Preconditioning using Rank-structured Sparse Matrix Factorization

Pieter Ghysels, Xiaoye Sherry Li, Gustavo Chávez, Yang Liu
Mathias Jacquelin, Esmond Ng

Lawrence Berkeley National Laboratory

pghysels@lbl.gov

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Introduction

Sparse direct factorization based solvers

- Are robust
- But expensive
  - Use a lot of memory: fill-in
  - Cost determined by dense algebra on largest dense submatrices

Introduction

Sparse direct factorization based solvers

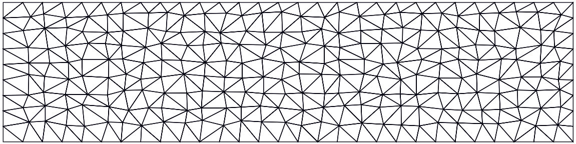
- Are robust
- But expensive
  - Use a lot of memory: fill-in
  - Cost determined by dense algebra on largest dense submatrices

Use low-rank approximation/compression for
fill-in in sparse LU factorization

- For many matrices from PDEs fill-in occurs in dense matrices with low rank off-diagonal blocks
  - Hierarchically Semi-Separable (HSS) matrices
- Gains in complexity (over exact direct solver)
- Fully algebraic sparse solver/preconditioner
- **STRUMPACK** software library

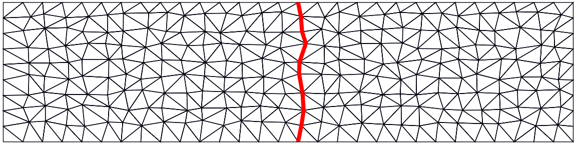
Jianlin Xia. *Randomized sparse direct solvers*. SIAM Journal on Matrix Analysis and Applications 34.1 (2013): 197-227.

Multifrontal Sparse LU Factorization [Duff & Reid '83]



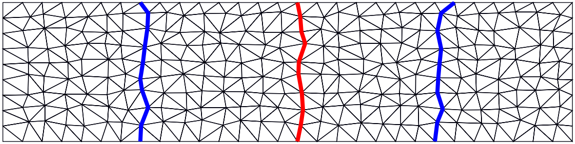
- Nested-dissection reordering
  - Separator/supernodal tree
  - (Par)Metis/(PT)Scotch graph partitioners
- For every separator
  - Dense frontal matrix
  - Partial LU factorization
  - Schur complement update
  - Extend-add: parent nodes "sum" Schur complements from the children
- Multifrontal solve
  - Forward and backward solve
  - Two traversals of the separator tree

Multifrontal Sparse LU Factorization [Duff & Reid '83]

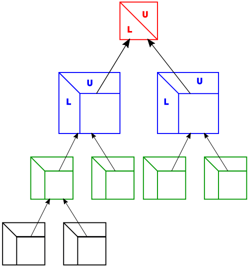


- Nested-dissection reordering
  - Separator/supernodal tree
  - (Par)Metis/(PT)Scotch graph partitioners
- For every separator
  - Dense frontal matrix
  - Partial LU factorization
  - Schur complement update
  - Extend-add: parent nodes "sum" Schur complements from the children
- Multifrontal solve
  - Forward and backward solve
  - Two traversals of the separator tree

Multifrontal Sparse LU Factorization [Duff & Reid '83]

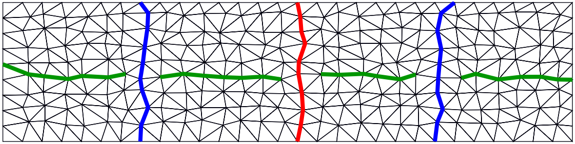


- Nested-dissection reordering
  - Separator/supernodal tree
  - (Par)Metis/(PT)Scotch graph partitioners
- For every separator
  - Dense frontal matrix
  - Partial LU factorization
  - Schur complement update
  - Extend-add: parent nodes "sum" Schur complements from the children
- Multifrontal solve
  - Forward and backward solve
  - Two traversals of the separator tree

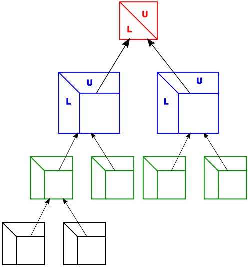


supernodal/separator tree

Multifrontal Sparse LU Factorization [Duff & Reid '83]



- Nested-dissection reordering
  - Separator/supernodal tree
  - (Par)Metis/(PT)Scotch graph partitioners
- For every separator
  - Dense frontal matrix
  - Partial LU factorization
  - Schur complement update
  - Extend-add: parent nodes "sum" Schur complements from the children
- Multifrontal solve
  - Forward and backward solve
  - Two traversals of the separator tree



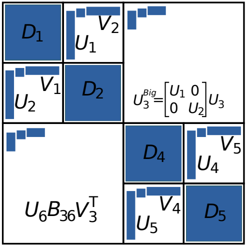
supernodal/separator tree

Hierarchically Semi-Separable (HSS) Matrices

- Data-sparse representation (like $\mathcal{H}$, $\mathcal{H}^2$, BLR, HODLR, ...)
- HSS is subset of $\mathcal{H}^2$-matrices, subset of $\mathcal{H}$-matrices
- Full rank matrix with low-rank off-diagonal blocks
- Hierarchical partitioning of the matrix

Hierarchically Semi-Separable (HSS) Matrices

- Data-sparse representation (like $\mathcal{H}$, $\mathcal{H}^2$, BLR, HODLR, ...)
- HSS is subset of $\mathcal{H}^2$-matrices, subset of $\mathcal{H}$-matrices
- Full rank matrix with low-rank off-diagonal blocks
- Hierarchical partitioning of the matrix



- Off-diagonal blocks are approximated as low-rank

$$A_{\nu_1,\nu_2} = A(I_{\nu_1}, I_{\nu_2}) = U_{\nu_1}^{\text{Big}} B_{\nu_1,\nu_2} (V_{\nu_2}^{\text{Big}})^*$$

- Diagonal blocks are full rank

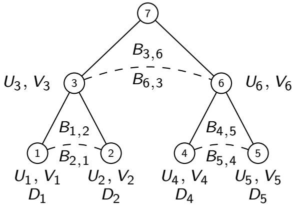
$$D_\tau = A(I_\tau, I_\tau)$$

- Column bases U^{Big} and row bases V^{Big} are nested

$$U_\tau^{\text{Big}} = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} U_\tau, \quad V_\tau^{\text{Big}} = \begin{bmatrix} V_{\nu_1} & 0 \\ 0 & V_{\nu_2} \end{bmatrix} V_\tau$$

Hierarchically Semi-Separable (HSS) Matrices

- Data-sparse representation (like $\mathcal{H}$, $\mathcal{H}^2$, BLR, HODLR, ...)
- HSS is subset of $\mathcal{H}^2$-matrices, subset of $\mathcal{H}$-matrices
- Full rank matrix with low-rank off-diagonal blocks
- Hierarchical partitioning of the matrix



- Off-diagonal blocks are approximated as low-rank

$$A_{\nu_1,\nu_2} = A(I_{\nu_1}, I_{\nu_2}) = U^{\text{Big}}_{\nu_1} B_{\nu_1,\nu_2} (V^{\text{Big}}_{\nu_2})^*$$

- Diagonal blocks are full rank

$$D_\tau = A(I_\tau, I_\tau)$$

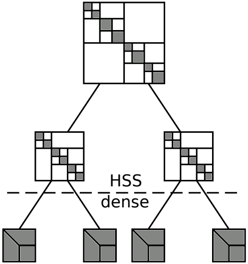
- Column bases U^{Big} and row bases V^{Big} are nested

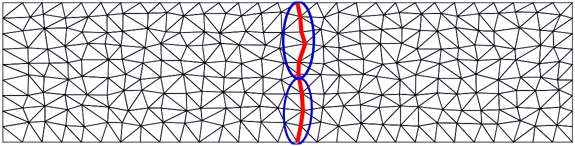
$$U^{\text{Big}}_\tau = \begin{bmatrix} U_{\nu_1} & 0 \\ 0 & U_{\nu_2} \end{bmatrix} U_\tau, \quad V^{\text{Big}}_\tau = \begin{bmatrix} V_{\nu_1} & 0 \\ 0 & V_{\nu_2} \end{bmatrix} V_\tau$$

- Fast HSS construction via randomized sampling
- Fast HSS ULV-like factorization (U and V^* unitary, L triangular)

Multifrontal HSS-enabled Sparse Solver/Preconditioner

- Only use HSS approximation for the largest frontal matrices
  - level(τ) $< \ell_s \rightarrow F_\tau$ is HSS
  - level(τ) $\geq \ell_s \rightarrow F_\tau$ is dense
- HSS partitioning based on recursive bisection of separator graph
  - Uses METIS partitioner
  - Goal is to reduce HSS ranks

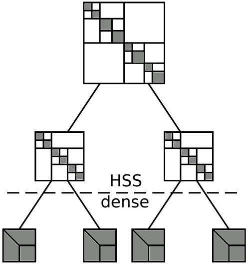


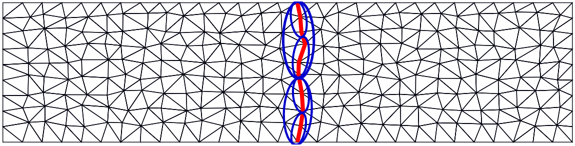


Multifrontal HSS-enabled Sparse Solver/Preconditioner

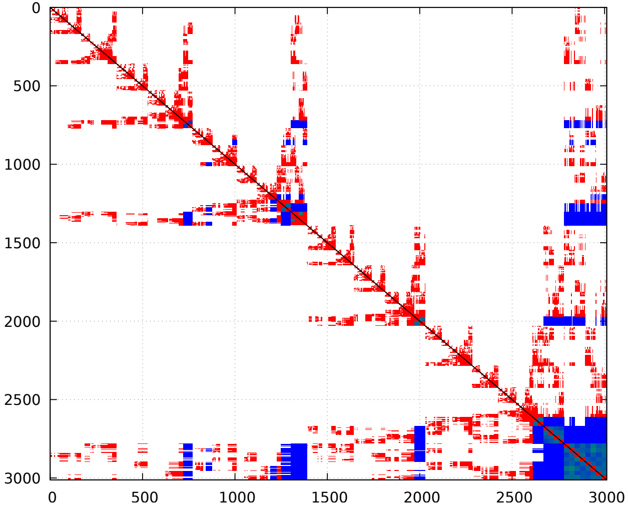
- Only use HSS approximation for the largest frontal matrices
  - level(τ) $< \ell_s \rightarrow F_\tau$ is HSS
  - level(τ) $\geq \ell_s \rightarrow F_\tau$ is dense

- HSS partitioning based on recursive bisection of separator graph
  - Uses METIS partitioner
  - Goal is to reduce HSS ranks

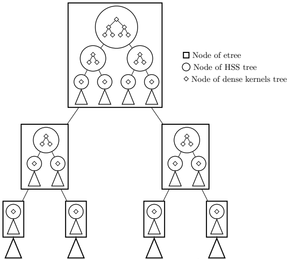




Multifrontal HSS-enabled Sparse Solver/Preconditioner



Sources of Parallelism

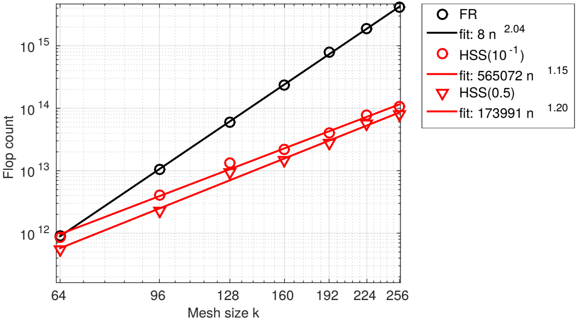


- Sources of parallelism
  - Supernodal/separator tree
  - HSS hierarchy
  - BLAS/LAPACK calls
- On-Node parallelism
  - Recursive traversal of trees using OpenMP tasks
  - Recursively split BLAS operations in smaller ones with OpenMP tasks
- Distributed memory parallelism
  - Proportional splitting of MPI communicators for sub-trees
  - ScaLAPACK for distributed levels of the trees

Work In Progress: SLATE integration

- On-node: SLATE with OpenMP tasks parallelism
- SLATE as ScaLAPACK alternative, with GPU off-loading

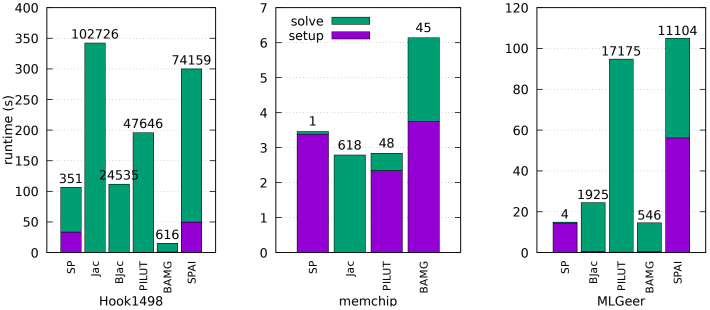
Fast Direct Solver for 3D Helmholtz – Flop Count



- Theory predicts $\mathcal{O}(n^{4/3} \log n)$ FLOPS for factorization
- HSS ranks grow with mesh dimension $\sim n^{\frac{1}{3}} = k$
- Use as a preconditioner

Preconditioner Setup and Solve Phases

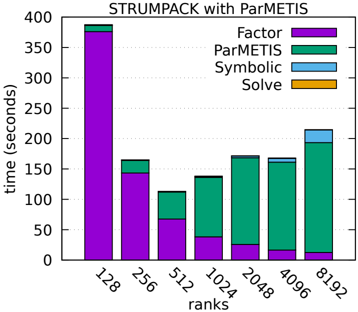
matrix	N	nnz	type	origin
	$(\times 10^6)$			
Hook_1498	1.5	59.3	SPD	struct. mechanics
memchip	2.7	13.3	non-sym	memory chip
ML_Geer	1.5	110.7	non-sym	poroelasticity

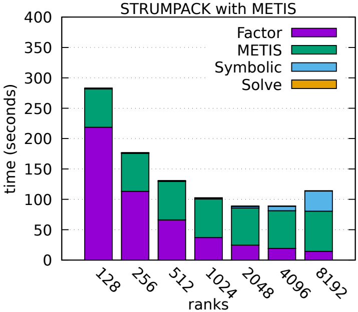


- 4 nodes, 96 Intel® Ivy Bridge cores
- For memchip, solver acts as direct method (small frontal matrices)
- AMG very efficient for many PDE based systems

ParMETIS Matrix Ordering Preprocessing Phase

Matrix reordering to reduce fill-in becomes a major bottleneck





- ParMETIS issues: poor scaling, worsening quality, . . .
- Possible remedy: only use a handful of compute nodes
- Good strong scaling for numerical factorization

The Graph Laplacian

Graph $G(V,E)$, vertices $V = \{v_i\}$, edges $E = \{e_{ij} = (v_i, v_j)\}$, $i, j \in \{i, \dots, n\}$

- Adjacency matrix

$$A_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Degree matrix

$$D = \text{diag}(d_i) = \text{diag}(\text{degree}(v_i))$$

  where $\text{degree}(v_i)$ is the number of edges incident to vertex v_i

- Graph Laplacian

$$L(G) = D - A = \begin{cases} -1 & \text{if } (v_i, v_j) \in E \\ d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- $L(G)$ is positive semi-definite
- $\lambda_0 = 0$, $v_0 = c^{st}$ (constant vector)
- If G is connected, then λ_0 has multiplicity 1

Spectral Bisection and the Fiedler Vector

Fiedler vector

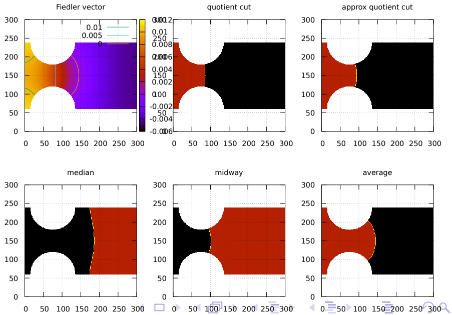
- Eigenvector v_1 belonging to λ_1, the smallest (nonzero) eigenvalue of $L(G)$
- Fiedler vector is very smooth

Spectral bisection based on Fiedler vector values $F_i \equiv F(v_i)$

$V_1 = \{\, v_i \mid F_i \leq c\}$

$V_2 = \{\, v_i \mid F_i > c\}$

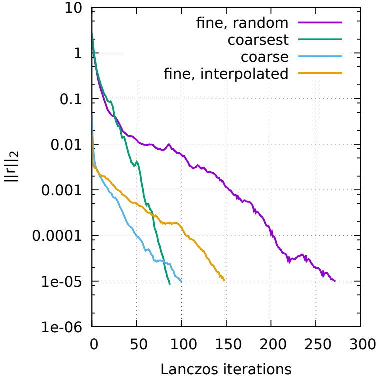
- $c = \text{mean}\{F_0, F_1, \ldots, F_n\} = 0$
- $c = \text{median}\{F_0, F_1, \ldots, F_n\}$
- $c = (\min(F_i) + \max(F_i))/2$
- $c = \text{argmin}(Q(V_1, V_2))$
- $\ldots$



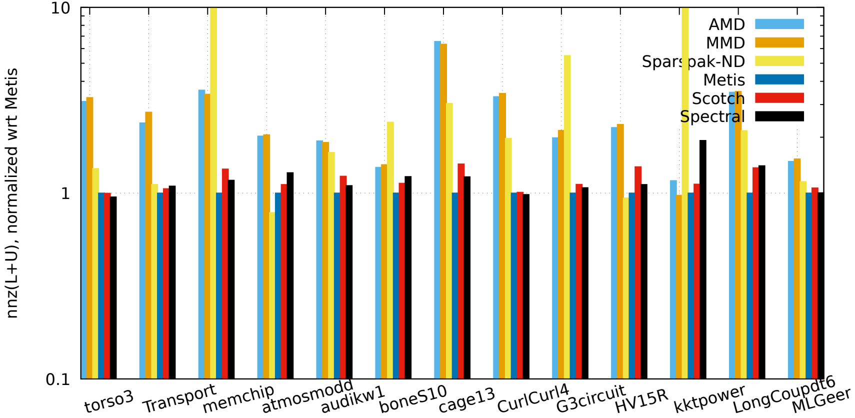
Computing the Fiedler Vector

- Graph coarsening
  - Group neighboring vertices
  - Define graph Laplacian on coarser graph
  - Weighted coarsening
- Compute Fiedler vector on coarser graph
  - Recursive call
  - LAPACK `ssyevx`/`dsyevx` ($|V| < 30$)
- Interpolate Fiedler vector to original graph
- Lanczos eigensolver on original graph
  - Start with interpolated Fiedler vector

Works well for *low accuracy*



Spectral Nested Dissection Ordering Quality Comparison



Typically slightly worse than Metis, comparable to Scotch

Shared Memory Parallel Implementation

- Exploit multiple levels of parallelism
  - Bisection
    - Fiedler vector computation
  - Recursive call for unconnected subgraphs
    - Reorder subgraphs concurrently
    - Spawn a new tasks for each subgraph
- For the top separator
  - Parallelism exclusively from Fiedler vector computation
- Lower down the recursion
  - Parallelism from many concurrent sub-graphs
- Multiple eigensolves going on concurrently
- Possible load imbalance
- Need for dynamic scheduling
- Carefully control granularity

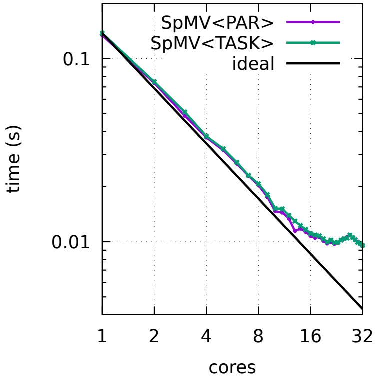
Nested Dissection with OpenMP Tasking

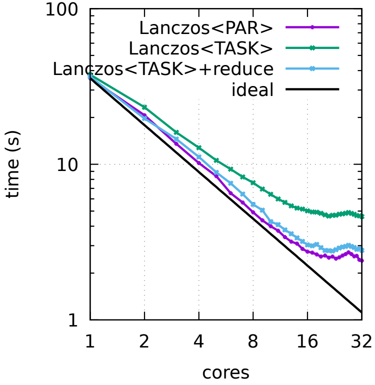
```
PPt<intt> nd_recursive(const Graph<intt>& g, const NDOptions& opts, int lvl) {
  // limit number of tasks
  auto par = (lvl < opts.max_task_lvl()) ?
    ThreadingModel::TASK : ThreadingModel::SEQ;

  if (g.n() <= opts.dissection_cutoff()) return amd(g);

  // handle nodes with degree 0 separately
  // check if g is connected, if not recursion on connected parts

  auto F = compute_Fiedler(par, g, opts);    // multilevel Fiedler computation
  auto c = get_cut_value(par, g, F, opts);            // minimize conductance
  auto part = vertex_separator(par, g, c, opts);  // edge to vertex separator
  auto [A, B] = g.extract_domains(par, part);

  PPt<intt> pA, pB;
#pragma omp task if(par==ThreadingModel::TASK) default(shared)
    pA = nd_recursive(A, opts, lvl+1);
#pragma omp task if(par==ThreadingModel::TASK) default(shared)
    pB = nd_recursive(B, opts, lvl+1);
#pragma omp taskwait

  return ...  // combine pA, pB
}
```

- `max_task_lvl = std::log2(omp_get_max_threads()) + 3;`

OpenMP Scaling

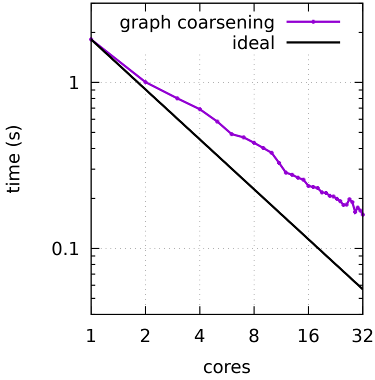
NERSC Cori Haswell – Flan_1565.mtx

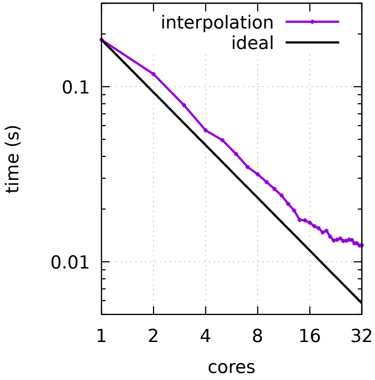




- SpMV: Application of Graph Laplacian to a vector
- PAR: OpenMP parallel for loop for each loop (axpy, spmv, dot, ..)
- TASK: single OpenMP parallel region, taskloop parallelism
- taskloop reduction not supported yet → manual implementation

OpenMP Scaling

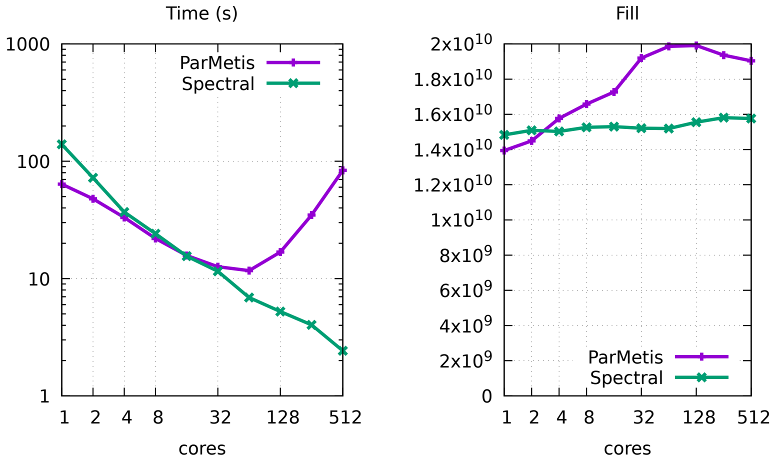




Distributed Memory Spectral Nested Disscetion

NERSC Cori, Haswell

Queen_1417.mtx, N=4,147,110 nnz=333,646,394

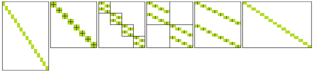


- Parallel multilevel Lanczos, split MPI communicator after bisection
- Spectral quality degrades: edge to vertex separator, parallel coarsening

Conclusions and Outlook

Rank-Structured Preconditioner STRUMPACK

- Preconditioner for range of PDE based problems
- Achieves (nearly) linear scaling for certain problems
- Not all problems compress well: other rank structured formats!
  - HODLR - Hierarchically Off-Diagonal Low-Rank
  - BLR - Block Low-Rank
  - Butterfly, based on FFT ideas



- Never form dense front: randomized sampling, ACA

Spectral Nested-Dissection

- Okay quality, worse than ParMetis, better than PTScotch
  - Improve with Kernighan-Lin or Fiduccia-Mattheyses?
- Efficient and scalable implementation: MPI+OpenMP (+GPU?)
- Prec LOBPCG, communication avoiding/hiding eigensolvers
- Integration in SuperLU, STRUMPACK
- Optimize for *data-sparsity* in rank-structured solver?
- K-way graph partitioning

The End

Thanks for listening!

Questions?