EXASCALE COMPUTING PROJECT

Argonne Training Program on Extreme-Scale Computing

# Direct Sparse Linear Solvers, Preconditioners

- SuperLU, STRUMPACK, with hands-on examples

ATPESC 2020

X. Sherry Li,  Pieter Ghysels
Lawrence Berkeley National Laboratory

August 4, 2020

# 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 types of factorizations
- Parallelism exploiting sparsity (trees, DAGs)
  - Task scheduling, avoiding communication
- Parallel performance

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

- Hierarchical matrices, Butterfly matrix

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

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

# Sparse matrix representation: Compressed Row Storage (CRS)

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

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

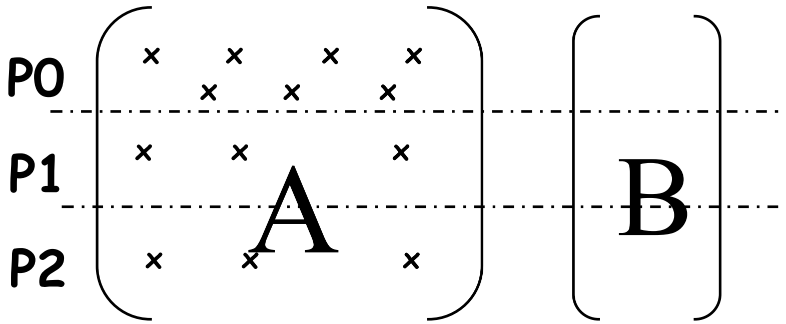
| | 1 | 3 | 5 | 8 | 11 | 13 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|
| nzval | 1 a | 2 b | c d 3 | e 4 f | 5 g | h i 6 j | k l 7 | |
| colind | 1 4 | 2 5 | 1 2 3 | 2 4 5 | 5 7 | 4 5 6 7 | 3 5 7 | |
| rowptr | 1 3 5 8 11 13 17 20 | | | | | | | |

*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_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 }**

# Algorithms: review of Gaussian Elimination (GE)

- First step of GE:

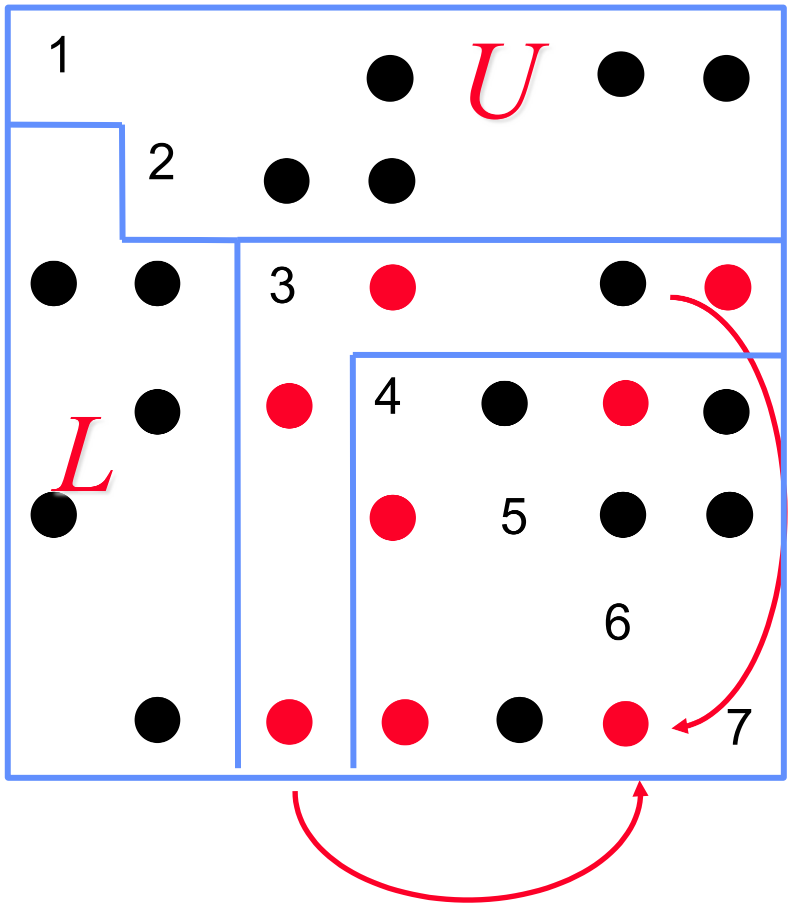
$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I \end{bmatrix} \cdot \begin{bmatrix} \alpha & w^T \\ 0 & C \end{bmatrix}$$

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

- 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

# Fill-in in sparse LU






EXASCALE COMPUTING PROJECT

# 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

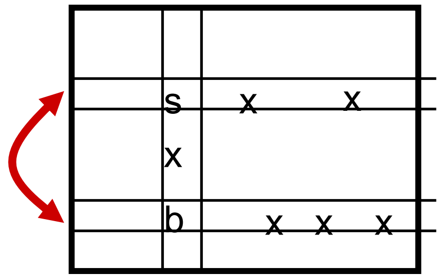
4. Perform factorization and triangular solutions (~80% time)
   - Numerical algorithms (F.P. operations only on nonzeros)
   - Usually dominate the total runtime

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)

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

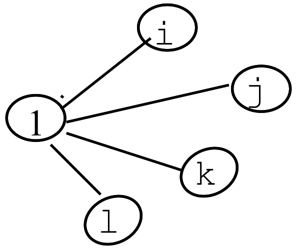
$$\Rightarrow \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & & & \\ 3 & & 3 & & \\ 4 & & & 4 & \\ 5 & & & & 5 \end{pmatrix} \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix}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

(no fill after elimination)

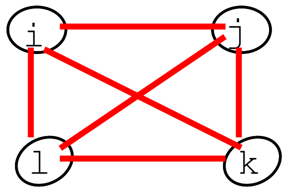
# Ordering to preserve sparsity : Minimum Degree

Local greedy strategy: minimize upper bound on fill-in

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




Eliminate 1 →



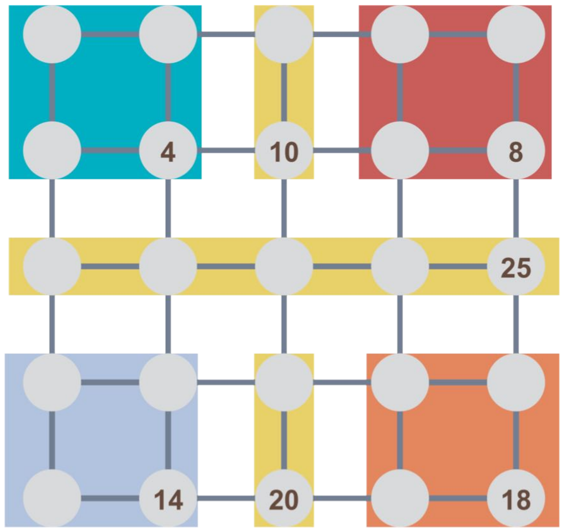



EXASCALE COMPUTING PROJECT

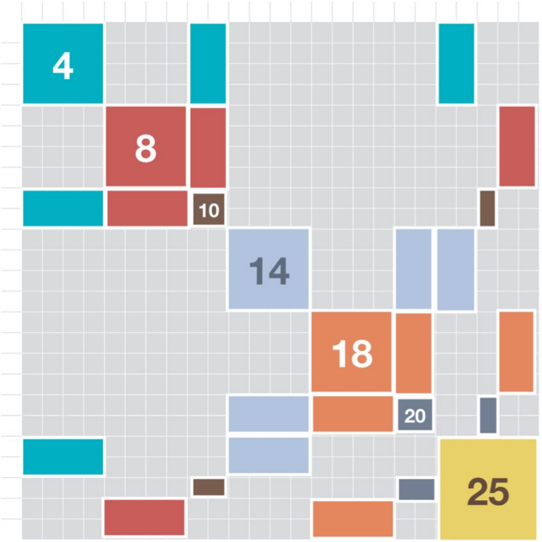
# Ordering to preserve sparsity : Nested Dissection

- 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]

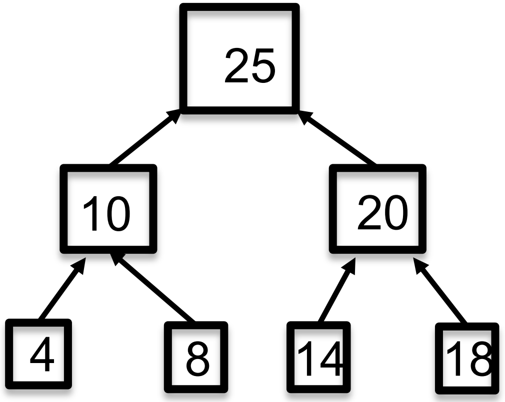
Geometry




Reordered Matrix

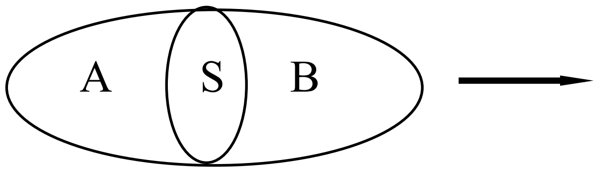



Separator Tree



# ND Ordering

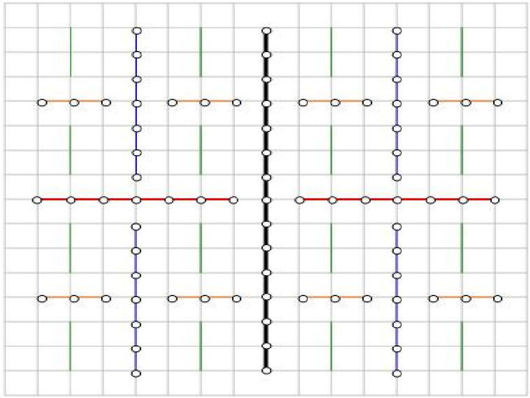
- Generalized nested dissection [Lipton/Rose/Tarjan ’79]
  – **Global graph partitioning: top-down, divide-and-conqure**
  – **Best for large problems**
  – **Parallel codes available: ParMetis, PT-Scotch**
    - First level



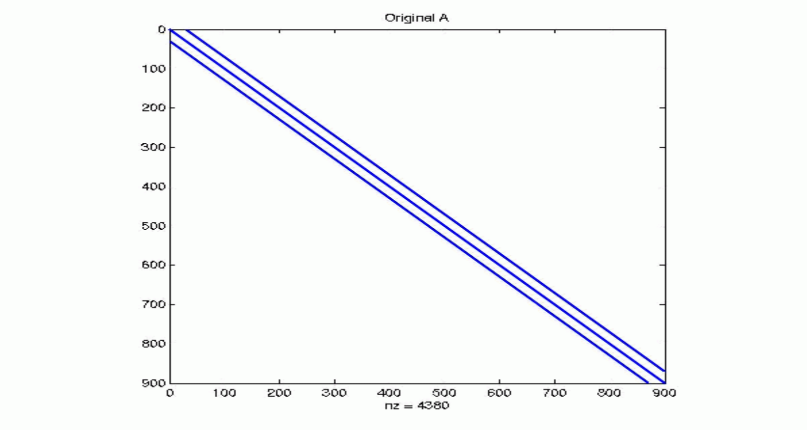

$$\begin{bmatrix} A & 0 & x \\ 0 & B & x \\ x & x & S \end{bmatrix}$$

    - 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]**

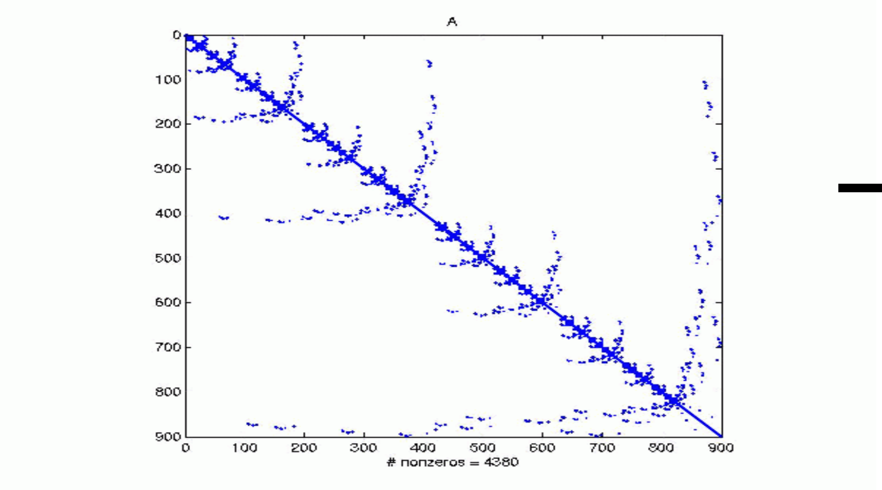
# ND Ordering



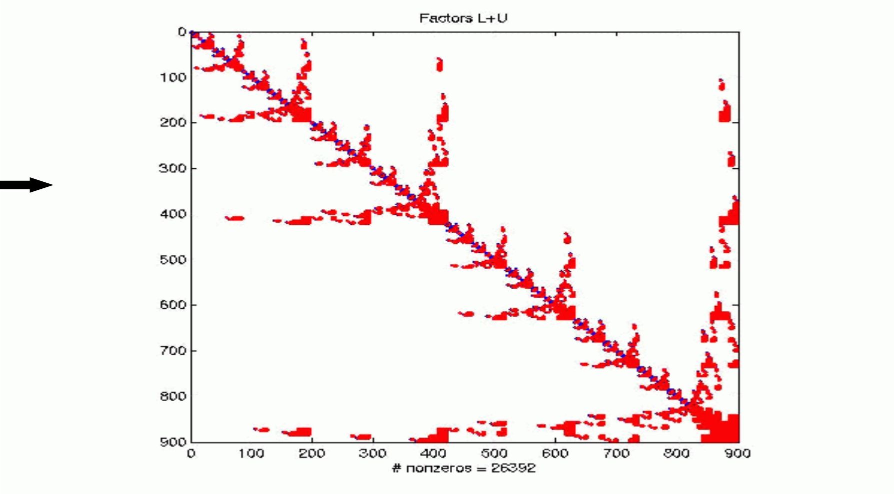
2D mesh




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

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

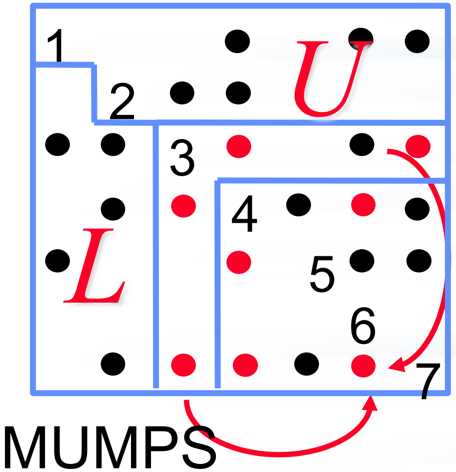
# Algorithm variants, codes …. depending on matrix properties

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

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

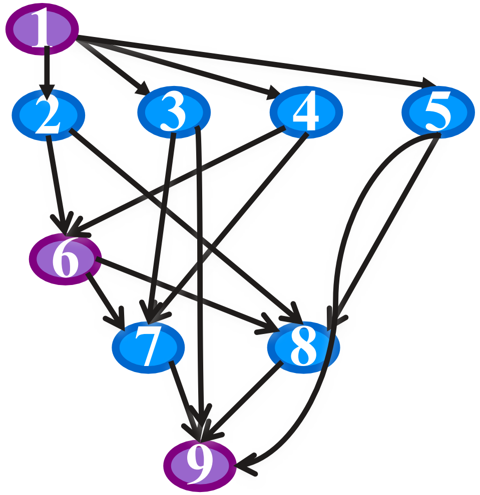
- Survey of sparse direct solvers (codes, algorithms, parallel capability):
  https://portal.nersc.gov/project/sparse/superlu/SparseDirectSurvey.pdf

# Sparse LU: two algorithm variants

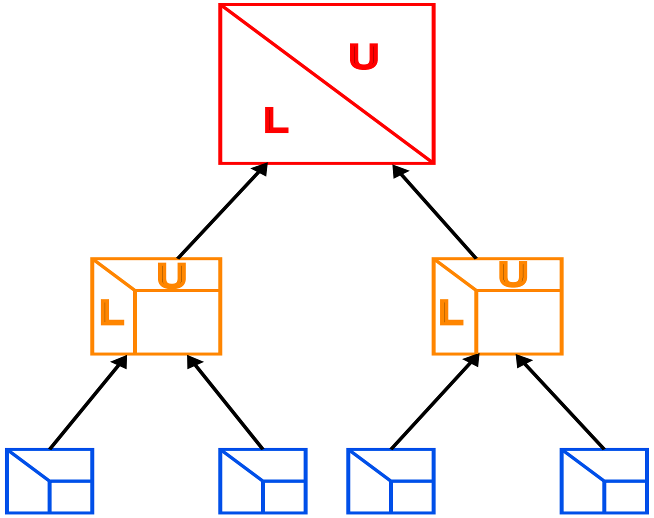



… depending on how updates are accumulated

DAG based
Supernodal: SuperLU




Tree based
Multifrontal: STRUMPACK, MUMPS



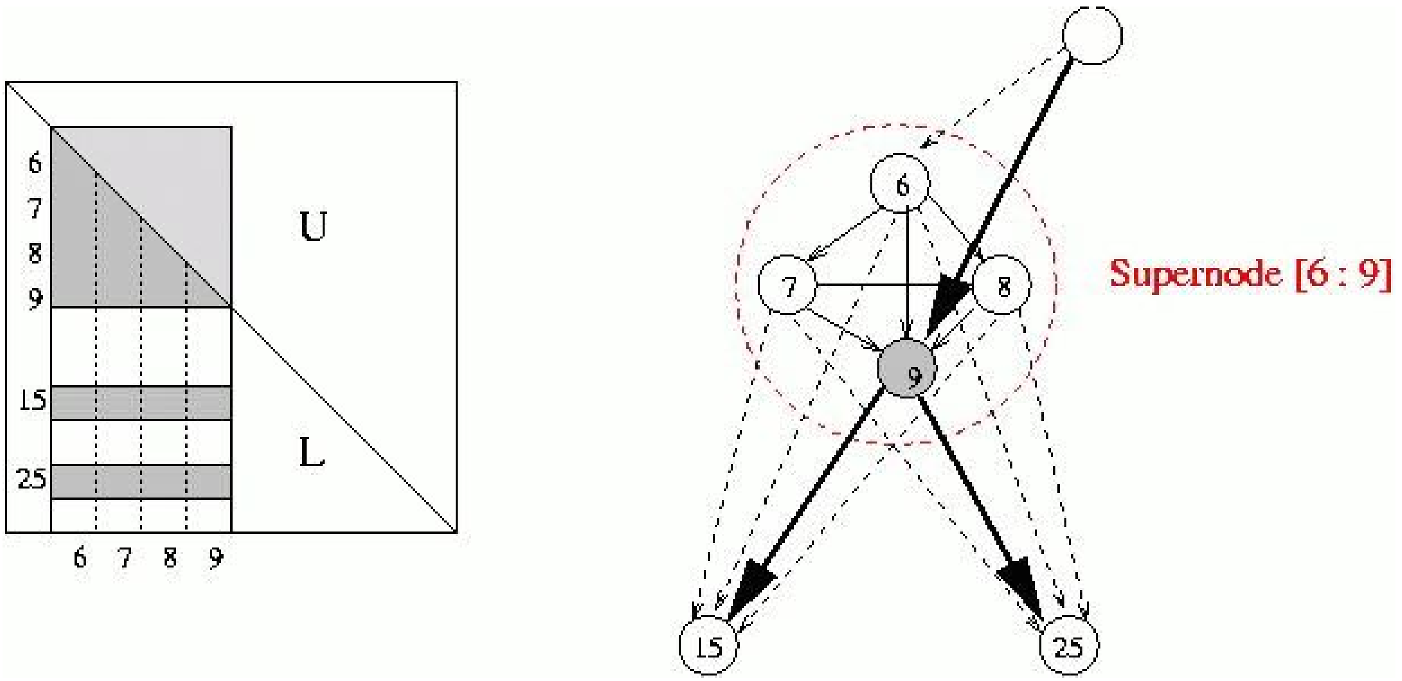

$S^{(j)} \leftarrow ((\ S^{(j)} - D^{(k1)}\ ) - D^{(k2)}\ ) - \dots$

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

# Supernode

Exploit dense submatrices in the factors

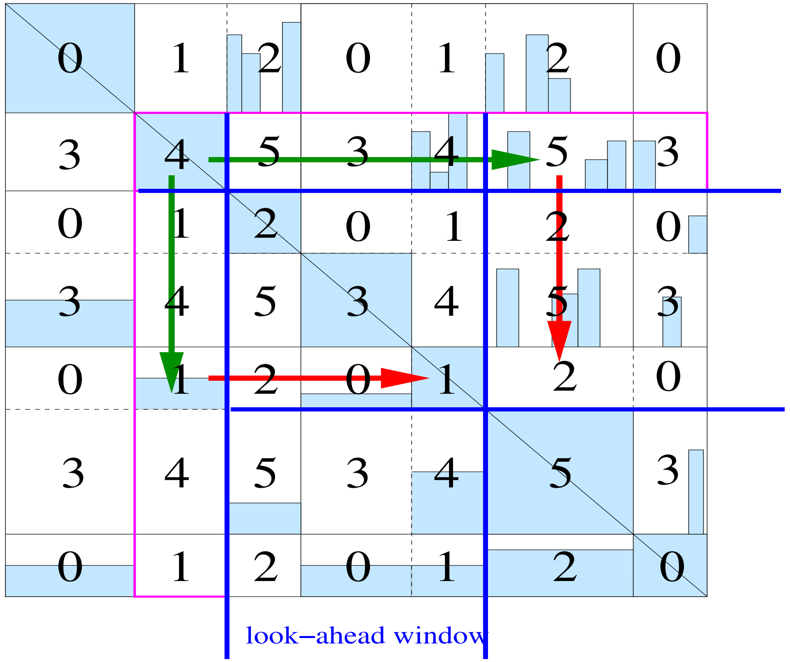
- Can use Level 3 BLAS
- Reduce inefficient indirect addressing (scatter/gather)
- Reduce graph traversal time using a coarser graph



# 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.

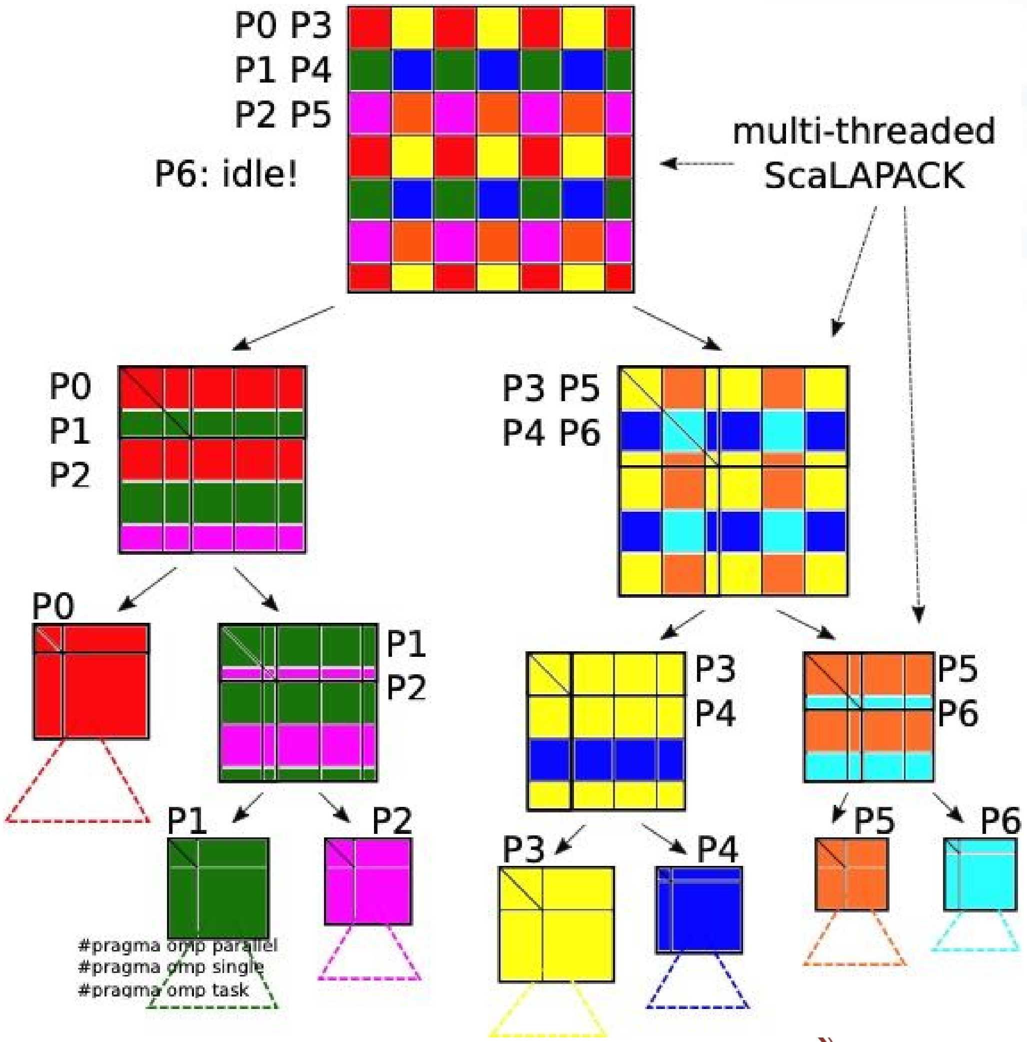



**MPI Process Grid**

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |

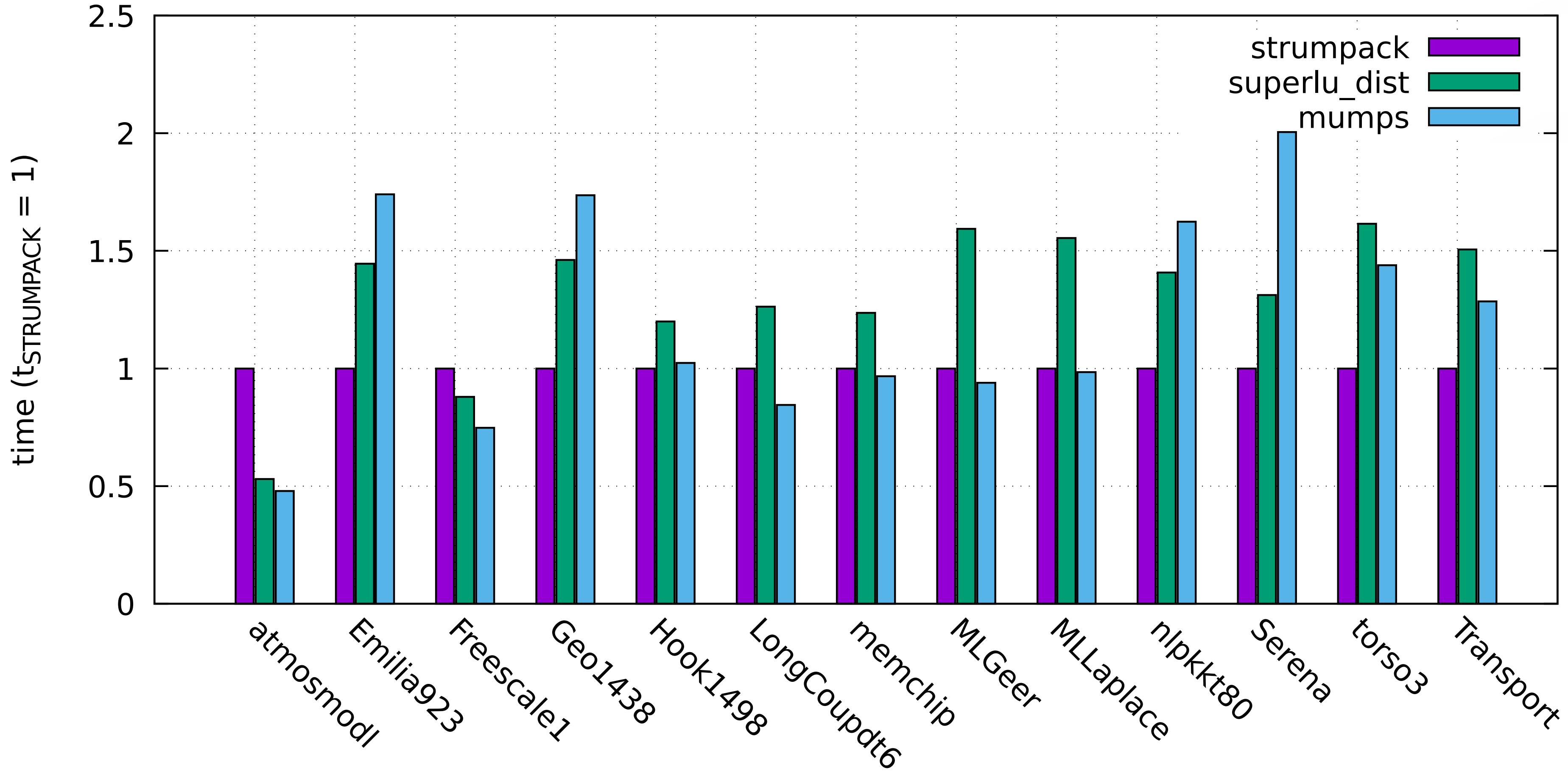
# Distributed separator-tree-based parallelism (internal to STRUMPACK)

- 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 | | |

# 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/

# Algorithm complexity (in bigO sense)

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

| | 2D problems<br>$N = k^2$ | | | 3D problems<br>$N = k^3$ | | |
|---|---|---|---|---|---|---|
| | 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 | $N$ | $N$ | $N$ | $N^\alpha$ 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, 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

## Rank Structured Solvers for Dense Linear Systems

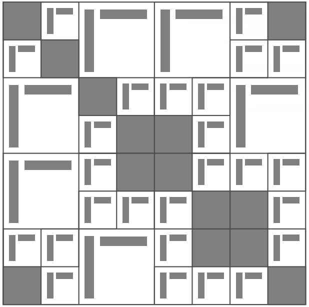
# Hierarchical Matrix Approximation

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

- Data-sparse, rank-structured, compressed

Hierarchical/recursive $2 \times 2$ matrix blocking, with blocks either:

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

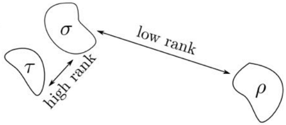


Use cases:

- 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

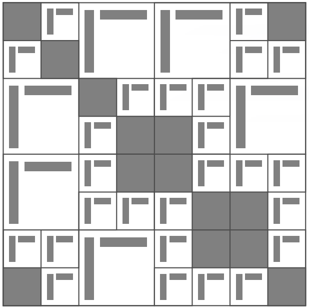



- Row cluster $\sigma$
- Column cluster $\tau$
- $\sigma \times \tau$ is compressible $\Leftrightarrow$

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

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

- 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



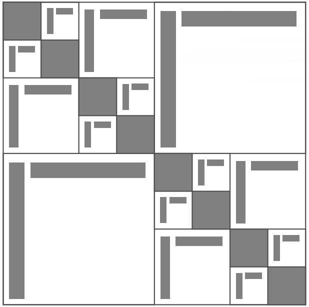
Hackbusch, W., 1999. *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices.* Computing, 62(2), pp.89-108.

# 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

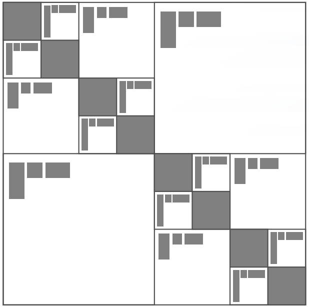
$$U_\sigma = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \hat{U}_\sigma$$

  with $\nu_1$ and $\nu_2$ children of $\sigma$ in the cluster tree.

- At lowest level

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

- Store only $\hat{U}_\sigma$, smaller than $U_\sigma$
- 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



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

# HSS: Hierarchically Semi Seperable

- Weak admissibility
- Off-diagonal blocks

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

- Nested bases

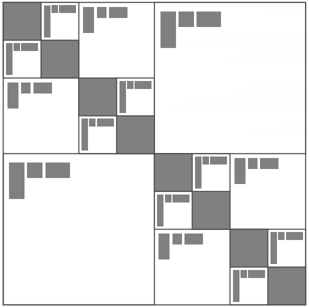
$$U_\sigma = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} \hat{U}_\sigma$$

with $\nu_1$ and $\nu_2$ children of $\sigma$ in the cluster tree.

- At lowest level

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

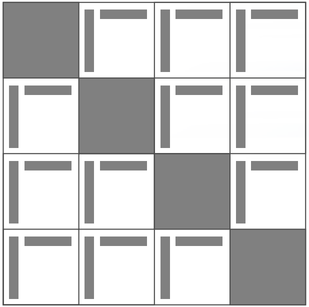
- Store only $\hat{U}_\sigma$, smaller than $U_\sigma$
- Complexity $\mathcal{O}(N) \leftrightarrow \mathcal{O}(N \log N)$ for HODLR
- HSS is special case of $\mathcal{H}^2$: $\mathcal{H}$ with nested bases



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

# BLR: Block Low Rank [1, 2]

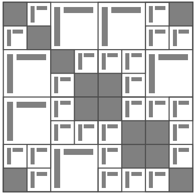
- Flat partitioning (non-hierarchical)
- Weak or strong admissibility
- Larger asymptotic complexity than $\mathcal{H}$, HSS, ...
- Works well in practice



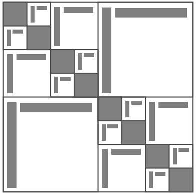

Mary, T. (2017). *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* (Doctoral dissertation).

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

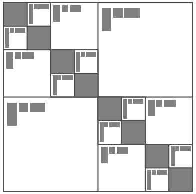
# Data-Sparse Matrix Representation Overview



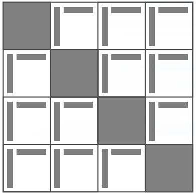
$\mathcal{H}$



HODLR



HSS



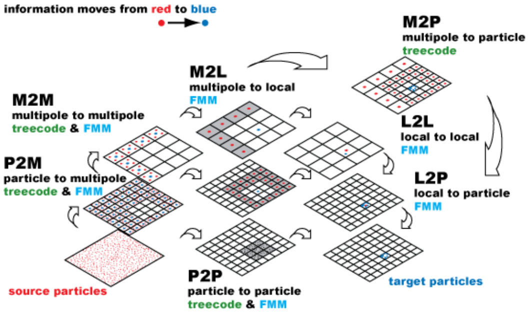
BLR

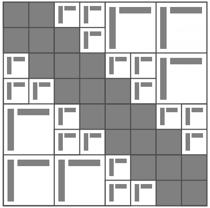
- Partitioning: **hierarchical** ($\mathcal{H}$, HODLR, HSS) or **flat** (BLR)
- Admissibility: **weak** (HODLR, HSS) or **strong** ($\mathcal{H}$, $\mathcal{H}^2$)
- Bases: **nested** (HSS, $\mathcal{H}^2$) or **not nested** (HODLR, $\mathcal{H}$, BLR)

# Fast Multipole Method [1]

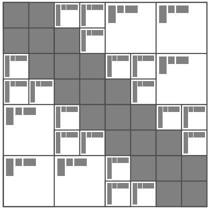
Particle methods like Barnes-Hut and FMM can be interpreted algebraically using hierarchical matrix algebra

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



Barnes-Hut



FMM

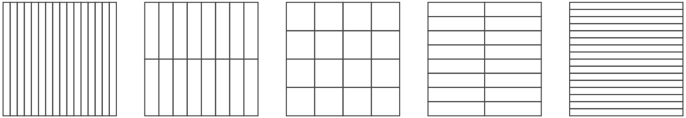

Greengard, L., and Rokhlin, V. *A fast algorithm for particle simulations.* Journal of computational physics 73.2 (1987): 325-348.



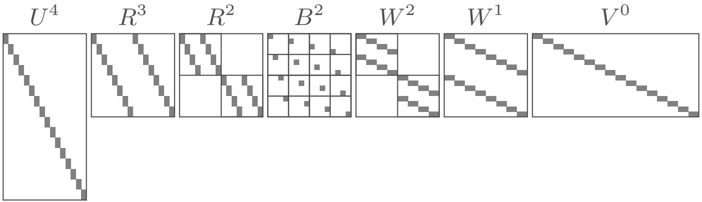

EXASCALE COMPUTING PROJECT

# Butterfly Decomposition [1]

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



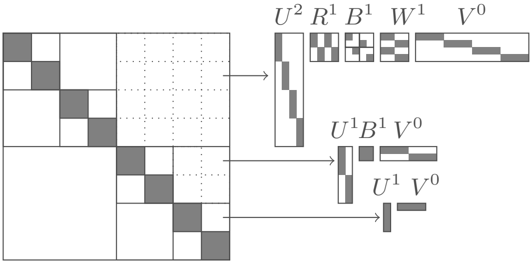
Multiplicative decomposition:




- 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



EXASCALE COMPUTING PROJECT

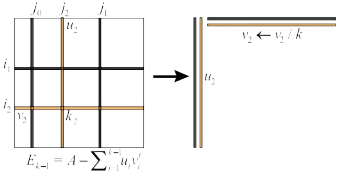
# Low Rank Approximation Techniques

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]

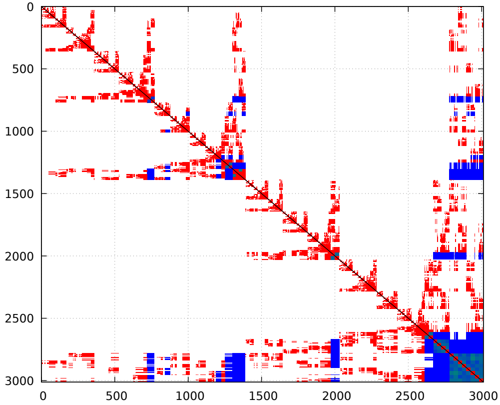
- Fast matrix-vector product: $S = A\Omega$
  Reduce dimension of $A$ by random projection with $\Omega$
- E.g., operator is sparse or rank structured, or the product of sparse and rank structured

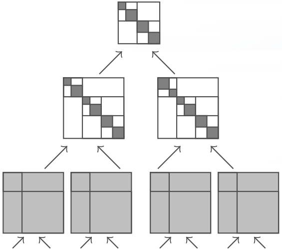
Halko, N., Martinsson, P.G., Tropp, J.A. (2011). *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.* SIAM Review, 53(2), 217-288.

# Approximate Multifrontal Factorization

# Sparse Multifrontal Solver/Preconditioner with Rank-Structured Approximations

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

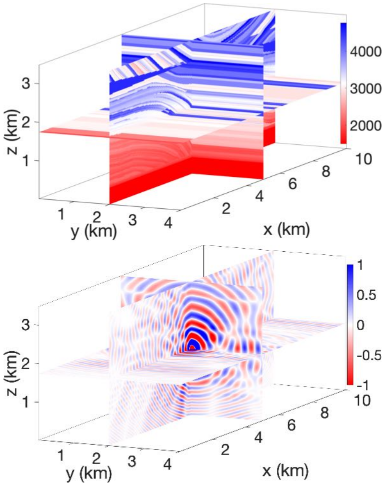


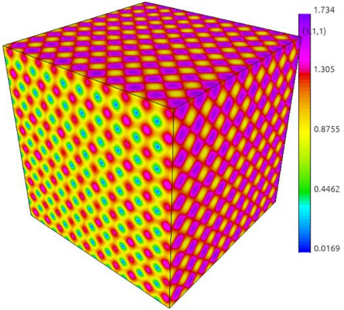
Only apply rank structured compression to largest fronts (dense sub-blocks), keep the rest as regular dense

# High Frequency Helmholtz and Maxwell

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



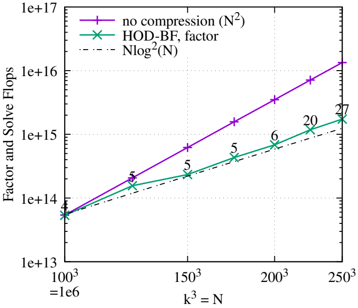

Marmousi2 geophyical elastic dataset



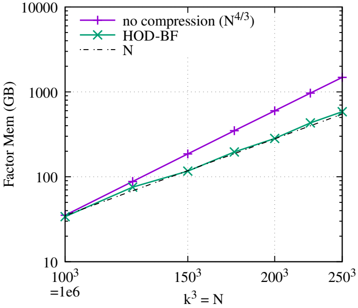

Indefinite Maxwell, using MFEM

# High Frequency Helmholtz and Maxwell

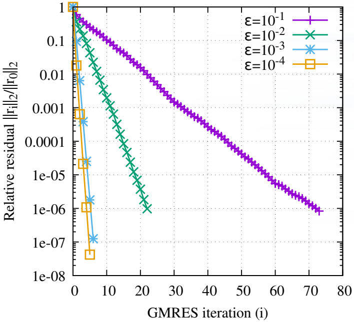
## Sparse multifrontal solver with HODBF compression




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 $\mathcal{O}(N^2)$

# 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`

# SuperU_DIST Hands-on session

# SuperLU_DIST with MFEM

xsdk-project.github.io/MathPackagesTraining2020/lessons/superlu_mfem/

Solve steady-state convection-diffusion equations

Get 2 compute nodes:  qsub -I -n 1 -t 30 -A  ATPESC2020 -q R.ATPESC2020_0806_1

cd HandsOnLessons/superlu_mfem

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

# SuperLU_DIST with MFEM

## xsdk-project.github.io/MathPackagesTraining2020/lessons/superlu_mfem/

- Convection-Diffusion equation (steady-state):

  HandsOnLessons/superlu_mfem/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)

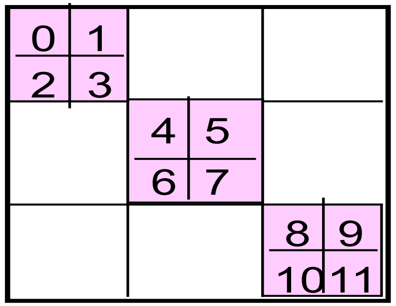
- Lessons learned
  - Direct solver can deal with ill-conditioned problems.
  - Performance may vary greatly with different elimination orders.

# SuperLU_DIST exercises:

## HandsOnLessons/superlu_mfem/superlu-dist/EXAMPLE

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

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




Block Jacobi preconditioner

# HandsOnLessons/superlu_mfem/superlu-dist/EXAMPLE

Four input matrices:

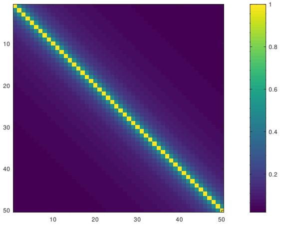
- g4.rua (16 dofs)
- g20.rua (400 dofs)
- big.rua (4960 dofs)
- stomach.rua (213k dofs, ~15 sec @ P=16)

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

# STRUMPACK Hands-On Session

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

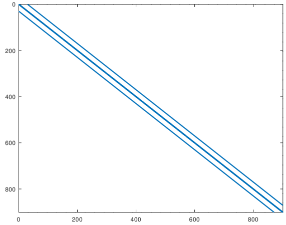
*HandsOnLessons/strumpack/run_testHODLR*




- See `HandsOnLessons/strumpack/README`
- Get a compute node:
```
qsub -I -n 1 -t 30 -A ATPESC2020
```
- Set OpenMP threads:
```
export OMP_NUM_THREADS=1
```
- Run example:
```
mpiexec -n 1 ./run_testHODLR 20000
```
- With description of command line parameters:
```
mpiexec -n 1 ./run_testHODLR 20000 --help
```
- Vary leaf size (smallest block size) and tolerance:
```
mpiexec -n 1 ./run_testHODLR 20000 --hodlr_rel_tol 1e-4 --hodlr_leaf_size 16
mpiexec -n 1 ./run_testHODLR 20000 --hodlr_rel_tol 1e-4 --hodlr_leaf_size 128
```
- Vary number of MPI processes:
```
mpiexec -n 12 ./run_testHODLR 20000 --hodlr_rel_tol 1e-8 --hodlr_leaf_size 16
mpiexec -n 12 ./run_testHODLR 20000 --hodlr_rel_tol 1e-8 --hodlr_leaf_size 128
```

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

*HandsOnLessons/strumpack/run_testMMdouble{MPIDist}*




- See `HandsOnLessons/strumpack/README`
- Get a compute node: `qsub -I -n 1 -t 30 -A ATPESC2020`
- Set OpenMP threads: `export OMP_NUM_THREADS=1`

- Run example:
```
mpiexec -n 1 ./run_testMMdouble pde900.mtx
```
- With description of command line parameters:
```
mpiexec -n 1 ./run_testMMDouble pde900.mtx --help
```
- Vary number of MPI processes:
```
mpiexec -n 1 ./run_testMMdouble pde900.mtx
mpiexec -n 12 ./run_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

***HandsOnLessons/strumpack/run_testPoisson3d{MPIDist}***

- See `HandsOnLessons/strumpack/README`
- Get a compute node: `qsub -I -n 1 -t 30 -A ATPESC2020`
- Set OpenMP threads: `export OMP_NUM_THREADS=1`

- Solve $40^3$ Poisson problem:

```
mpiexec -n 1 ./run_testPoisson3d 40 --help
```

- Enable BLR compression (sequential):

```
mpiexec -n 1 ./run_testPoisson3d 40 --sp_compression BLR --help
mpiexec -n 1 ./run_testPoisson3d 40 --sp_compression BLR --blr_rel_tol 1e-2
mpiexec -n 1 ./run_testPoisson3d 40 --sp_compression BLR --blr_rel_tol 1e-4
mpiexec -n 1 ./run_testPoisson3d 40 --sp_compression BLR --blr_leaf_size 128
mpiexec -n 1 ./run_testPoisson3d 40 --sp_compression BLR --blr_leaf_size 256
```

- Parallel, with HSS/HODLR compression:

```
mpiexec -n 12 ./run_testPoisson3dMPIDist 40
mpiexec -n 12 ./run_testPoisson3dMPIDist 40 --sp_compression HSS \
     --sp_compression_min_sep_size 1000 --hss_rel_tol 1e-2
mpiexec -n 12 ./run_testPoisson3dMPIDist 40 --sp_compression HODLR \
     --sp_compression_min_sep_size 1000 --hodlr_leaf_size 128
```

# Thank you!

# EXTRA SLIDES

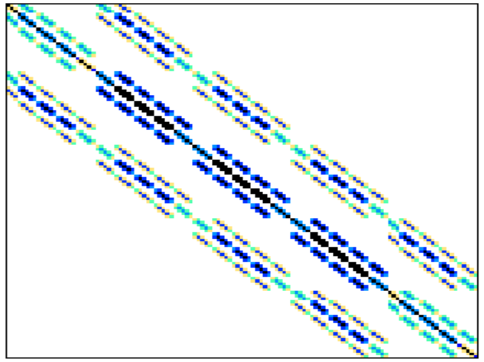


37

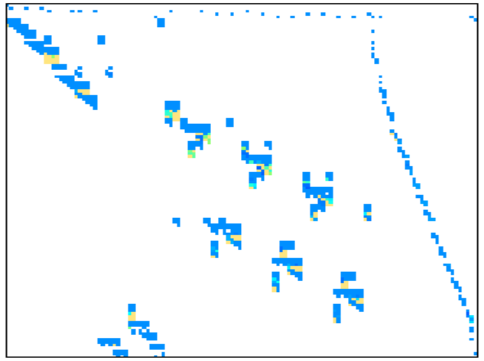
# Direct solvers can support wide range of applications

- fluid dynamics, structural mechanics, chemical process simulation, circuit simulation, electromagnetic fields, magneto-hydrodynamics, seismic-imaging, economic modeling, optimization, data analysis, statistics, . . .
- (non)symmetric, indefinite, ill-conditioned …
- Example: A of dimension $10^6$, 10~100 nonzeros per row
- Matlab: > spy(A)

Boeing/msc00726 (structural eng.)

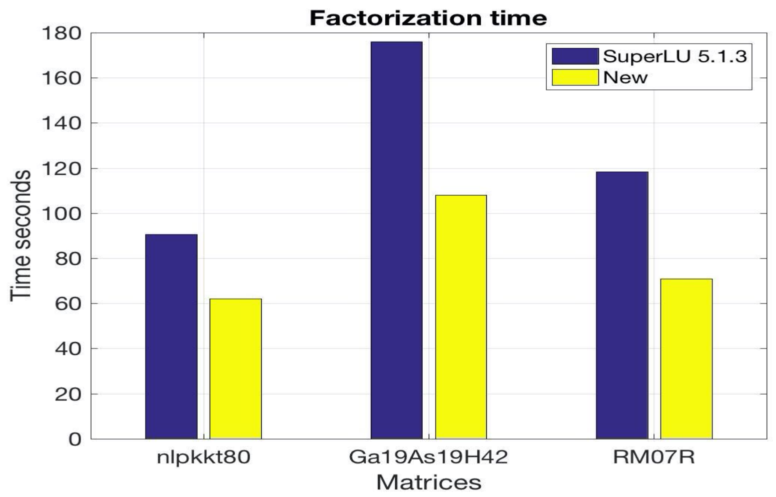


Mallya/lhr01 (chemical eng.)



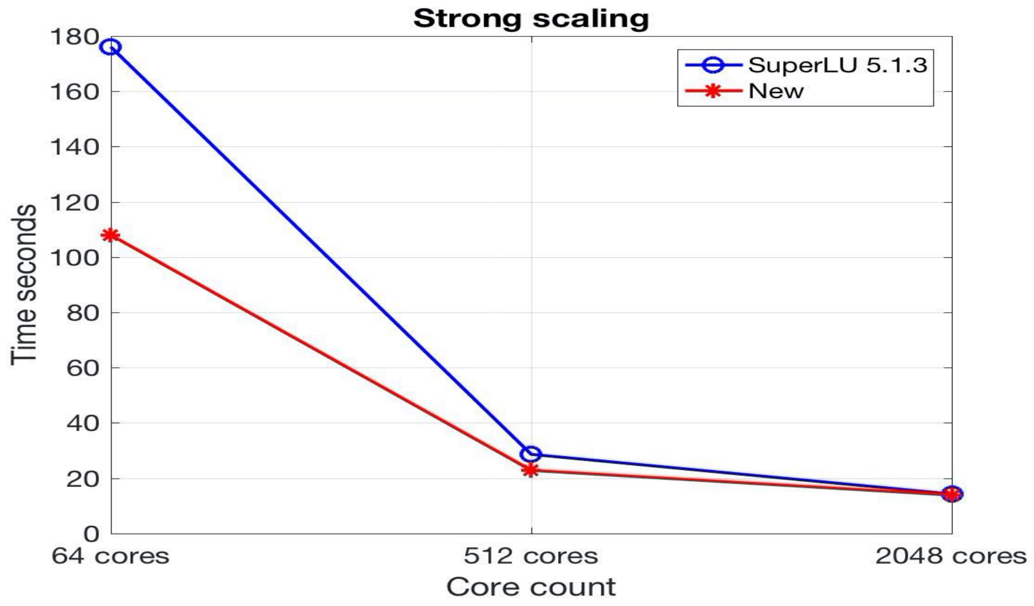
# SuperLU_DIST performance on Intel KNL

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




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

- **Strong scaling to 32 nodes**




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

# *Examples in examples/*

See README

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

# Available direct solvers

- Survey of different types of factorization codes

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

  - $LL^T$ (s.p.d.)
  - $LDL^T$ (symmetric indefinite)
  - LU (nonsymmetric)
  - QR (least squares)
  - Sequential, shared-memory (multicore), distributed-memory, out-of-core, few are GPU-enabled …

- Distributed-memory codes:
  - SuperLU_DIST (Li, Demmel, Grigori, Liu, Sao, Yamazaki)
    - accessible from PETSc, Trilinos, . . .
  - MUMPS, PasTiX, WSMP, . . .



41

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

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

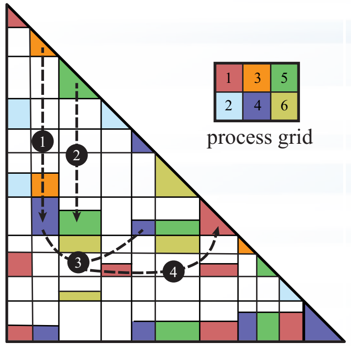
† Uses QR storage to statically accommodate any LU fill-in

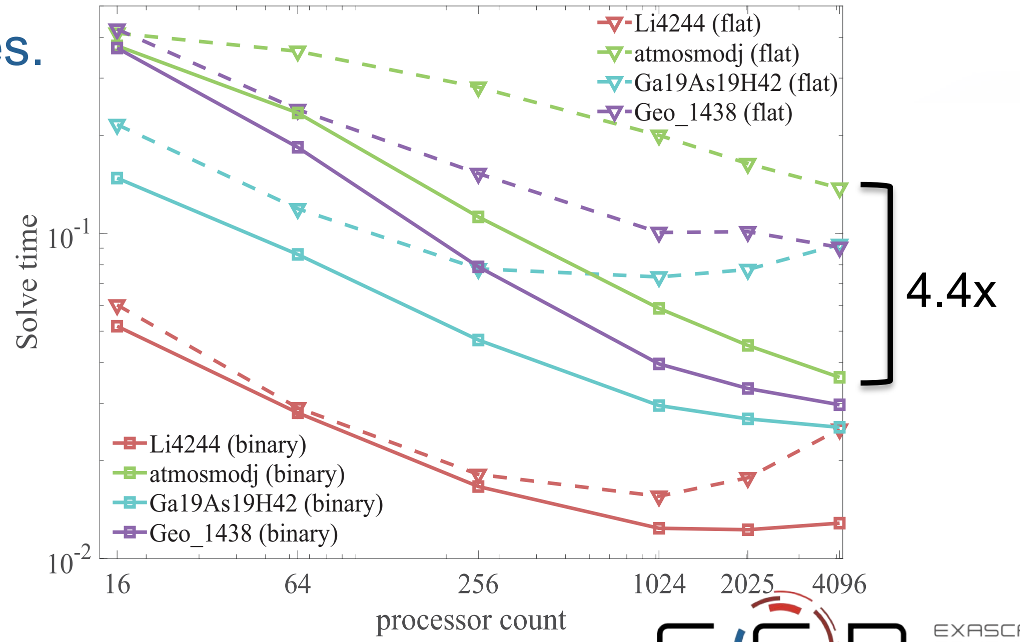
Abbreviations used in the table:

SPD = symmetric and positive definite
Sym = symmetric and may be indefinite
Sym-pat = symmetric nonzero pattern but unsymmetric values
Unsym = unsymmetric
HSL = Harwell Subroutine Library: http://www.cse.clrc.ac.uk/Activity/HSL

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

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





EXASCALE COMPUTING PROJECT

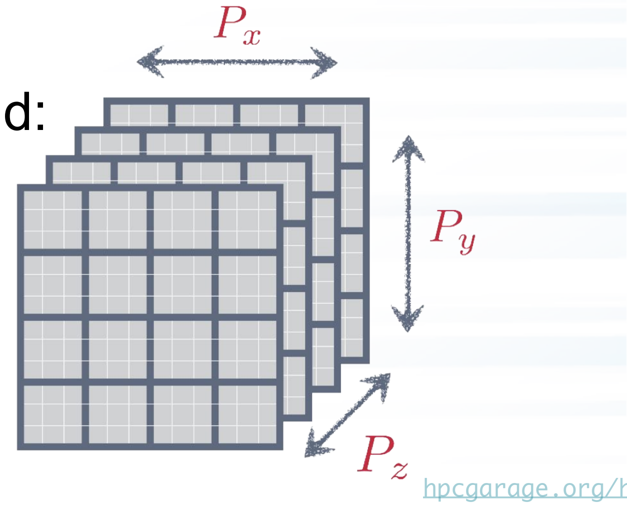
# Communication-avoid 3D sparse LU in SuperLU_DIST

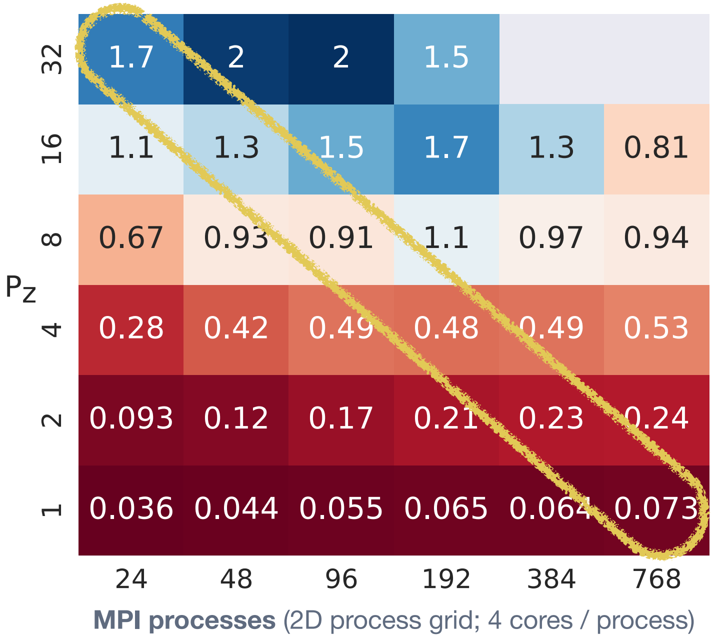
(P. Sao, X.S. Li, R. Vuduc, IPDPS 2018)

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

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

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




**2D to 3D:**
→ **23x speedup**

**Teraflop/s**
(32x procs → 2x speedup)

# Combining 3D algorithm with GPU acceleration

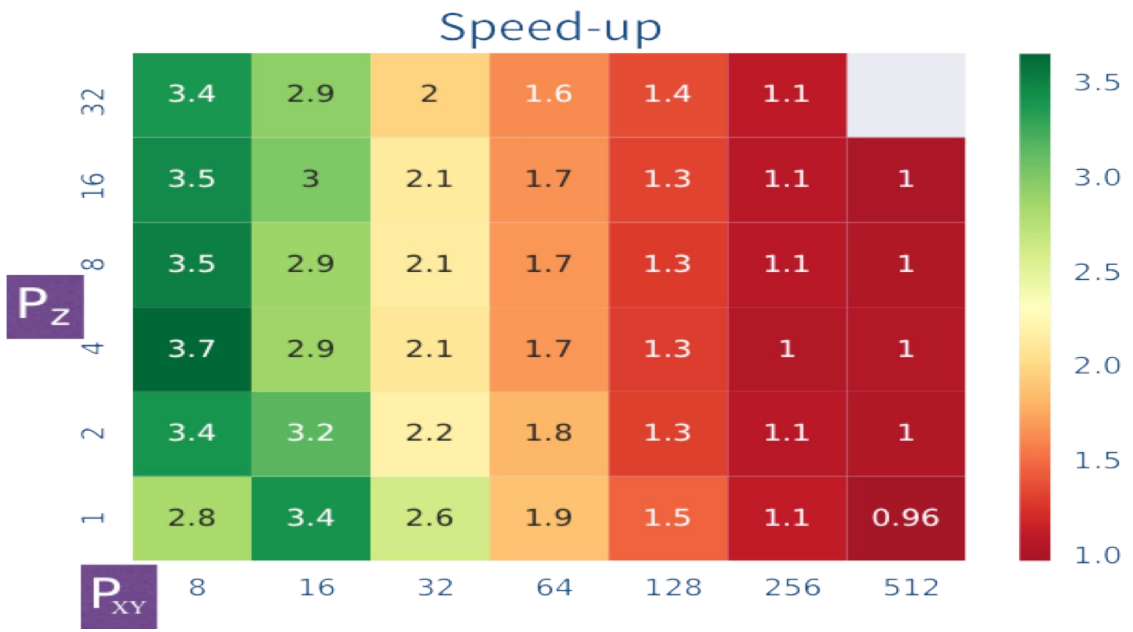
(*Sao, Vuduc, Li, JPDC preprint, 2018)*

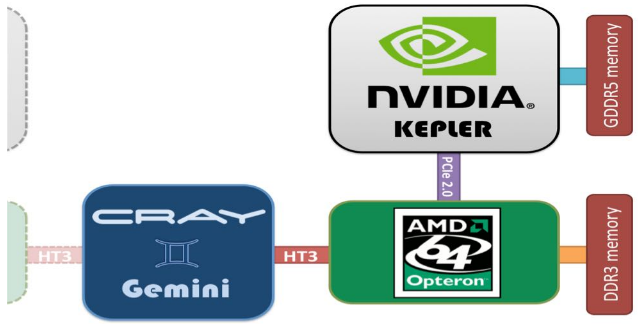
- Co-processor acceleration to reduce intra-node communication
  - *Sao, Vuduc, Li (EuroPar'14); Sao, Liu, Vuduc, Li (IPDPS'15)*
  - Offload Schur-complement update to GPU
- Empirical study on Cray XK7 (titan @ OLCF)

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

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

nlpkkt80



# SuperLU Installation

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

  Two ways of building:

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

# Use multicore, GPU

- Instructions in top-level README.
- To use OpenMP parallelism:

  Export OMP_NUM_THREADS=<##>

- To enable Nvidia GPU access, need to take the following 2 step:
  1. set the following Linux environment variable:

     ```
     export ACC=GPU
     ```

  2. Add the CUDA library location in make.inc: (see sample make.inc)

     ```
     ifeq "${ACC}" "GPU”
          CUDA_FLAGS = -DGPU_ACC
          INCS += -I<CUDA directory>/include
          LIBS += -L<CUDA directory>/lib64 -lcublas –lcudart
     endif
     ```

# Flowchart of iterative methods

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