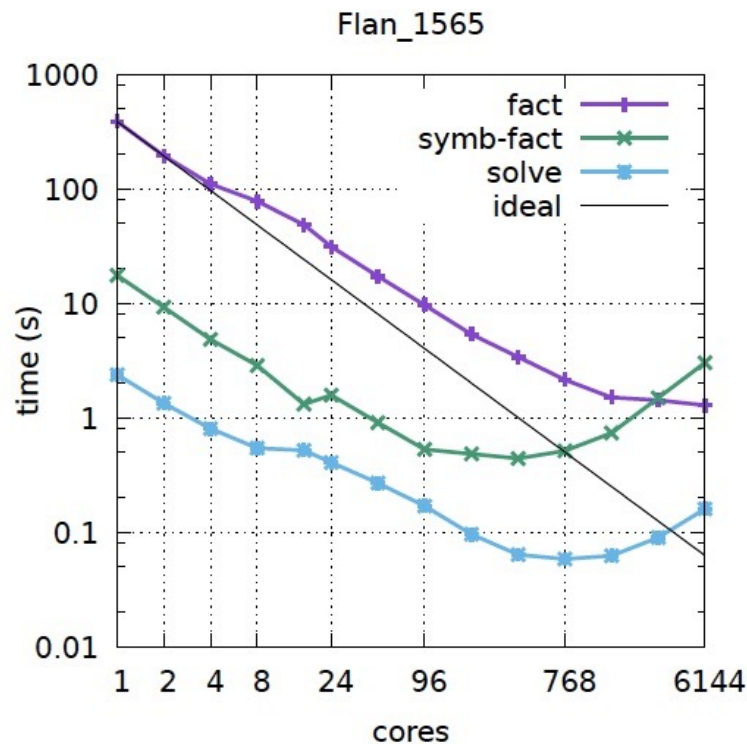
STRUMPACK-dense: parallel weak scaling

- Root node of the multifrontal factorization of a discretized Helmholtz problem (frequency domain, PML boundary, 10Hz).
- For many PDEs on mesh $K \times K \times K$, max. off-diagonal rank $O(K)$.

K (mesh: K^3)	100	200	300	400	500
Matrix size K^2	10,000	40,000	90,000	160,000	250,000
MPI tasks	64	256	1,024	4,096	8,192
Max. rank	313	638	903	1289	1625
Speedup over ScaLAPACK LU	1.8	4.0	5.4	4.8	3.9

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**



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 matrix-market 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.

Summary

- **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**
 - Integrate into more applications
 - Hybrid model of parallelism for multicore/vector nodes, differentiate intra-node and inter-node parallelism
 - Hybrid programming models, hybrid algorithms
 - More parallel structured matrix preconditioners:
 - HODLR, H/H^2 , butterfly, ...


SuperLU_DIST handson

- **Convection-Diffusion equation (steady-state):** $\nabla \cdot (\kappa \nabla u) - \nabla \cdot (\vec{v} u) + R = 0$
/projects/ATPESC2017/NumericalPackages/handson/mfem/examples/atpesc/superlu
- **GMRES iterative solver**
\$./convdiff (default velocity = 100)
\$./convdiff --velocity 1000 (no convergence)
- **Switch to SuperLU direct solver**
\$./convdiff -slu --velocity 1000
- **Experiment with different orderings: --slu-colperm**
 - 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.

EXTRA SLIDES

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 (GESD)
 - 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

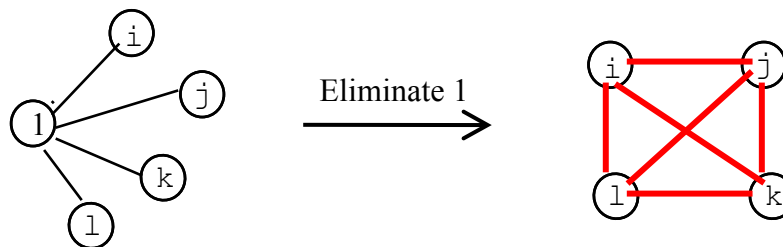


	s	x	x
	x		
	b	x	x

Ordering : Minimum Degree

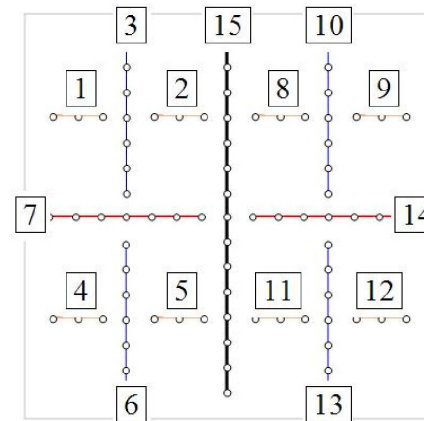
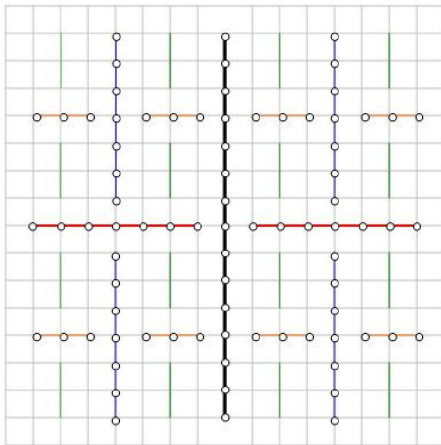
Local greedy: minimize upper bound on fill-in

$$\begin{array}{c}
 \begin{array}{c} 1 \\ i \\ j \\ k \\ l \end{array} \begin{bmatrix} x & & & & \\ & x & & & \\ & & x & & \\ & & & x & \\ & & & & x \end{bmatrix}
 \end{array}
 \xrightarrow{\text{Eliminate } 1}
 \begin{array}{c}
 \begin{array}{c} 1 \\ i \\ j \\ k \\ l \end{array} \begin{bmatrix} x & & & & \\ & x & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot & \\ & & & & & \cdot \end{bmatrix}
 \end{array}$$



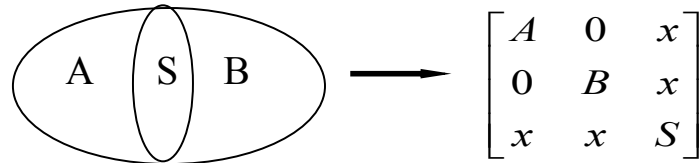
Ordering : Nested Dissection

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



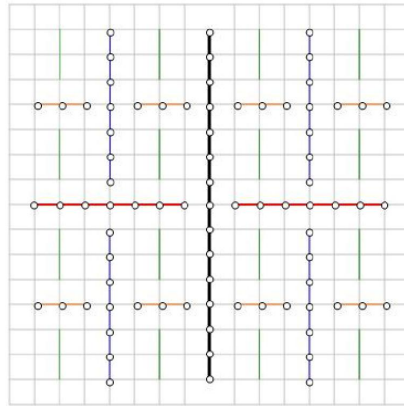
ND Ordering

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

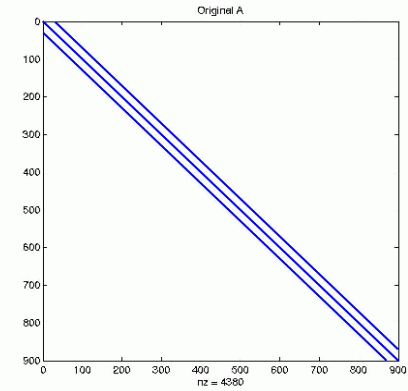


- Recurse on A and B
- Goal: find the smallest possible separator S at each level
 - Multilevel schemes:
 - **Chaco [Hendrickson/Leland '94], Metis [Karypis/Kumar '95]**
 - Spectral bisection [Simon et al. '90-'95]
 - Geometric and spectral bisection [Chan/Gilbert/Teng '94]

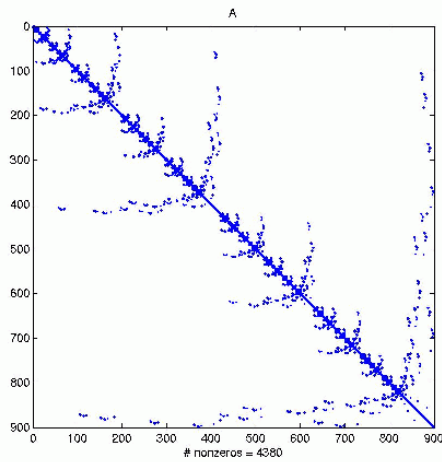
ND Ordering



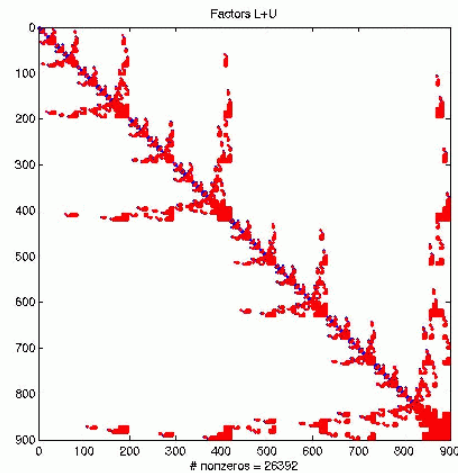
2D mesh



A, with row-wise ordering



A, with ND ordering



L & U factors

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 $A^T + A$
- **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 ATA, but no need to compute ATA