A comparison of parallel rank-structured solvers

François-Henry Rouet

Livermore Software Technology Corporation, Lawrence Berkeley National Laboratory

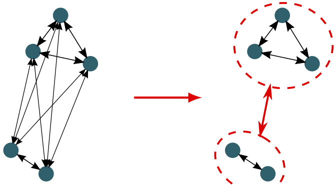
Joint work with:
- LSTC: J. Anton, C. Ashcraft, C. Weisbecker
- LBNL: P. Ghysels, X. S. Li
- MUMPS project: P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, T. Mary

LSTC Livermore Software Technology Corp.

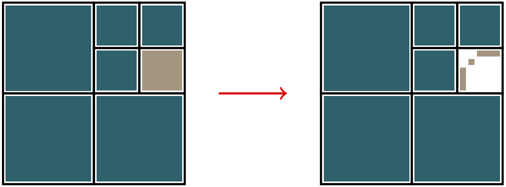
SIAM Conference on Parallel Processing, April 15th, 2016

Low-rankness

- Low-rank/structured methods rely on data sparsity, similar to the Fast Multipole Method.



- In algebraic terms: some off-diagonal blocks of the input matrix are low-rank; they can be compressed.



- NB: sometimes this applies to intermediate matrices (not the input matrix), e.g., in sparse factorizations.

Classes of structured matrices

Most structured matrices belong to the class of Hierarchical matrices ($\mathcal{H}-$matrices) [Hackbusch, Bebendorf, Börm, Grasedyck. . .].

- $\mathcal{H}^2$ (Hackbusch, Börm, et al.)
- HSS (Chandrasekaran, Jia, et al.)
- HODLR (Darve et al.)
- BLR (Amestoy, Ashcraft, et al.)
- $+$ SSS, MHS, . . .

In this talk:

- We review some algorithmic and implementation differences.
- We discuss a comparison of HSS and BLR.

Differences

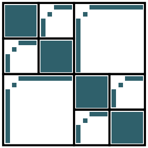
Three criteria differentiate all the low-rank formats:

Differences

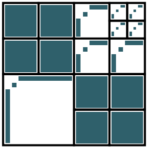
Three criteria differentiate all the low-rank formats:

- Clustering/partitioning: off-diagonal blocks can be refined or not.

The partitioning is defined by a single tree whose leaves cluster $[1, n]$.



vs



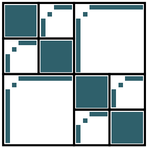
The partitioning is defined by the product of two trees (rows $\times$ columns).

Differences

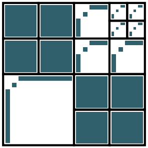
Three criteria differentiate all the low-rank formats:

- Clustering/partitioning: off-diagonal blocks can be refined or not.

The partitioning is defined by a single tree whose leaves cluster $[1, n]$.



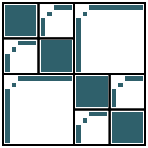
vs



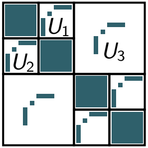
The partitioning is defined by the product of two trees (rows $\times$ columns).

- Nested basis or not.

Blocks have independent compressed representations (bases).



vs



Shared information:

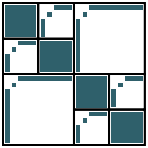
$$U_3^{\text{big}} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} U_3$$

Differences

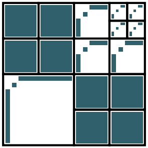
Three criteria differentiate all the low-rank formats:

- Clustering/partitioning: off-diagonal blocks can be refined or not.

The partitioning is defined by a single tree whose leaves cluster $[1, n]$.



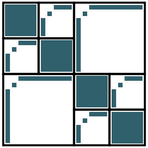
vs



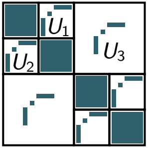
The partitioning is defined by the product of two trees (rows $\times$ columns).

- Nested basis or not.

Blocks have independent compressed representations (bases).



vs

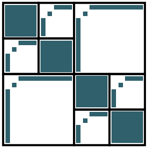


Shared information:

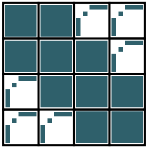
$$U_3^{\text{big}} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} U_3$$

- Buffer zone next to the diagonal or not ("strong admissibility").

Assumes interaction between two clusters is low-rank.



vs

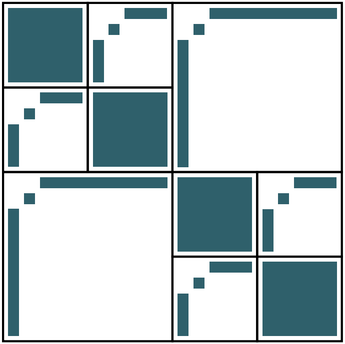


Blocks next to the diagonal not "admitted" (compressed).

Main classes of hierarchical matrices

HODLR (Darve et al.)

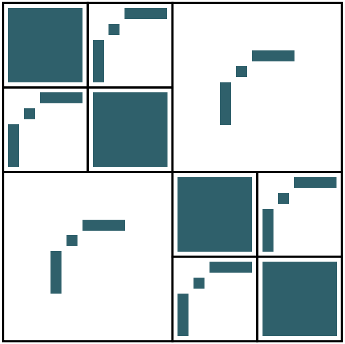
- No nested bases.
- No off-diagonal refinement.
- No buffer zone.



Main classes of hierarchical matrices

HSS (Chandrasekaran, Jia. . .)

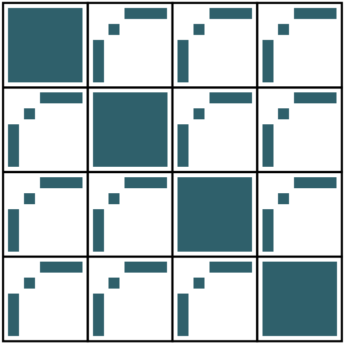
- Nested bases.
- No off-diagonal refinement.
- No buffer zone.



Main classes of hierarchical matrices

BLR (Amestoy, Ashcraft, et al.)

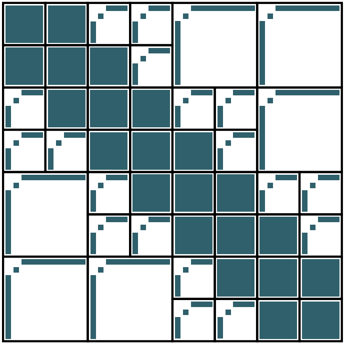
- No nested bases.
- Refine off-diagonal blocks.
- Can do buffer zone.



Main classes of hierarchical matrices

Barnes-Hut ("tree code")

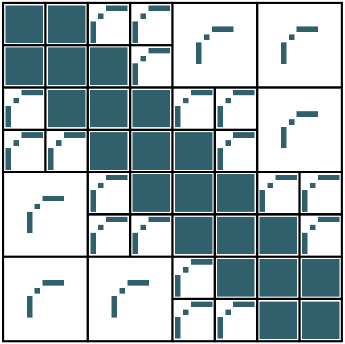
- No nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



Main classes of hierarchical matrices

Fast Multipole Method (Greengard & Rokhlin)

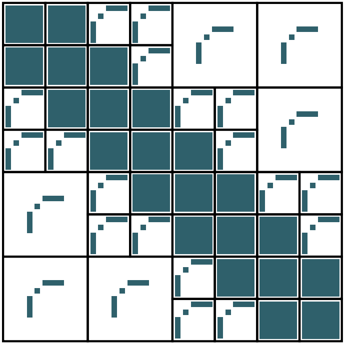
- Nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



Main classes of hierarchical matrices

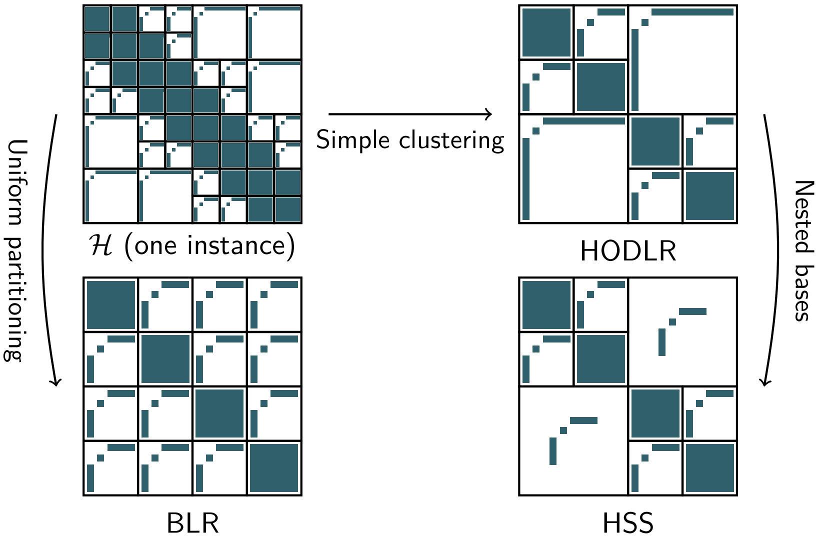
Fast Multipole Method (Greengard & Rokhlin)

- Nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



$\mathcal{H} \rightarrow \mathcal{H}^2 \equiv$ Barnes-Hut $\rightarrow$ FMM

The four formats



Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).

Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).
- Rank-Revealing QR (Householder or Gram-Schmidt). Cost $O(mnk)$. Strong RRQR might reduce ranks but is more costly.

Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).
- Rank-Revealing QR (Householder or Gram-Schmidt). Cost $O(mnk)$. Strong RRQR might reduce ranks but is more costly.
- Interpolative Decomposition (ID) is RRQR + 1 step:

$$B = QR\Pi^{-1} = Q\left[R_1 R_2\right]\Pi^{-1} = (QR_1)\left[I \ R_1^{-1}R_2\right]\Pi^{-1} = B(:,J)X$$

Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).
- Rank-Revealing QR (Householder or Gram-Schmidt). Cost $O(mnk)$. Strong RRQR might reduce ranks but is more costly.
- Interpolative Decomposition (ID) is RRQR + 1 step:

  $$B = QR\Pi^{-1} = Q\,[R_1 R_2]\,\Pi^{-1} = (QR_1)\left[I \; R_1^{-1} R_2\right]\Pi^{-1} = B(:, J)X$$

- Adaptive Cross Approximation (Bebendorf) is essentially rank-revealing LU and a similar trick to get

  $$B = X\,B(I, J)\,Y$$

  Cost $O(k^2 n)$. In some applications people choose I, J a priori.

Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).
- Rank-Revealing QR (Householder or Gram-Schmidt). Cost $O(mnk)$. Strong RRQR might reduce ranks but is more costly.
- Interpolative Decomposition (ID) is RRQR + 1 step:

$$B = QR\Pi^{-1} = Q\,[R_1 R_2]\,\Pi^{-1} = (QR_1)\left[I\ R_1^{-1}R_2\right]\Pi^{-1} = B(:,J)X$$

- Adaptive Cross Approximation (Bebendorf) is essentially rank-revealing LU and a similar trick to get

$$B = X\,B(I,J)\,Y$$

  Cost $O(k^2 n)$. In some applications people choose I, J a priori.
- CUR (Mahoney & Drineas), or pseudo-skeleton decomposition, is essentially a two-sided ID:

$$B = CUR = B(:,J)\,U\,B(I,:)$$

Compression kernel

Compression of an $m \times n$ block B:

- SVD: optimal but costly ($O(mn^2)$).
- Rank-Revealing QR (Householder or Gram-Schmidt). Cost $O(mnk)$. Strong RRQR might reduce ranks but is more costly.
- Interpolative Decomposition (ID) is RRQR + 1 step:

  $$B = QR\Pi^{-1} = Q\left[R_1 R_2\right]\Pi^{-1} = (QR_1)\left[I \ R_1^{-1}R_2\right]\Pi^{-1} = B(:,J)X$$

- Adaptive Cross Approximation (Bebendorf) is essentially rank-revealing LU and a similar trick to get

  $$B = X\,B(I,J)\,Y$$

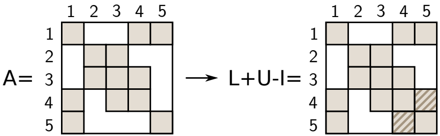
  Cost $O(k^2 n)$. In some applications people choose I, J a priori.
- CUR (Mahoney & Drineas), or pseudo-skeleton decomposition, is essentially a two-sided ID:

  $$B = CUR = B(:,J)\,U\,B(I,:)$$

- BDLR (Darve et al.) is a new technique that looks at the underlying graph to pick some interesting rows/columns.

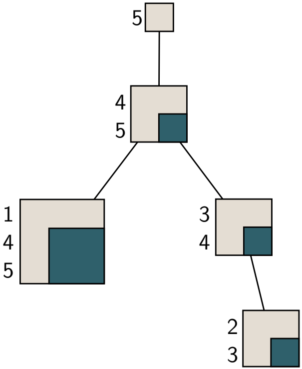
Low-rank representations and sparse solvers

Many LR techniques embedded in the multifrontal method [Duff & Reid '83]. Related: "sweeping preconditioner" [Ying, Engquist,...], elliptic solver [Chavez et al. '16]...



Traverse the tree bottom-up; at each node (a.k.a. frontal matrix):

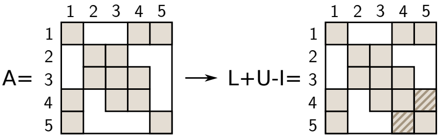
- Partial factorization (yields parts of L/U).
- Compute a contribution block (Schur complement) to be used at the parent.



Elimination tree

Low-rank representations and sparse solvers

Many LR techniques embedded in the multifrontal method [Duff & Reid '83]. Related: "sweeping preconditioner" [Ying, Engquist,. . .], elliptic solver [Chavez et al. '16]. . .

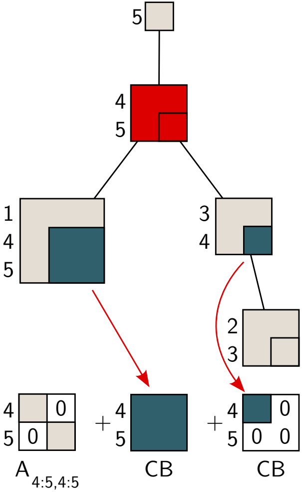


Traverse the tree bottom-up; at each node (a.k.a. frontal matrix):

- Partial factorization (yields parts of L/U).
- Compute a contribution block (Schur complement) to be used at the parent.

  Assembly/extend-add operation:

  $F = A_{I,J} \oplus \text{CB}_{\text{child 1}} \oplus \text{CB}_{\text{child 2}} \oplus \dots$



Extend-add and low-rank representations

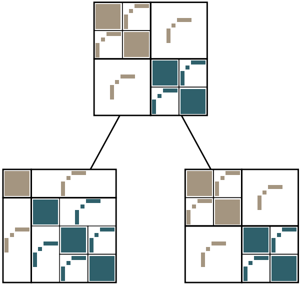
How to do extend-add with compressed matrices?

- BLR: contribution blocks not compressed [Amestoy et al. '12].

Extend-add and low-rank representations

How to do extend-add with compressed matrices?

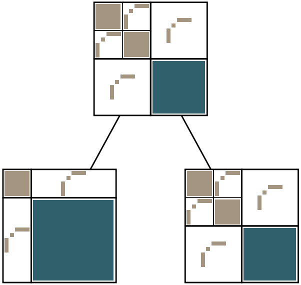
- BLR: contribution blocks not compressed [Amestoy et al. '12].
- HSS, [Jia et al. '09], serial 2D code. HSS extend-add. Slow and hard to parallelize. Hard to extend to algebraic.



Extend-add and low-rank representations

How to do extend-add with compressed matrices?

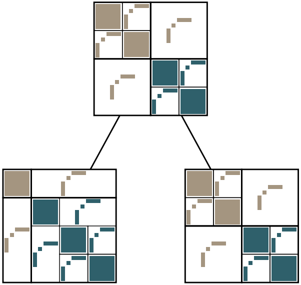
- BLR: contribution blocks not compressed [Amestoy et al. '12].
- HSS, [Jia et al. '09], serial 2D code. HSS extend-add. Slow and hard to parallelize. Hard to extend to algebraic.
- HSS, [Wang et al. '15], parallel geometric code. Contribution blocks not compressed. Simple but penalty on memory footprint.



Extend-add and low-rank representations

How to do extend-add with compressed matrices?

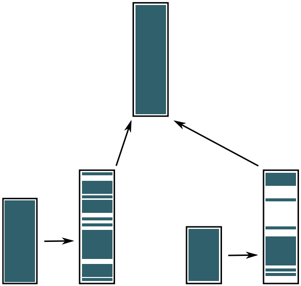
- BLR: contribution blocks not compressed [Amestoy et al. '12].
- HSS, [Jia et al. '09], serial 2D code. HSS extend-add. Slow and hard to parallelize. Hard to extend to algebraic.
- HSS, [Wang et al. '15], parallel geometric code. Contribution blocks not compressed. Simple but penalty on memory footprint.
- HSS, [Ghysels et al. '15], parallel algebraic code with randomized sampling:
  1. After partial factorization, compute $Y = CB \cdot R$ with R random tall-skinny matrix. Y is a sample of the Schur Complement.
  2. At parent node, compute a sample of the frontal matrix F as:
     $$F \cdot R = A \cdot R \updownarrow Y_1 \updownarrow Y_2 \ldots$$
     Extend-add with "extension" only along rows.



Extend-add and low-rank representations

How to do extend-add with compressed matrices?

- BLR: contribution blocks not compressed [Amestoy et al. '12].
- HSS, [Jia et al. '09], serial 2D code. HSS extend-add. Slow and hard to parallelize. Hard to extend to algebraic.
- HSS, [Wang et al. '15], parallel geometric code. Contribution blocks not compressed. Simple but penalty on memory footprint.
- HSS, [Ghysels et al. '15], parallel algebraic code with randomized sampling:
  1. After partial factorization, compute $Y = CB \cdot R$ with R random tall-skinny matrix. Y is a sample of the Schur Complement.
  2. At parent node, compute a sample of the frontal matrix F as:
     $$F \cdot R = A \cdot R \updownarrow Y_1 \updownarrow Y_2 \dots$$
     Extend-add with "extension" only along rows.



Software packages – 1/2

Code	License	Authors	Format	Arch	Matrix
HLIBPro 2.4	Commercial (free academia)	Kriemann et al.	$\mathcal{H}$, $\mathcal{H}^2$	Shared (TBB), Dist. (MPI)	Dense, Sparse
HODLR 3.14	None	Ambikasaran, Darve	HODLR	Serial	Dense
MUMPS 5.X dev	Cecill-C $\simeq$ GPL	Amestoy, L'Excellent, et al.	BLR	Dist. (MPI), Shared (OpenMP)	Sparse (dense)
STRUMPACK -dense 1.1.1	BSD	R., Li , Ghysels	HSS	Dist. (MPI)	Dense
STRUMPACK -sparse 0.9.4	BSD	Ghysels, Li, R.	HSS	Shared (OpenMP)	Sparse

Other codes: H2lib [Boerm et al.], AHMED [Bebendorf & Rjasanov], BEM++ [Smigaj et al.], DMHM [Poulson & Li], H2tools [Mikhalev et al.]...

Software packages – 2/2

Code	Matrix	Clustering	Compress	Factor	Solve	Extract	Matvec
HLIBPro	Dense	✓(geo)	✓	✓	✓	✓	✓
	Sparse	✓(graph)	✓	✓	✓	✓	✓
HODLR	Dense		✓	✓	✓	✓	✓
MUMPS	Sparse	✓(graph)		✓	✓		
	Dense			✓	✓		
STRUMPACK	Dense		✓	✓	✓	✓	✓
	Sparse	✓(graph)		✓	✓		

Software packages – 2/2

Code	Matrix	Clustering	Compress	Factor	Solve	Extract	Matvec
HLIBPro	Dense	✓(geo)	✓	✓	✓	✓	✓
	Sparse	✓(graph)	✓	✓	✓	✓	✓
HODLR	Dense		✓	✓	✓	✓	✓
MUMPS	Sparse	✓(graph)		✓	✓		
	Dense			✓	✓		
STRUMPACK	Dense		✓	✓	✓	✓	✓
	Sparse	✓(graph)		✓	✓		

HLIBPro also has:

- $\mathcal{H}$-matrix addition and multiplication,
- BEM-specific features,
- Iterative solvers,
- Visualization. . .

HODLR: there is a new code by A. Aminfar with sparse features.

STRUMPACK: HSS algorithms based on randomized sampling [Martinsson]. Sparse MPI+OpenMP solver to be released soon (P. Ghysel's talk).

MUMPS: BLR features implemented in the dissertations of C. Weisbecker and T. Mary, to be released soon.

Algorithmic and implementation differences

- For dense matrices:
  - HODLR, HLIBPro and STRUMPACK 1/ compress the entire matrix then 2/ perform a structured factorization (e.g., ULV factorization for HSS).
  - MUMPS interleaves compressions and factorizations of panels.

Algorithmic and implementation differences

- For dense matrices:
  - HODLR, HLIBPro and STRUMPACK 1/ compress the entire matrix then 2/ perform a structured factorization (e.g., ULV factorization for HSS).
  - MUMPS interleaves compressions and factorizations of panels.
- Compression kernel:
  - MUMPS and STRUMPACK use QR with column pivoting.
  - HODLR and HLIBPro use ACA.

Algorithmic and implementation differences

- For dense matrices:
  - HODLR, HLIBPro and STRUMPACK 1/ compress the entire matrix then 2/ perform a structured factorization (e.g., ULV factorization for HSS).
  - MUMPS interleaves compressions and factorizations of panels.
- Compression kernel:
  - MUMPS and STRUMPACK use QR with column pivoting.
  - HODLR and HLIBPro use ACA.
- Compression threshold:
  - HLIBPro, HODLR and STRUMPACK use a relative threshold.
  - MUMPS uses an absolute threshold on singular values.

Algorithmic and implementation differences

- For dense matrices:
  - HODLR, HLIBPro and STRUMPACK 1/ compress the entire matrix then 2/ perform a structured factorization (e.g., ULV factorization for HSS).
  - MUMPS interleaves compressions and factorizations of panels.
- Compression kernel:
  - MUMPS and STRUMPACK use QR with column pivoting.
  - HODLR and HLIBPro use ACA.
- Compression threshold:
  - HLIBPro, HODLR and STRUMPACK use a relative threshold.
  - MUMPS uses an absolute threshold on singular values.
- Interface:
  - HLIBPro and HODLR require only a function that defines $A_{i,j}$.
  - MUMPS requires an explicit matrix A.
  - STRUMPACK can take either an explicit matrix, either an element function and samples of the row and column spaces of the matrix: $S_r = A \cdot R_r$, $S_c = A^T \cdot R_c$.

Comparison for dense problems (SIAM LA15) – 1/5

Settings:

- 10 dense matrices from various applications.
- $N = 10,000 - 20,000$.
- Benchmark: preconditioned GMRES.
- HODLR vs HLIBPro vs MUMPS-BLR vs STRUMPACK.

Comparison for dense problems (SIAM LA15) – 2/5

Quantum Chemistry Toeplitz matrix, $n = 12,500$.

Solver	Time(s)	Mem(MB)	#Iters
LAPACK	63.5	1192.1	1
HODLR 10^{-14}	0.7	42.5	2
HODLR 10^{-08}	0.5	27.5	3
HODLR 10^{-02}	60.0	10.7	600
HLIBPro 10^{-14}	3.6	42.8	2
HLIBPro 10^{-08}	2.2	30.3	2
HLIBPro 10^{-02}	1.4	16.3	6
MUMPS-BLR 10^{-14}	8.3	64.3	2
MUMPS-BLR 10^{-08}	9.0	58.4	3
MUMPS-BLR 10^{-02}	12.2	53.6	17
STRUMPACK 10^{-14}	0.7	40.7	1
STRUMPACK 10^{-08}	0.5	22.7	3
STRUMPACK 10^{-02}	1.0	7.9	65

Very structured problem, hierarchical formats outperform BLR.

Nested basis structure gives the edge to HSS.

Comparison for dense problems (SIAM LA15) – 3/5

Covariance matrix, $n = 10,648$ ($22 \times 22 \times 22$ mesh).

Solver	Time(s)	Mem(MB)	#Iters
LAPACK	41.0	865.0	1
HODLR 10^{-14}	448.2	2250.1	2
HODLR 10^{-08}	89.6	935.7	9
HODLR 10^{-02}	NoCV	10.9	NoCV
HLIBPro 10^{-14}	247.8	764.7	1
HLIBPro 10^{-08}	191.5	577.5	3
HLIBPro 10^{-02}	NoCV	30.8	NoCV
MUMPS-BLR 10^{-14}	48.9	865.0	2
MUMPS-BLR 10^{-08}	35.7	737.0	3
MUMPS-BLR 10^{-02}	49.6	203.3	130
STRUMPACK 10^{-14}	277.7	1651.9	2
STRUMPACK 10^{-08}	97.8	945.1	6
STRUMPACK 10^{-02}	111.1	648.8	436

No compression with hierarchical formats except with large ε.

Some limited gains with BLR.

Comparison for dense problems (SIAM LA15) – 4/5

BEM Acoustic Sphere, $n = 10,002$.

Solver	Time(s)	Mem(MB)	#Iters
LAPACK	53.0	921.8	1
HODLR 10^{-14}	1.5	23.1	4
HODLR 10^{-08}	0.8	9.5	5
HODLR 10^{-02}	1.0	7.2	8
HLIBPro 10^{-14}	1.5	13.7	7
HLIBPro 10^{-08}	1.4	11.4	7
HLIBPro 10^{-02}	1.2	9.3	7
MUMPS-BLR 10^{-14}	11.1	48.3	1
MUMPS-BLR 10^{-08}	9.1	40.6	2
MUMPS-BLR 10^{-02}	9.8	38.5	5
STRUMPACK 10^{-14}	245.8	501.8	1
STRUMPACK 10^{-08}	8.8	22.7	2
STRUMPACK 10^{-02}	1.9	10.9	5

HODLR, HLIBPro, ahead.

No clear nested basis structure in this problem.

Comparison for dense problems (SIAM LA15) – 5/5

For our test suite (10 problems):

- Problems with very low-ranks (Toeplitz, 2D Laplacian): HLIBPro/HODLR/STRUMPACK dominate.
- Problems with large ranks (in A_{12}, A_{21}) (Covariance, 3D Laplacian): MUMPS-BLR faster.
- Some problems: no clear result, depends on threshold.
- MUMPS-BLR limits the worst-case: no huge increase in run time or memory for 10^{-14}. It rejects blocks with large ranks. This is possible with HSS etc. but these codes don't do it.

Sparse problems – Regular grids

We compared HSS (STRUMPACK) and BLR (MUMPS) for 2D Poisson and 3D Helmholtz.

- 2D Poisson: HSS slightly lower asymptotic complexity but higher prefactor.
- 3D Helmholtz: asymptotic behaviors very similar.

Cf. next talk (Theo Mary) for experimental results with BLR and complexity study.

Sparse problems – General problems

We experimented with 10 medium-sized matrices from UFL.
(A22, Geo_1438, tdr190k, atmosmodd, nlpkkt80, Serena, torso3, Cube_Coup_dt0, spe10-aniso, Transport)

Lessons learnt:

- HSS can't be used as a direct solver ($\varepsilon \simeq 10^{-16}$). Aggressive settings needed. Cf. P. Ghysels' talk for HSS vs ILU and other preconditioners.
- Gains with BLR as a function of ε more consistent.
- BLR has a wider range of applications. HSS more restricted (especially for sparse problems, not as much for dense).
- Parallelism, efficiency: HSS more complicated communication pattern (tree traversal). BLR similar to a traditional factorization; better flop rate.

End

Thank you for your attention!

Any questions?