EXASCALE COMPUTING PROJECT

Argonne Training Program on Extreme-Scale Computing

Direct Sparse Linear Solvers, Preconditioners

- SuperLU, STRUMPACK, with hands-on examples

ATPESC 2021

X. Sherry Li, Pieter Ghysels
Lawrence Berkeley National Laboratory

August 10, 2021

Tutorial Content

Part 1. Sparse direct solvers: SuperLU and STRUMPACK (30 min)

- Sparse matrix representations
- Algorithms
  - Gaussian elimination, sparsity and graph, ordering, symbolic factorization
- Different organizations of elimination algorithms
- Parallelism exploiting sparsity (trees, DAGs)
  - Task scheduling, avoiding communication

Part 2. Rank-structured approximate factorizations: STRUMPACK (15 min)

- Hierarchical matrices, Butterfly matrix

Part 3. Hands-on examples in SuperLU or STRUMPACK (15 min)

Algorithms: review of Gaussian Elimination (GE)

- First step of GE:

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

$$C = B - \frac{v \cdot w^T}{\alpha}$$

- Repeat GE on C
- Result in LU factorization (A = LU)
  - L lower triangular with unit diagonal, U upper triangular

- Then, x is obtained by solving two triangular systems with L and U, easier to solve

EXASCALE COMPUTING PROJECT

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

Exploit sparsity

1) Structural sparsity
   - Defined by {0, 1} structure (Graphs)
   - LU factorization ~ $O(N^2)$ flops, for many 3D discretized PDEs

2) Data sparsity (usually with approximation)
   - On top of 1), can find data-sparse structure in dense (sub)matrices (often involve approximation)
   - LU factorization ~ O(N polylog(N))

SuperLU: only structural sparsity

STRUMPACK: both structural and data sparsity

PDE discretization leads to sparse matrices

- Poisson equation in 2D (continuum)

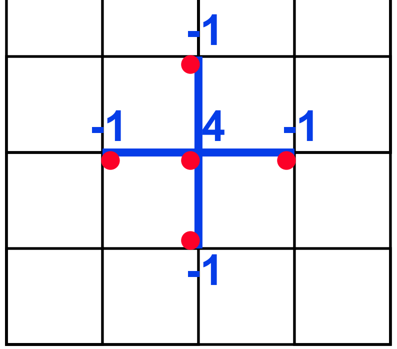
$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y), \quad (x,y) \in R$$

$$u(x,y) = g(x,y), \quad \text{(x,y) on the boundary}$$

- Stencil equation (discretized)

$$4 \cdot u(i,j) - u(i-1,j) - u(i+1,j) - u(i,j-1) - u(i,j+1) = f(i,j)$$

Graph and "stencil"



$$A = \begin{pmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{pmatrix}$$

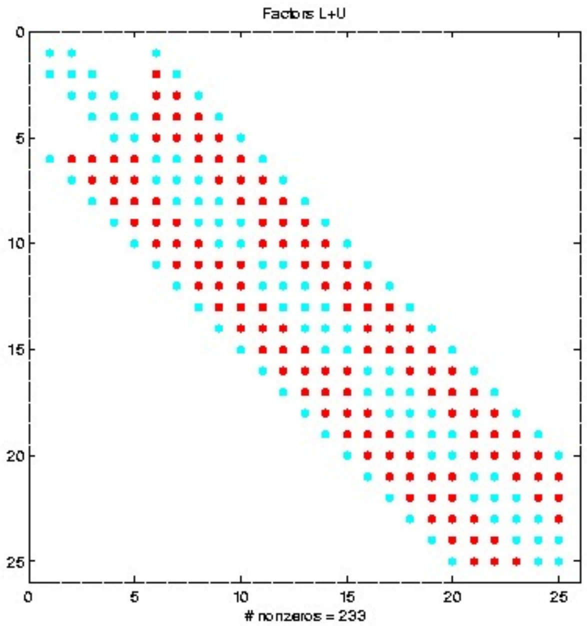
Fill-in in Sparse GE

Original zero entry A_{ij} becomes nonzero in L or U

– Red: fill-ins (Matlab: spy())

Natural order: NNZ = 233

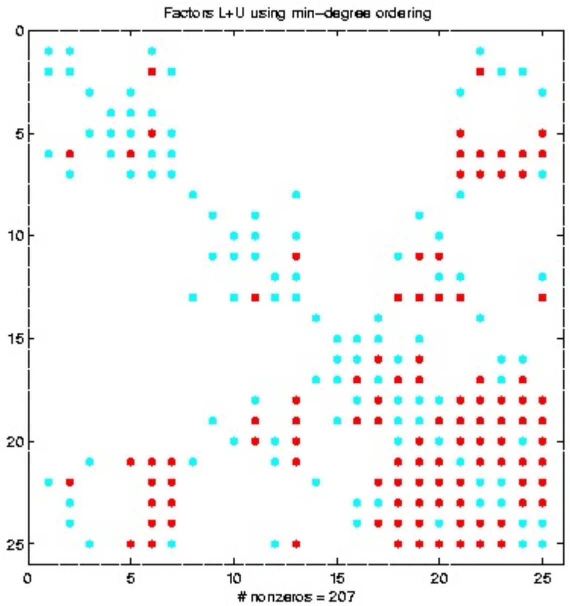
Band solver



Fill-in: $O(N^{3/2})$
Flops: $O(N^2)$

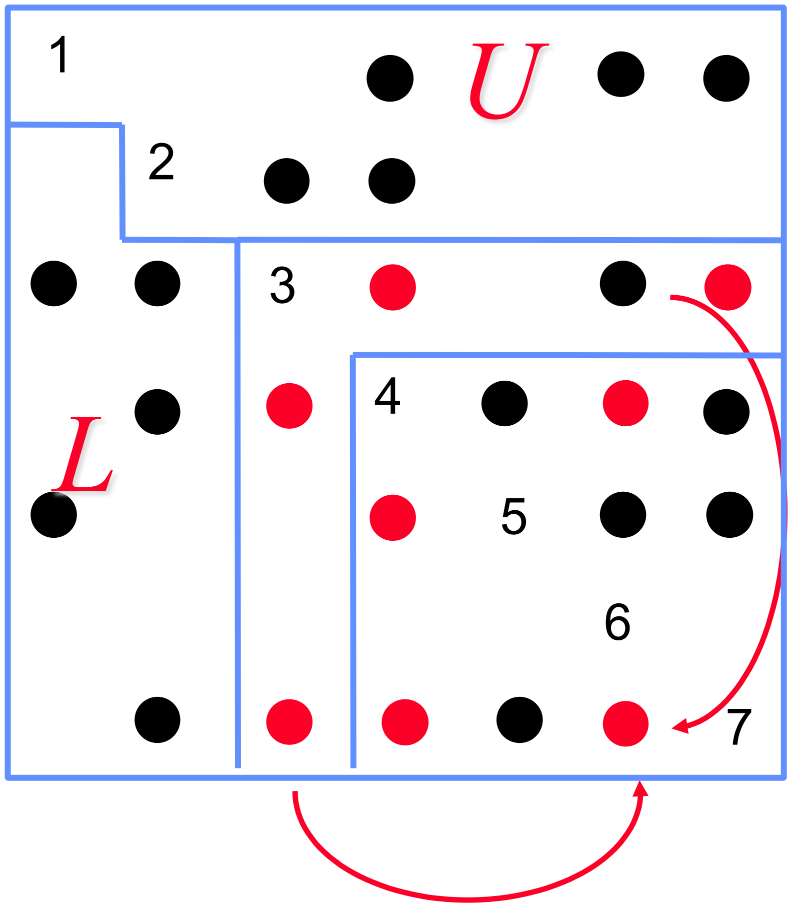
Minimum Degree order: NNZ = 207

General sparse solver



Fill-in: $O(N \log(N))$
Flops: $O(N^{3/2})$

Fill-in in sparse LU



Store general sparse matrix: 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												

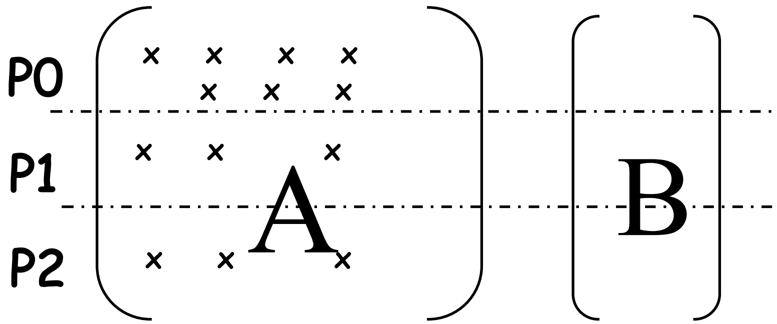
Many other data structures: "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", R. Barrett et al.

EXASCALE COMPUTING PROJECT

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



Local A stored in *Compressed Row Format*

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:

P0	s		u		u
	l	u			
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 }**

Direct solver solution phases

1. Preprocessing: 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. Preprocessing: predict the fill-in positions in L & U (~10% time)
   - Symbolic factorization (combinatorial algorithms)

3. Preprocessing: Design efficient data structure for quick retrieval of the nonzeros
   - Compressed storage schemes

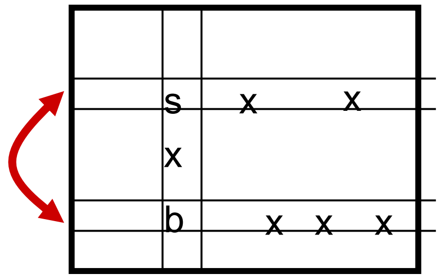
4. Perform factorization and triangular solutions (~80% time)
   - Numerical algorithms (F.P. operations only on nonzeros)
   - Usually dominate the total runtime

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

Numerical pivoting for stability

- 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 dense LU, sequential SuperLU and SuperLU_MT (GEPP)
  - Can force diagonal pivoting (controlled by diagonal threshold)
  - Hard to implement scalably for sparse factorization



Relaxed pivoting strategies:

- **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 $\sqrt{\varepsilon}\|A\|$, w/o changing data structures for L & U
  - If needed, use a few steps of iterative refinement after the first solution
  - quite stable in practice
- **Restricted pivoting**

Can we reduce fill? -- various ordering algorithms

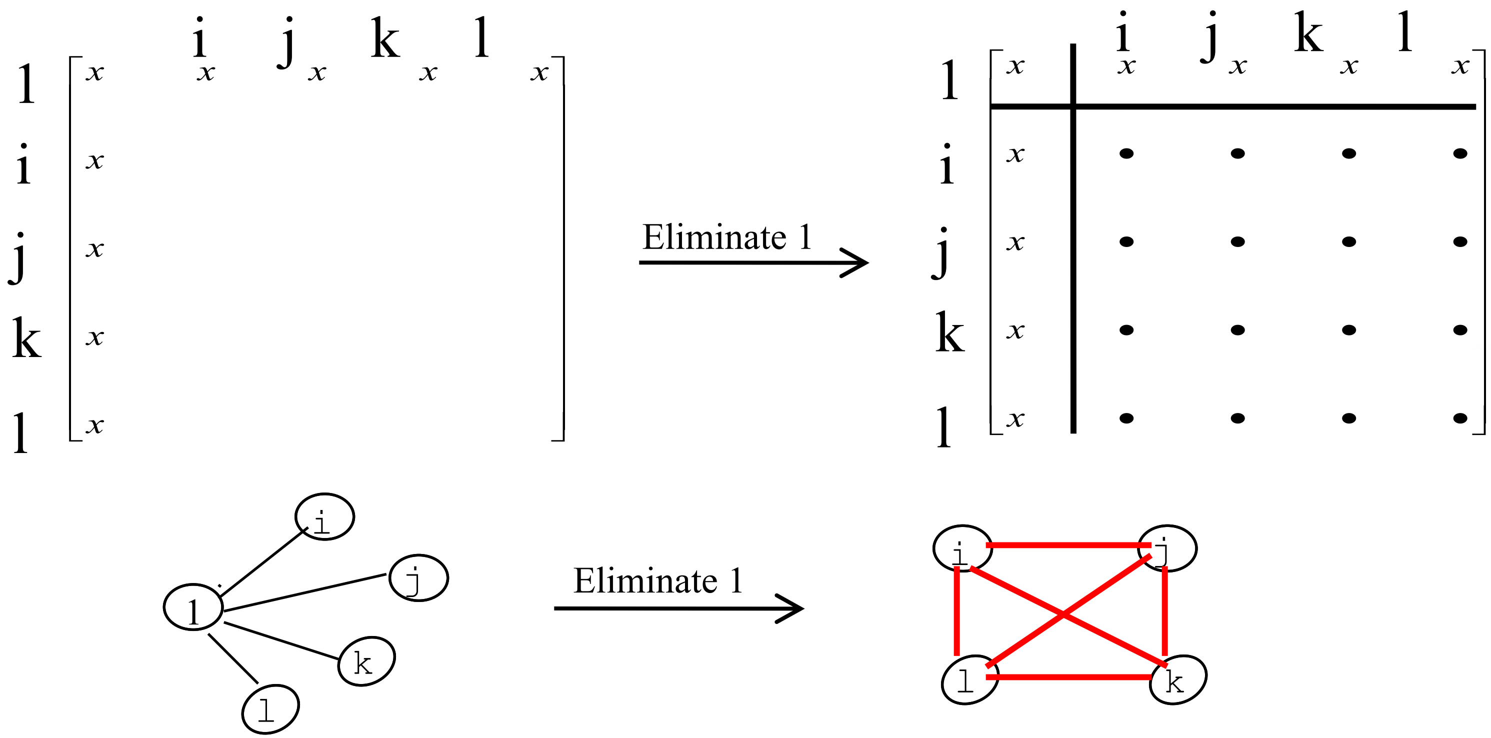
- Reordering (= permutation of equations and variables)

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & & & \\ 3 & & 3 & & \\ 4 & & & 4 & \\ 5 & & & & 5 \end{pmatrix} \text{(all filled after elimination)}$$

$$\Rightarrow \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & & & \\ 3 & & 3 & & \\ 4 & & & 4 & \\ 5 & & & & 5 \end{pmatrix} \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix} = \begin{pmatrix} 5 & & & & 5 \\ & 4 & & & 4 \\ & & 3 & & 3 \\ & & & 2 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

(no fill after elimination)

Ordering to preserve sparsity : Minimum Degree



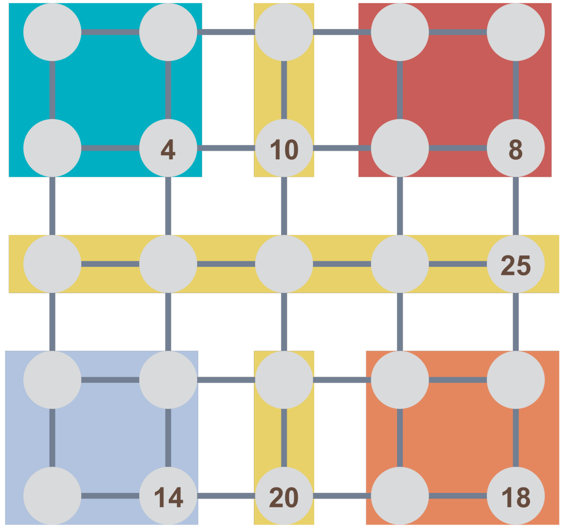
- Local greedy strategy: minimize upper bound on fill-in at each elimination step
- Algorithm: Repeat N steps:
  - Choose a vertex with minimum degree to eliminate
  - Update the remaining graph

Quotient graph [ ], approximate degree [ ]

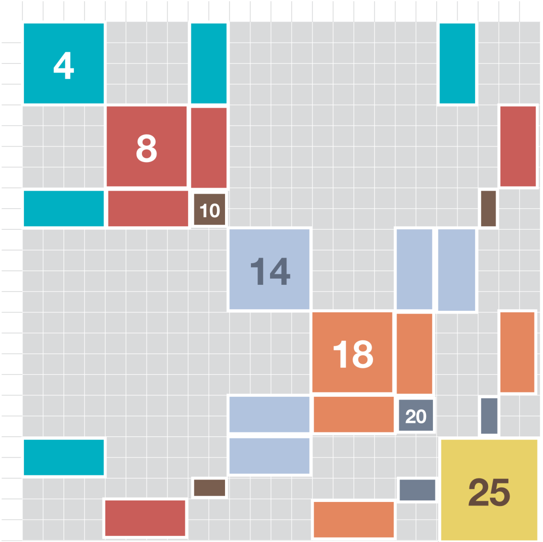
Ordering to preserve sparsity : Nested Dissection

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

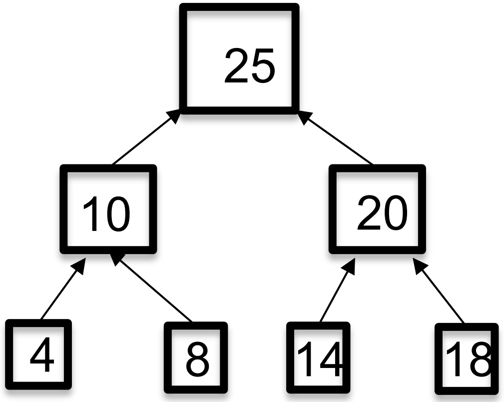
Geometry



Reordered Matrix

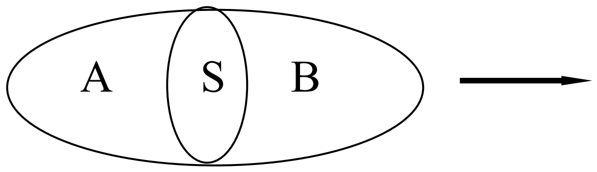


Separator Tree



ND Ordering

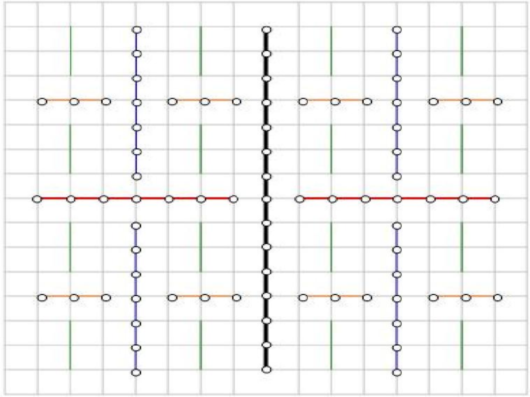
- Generalized nested dissection [Lipton/Rose/Tarjan ’79]
  - **Global graph partitioning: top-down, divide-and-conqure**
  - **Best for large problems**
  - **Parallel codes available: ParMetis, PT-Scotch**
    - First level



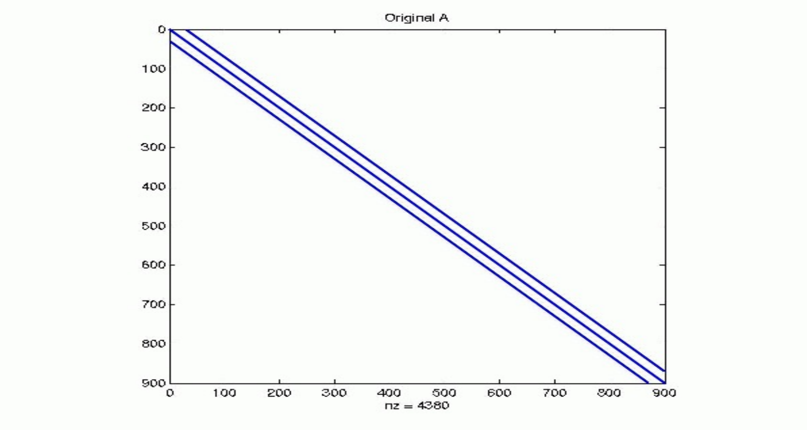
$$\begin{bmatrix} A & 0 & x \\ 0 & B & x \\ x & x & S \end{bmatrix}$$

    - 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, Ghysels et al. 2019-]**
  - **Geometric and spectral bisection [Chan/Gilbert/Teng `94]**

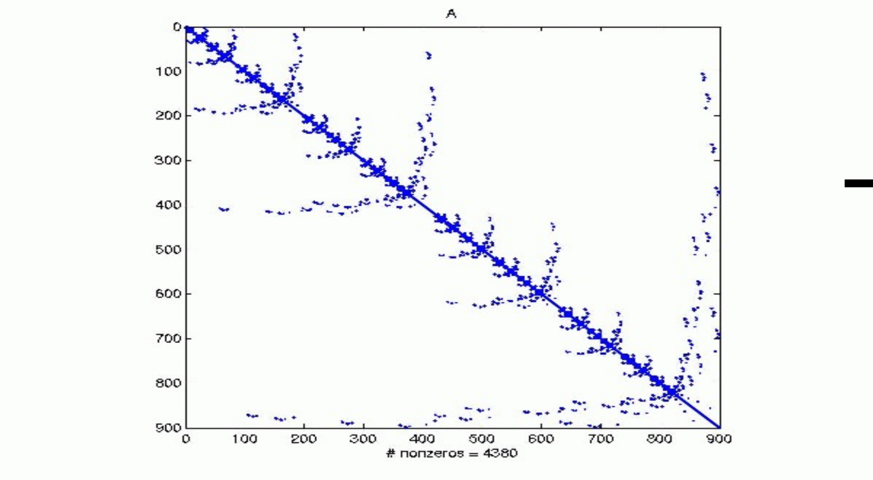
ND Ordering



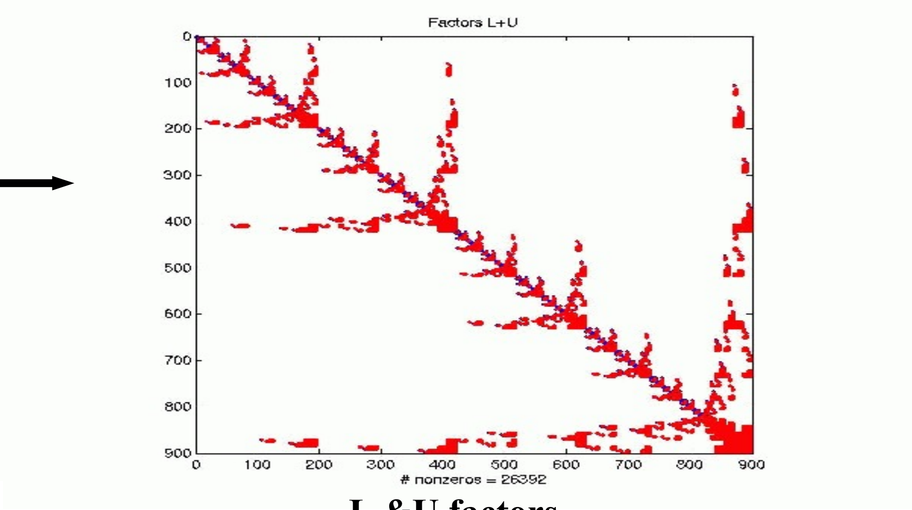
2D mesh



A, with row-wise ordering



A, with ND ordering



L &U factors

Ordering for LU with non-symmetric patterns

- 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 A^TA, but no need to compute A^TA

User-controllable options in SuperLU_DIST

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_MC64**, LargeDiag_HWPM, MY_PERMR }
- ColPerm: { NATURAL, MMD_ATA, MMD_AT_PLUS_A, COLAMD, **METIS_AT_PLUS_A,** PARMETIS, ZOLTAN, MY_PERMC }

Call set_default_options_dist(&options) to set default values.

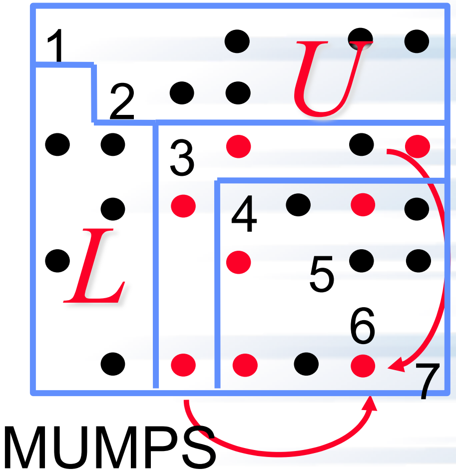
Algorithm variants, codes …. depending on matrix properties

Matrix properties	Supernodal (updates in-place)	Multifrontal (partial updates passing around)
Symmetric Pos. Def.: Cholesky LL' indefinite: LDL'	symPACK (DAG)	MUMPS (tree)
Symmetric pattern, non-symmetric value	PARDISO (DAG)	MUMPS (tree) STRUMPACK (binary tree)
Non-symmetric everything	SuperLU (DAG) PARDISO (DAG)	UMFPACK (DAG)

- Remarks:
    - SuperLU, MUMPS, UMFPACK can use any sparsity-reducing ordering
    - STRUMPACK can only use nested dissection (restricted to binary tree)

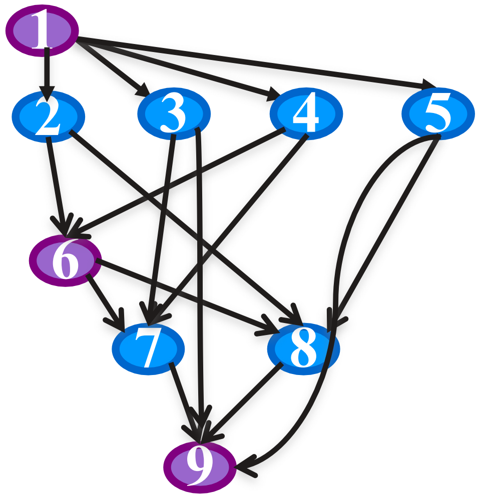
- Survey of sparse direct solvers (codes, algorithms, parallel capability):
  https://portal.nersc.gov/project/sparse/superlu/SparseDirectSurvey.pdf

Sparse LU: two algorithm variants



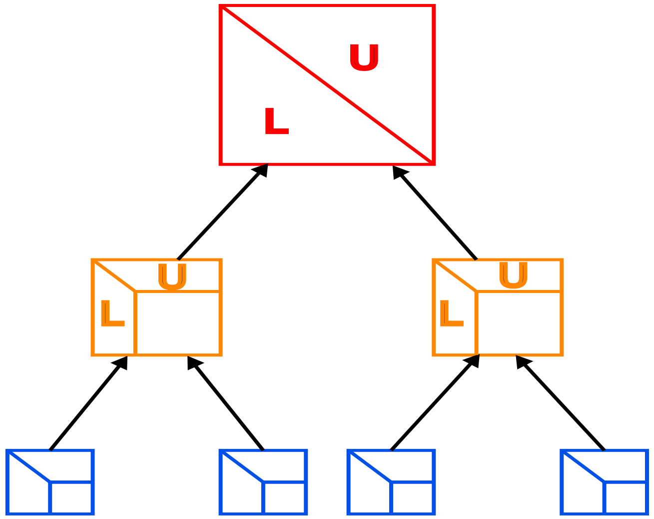
… depending on how updates are accumulated

DAG based
Supernodal: SuperLU



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

Tree based
Multifrontal: STRUMPACK, MUMPS

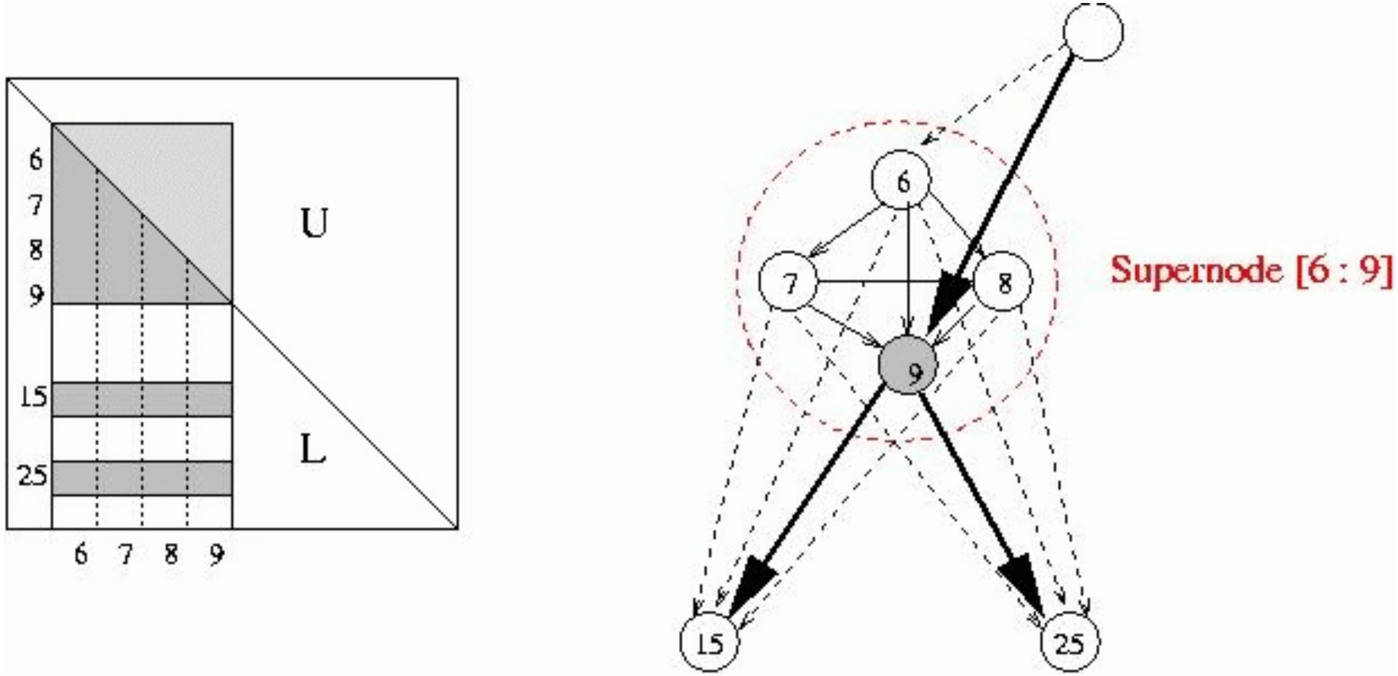


$S^{(j)} \leftarrow A^{(j)} - (..(D^{(k1)} + D^{(k2)}) + \dots)$

Supernode

Exploit dense submatrices in the factors

- Can use Level 3 BLAS
- Reduce inefficient indirect addressing (scatter/gather)
- Reduce graph traversal time using a coarser graph

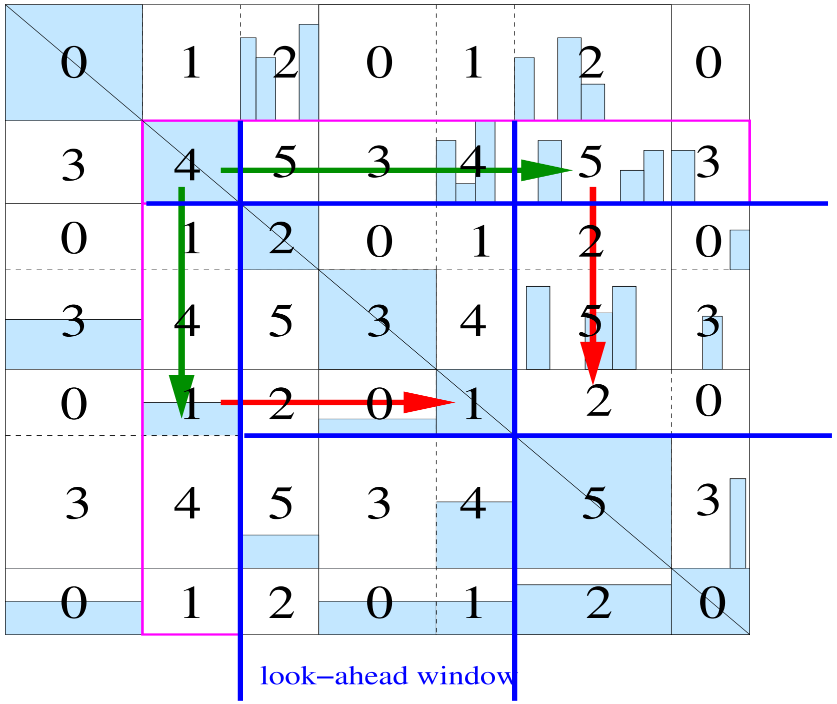


Distributed L & U factored matrices (internal to SuperLU)

- 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.

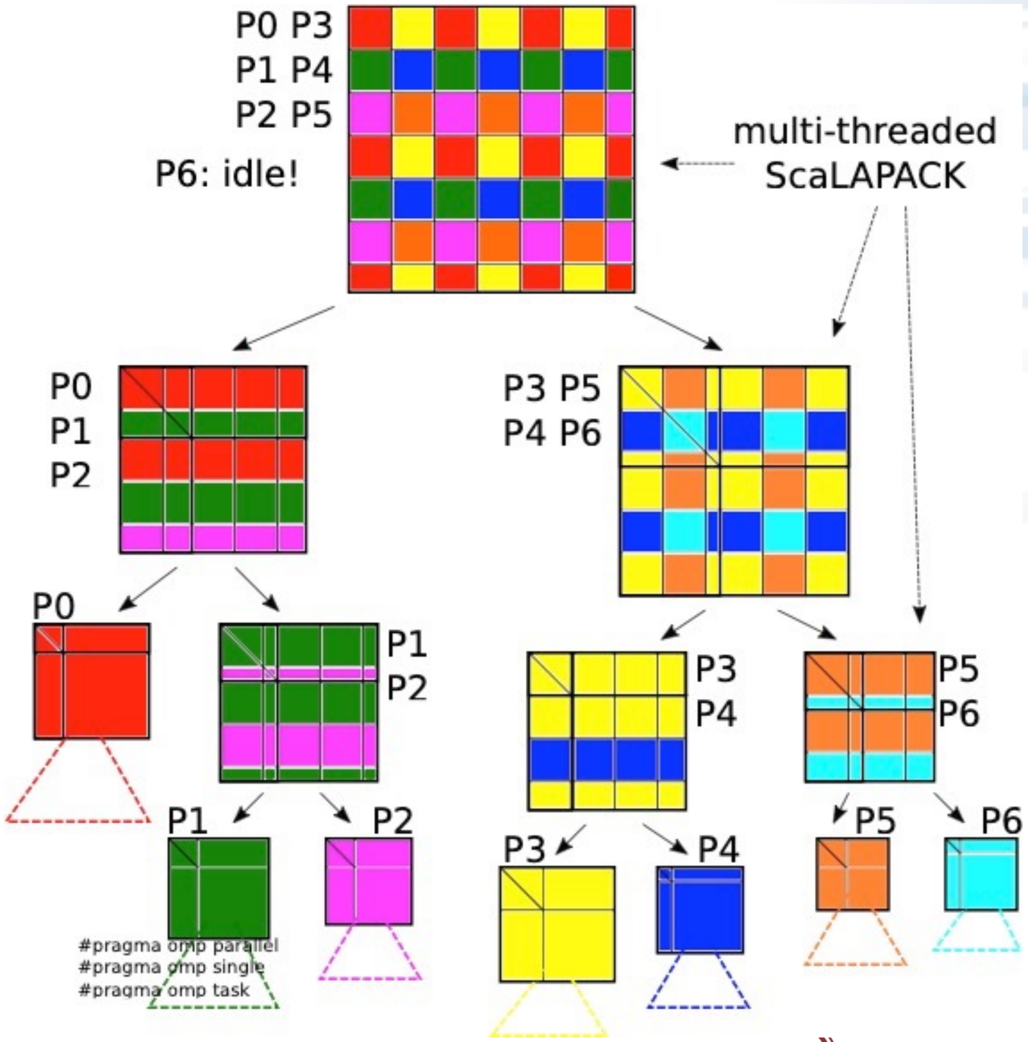


MPI Process Grid

0	1	2
3	4	5

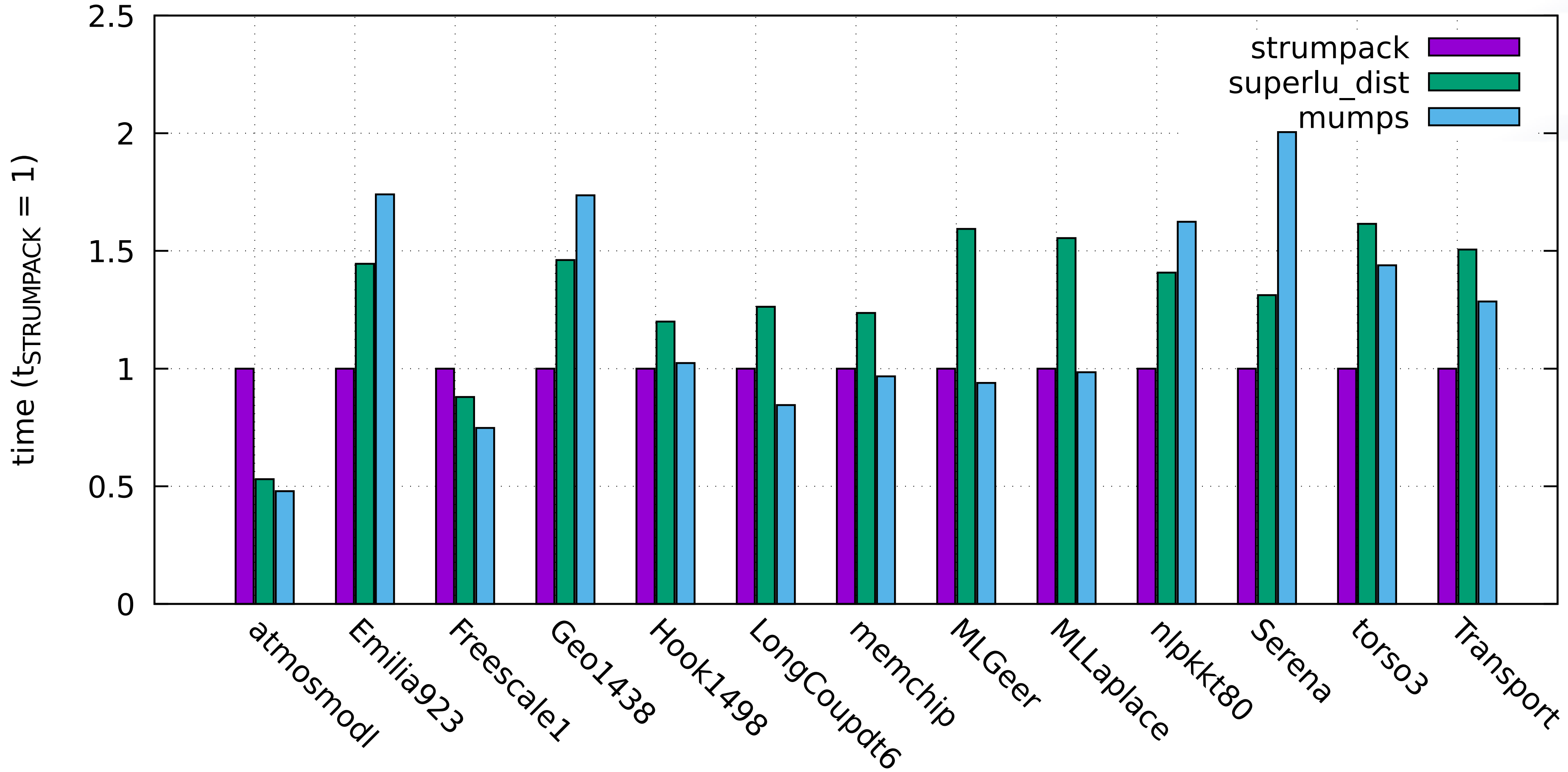
Distributed separator-tree-based parallelism (internal to STRUMPACK)

- Supernode = separator = frontal matrix
- Map sub-tree to sub-process grid
  - Proportional to estimated work
- ScaLAPACK 2D block cyclic layout at each node
- Multi-threaded ScaLAPACK through system MT-BLAS
- Allow idle processes for better communication
  - e.g.: 2x3 process grid is better than 1x7



Comparison of LU time from 3 direct solvers

- Pure MPI on 8 nodes Intel Ivy Bridge, 192 cores (2x12 cores / node), NERSC Edison
- METIS ordering



SuperLU_DIST recent improvements

- GPU
- Communication avoiding & hiding

SpLU	2D algorithm (baseline) →	+ GPU off-load (master) **3x**	
	↓ 3D Comm-Avoiding **27x** @ 32,000 cores →	**3.5x** @ 4096 Titan nodes (Version-7)	
SpTRSV	2D algorithm (baseline) →	GPU (gpu_trisolve) **8.5x** @1 Summit GPU →	1-sided MPI (trisolve-fompi) **2.4x** @12,000 KNL cores
	↓ 3D Comm-Avoiding **7x** @ 12,000 cores		

EXASCALE COMPUTING PROJECT

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**

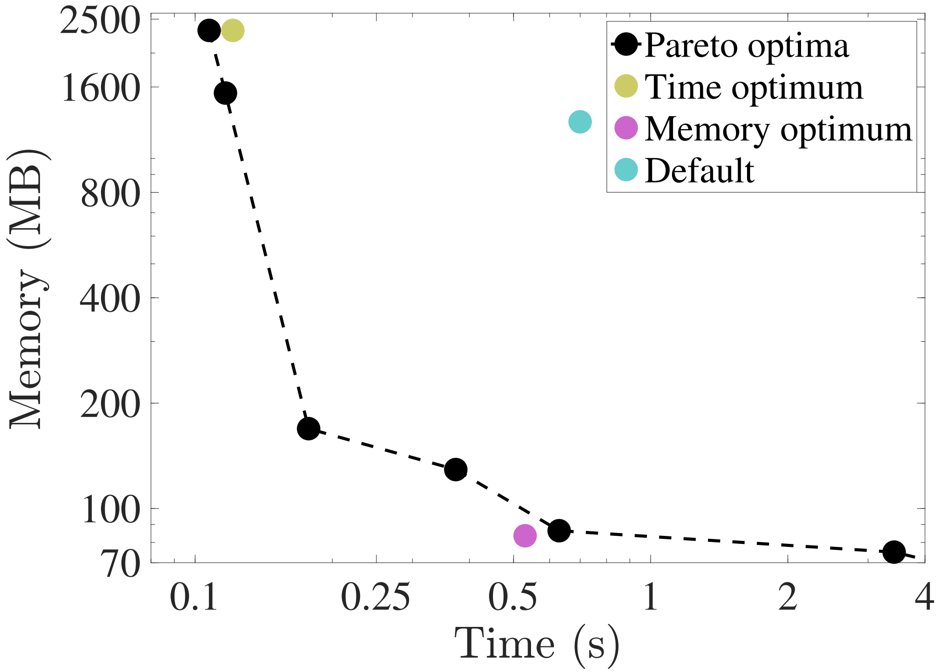
    (Parameters modifiable in routine sp_ienv())
    - **Larger blocks better for uniprocessor**
    - **Smaller blocks better for parallellism and load balance**

- **GPTune:** ML algorithms for selection of best parameters
  - https://github.com/gptune/GPTune/

GPTune: multi-objective autotuning for SuperLU_DIST

https://github.com/gptune/GPTune/

- $\mathbb{IS} = [\text{matrix name}], \quad \mathbb{PS} = [\text{COLPERM, NSUP, NREL, nprow}],$
- Multi-objective: $\mathbb{OS} = [\text{time, memory}]$
  Single-objective: $\mathbb{OS} = [\text{time}]$ or $[\text{memory}]$
- Returns multiple tuning parameter configurations.
- Pareto optimal: best time and memory tradeoff (no other $\mathbb{PS}$ points dominate over this point in both objectives)



matrix ”Si2”

Algorithm complexity (in bigO sense)

- Dense LU: $O(N^3)$
- Model PDEs with regular mesh, nested dissection ordering

	2D problems $N = k^2$			**3D problems $N = k^3$**		
	Factor flops	Solve flops	Memory	Factor flops	Solve flops	Memory
Exact sparse LU	$N^{3/2}$	N log(N)	N log(N)	N^2	$N^{4/3}$	$N^{4/3}$
STRUMPACK with low-rank compression	N	N	N	N^{α} polylog(N) ($\alpha < 2$)	N log(N)	N log(N)

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)
  - **$O(N^2)$ 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://portal.nersc.gov/project/sparse/superlu/**

- STRUMPACK: (inexact) direct solver, preconditioner.
  (P. Ghysels, L. Claus, Y. Liu, G. Chavez, 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 + CUDA; Provide Fortran interface.**
  - **Real, complex.**
  - **http://portal.nersc.gov/project/sparse/strumpack/**

References

- Short course, “Factorization-based sparse solvers and preconditioners”, 4th Gene Golub SIAM Summer School, 2013.https://archive.siam.org/students/g2s3/2013/index.html
  - **10 hours lectures, hands-on exercises**
  - **Extended summary: http://crd-legacy.lbl.gov/~xiaoye/g2s3-summary.pdf**

  (in book “Matrix Functions and Matrix Equations”, https://doi.org/10.1142/9590)

- SuperLU: portal.nersc.gov/project/sparse/superlu
- STRUMPACK: portal.nersc.gov/project/sparse/strumpack/
- ButterflyPACK: https://github.com/liuyangzhuan/ButterflyPACK

Rank-structured Approximate Factorizations in STRUMPACK

- “inexact” direct solvers
- strong preconditioners

SuperU_DIST Hands-on session

SuperLU_DIST with MFEM

xsdk-project.github.io/MathPackagesTraining2021/lessons/superlu_mfem/

Solve steady-state convection-diffusion equations

Get 1 compute node: qsub -I -n 1 -t 10 -A ATPESC2021 -q training

cd track-5-numerical/superlu/superlu_mfem_dist

- run 1: ./convdiff | tee run1.out
- run 2: ./convdiff --velocity 1000 | tee run2.out
- run 3: ./convdiff --velocity 1000 -slu -cp 0 | tee run3.out
- run 4: ./convdiff --velocity 1000 -slu -cp 2 | tee run4.out
- run 5: ./convdiff --velocity 1000 -slu -cp 4 | tee run5.out
- run 5.5: mpiexec -n 1 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 | tee run55.out
- run 6: mpiexec -n 12 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 | tee run6.out
- run 7: mpiexec -n 12 ./convdiff --refine 3 --velocity 1000 -slu -cp 4 -2rhs | tee run7.out

Summary of SuperLU_DIST with MFEM
xsdk-project.github.io/MathPackagesTraining2021/lessons/superlu_mfem/

- Convection-Diffusion equation (steady-state): convdiff.cpp
- 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.

SuperLU_DIST MPI + GPU
track-5-numerical/superlu/EXAMPLE

See README file (e.g. mpiexec -n 8 ./pddrive3d -r 2 -c 2 -d 2 stomach.rua)

$ export OMP_NUM_THREADS=1

MPI:

- run 1: export SUPERLU_ACC_OFFLOAD=0; mpiexec -n 1 pddrive3d stomach.rua | tee run1.out
- run 2: export SUPERLU_ACC_OFFLOAD=0; mpiexec -n 2 pddrive3d -c 2 stomach.rua | tee run2.out

+GPU:

- run 3: export SUPERLU_ACC_OFFLOAD=1; mpiexec -n 1 pddrive3d stomach.rua | tee run3.out
- run 4: export SUPERLU_ACC_OFFLOAD=1; mpiexec -n 2 pddrive3d -c 2 stomach.rua | tee run4.out

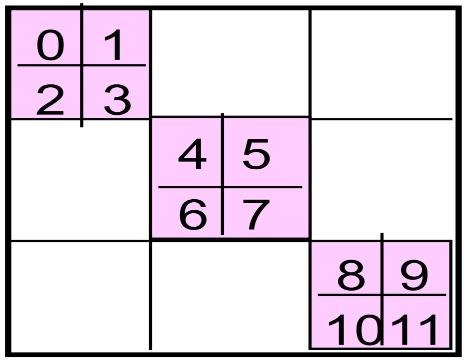
Factorization seconds	no GPU	w/ GPU
MPI = 1	23.7	8.3
MPI = 2	14.7	6.7

SuperLU_DIST other examples

track-5-numerical/superlu/EXAMPLE

See README file (e.g. mpiexec -n 12 ./pddrive1 -r 3 -c 4 stomach.rua)

- 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

track-5-numerical/superlu/EXAMPLE

Four input matrices:

- g4.rua (16 dofs)
- g20.rua (400 dofs)
- big.rua (4960 dofs)
- stomach.rua (213k dofs)

- Can get many other test matrices at SuiteSparse
  https://sparse.tamu.edu

Thank you!

Rank Structured Solvers for Dense Linear Systems

EXASCALE COMPUTING PROJECT

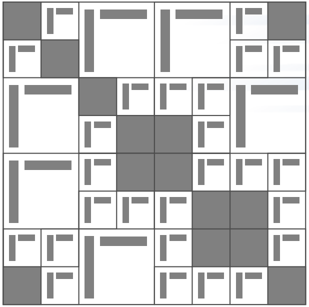
Hierarchical Matrix Approximation

$\mathcal{H}$-matrix representation [1]

- Data-sparse, rank-structured, compressed

Hierarchical/recursive 2×2 matrix blocking, with blocks either:

- Low-rank: $A_{IJ} \approx UV^\top$
- Hierarchical
- Dense (at lowest level)



Use cases:

- Boundary element method for integral equations
- Cauchy, Toeplitz, kernel, covariance, . . . matrices
- Fast matrix-vector multiplication
- $\mathcal{H}$-LU decomposition
- Preconditioning

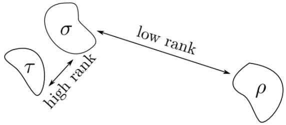
Hackbusch, W., 1999. *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices.* Computing, 62(2), pp.89-108.

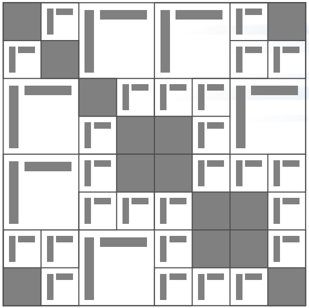
Admissibility Condition

- Row cluster σ
- Column cluster τ
- $\sigma \times \tau$ is compressible $\Leftrightarrow$

$$\frac{\max(\mathrm{diam}(\sigma), \mathrm{diam}(\tau))}{\mathrm{dist}(\tau, \sigma)} \leq \eta$$

  - $\mathrm{diam}(\sigma)$: diameter of physical domain corresponding to σ
  - $\mathrm{dist}(\sigma, \tau)$: distance between σ and τ





- Weaker interaction between clusters leads to smaller ranks
- Intuitively larger distance, greater separation, leads to weaker interaction
- Need to cluster and order degrees of freedom to reduce ranks

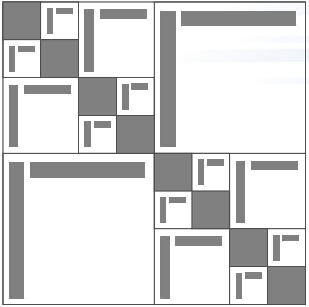
Hackbusch, W., 1999. *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices.* Computing, 62(2), pp.89-108.

HODLR: Hierarchically Off-Diagonal Low Rank

- Weak admissibility

$$\sigma \times \tau \text{ is compressible } \Leftrightarrow \sigma \neq \tau$$

  Every off-diagonal block is compressed as low-rank, even interaction between neighboring clusters (no separation)



Compared to more general $\mathcal{H}$-matrix

- Simpler data-structures: same row and column cluster tree
- More scalable parallel implementation
- Good for 1D geometries, e.g., boundary of a 2D region discretized using BEM or 1D separator
- Larger ranks

HSS: Hierarchically Semi Seperable

- Weak admissibility
- Off-diagonal blocks

$$A_{\sigma,\tau} \approx U_\sigma B_{\sigma,\tau} V_\tau^\top$$

- Nested bases

$$U_\sigma = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \hat{U}_\sigma$$

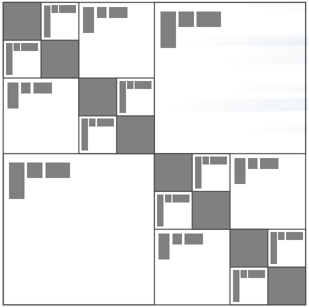
with ν_1 and ν_2 children of σ in the cluster tree.

- At lowest level

$$U_\sigma \equiv \hat{U}_\sigma$$

- Store only $\hat{U}_\sigma$, smaller than U_σ
- Complexity $\mathcal{O}(N) \leftrightarrow \mathcal{O}(N \log N)$ for HODLR
- HSS is special case of $\mathcal{H}^2$: $\mathcal{H}$ with nested bases

$$\begin{bmatrix} \begin{matrix} D_0 & U_0 B_{0,1} V_1^* \\ U_1 B_{1,0} V_0^* & D_1 \end{matrix} & U_2 B_{2,5} V_5^* \\ U_5 B_{5,2} V_2^* & \begin{matrix} D_3 & U_3 B_{3,4} V_4^* \\ U_4 B_{4,3} V_3^* & D_4 \end{matrix} \end{bmatrix}$$



HSS: Hierarchically Semi Seperable

- Weak admissibility
- Off-diagonal blocks

$$A_{\sigma,\tau} \approx U_\sigma B_{\sigma,\tau} V_\tau^\top$$

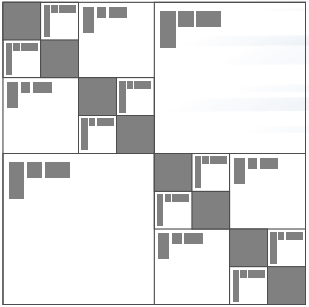
- Nested bases

$$U_\sigma = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \hat{U}_\sigma$$

  with ν_1 and ν_2 children of σ in the cluster tree.
- At lowest level

$$U_\sigma \equiv \hat{U}_\sigma$$

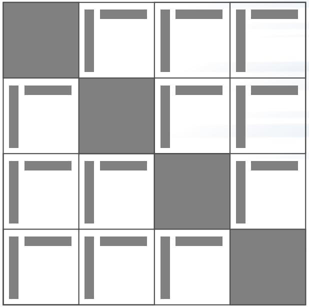
- Store only $\hat{U}_\sigma$, smaller than U_σ
- Complexity $\mathcal{O}(N) \leftrightarrow \mathcal{O}(N \log N)$ for HODLR
- HSS is special case of $\mathcal{H}^2$: $\mathcal{H}$ with nested bases



$$\begin{bmatrix} \begin{matrix} D_0 & U_0 B_{0,1} V_1^* \\ U_1 B_{1,0} V_0^* & D_1 \end{matrix} & \begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix} \hat{U}_2 B_{2,5} \hat{V}_5^* \begin{bmatrix} V_3^* & 0 \\ 0 & V_4^* \end{bmatrix} \\ \begin{bmatrix} U_3 & 0 \\ 0 & U_4 \end{bmatrix} \hat{U}_5 B_{5,2} \hat{V}_2^* \begin{bmatrix} V_0^* & 0 \\ 0 & V_1^* \end{bmatrix} & \begin{matrix} D_3 & U_3 B_{3,4} V_4^* \\ U_4 B_{4,3} V_3^* & D_4 \end{matrix} \end{bmatrix}$$

BLR: Block Low Rank [1, 2]

- Flat partitioning (non-hierarchical)
- Weak or strong admissibility
- Larger asymptotic complexity than $\mathcal{H}$, HSS, . . .
- Works well in practice



Mary, T. (2017). *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* (Doctoral dissertation).

Amestoy, Patrick, et al. (2015). *Improving multifrontal methods by means of block low-rank representations.* SISC 37.3 : A1451-A1474.

Data-Sparse Matrix Representation Overview

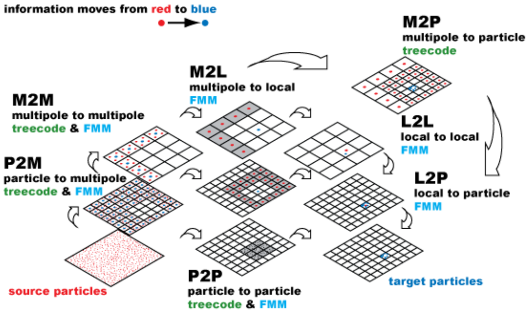
$\mathcal{H}$ HODLR HSS BLR

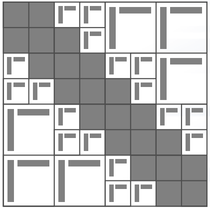
- Partitioning: **hierarchical** ($\mathcal{H}$, HODLR, HSS) or **flat** (BLR)
- Admissibility: **weak** (HODLR, HSS) or **strong** ($\mathcal{H}$, $\mathcal{H}^2$)
- Bases: **nested** (HSS, $\mathcal{H}^2$) or **not nested** (HODLR, $\mathcal{H}$, BLR)

Fast Multipole Method [1]

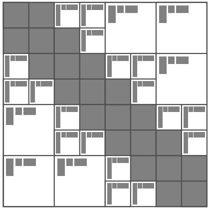
Particle methods like Barnes-Hut and FMM can be interpreted algebraically using hierarchical matrix algebra

- Barnes-Hut $\mathcal{O}(N \log N)$
- Fast Multipole Method $\mathcal{O}(N)$





Barnes-Hut

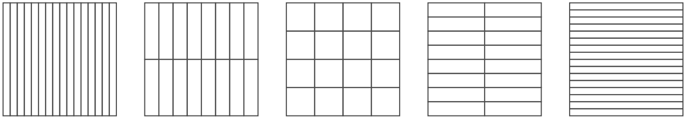


FMM

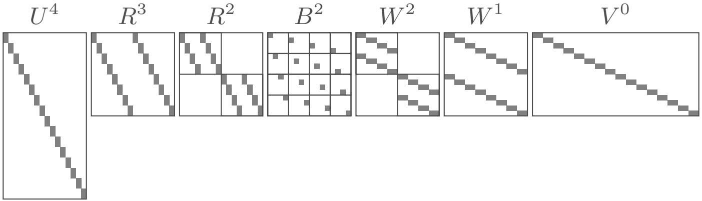
[1] Greengard, L., and Rokhlin, V. *A fast algorithm for particle simulations.* Journal of computational physics 73.2 (1987): 325-348.

Butterfly Decomposition [1]

Complementary low rank property: sub-blocks of size $\mathcal{O}(N)$ are low rank:



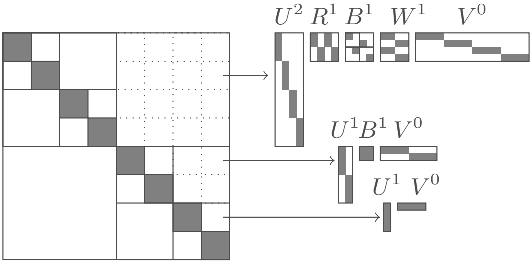
Multiplicative decomposition:



- Multilevel generalization of low rank decomposition
- Based on FFT ideas, motivated by high-frequency problems

Michielssen, E., and Boag, A. *Multilevel evaluation of electromagnetic fields for the rapid solution of scattering problems.* Microwave and Optical Technology Letters 7.17 (1994): 790-795.

HODBF: Hierarchically Off-Diagonal Butterfly



- HODLR but with low rank replaced by Butterfly decomposition
- Reduces ranks of large off-diagonal blocks

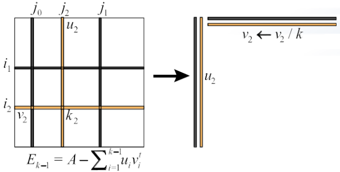
Low Rank Approximation Techniques

Traditional approaches need entire matrix

- Truncated Singular Value Decomposition (TSVD): $A \approx U\Sigma V^T$
  - Optimal, but expensive
- Column Pivoted QR: $AP \approx QR$
  - Less accurate than TSVD, but cheaper

Adaptive Cross Approximation

- No need to compute every element of the matrix
- Requires certain assumptions on input matrix
- Left-looking LU with rook pivoting



Randomized algorithms [1]

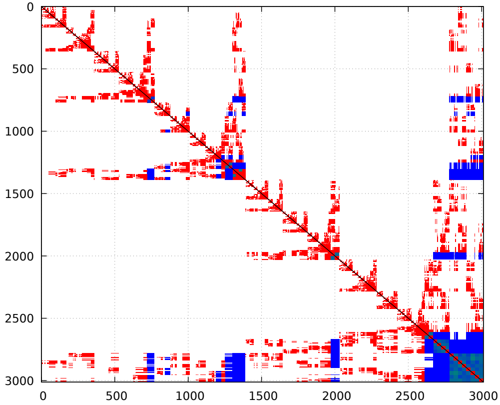
- Fast matrix-vector product: $S = A\Omega$
  Reduce dimension of A by random projection with Ω
- E.g., operator is sparse or rank structured, or the product of sparse and rank structured

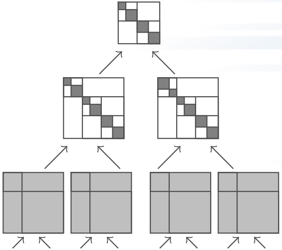
Halko, N., Martinsson, P.G., Tropp, J.A. (2011). *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.* SIAM Review, 53(2), 217-288.

Approximate Multifrontal Factorization

Sparse Multifrontal Solver/Preconditioner with Rank-Structured Approximations

L and U factors, after nested-dissection ordering,
compressed blocks in blue

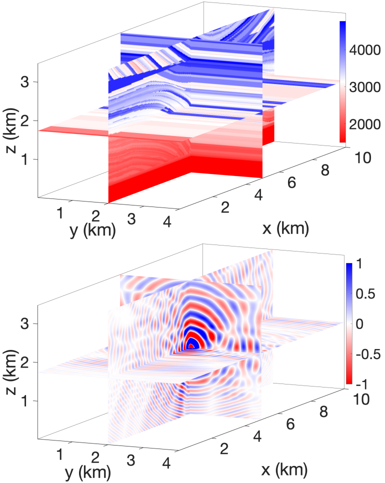




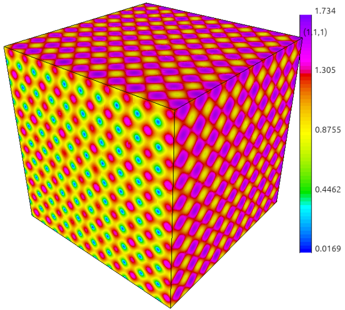
Only apply rank structured compression to largest fronts (dense sub-blocks), keep the rest as regular dense

High Frequency Helmholtz and Maxwell

Regular $k^3 = N$ grid, fixed number of discretization points per wavelength



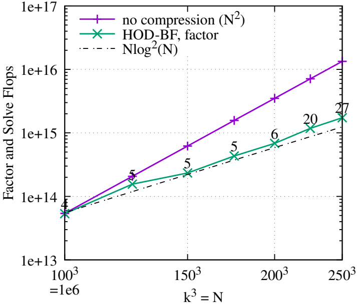
Marmousi2 geophyical elastic dataset



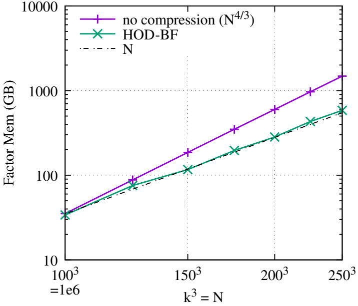
Indefinite Maxwell, using MFEM

High Frequency Helmholtz and Maxwell

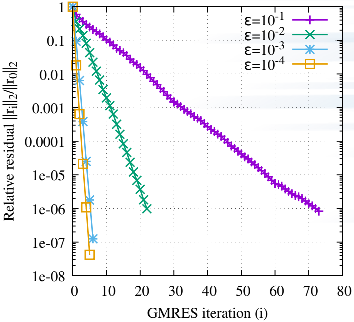
Sparse multifrontal solver with HODBF compression



Operations for factor and solve phases, $\varepsilon = 10^{-3}$.



Memory usage for the sparse triangular factors.



GMRES convergence for $k = 200$.

- Highly oscillatory problems are hard for iterative solvers
- Typically solved with sparse direct solvers, but scale as $\mathcal{O}(N^2)$

EXASCALE COMPUTING PROJECT

Software: ButterflyPACK

- Butterfly
- Hierarchically Off-Diagonal Low Rank (HODLR)
- Hierarchically Off-Diagonal Butterfly (HODBF)
- Hierarchical matrix format ($\mathcal{H}$)
  - Limited parallelism

- Fast compression, using randomization
- Fast multiplication, factorization & solve
- Fortran2008, MPI, OpenMP

`https://github.com/liuyangzhuan/ButterflyPACK`

Software: STRUMPACK

STRUctured Matrix PACKage

- Fully algebraic solvers/preconditioners
- Sparse direct solver (multifrontal LU factorization)
- Approximate sparse factorization preconditioner
- Dense
  - HSS: Hierarchically Semi-Separable
  - BLR: Block Low Rank (sequential only)
  - ButterflyPACK integration/interface:
    - Butterfly
    - HODLR
    - HODBF
- C++, MPI + OpenMP + CUDA, real & complex, 32/64 bit integers
- BLAS, LAPACK, Metis
- Optional: MPI, ScaLAPACK, ParMETIS, (PT-)Scotch, cuBLAS/cuSOLVER, SLATE, ZFP

https://github.com/pghysels/STRUMPACK
https://portal.nersc.gov/project/sparse/strumpack/master/

Other Available Software

HiCMA	https://github.com/ecrc/hicma
HLib	http://www.hlib.org/
HLibPro	https://www.hlibpro.com/
H2Lib	http://www.h2lib.org/
HACApK	https://github.com/hoshino-UTokyo/hacapk-gpu
MUMPS	http://mumps.enseeiht.fr/
PaStiX	https://gitlab.inria.fr/solverstack/pastix
ExaFMM	http://www.bu.edu/exafmm/

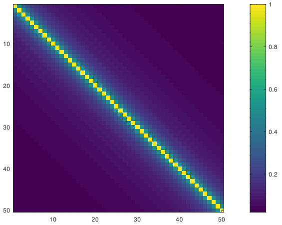
See also:
https://github.com/gchavez2/awesome_hierarchical_matrices

STRUMPACK Hands-On Session

EXASCALE COMPUTING PROJECT

HODLR Compression of Toeplitz Matrix $T(i,j) = \frac{1}{1+|i-j|}$

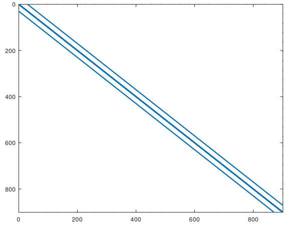
track-5-numerical/rank_structured_strumpack/build/testHODLR



- See `track-5-numerical/rank_structured_strumpack/README`
- Get a compute node:
  ```
  qsub -I -n 1 -t 30 -A ATPESC2021 -q training
  ```
- Set OpenMP threads:
  ```
  export OMP_NUM_THREADS=1
  ```
- Run example:
  ```
  mpiexec -n 1 ./build/testHODLR 20000
  ```
- With description of command line parameters:
  ```
  mpiexec -n 1 ./build/testHODLR 20000 --help
  ```
- Vary leaf size (smallest block size) and tolerance:
  ```
  mpiexec -n 1 ./build/testHODLR 20000 --hodlr_rel_tol 1e-4 --hodlr_leaf_size 16
  mpiexec -n 1 ./build/testHODLR 20000 --hodlr_rel_tol 1e-4 --hodlr_leaf_size 128
  ```
- Vary number of MPI processes:
  ```
  mpiexec -n 12 ./build/testHODLR 20000 --hodlr_rel_tol 1e-8 --hodlr_leaf_size 16
  mpiexec -n 12 ./build/testHODLR 20000 --hodlr_rel_tol 1e-8 --hodlr_leaf_size 128
  ```

Solve a Sparse Linear System with Matrix `pde900.mtx`

track-5-numerical/rank_structured_strumpack/build/testMMdouble{MPIDist}



- See `track-5-numerical/rank_structured_strumpack/README`
- Get a compute node:
  ```
  qsub -I -n 1 -t 30 -A ATPESC2021 -q training
  ```
- Set OpenMP threads: `export OMP_NUM_THREADS=1`
- Run example:
  ```
  mpiexec -n 1 ./build/testMMdouble pde900.mtx
  ```
- With description of command line parameters:
  ```
  mpiexec -n 1 ./build/testMMDouble pde900.mtx --help
  ```
- Enable/disable GPU off-loading:
  ```
  mpiexec -n 1 ./build/testMMDouble pde900.mtx --sp_disable_gpu
  ```
- Vary number of MPI processes:
  ```
  mpiexec -n 1 ./build/testMMdouble pde900.mtx
  mpiexec -n 12 ./build/testMMdoubleMPIDist pde900.mtx
  ```
- Other sparse matrices, in matrix market format:
  NIST Matrix Market: `https://math.nist.gov/MatrixMarket`
  SuiteSparse: `http://faculty.cse.tamu.edu/davis/suitesparse.html`

Solve 3D Poisson Problem

track-5-numerical/rank_structured_strumpack/build/testPoisson3d{MPIDist}

- See `track-5-numerical/rank_structured_strumpack/README`
- Get a compute node: `qsub -I -n 1 -t 30 -A ATPESC2021 -q training`
- Set OpenMP threads: `export OMP_NUM_THREADS=1`

- Solve 40^3 Poisson problem:

```
mpiexec -n 1 ./build/testPoisson3d 40 --help --sp_disable_gpu
```

- Enable BLR compression (sequential):

```
mpiexec -n 1 ./build/testPoisson3d 40 --sp_compression BLR --help
mpiexec -n 1 ./build/testPoisson3d 40 --sp_compression BLR --blr_rel_tol 1e-2
mpiexec -n 1 ./build/testPoisson3d 40 --sp_compression BLR --blr_rel_tol 1e-4
mpiexec -n 1 ./build/testPoisson3d 40 --sp_compression BLR --blr_leaf_size 128
mpiexec -n 1 ./build/testPoisson3d 40 --sp_compression BLR --blr_leaf_size 256
```

- Parallel, with HSS/HODLR compression:

```
mpiexec -n 12 ./build/testPoisson3dMPIDist 40
mpiexec -n 12 ./build/testPoisson3dMPIDist 40 --sp_compression HSS \
     --sp_compression_min_sep_size 1000 --hss_rel_tol 1e-2
mpiexec -n 12 ./build/testPoisson3dMPIDist 40 --sp_compression HODLR \
     --sp_compression_min_sep_size 1000 --hodlr_leaf_size 128
```