ARGONNE ATPESC2024 EXTREME-SCALE COMPUTING

Direct Sparse Linear Solvers, Preconditioners

SuperLU, STRUMPACK, with hands-on examples

Pieter Ghysels, AMD
Sherry Li, Lawrence Berkeley National Laboratory

Agenda [11:15 – 12:30]

Setup for SuperLU hands-on (5 min)

Overview of sparse direct solvers + SuperLU (30 min)

Setup for STRUMPACK hands-on (5 min)

STRUMPACK with compression techniques (30 min)

polaris setup

https://xsdk-project.github.io/MathPackagesTraining2024/setup_instructions/

- Copy all examples to your home:

cd ~

rsync -a /eagle/ATPESC2024/EXAMPLES/track-5-numerical .

- Get a single GPU node

qsub -I -l select=1 -l filesystems=home:eagle -l walltime=1:00:00 -q ATPESC -A ATPESC2024

- Follow SuperLU lesson at:

https://xsdk-project.github.io/MathPackagesTraining2024/lessons/superlu_dist

SpLU tiime on Polaris (AMD EPYC 7543P, NVIDIA A100)

export matdir=/eagle/ATPESC2024/usr/MathPackages/datafiles

export OMP_NUM_THREADS=1

- 3D algorithm: mpiexec -n 2 pddive3d -r 1 -c 1 -d 2 ${matdir}/<matrix file> | tee output
  - Offload GEMM and Scatter in Schur-complement, panel factor still on CPU
- 2D algorithm: mpiexec -n 2 pddrive -r 1 -c 2 ${matdir}/<matrix file> | tee output
  - Only offload GEMM

		3D code 1x1x1	3D code 1x1x2	2D code 1x1	2D code 1x2
Li4244.bin	CPU (SUPERLU_ACC_OFFLOAD=0) +GPU	201.0 3.2	118.6 1.8	198.8 130.9	103.2 73.2
nd24k.mtx	CPU (SUPERLU_ACC_OFFLOAD=0) +GPU	167.2 4.1	110.9 --	178.1 152.7	97.8 87.0

Algorithm tour of sparse direct solvers

(illustration with SuperLU_DIST)

Gaussian Elimination (GE) to solve Ax=b

- 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

Growth factor:

$$g_n = \frac{max_{i,j,k} \left| a_{i,j}^{(k)} \right|}{max_{i,j} \left| a_{i,j} \right|} \leq 2^{n-1}$$

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

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: combine structural and data sparsity

PDE discretization leads to sparse matrices

- Poisson equation in 2D (continuum)

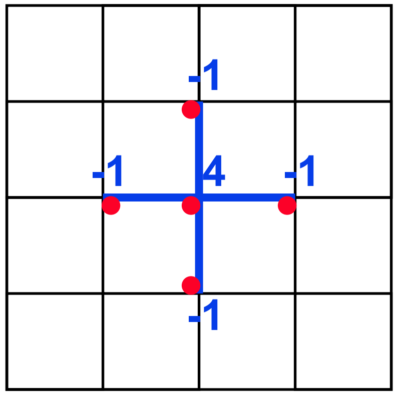
$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y), \quad (x,y) \in R$$

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

- 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"



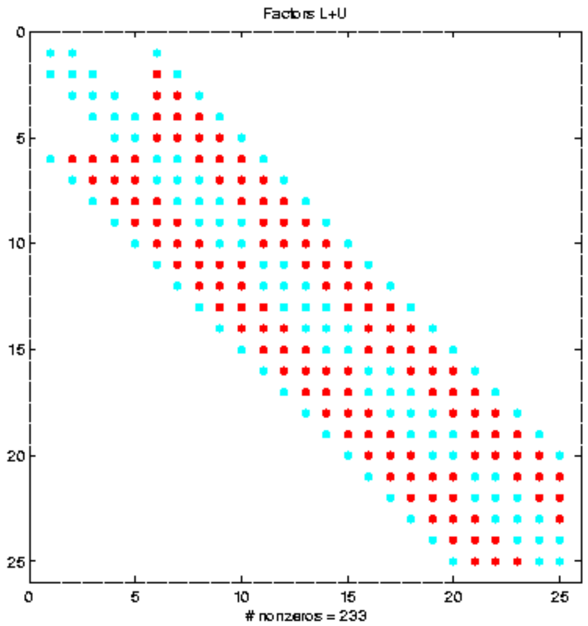
$$A = \left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)$$

Fill-in in Sparse GE

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

– Red: fill-ins (Matlab: spy())

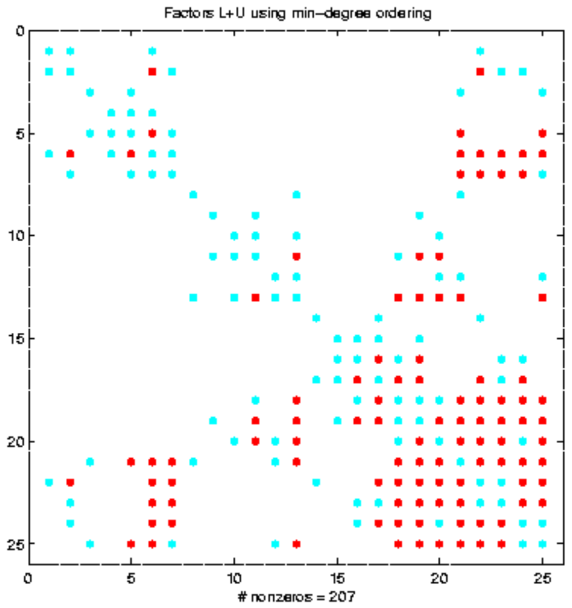
Natural order: NNZ = 233



Band solver

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

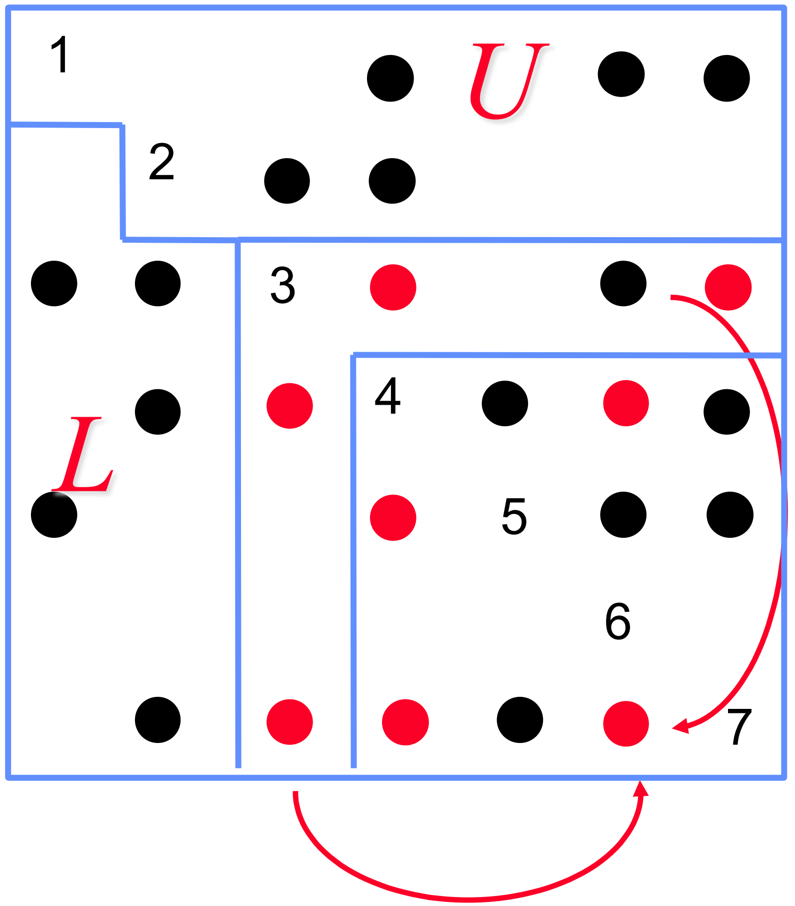
Minimum Degree order: NNZ = 207



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**

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

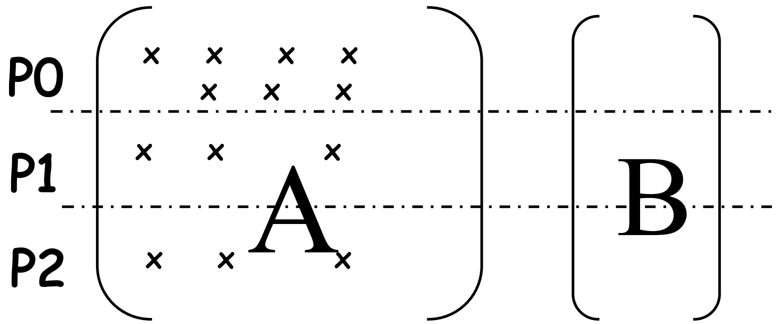
	1	3	5	8	11	13	17	20
nzval	1 a	2 b	c d 3	e 4 f	5 g	h i 6 j	k l 7	
colind	1 4	2 5	1 2 3	2 4 5	5 7	4 5 6 7	3 5 7	
rowptr	1 3 5 8 11 13 17 20							

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

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   nnz_loc;  // number of nonzeros in the local submatrix
    int   m_loc;    // number of rows local to this processor
    int   fst_row;  // global index of the first row
    void  *nzval;   // pointer to array of nonzero values, packed by row
    int   *colind;  // pointer to array of column indices of the nonzeros
    int   *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

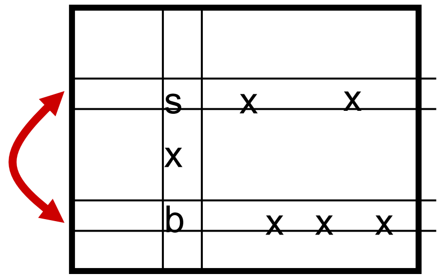
4. Perform factorization and triangular solutions (~80% time)
    - Numerical algorithms (F.P. operations only on nonzeros)
    - Usually dominate the total runtime

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

Numercial 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)
  1. Before factor, scale and permute A to maximize diagonal: $P_r D_r A D_c = A'$
  2. During factor A' = LU, replace tiny pivots by $\sqrt{\varepsilon}\|A\|$, w/o changing data structures for L & U
  3. If needed, use a few steps of iterative refinement after the first solution

- **Restricted pivoting**
- …

Can we reduce fill? -- various ordering algorithms

- Reordering (= permutation of equations and variables)

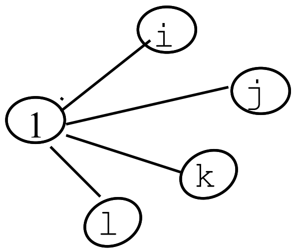
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & & & \\ 3 & & 3 & & \\ 4 & & & 4 & \\ 5 & & & & 5 \end{pmatrix} \text{(all filled after elimination)}$$

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

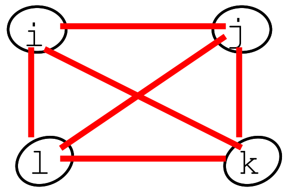
(no fill after elimination)

Ordering to preserve sparsity : Minimum Degree

$$\begin{array}{cc} & \begin{array}{ccccc} & i & j & k & l \end{array} \\ \begin{array}{c} 1\\ i\\ j\\ k\\ l \end{array} & \begin{bmatrix} x & x & x & x & x \\ x & & & & \\ x & & & & \\ x & & & & \\ x & & & & \end{bmatrix} \end{array} \xrightarrow{\text{Eliminate 1}} \begin{array}{cc} & \begin{array}{ccccc} & i & j & k & l \end{array} \\ \begin{array}{c} 1\\ i\\ j\\ k\\ l \end{array} & \left[\begin{array}{c|cccc} x & x & x & x & x \\ \hline x & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet \end{array}\right] \end{array}$$



Eliminate 1 →



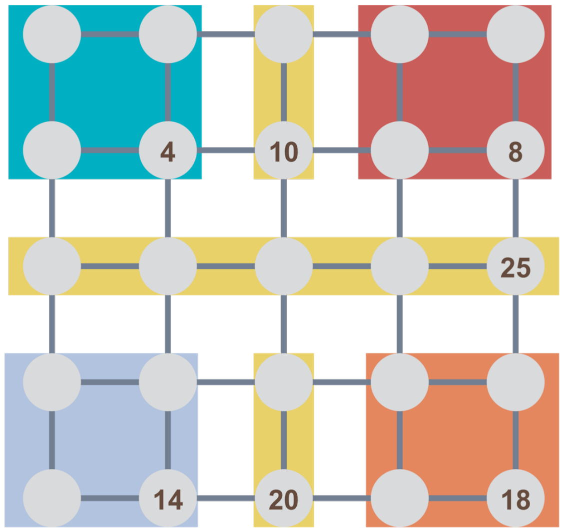
- Local greedy strategy: minimize upper bound on fill-in at each elimination step
- Algorithm: Repeat N steps:
  - Choose a vertex with minimum degree to eliminate
  - Update the remaining graph

Fast implementation: Quotient graph, approximate degree

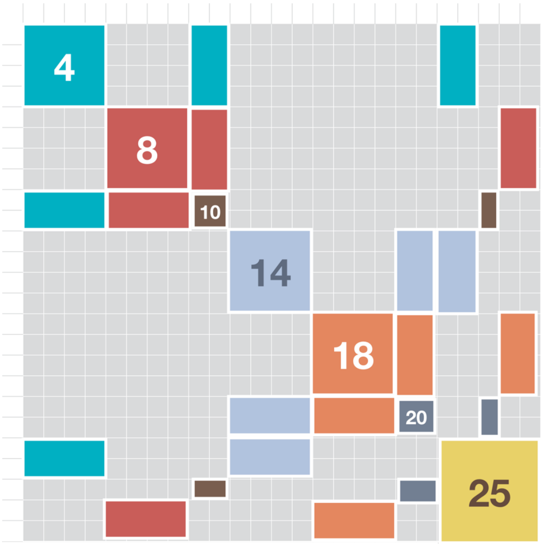
Ordering to preserve sparsity : Nested Dissection

- 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]

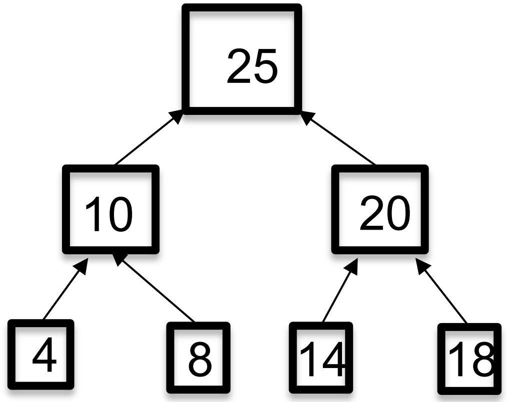
Geometry



Reordered Matrix

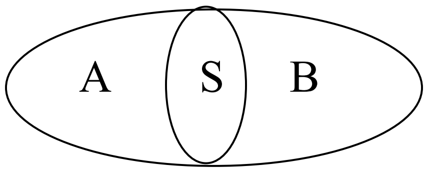


Separator Tree



ND Ordering

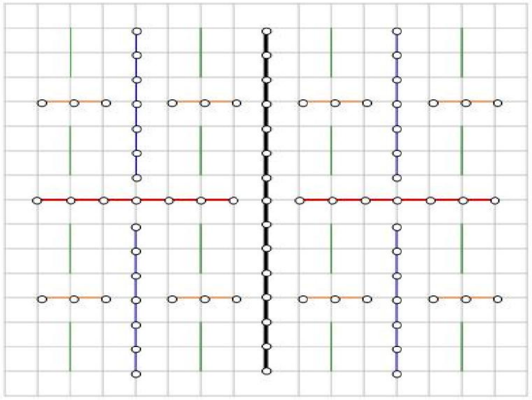
- Generalized nested dissection [Lipton/Rose/Tarjan '79]
  - **Global graph partitioning: top-down, divide-and-conqure**
  - **Best for large problems**
  - **Parallel codes available: ParMetis, PT-Scotch**
    - First level



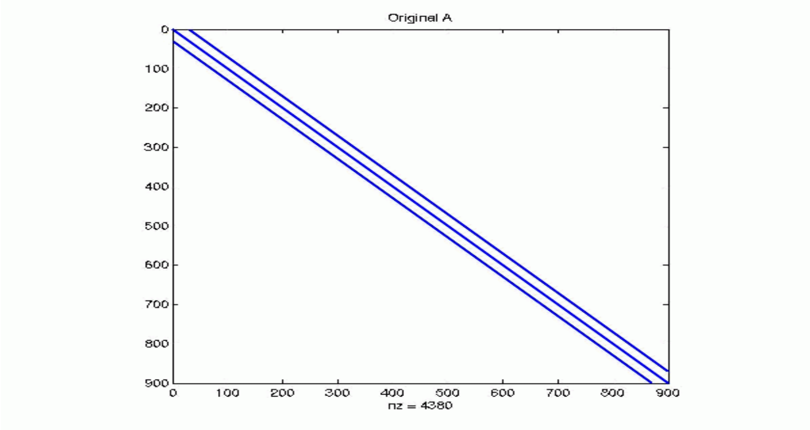
$$\begin{bmatrix} A & 0 & x \\ 0 & B & x \\ x & x & S \end{bmatrix}$$

    - 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]**

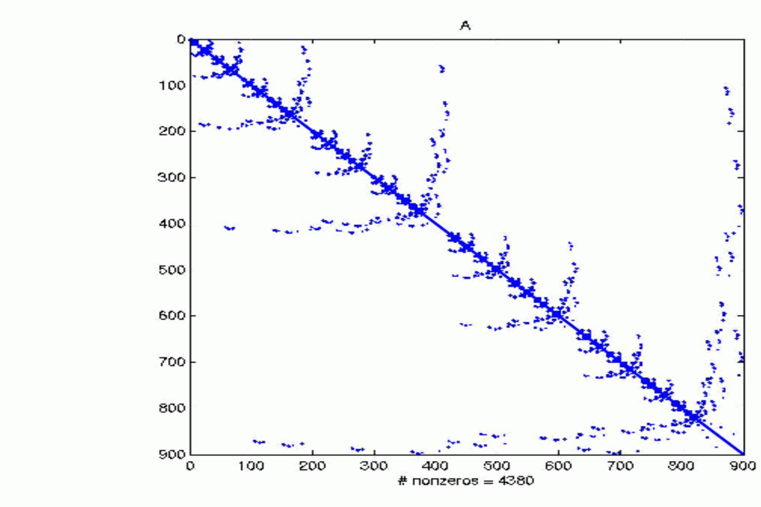
ND Ordering



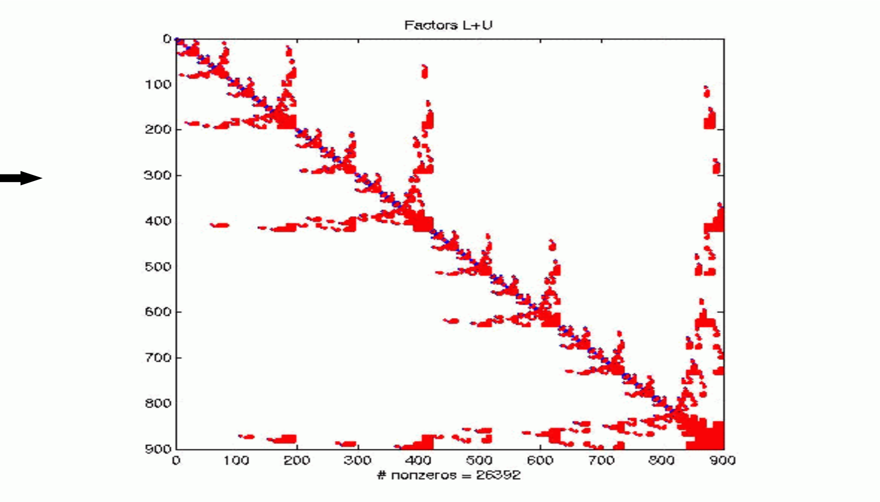
2D mesh



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 graph of 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

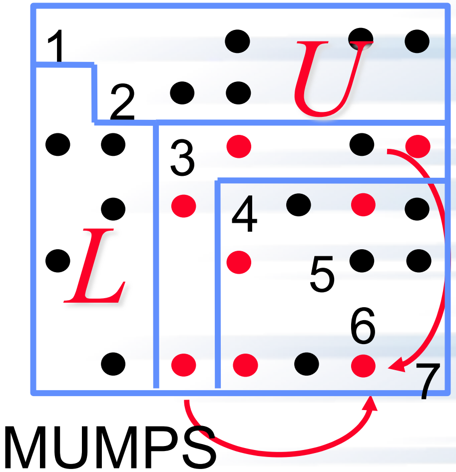
Algorithm variants, codes …. depending on matrix properties

Matrix properties	Supernodal (updates in-place)	Multifrontal (partial updates passing to later)
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

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

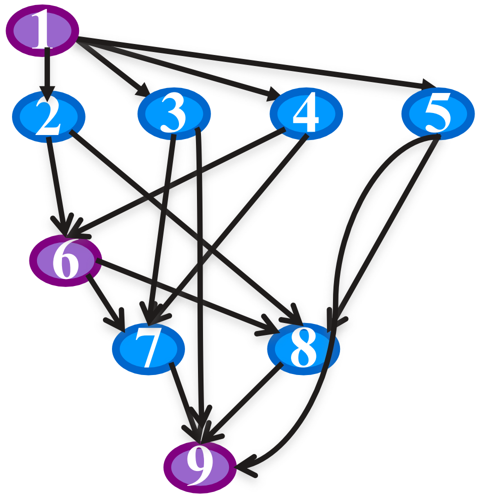
Sparse LU: two algorithm variants



… depending on how updates are accumulated

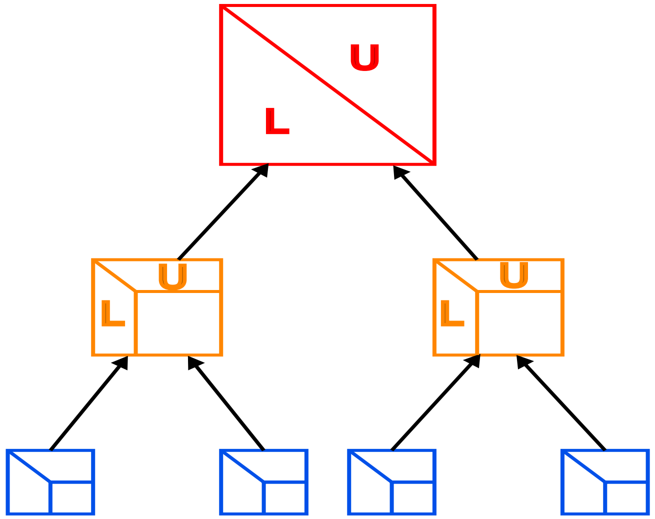
DAG based

Supernodal: SuperLU



Tree based

Multifrontal: STRUMPACK, MUMPS



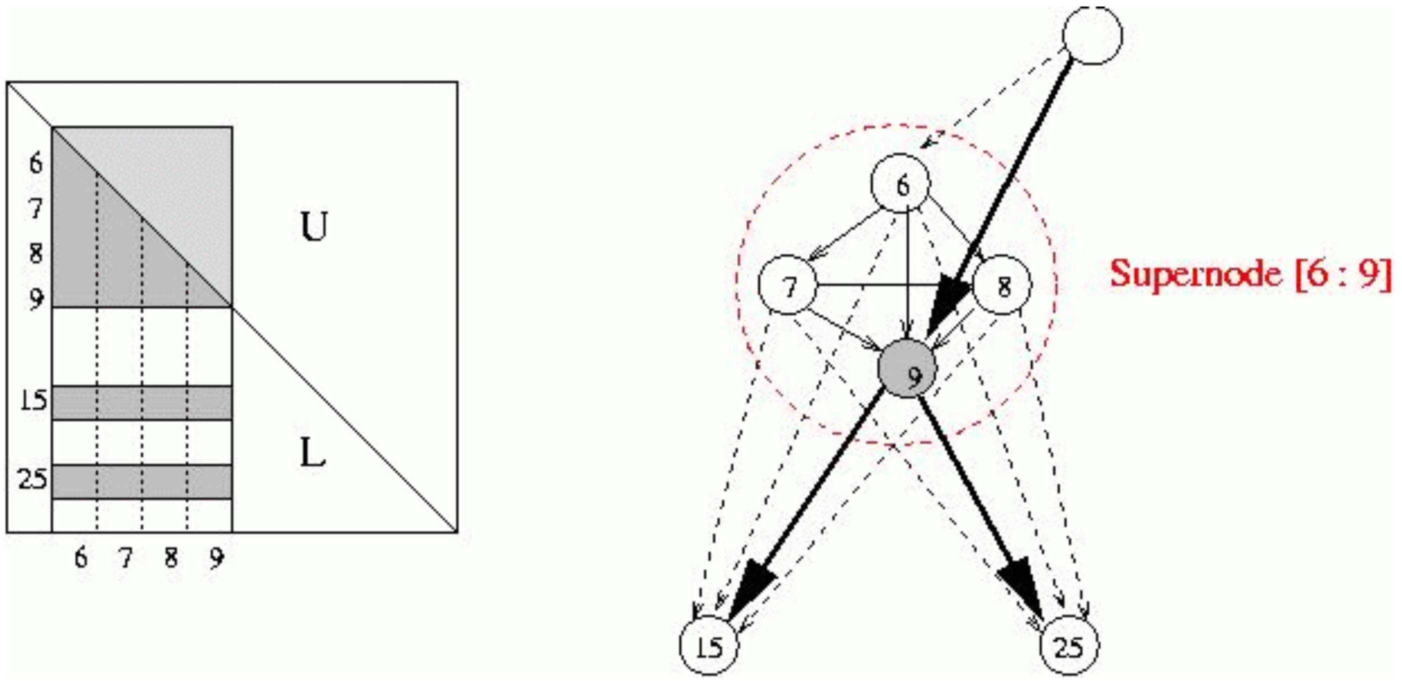
$S^{(j)} \leftarrow ((A^{(j)} - D^{(k1)}) - D^{(k2)}) - \ldots)$

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

Supernode

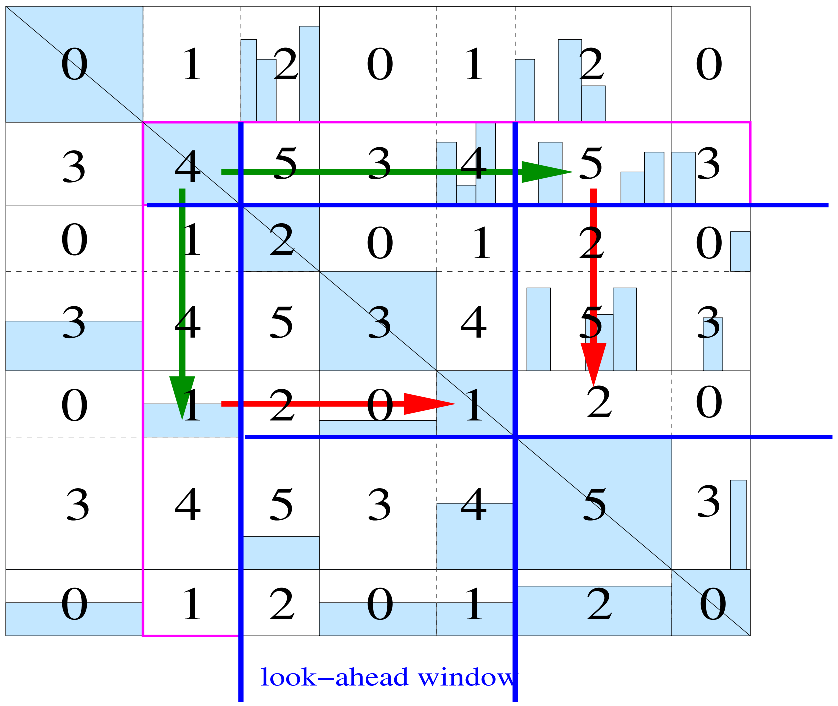
Exploit dense submatrices in the factors

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



2D 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.

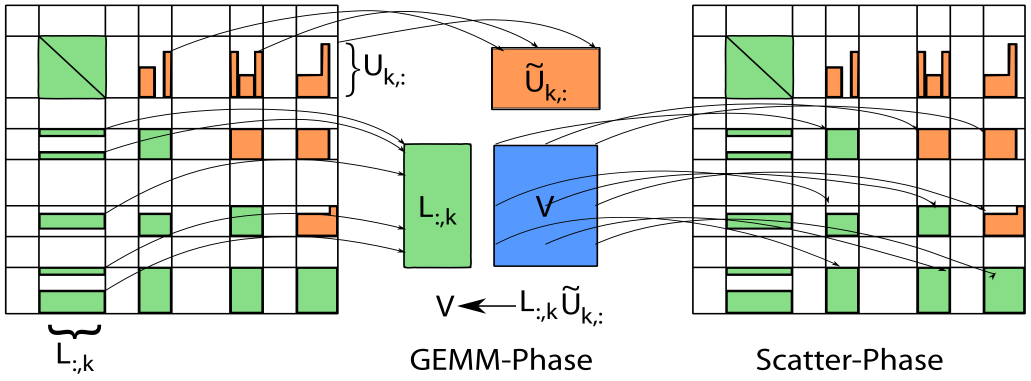


MPI Process Grid

0	1	2
3	4	5

Per-rank Schur complement update

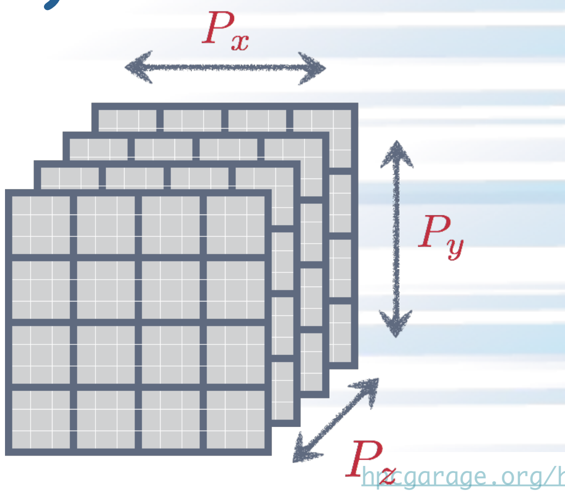
Loop through N steps: (Gaussian Elimination)
FOR (k = 1, N) {
1) **Gather** sparse blocks A(:, k) and A(k,:) into dense work[]
2) Call dense **GEMM** on work[]
3) **Scatter** work[] into remaining sparse blocks
}



Communication-avoiding 3D SpLU (accessible from 'pddrive3d')

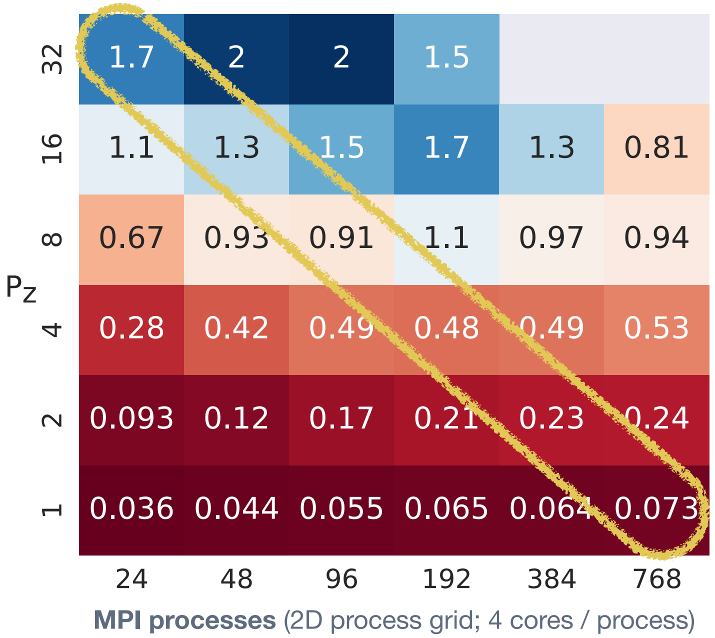
[Sao, Li, Vuduc, JPDC 2019]

3D process grid: $\{P_{XY}, P_Z\}$



- 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
  - Non-planar graph: up to 3.3x faster, 2x more mem



2D to 3D:
→ **23x speedup**

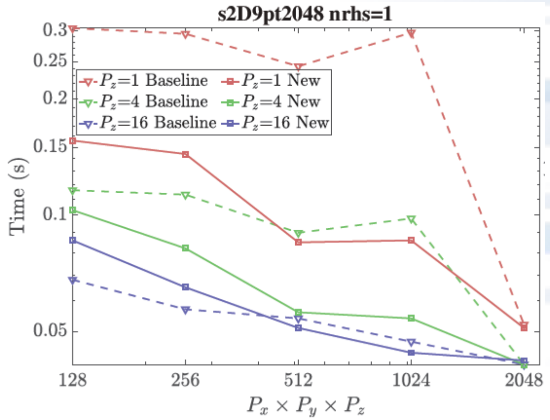
Teraflop/s
(32x procs → 2x speedup)

Communication-avoiding 3D SpTRSV

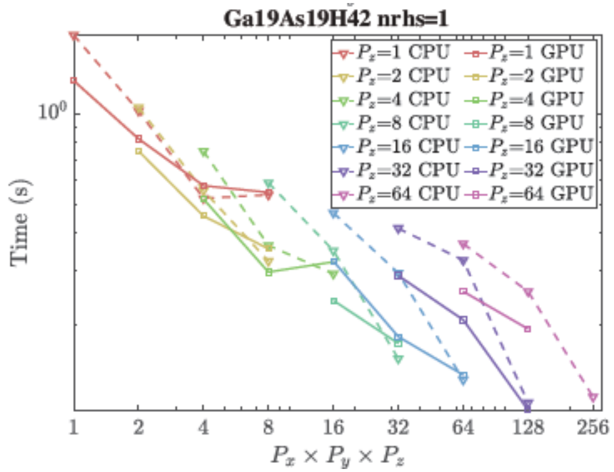
[Liu, Ding, Wiliams, Li; SC2023]

- Communication optimization for 3D SpTRSV
  - Trade inter-grid synchronization with replicated computation
  - Sparse Allreduce operations for inter-grid communication
  - Communication tree for intra-gird communication
  - NVSHMEM for GPU-initiated one-sided communication in each 2D grid
- Strong scaling
  - New 3D CPU SpTRSV achieves up to 3.4X speedups compared to baseline 3D SpTRSV on Cori using 2048 CPU cores.
  - New 3D GPU SpTRSV achieves up to 6.5X speedups compared to new 3D CPU SpTRSV on Crusher and Perlmutter with up to 256 GPUs.

New vs baseline (Cori)



A100 vs. EPYC 7763 with NVSHMEM (Perlmutter)

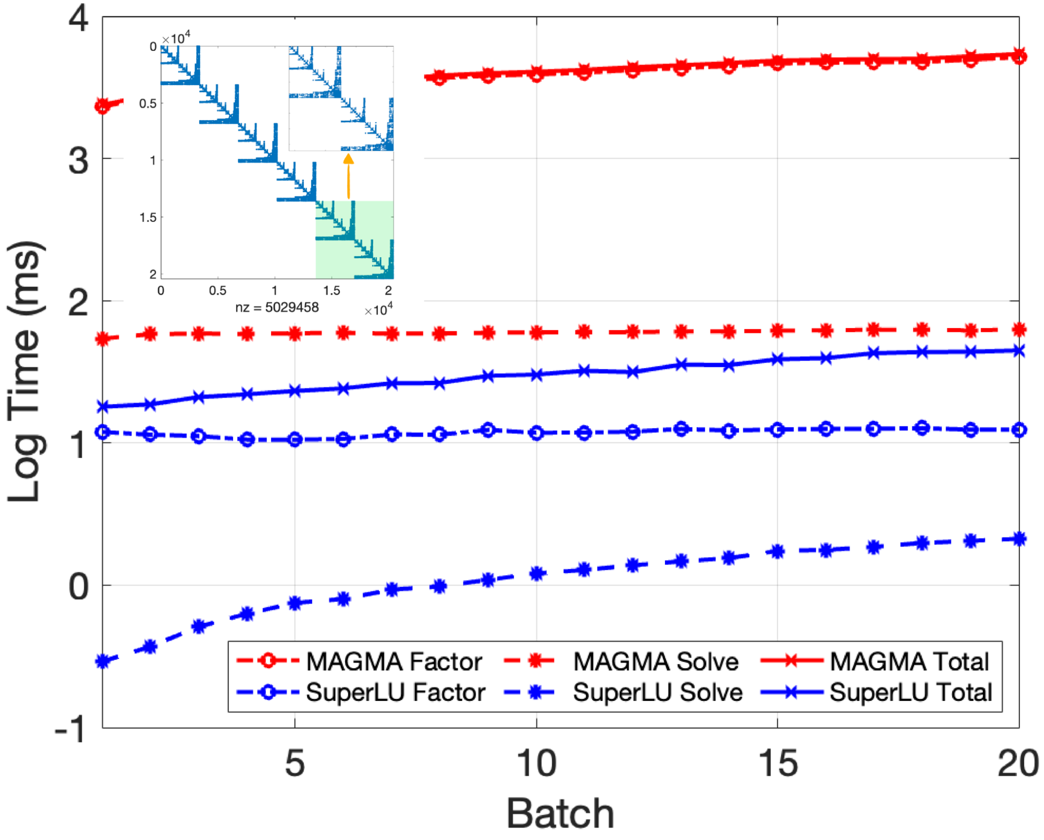


P_{XY} = 4x1

Batched SpLU and SpTRSV on GPUs

- Batched linear systems of matrices with same sparsity pattern but different values

```
pdgssvx3d_csc_batch(&options, batchCount,
            m, n, Astore->nnz, nrhs, SparseMatrix_handles,
            RHSptr, ldRHS, ReqPtr, CeqPtr, RpivPtr, CpivPtr,
            DiagScale, F, Xptr, ldX, Berrs, &grid, &stat, &info);
```



Example using duplicated matrices:
mpiexec -n 1 pddrive3d -r 1 -c 1 -b 10 ${matdir}/<matrix file>

SuperLU_DIST Instalation (CMAKE as an Example)

```
mkdir build
cd build
cmake .. \
      -DTPL_ENABLE_PARMETISLIB=ON | OFF \
      -DTPL_ENABLE_INTERNAL_BLASLIB=OFF | ON \
      -DTPL_ENABLE_LAPACKLIB=OFF | ON \
      -DTPL_ENABLE_COMBBLASLIB=OFF | ON \
      -DTPL_ENABLE_CUDALIB=OFF | ON \
      -DTPL_ENABLE_HIPLIB=OFF | ON \
      -DTPL_ENABLE_MAGMALIB=ON | OFF \
      -Denable_complex16=OFF | ON \
      -DXSDK_INDEX_SIZE=32 | 64 \
      -DTPL_ENABLE_NVSHMEM=OFF | ON \
      -DBUILD_SHARED_LIBS=OFF | ON \
      -DCMAKE_INSTALL_PREFIX=<...> \
      -DCMAKE_C_COMPILER=< MPI C compiler> \
      -DCMAKE_C_FLAGS="..." \
      -DCMAKE_CXX_COMPILER=< MPI C++ compiler> \
      -DCMAKE_CXX_FLAGS="..." \
      -DCMAKE_Fortran_FLAGS="..." \
      -DCMAKE_CUDA_FLAGS="..." \
      -DHIP_HIPCC_FLAGS="..." \
      -DXSDK_ENABLE_Fortran=OFF | ON \
      -DCMAKE_Fortran_COMPILER=< MPI F90 compiler> \
      -DCMAKE_CUDA_ARCHITECTURES=80
```

use parmetis/metis for fill-in reduction ordering

use internal or vendor blas

Lapack needed for GPU SpTRSV

NVIDIA GPU

AMD GPU

MAGMA needed for batched SpLU

Integer type

NVSHMEM needed for multi-GPU SpTRSV

User-controllable options in SuperLU_DIST

For stability and efficiency, need to solve transformed linear system:

$$P_c (P_r (D_r A D_c)) P_c^T P_c D_c^{-1} x = P_c P_r D_r b$$

"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.

Runtime environment variables in SuperLU_DIST

```
export SUPERLU_ACC_OFFLOAD=1 | 0     # perform GPU SpLU
export GPU3DVERSION=1 | 0            # whether to GPU offload all computations in SpLU
export SUPERLU_ACC_SOLVE=0 | 1       # perform GPU SpTRSV
export SUPERLU_MAXSUP=256            # max supernode size
export SUPERLU_RELAX=64              # upper bound for relaxed supernode size
export SUPERLU_MAX_BUFFER_SIZE=10000000 # buffer size in words on GPU
export SUPERLU_NUM_LOOKAHEADS=10     # lookahead window size
export SUPERLU_NUM_GPU_STREAMS=1     # number of CUDA/HIP streams
export SUPERLU_MPI_PROCESS_PER_GPU=1 # number of MPIs per GPU
```

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 by environment variables)**
    - **Larger blocks better for uniprocessor**
    - **Smaller blocks better for parallellism and load balance**

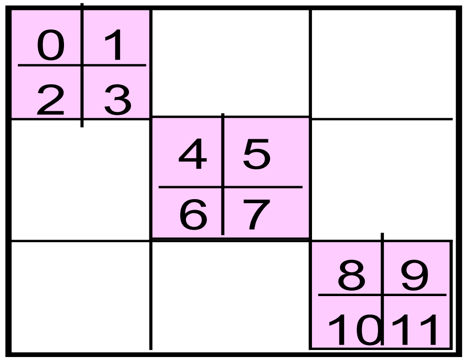
- GPTune: Statistical learning algorithms for selection of best parameters
  - gptune.lbl.gov

SuperLU_DIST other examples

superlu_dist/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

Summary: 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)

Code	Technique	Scope	Contact	
Serial platforms (possibly on GPU				
CHOLMOD	Left-looking	SPD	Davis	[8]
GLU3.0	Left-looking	Unsym (GPU)	Peng	[36]
KLU	Left-looking	Unsym	Davis	[11]
MA57	Multifrontal	Sym	HSL	[19]
MA41	Multifrontal	Sym-pat	HSL	[1]
MA42	Frontal	Unsym	HSL	[20]
MA67	Multifrontal	Sym	HSL	[17]
MA48	Right-looking	Unsym	HSL	[18]
Oblio	Left/right/Multifr.	sym, Out-core	Dobrian	[14]
SPARSE	Right-looking	Unsym	Kundert	[32]
SPARSPAK	Left-looking	SPD, Unsym, QR	George et al.	[22]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SSIDS	Multifrontal	Sym (GPU)	Hogg	[28]
SuperLLT	Left-looking	SPD	Ng	[35]
SuperLU	Left-looking	Unsym	Li	[12]
UMFPACK	Multifrontal	Unsym	Davis	[9]
Shared memory parallel machines (possibly on GPU)				
BCSLIB-EXT	Multifrontal	Sym, Unsym, QR	Ashcraft et al.	[6]
Cholesky	Left-looking	SPD	Rothberg	[31]
MF2	Multifrontal	Sym, Sym-pat, Out-core (GPU)	Lucas	[34]
MA41	Multifrontal	Sym-pat	HSL	[2]
MA49	Multifrontal	QR	HSL	[4]
PanelLLT	Left-looking	SPD	Ng	[24]
PARASPAR	Right-looking	Unsym	Zlatev	[41]
PARDISO	Left-Right looking	Sym-pat	Schenk	[39]
SPOOLES	Left-looking	Sym, Sym-pat	Ashcraft	[5]
SuiteSparseQR	Multifrontal	Rank-revealing QR	Davis	[10]
SuperLU_MT	Left-looking	Unsym	Li	[13]
TAUCS	Left/Multifr.	Sym, Unsym, Out-core	Toledo	[7]
WSMP	Multifrontal	SPD, Unsym	Gupta	[25]
Distributed memory parallel machines				
Clique	Multifrontal	Sym (no pivoting)	Poulson	[37]
MF2	Multifrontal	Sym, Sym-pat, Out-core, GPU	Lucas	[34]
DSCPACK	Multifrontal	SPD	Raghavan	[26]
MUMPS	Multifrontal	Sym, Sym-pat	Amestoy	[3]
PARDISO	Left-Right looking	Sym-pat, Unsym	Schenk	[39]
PaStiX	Left-Right looking	SPD, Sym, Sym-pat	Ramet	[29]
PSPASES	Multifrontal	SPD	Gupta	[23]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
STRUMPACK	Multifrontal	Unsym, Sym-pat (GPU)	Ghysels	[40]
SuperLU_DIST	Right-looking	Unsym (GPU)	Li	[33]
symPACK	Left-Right looking	SPD	Jacquelin	[30]
S+	Right-looking†	Unsym	Yang	[21]
WSMP	Multifrontal	SPD, Unsym	Gupta	[25]

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

Survey of sparse direct solver codes

portal.nersc.gov/project/sparse/superlu/SparseDirectSurvey.pdf

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
  - Users Guide, papers, FAQ, code documentation, …

Rank Structured Solvers for Dense Linear Systems

EXASCALE COMPUTING PROJECT

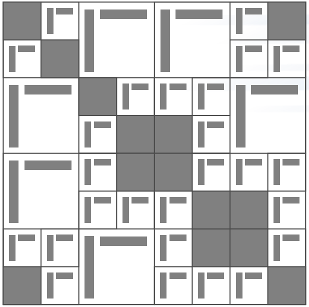
Hierarchical Matrix Approximation

$\mathcal{H}$-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)

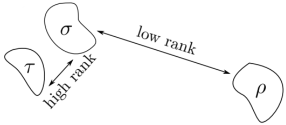


Use cases:

- 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

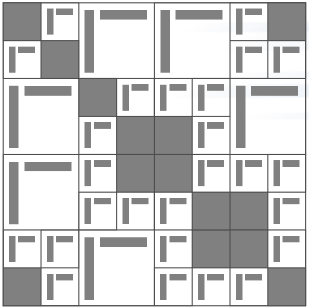


- Row cluster σ
- Column cluster τ
- $\sigma \times \tau$ is compressible $\Leftrightarrow$

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

  - diam(σ): diameter of physical domain corresponding to σ
  - dist(σ, τ): 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



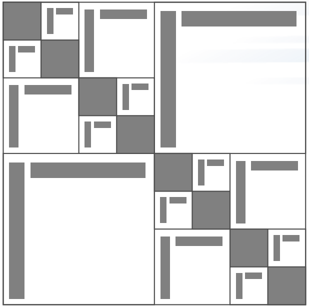
Hackbusch, W., 1999. *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices.* Computing, 62(2), pp.89-108.

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

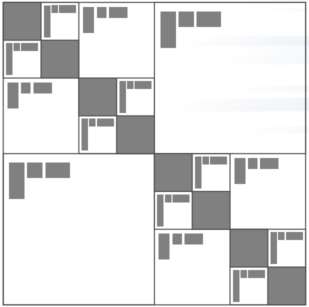
$$U_\sigma = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \hat{U}_\sigma$$

  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



$$\left[\begin{array}{cccc} D_0 & U_0 B_{0,1} V_1^* & \multicolumn{2}{c}{\multirow{2}{*}{$\color{red}{U_2} B_{2,5} \color{red}{V_5^*}$}} \\ U_1 B_{1,0} V_0^* & D_1 & & \\ \multicolumn{2}{c}{\multirow{2}{*}{$\color{red}{U_5} B_{5,2} \color{red}{V_2^*}$}} & D_3 & U_3 B_{3,4} V_4^* \\ & & U_4 B_{4,3} V_3^* & D_4 \end{array}\right]$$

HSS: Hierarchically Semi Seperable

- Weak admissibility
- Off-diagonal blocks

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

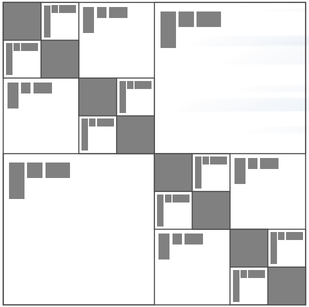
- Nested bases

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

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

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

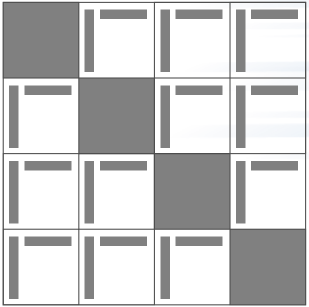
- Store only $\hat{U}_\sigma$, smaller than U_σ
- Complexity $\mathcal{O}(N) \leftrightarrow \mathcal{O}(N \log N)$ for HODLR
- HSS is special case of $\mathcal{H}^2$: $\mathcal{H}$ with nested bases



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

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

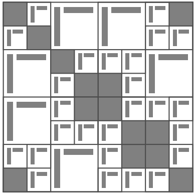


Mary, T. (2017). *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* (Doctoral dissertation).

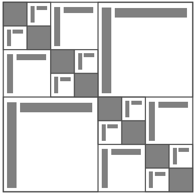
Amestoy, Patrick, et al. (2015). *Improving multifrontal methods by means of block low-rank representations.* SISC 37.3 : A1451-A1474.

EXASCALE COMPUTING PROJECT

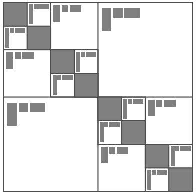
Data-Sparse Matrix Representation Overview



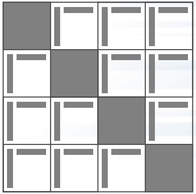
$\mathcal{H}$



HODLR



HSS



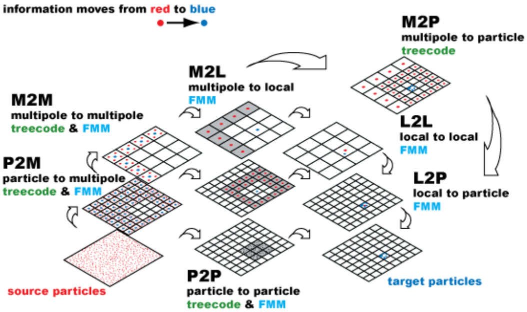
BLR

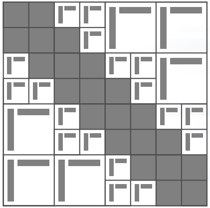
- Partitioning: **hierarchical** ($\mathcal{H}$, HODLR, HSS) or **flat** (BLR)
- Admissibility: **weak** (HODLR, HSS) or **strong** ($\mathcal{H}$, $\mathcal{H}^2$)
- Bases: **nested** (HSS, $\mathcal{H}^2$) or **not nested** (HODLR, $\mathcal{H}$, BLR)

Fast Multipole Method [1]

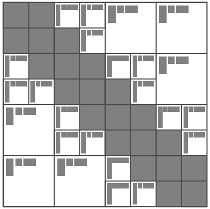
Particle methods like Barnes-Hut and FMM can be interpreted algebraically using hierarchical matrix algebra

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





Barnes-Hut

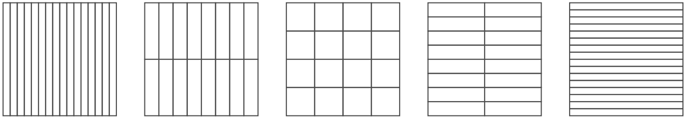


FMM

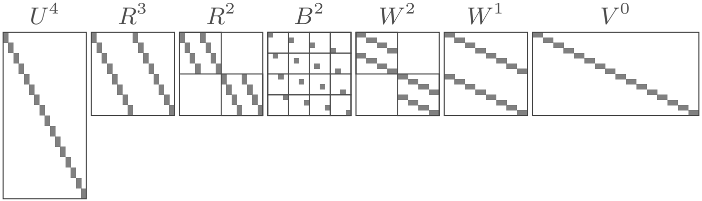
Greengard, L., and Rokhlin, V. *A fast algorithm for particle simulations.* Journal of computational physics 73.2 (1987): 325-348.

Butterfly Decomposition

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



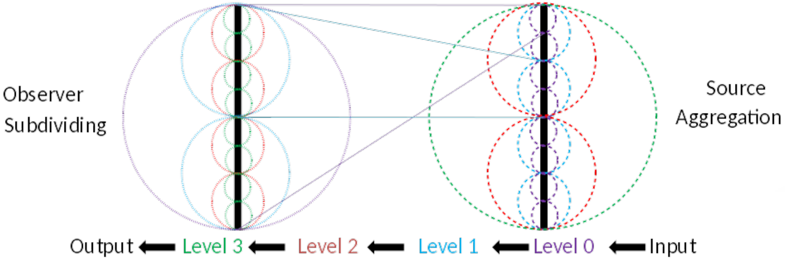
Multiplicative decomposition:

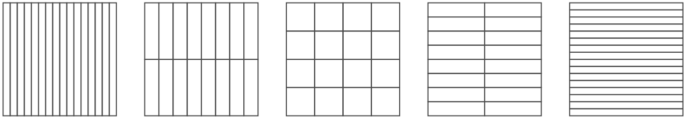


- Multilevel generalization of low rank decomposition
- Based on FFT ideas, motivated by high-frequency problems

EXASCALE COMPUTING PROJECT

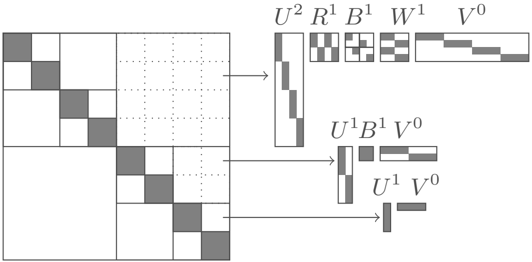
Butterfly Decomposition Intuition [1]





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

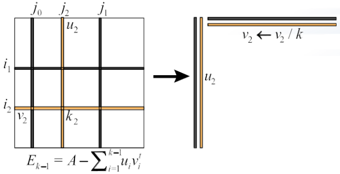
Low Rank Approximation Techniques

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.

Block Low Rank on GPU

Schur complement updates into dense block A_{ij}:

$$
\begin{aligned}
A_{ij} &\leftarrow A_{ij} + \sum_k A_{ik} A_{kj} \\
&\leftarrow A_{ij} + \sum_k (U_{ik} V_{ik})(U_{kj} V_{kj})
\end{aligned}
$$

For $k = 1 \ldots$:

- batch 1

$$\hat{T}_{ij} = V_{ik} U_{kj} \quad \forall i, j$$

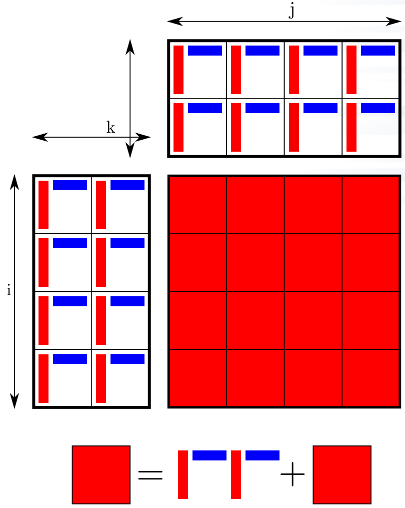
- batch 2 (depending on the rank)

$$\tilde{T}_{ij} = \begin{cases} U_{ik} \hat{T}_{ij} & if.. \\ \hat{T}_{ij} V_{kj} & else \end{cases} \quad \forall i, j$$

- batch 3

$$A_{ij} \leftarrow A_{ij} + \begin{cases} \tilde{T}_{ij} V_{kj} & if... \\ U_{ik} \tilde{T}_{ij} & else \end{cases} \quad \forall i, j$$

($3 \times$ `magmablas_dgemm_vbatched`)



$C_{ij} = \alpha \left(U_{ik} \left(V_{ik} U_{kj}\right)\right) V_{kj} + \beta C_{ij}$

Block Low Rank on GPU

Device allocations, data transfers

- `cudaMalloc`/`cudaFree` per BLR front

Low rank compression on GPU

- SVD (`cusolverDnDgesvdj`/`magma_dgesvd`) is expensive
- ARA (Adaptive Rand. Approx.) from KBLAS is much faster

BLR algorithmic variants

- FSUC (Factor/Solve/Update/Compress), FSCU, FCSU, CFSU
- LUAR (Low-rank Update Accumulation and Recompression)
- Matrix-free ARA

$$\left(A_{ij} + \sum_k U_{ik}\left(V_{ik}U_{kj}\right)V_{kj}\right)\mathbf{R} = A_{ij}\mathbf{R} + \sum_k U_{ik}\left(V_{ik}\left(U_{kj}\left(V_{kj}\mathbf{R}\right)\right)\right)$$

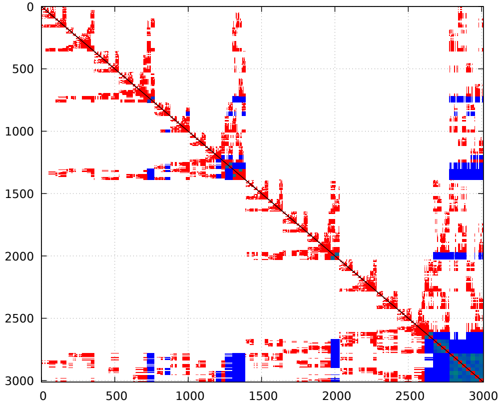
- ACA (Adaptive Cross Approx.), blocked ACA
- Pivoting

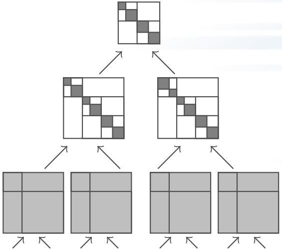
Mary, T. (2017). *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* (Doctoral dissertation).

Approximate Multifrontal Factorization

Sparse Multifrontal Solver/Preconditioner with Rank-Structured Approximations

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

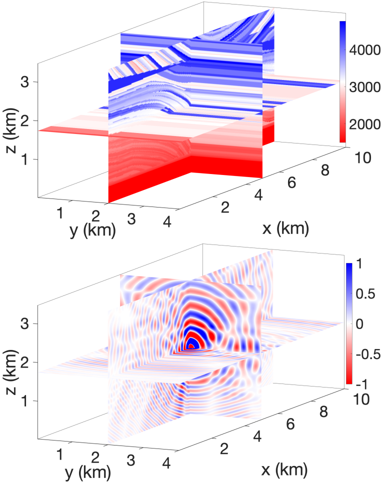




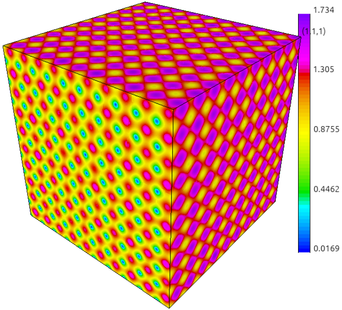
Only apply rank structured compression to largest fronts (dense sub-blocks), keep the rest as regular dense

High Frequency Helmholtz and Maxwell

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



Marmousi2 geophyical elastic dataset



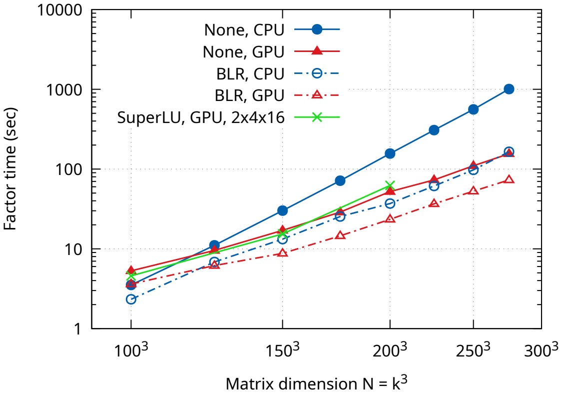
Indefinite Maxwell, using MFEM

Multi-GPU BLR Preconditioning, 32 nodes of Perlmutter

$$\left(\sum_i \rho(\mathbf{x}) \frac{\partial}{\partial x_i} \frac{1}{\rho(\mathbf{x})} \frac{\partial}{\partial x_i}\right) p(\mathbf{x}) + \frac{\omega^2}{\kappa^2(\mathbf{x})} p(\mathbf{x}) = -f(\mathbf{x})$$

FD on staggered grids, 27-point stencil and 8 PML layers, 15ppw.

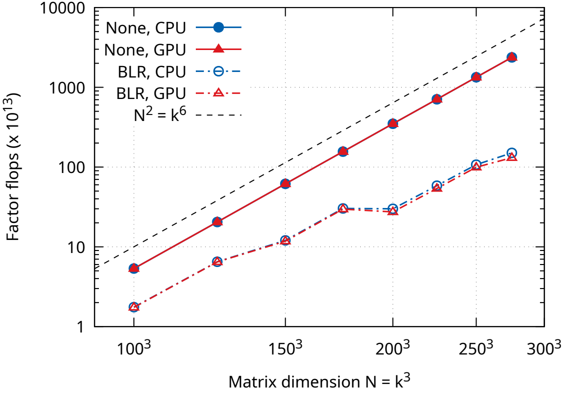
GMRes(30) with $\varepsilon = 10^{-6}$. BLR with $\varepsilon_{\text{rel}} = 10^{-2}$, only if $F_{11} \geq 2k \times 2k$.



Multi-GPU BLR Preconditioning, 32 nodes of Perlmutter

$$\left(\sum_i \rho(\mathbf{x}) \frac{\partial}{\partial x_i} \frac{1}{\rho(\mathbf{x})} \frac{\partial}{\partial x_i}\right) p(\mathbf{x}) + \frac{\omega^2}{\kappa^2(\mathbf{x})} p(\mathbf{x}) = -f(\mathbf{x})$$

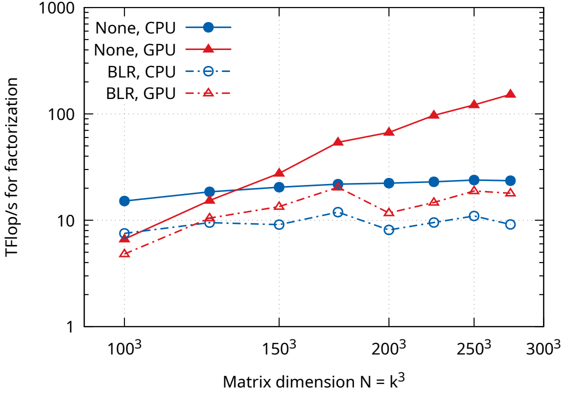
FD on staggered grids, 27-point stencil and 8 PML layers, 15ppw.
GMRes(30) with $\varepsilon = 10^{-6}$. BLR with $\varepsilon_{\text{rel}} = 10^{-2}$, only if $F_{11} \geq 2k \times 2k$.



Multi-GPU BLR Preconditioning, 32 nodes of Perlmutter

$$\left(\sum_i \rho(\mathbf{x}) \frac{\partial}{\partial x_i} \frac{1}{\rho(\mathbf{x})} \frac{\partial}{\partial x_i}\right) p(\mathbf{x}) + \frac{\omega^2}{\kappa^2(\mathbf{x})} p(\mathbf{x}) = -f(\mathbf{x})$$

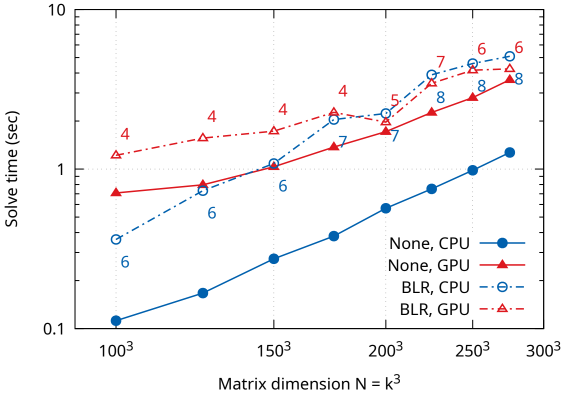
FD on staggered grids, 27-point stencil and 8 PML layers, 15ppw.
GMRes(30) with $\varepsilon = 10^{-6}$. BLR with $\varepsilon_{\text{rel}} = 10^{-2}$, only if $F_{11} \geq 2k \times 2k$.



Multi-GPU BLR Preconditioning, 32 nodes of Perlmutter

$$\left(\sum_i \rho(\mathbf{x}) \frac{\partial}{\partial x_i} \frac{1}{\rho(\mathbf{x})} \frac{\partial}{\partial x_i}\right) p(\mathbf{x}) + \frac{\omega^2}{\kappa^2(\mathbf{x})} p(\mathbf{x}) = -f(\mathbf{x})$$

FD on staggered grids, 27-point stencil and 8 PML layers, 15ppw.
GMRes(30) with $\varepsilon = 10^{-6}$. BLR with $\varepsilon_{\mathrm{rel}} = 10^{-2}$, only if $F_{11} \geq 2k \times 2k$.



EXASCALE COMPUTING PROJECT

BLR Preconditioning

			no compression				BLR($\varepsilon_{\mathrm{rel}} = 10^{-2}$)							
			CPU		A100		CPU				A100			
matrix	N $\times 10^3$	nnz $\times 10^3$	fact (sec)	solve (sec)	fact (sec)	solve (sec)	fact (sec)	solve (sec)	its	comp (%)	fact (sec)	solve (sec)	its	comp (%)
Serena	1,391	64,531	229.6	1.07	17.9	1.2	76.4	5.2	10	34.4	17.3	3.4	6	39.7
Geo_1438	1,437	63,156	151.9	1.04	12.7	1.0	60.4	7.2	13	45.6	16.9	4.5	7	54.2
Hook_1498	1,498	60,917	76.1	0.70	7.4	0.7	29.7	14.5	35	46.7	12.5	4.1	10	52.0
ML_Geer	1,504	110,879	23.6	0.51	2.0	0.3	11.5	10.1	27	64.6	8.7	4.0	11	66.6
Transport	1,602	23,500	40.9	0.63	3.2	0.3	21.3	10.8	25	52.0	8.8	4.5	11	58.1
Flan_1565	1,565	117,406	32.8	0.7	3.0	0.4	20.5	40.6	86	62.3	12.3	25.0	54	65.7
Cube_Coup_dt0	2,164	129,133	OOM	OOM	62.1	2.4	223.9	18.1	18	31.0	46.0	7.3	7	38.5

Table: Multifrontal solver with BLR compression tolerance $\varepsilon = 10^{-2}$.
GMRES(30) relative tolerance is 10^{-6}.
CPU uses 8 cores of AMD EPYC 7763.
PVC: Intel Data Center GPU Max Series ('Ponte Vecchio').
cuDSS: new NVIDIA sparse direct solver `https://developer.nvidia.com/cudss`.

BLR Preconditioning

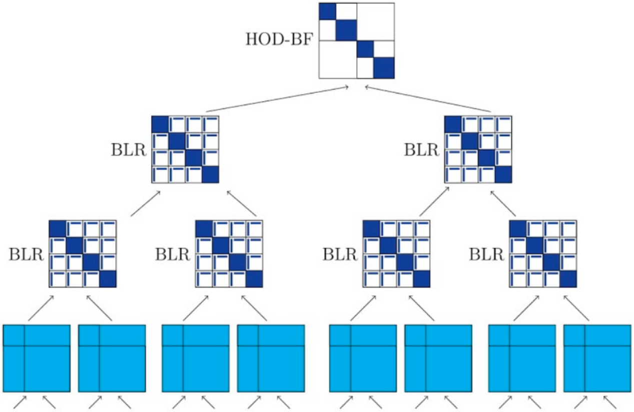
			no compression							
			CPU		A100		cuDSS A100		PVC	
matrix	N $\times 10^3$	nnz $\times 10^3$	fact (sec)	solve (sec)	fact (sec)	solve (sec)	fact (sec)	solve (sec)	fact (sec)	solve (sec)
Serena	1,391	64,531	229.6	1.07	17.9	1.2	48.2	0.4	14.3	0.6
Geo_1438	1,437	63,156	151.9	1.04	12.7	1.0	33.0	0.3	10.6	0.6
Hook_1498	1,498	60,917	76.1	0.70	7.4	0.7	15.4	0.2	6.5	0.4
ML_Geer	1,504	110,879	23.6	0.51	2.0	0.3	5.4	0.1	4.5	0.3
Transport	1,602	23,500	40.9	0.63	3.2	0.3	10.8	0.2	5.0	0.3
Flan_1565	1,565	117,406	32.8	0.7	3.0	0.4	8.8	0.1	5.7	0.4
Cube_Coup_dt0	2,164	129,133	OOM	OOM	62.1	2.4	80.7	0.6	23.2	0.9

Table: Multifrontal solver with BLR compression tolerance $\varepsilon = 10^{-2}$.
GMRES(30) relative tolerance is 10^{-6}.
CPU uses 8 cores of AMD EPYC 7763.
PVC: Intel Data Center GPU Max Series ('Ponte Vecchio').
cuDSS: new NVIDIA sparse direct solver `https://developer.nvidia.com/cudss`.

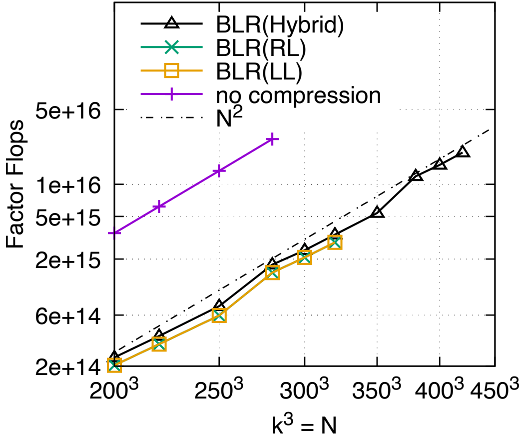
Combining Block Low Rank and Hierarchically Off-Diagonal Butterfly

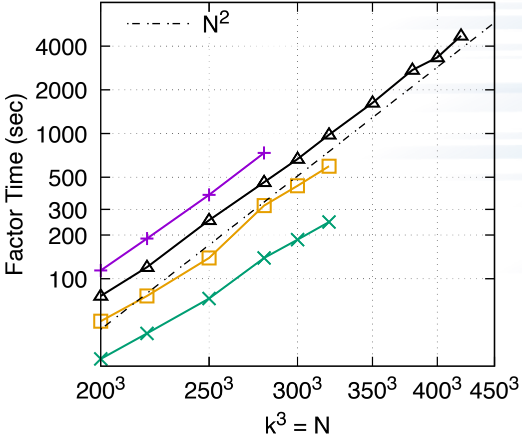
Rank-structured compression of largest dense blocks in the multifrontal/assembly tree

- Largest: HOD-BF
- Medium: BLR
- Smaller: dense



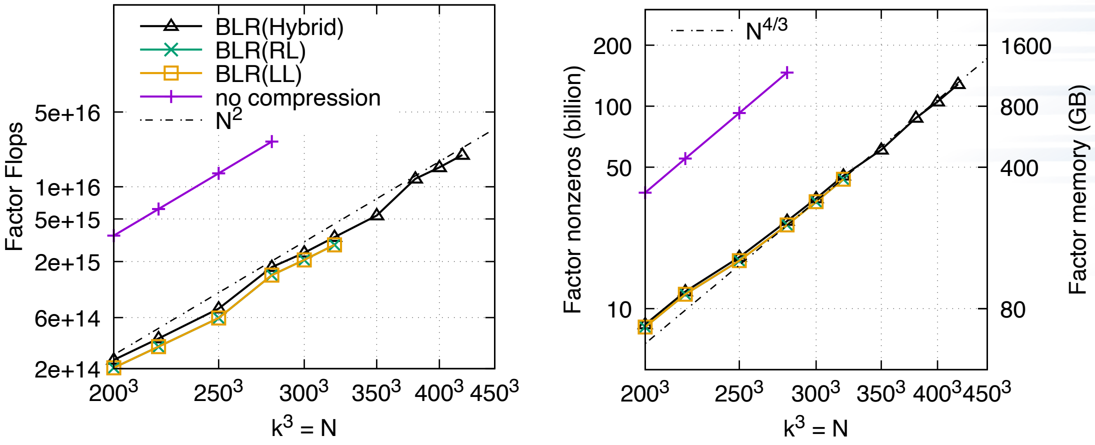
High Frequency 3D Helmholtz – BLR





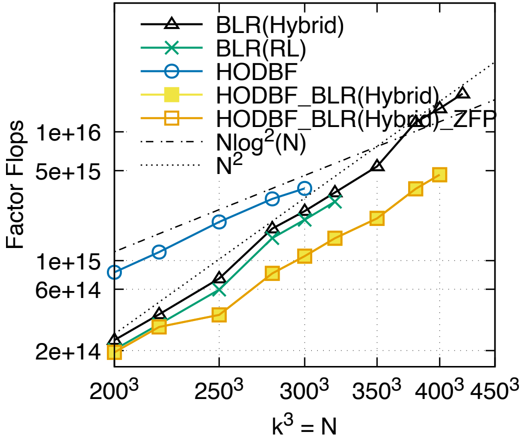
Right/left-looking BLR have different communication patterns
Hybrid BLR reduces peak memory by not forming dense front

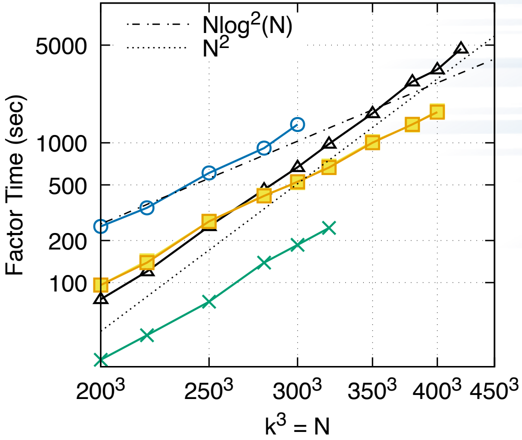
High Frequency 3D Helmholtz – BLR



Right/left-looking BLR have different communication patterns
Hybrid BLR reduces peak memory by not forming dense front

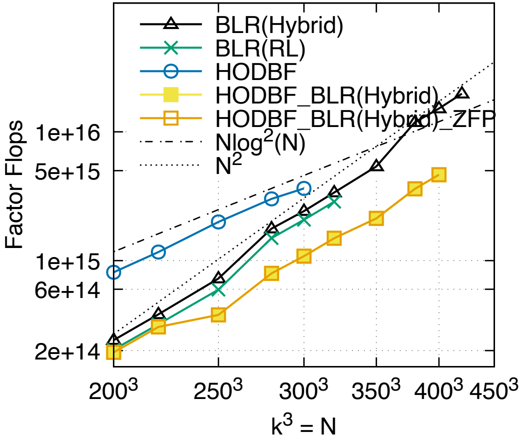
High Frequency 3D Helmholtz – HODBF & BLR & ZFP

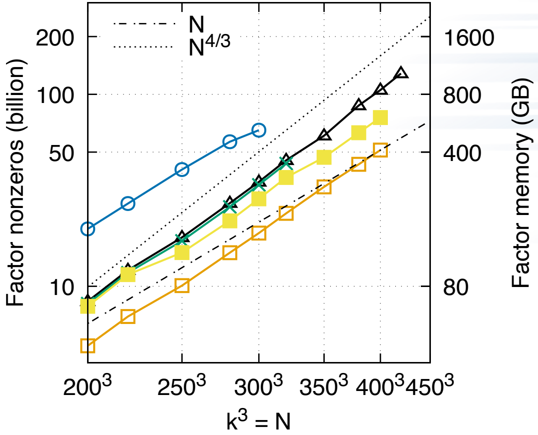




EXASCALE COMPUTING PROJECT

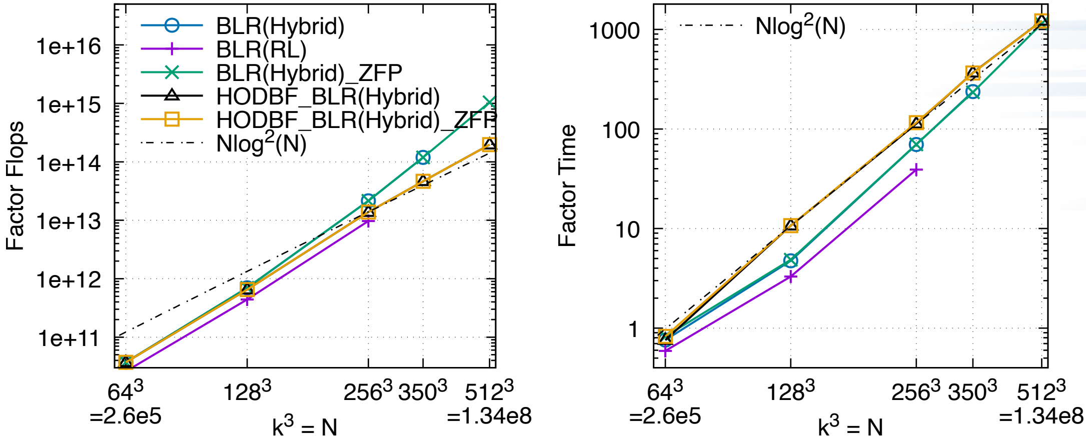
High Frequency 3D Helmholtz – HODBF & BLR & ZFP





Singularly Perturbed PDE – HODBF & BLR & ZFP

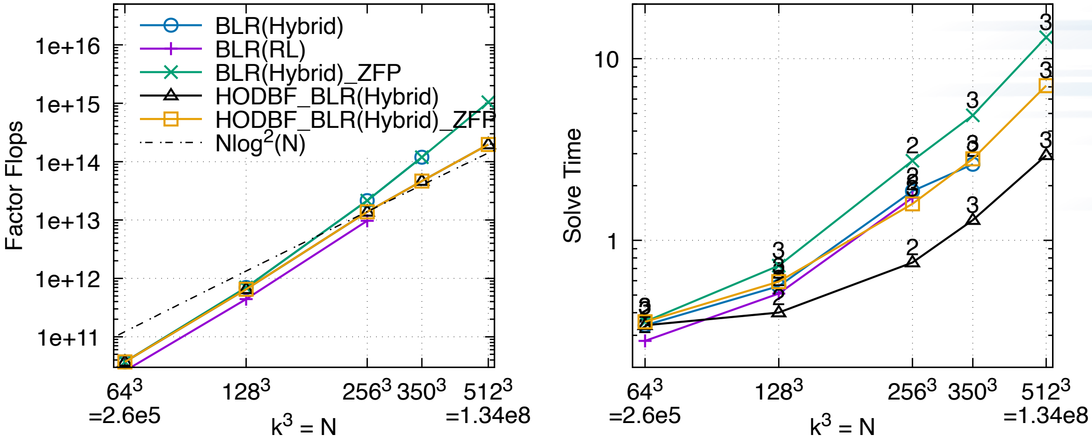
$$-\delta^2 \Delta u + u = f, \quad \text{on} \quad \Omega = (0,1)^3, \quad \text{and} \quad u(\partial\Omega) = g,$$



EXASCALE COMPUTING PROJECT

Singularly Perturbed PDE – HODBF & BLR & ZFP

$$-\delta^2 \Delta u + u = f, \quad \text{on} \quad \Omega = (0,1)^3, \quad \text{and} \quad u(\partial\Omega) = g,$$



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
  - 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

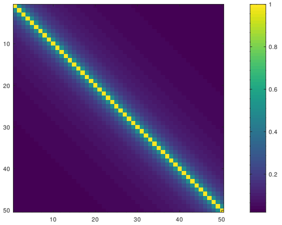
HiCMA	`https://github.com/ecrc/hicma`
HLib	`http://www.hlib.org/`
HLibPro	`https://www.hlibpro.com/`
H2Lib	`http://www.h2lib.org/`
HACApK	`https://github.com/hoshino-UTokyo/hacapk-gpu`
MUMPS	`http://mumps.enseeiht.fr/`
PaStiX	`https://gitlab.inria.fr/solverstack/pastix`
ExaFMM	`http://www.bu.edu/exafmm/`

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

STRUMPACK Hands-On Session

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 -l select=1 -l filesystems=home:eagle -l walltime
  ```
- Set OpenMP threads:
  ```
  export OMP_NUM_THREADS=1
  ```

- Run example:
  ```
  mpiexec -n 1 ./build/testHODLR 2000
  ```
- With description of command line parameters:
  ```
  mpiexec -n 1 ./build/testHODLR 2000 --help
  ```
- Vary leaf size (smallest block size) and tolerance:
  ```
  mpiexec -n 1 ./build/testHODLR 2000 --structured_rel_tol 1e-4 --structured_leaf_size 16
  mpiexec -n 1 ./build/testHODLR 2000 --structured_rel_tol 1e-4 --structured_leaf_size 12
  ```
- Vary number of MPI processes:
  ```
  mpiexec -n 12 ./build/testHODLR 2000 --structured_rel_tol 1e-8 --structured_leaf_size 1
  mpiexec -n 12 ./build/testHODLR 2000 --structured_rel_tol 1e-8 --structured_leaf_size 1
  ```

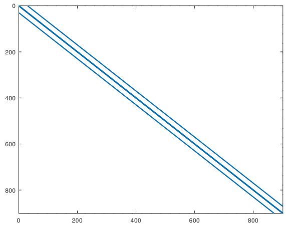
Solve a Sparse Linear System with Matrix `pde900.mtx`

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

- See `track-5-numerical/rank_structured_strumpack/README`
- Get a compute node:

```
qsub -I -l select=1 -l filesystems=home:eagle -l walltime=
```

- 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 -l select=1 -l filesystems=home:eagle -l walltime=1:00:00 -q ATPESC -A ATPESC202
```

- 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:

```
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
```