A comparison of different low-rank approximation techniques

François-Henry Rouet

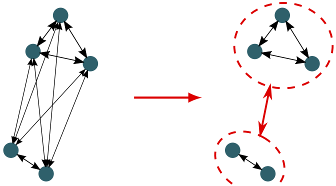
Lawrence Berkeley National Laboratory

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

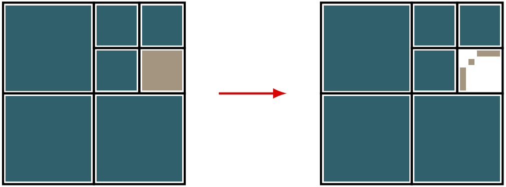
SIAM Conference on Applied Linear Algebra, October 29th, 2015

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 compare four different rank-structured software packages for dense problems (four different classes of matrices).

Differences

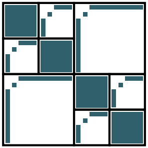
Three criteria differentiate all the low-rank formats:

Differences

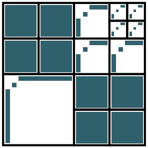
Three criteria differentiate all the low-rank formats:

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

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



vs



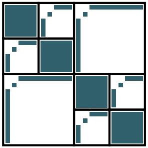
The partitioning is defined by the product of two trees (rows $\times$ columns).

Differences

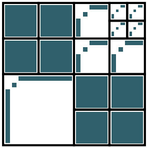
Three criteria differentiate all the low-rank formats:

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

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



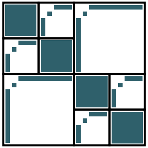
vs



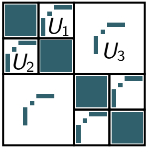
The partitioning is defined by the product of two trees (rows $\times$ columns).

- Nested basis or not.

Blocks have independent compressed representations (bases).



vs



Shared information:

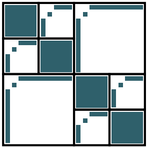
$$U_3^{\text{big}} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} U_3$$

Differences

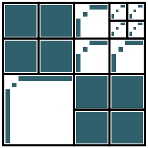
Three criteria differentiate all the low-rank formats:

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

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



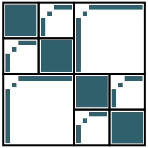
vs



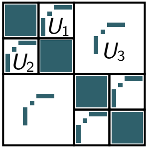
The partitioning is defined by the product of two trees (rows $\times$ columns).

- Nested basis or not.

Blocks have independent compressed representations (bases).



vs

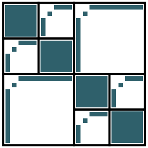


Shared information:

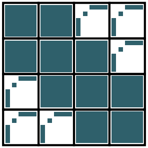
$$U_3^{\text{big}} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} U_3$$

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

Assumes interaction between two clusters is low-rank.



vs

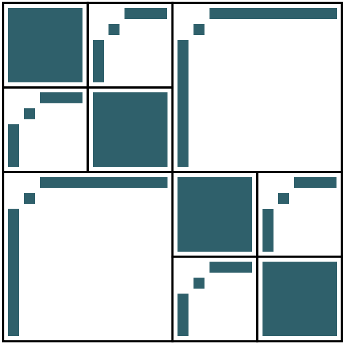


Blocks next to the diagonal not “admitted” (compressed).

Main classes of hierarchical matrices

HODLR (Darve et al.)

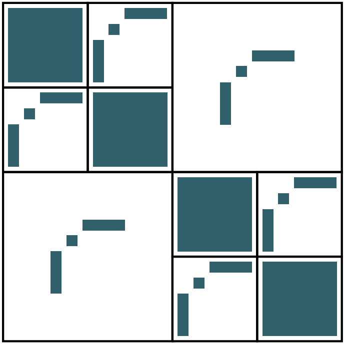
- No nested bases.
- No off-diagonal refinement.
- No buffer zone.



Main classes of hierarchical matrices

HSS (Chandrasekaran, Jia. . .)

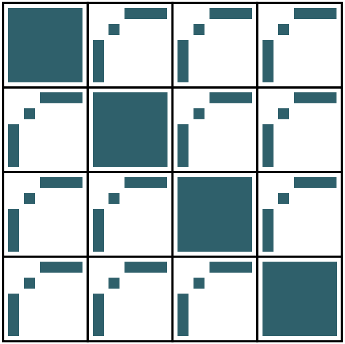
- Nested bases.
- No off-diagonal refinement.
- No buffer zone.



Main classes of hierarchical matrices

BLR (Amestoy, Ashcraft, et al.)

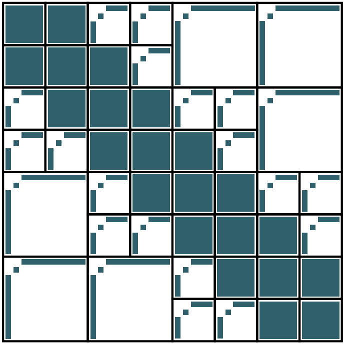
- No nested bases.
- Refine off-diagonal blocks.
- Can do buffer zone.



Main classes of hierarchical matrices

Barnes-Hut ("tree code")

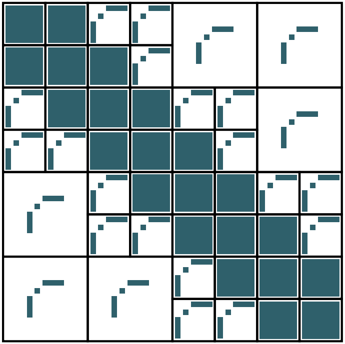
- No nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



Main classes of hierarchical matrices

Fast Multipole Method (Greengard & Rokhlin)

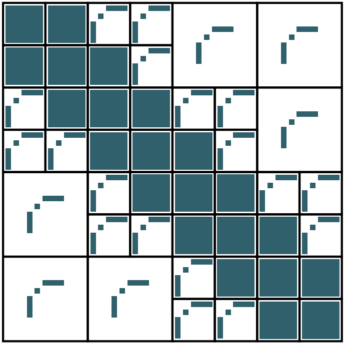
- Nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



Main classes of hierarchical matrices

Fast Multipole Method (Greengard & Rokhlin)

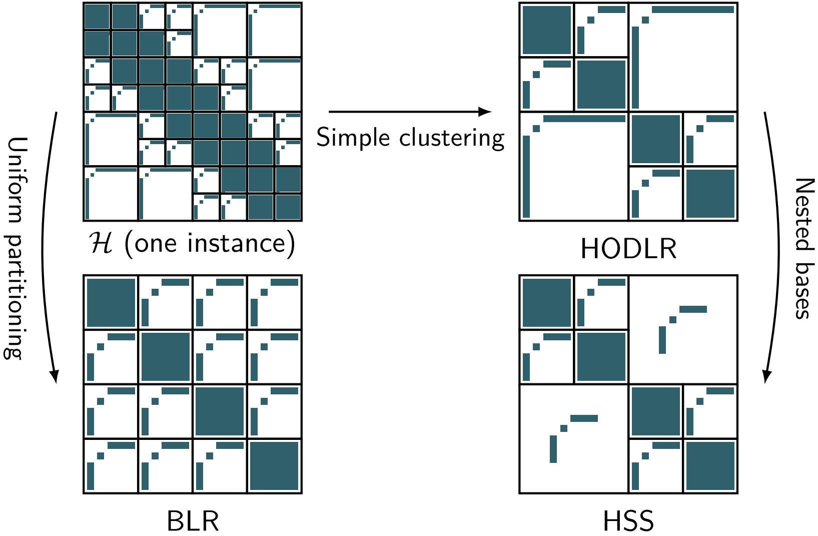
- Nested bases.
- Refine off-diagonal blocks.
- Buffer zone.



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

The four formats

In this talk, we examine:



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:

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

- 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:

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

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

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

*2.4 released yesterday!

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

- Workflow (dense case):
  - 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

- Workflow (dense case):
  - 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

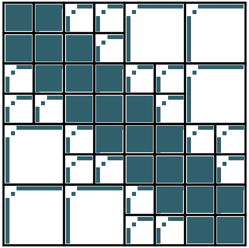
- Workflow (dense case):
  - HODLR, HLIBPro and STRUMPACK 1/ compress the entire matrix then 2/ perform a structured factorization (e.g., ULV factorization for HSS).
  - MUMPS interleaves compressions and factorizations of panels.
- Compression kernel:
  - MUMPS and STRUMPACK use QR with column pivoting.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

- Workflow (dense case):
  - 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: $S_r = A \cdot R_r$, $S_c = A^T \cdot R_c$.

The comparison

- Test: solving a linear system with GMRES (PETSc), preconditioned by HODLR / HLIBPro / MUMPS-BLR / STRUMPACK. **Sequential execution**.
- Three different compression thresholds: 10^{-14}, 10^{-8}, 10^{-2}.
- Block sizes, leaf sizes, tree levels: the best for each code.
- All the matrices are built/permuted in a way that reveals low-rankness and most problems don't have an underlying geometry. In HLIBPro, geometric clustering is disabled and the default partitioning/admissibility condition is used:



Test problems – 1/2

All problems are dense.

- Quantum Chemistry Toeplitz matrix (from J. Jones, D. Haxton, LBNL). Ranks grow slowly with matrix size.
- "Simple" Toeplitz matrix. $A_{i,j} = i - j$ for $i \neq j$. For any partitioning/decomposition, ranks should be 2.
- Covariance matrix (from U. Villa, LLNL). Associated with a 3D mesh, used to generate "random Gaussian fields".
- Root node of the multifrontal factorization of a 2D Laplacian problem (5-point FD). Max off-diagonal rank is expected to be very small, almost constant with problem size.
- Root node of the multifrontal factorization of a 3D Laplacian problem (7-point FD). Max off-diagonal rank grows as $\sqrt{n}$ (and is big w.r.t problem size).

Test problems – 2/2

- BEM Acoustic Sphere (from G. Sylvand, Airbus). Frequency 510 MHz, radius 1 meter, discretization step wavelength/10.
- Artificial HODLR matrix. Each block has rank 3% of its size.
- Two-electron integrals matrix (from J. McClean, LBNL). Corresponds to a CxHy molecule (e.g., C8H18), generated from a rank-4 tensor.
- FMM matrix (from Rio Yokota, Tokyo), Laplacian kernel for a 3D problem, random particules in a cube.
- Matrix associated with the pendigits dataset (from M. Mahoney, UC Berkeley). Gaussian kernel of a dataset of handwriting samples.

Results – 1/10

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

Solver	Times (s)			Mem (MB)		Iter	Rank
	Compr.	Facto.	Total	Compr.	Facto.		
LAPACK	-	63.0	63.5	-	1192.1	1	-
HODLR 10^{-14}	0.4	0.3	0.7	40.9	42.5	2	30
HODLR 10^{-08}	0.2	0.1	0.5	27.0	27.5	3	18
HODLR 10^{-02}	0.2	0.1	60.0	10.6	10.7	600	1
HLIBPro 10^{-14}	0.4	3.0	3.6	43.0	42.8	2	-
HLIBPro 10^{-08}	0.3	1.2	2.2	31.0	30.3	2	-
HLIBPro 10^{-02}	0.1	0.6	1.4	16.3	16.3	6	-
MUMPS-BLR 10^{-14}	-	7.6	8.3	-	64.3	2	-
MUMPS-BLR 10^{-08}	-	7.5	9.0	-	58.4	3	-
MUMPS-BLR 10^{-02}	-	4.9	12.2	-	53.6	17	-
STRUMPACK 10^{-14}	0.3	0.09	0.7	14.3	40.7	1	84
STRUMPACK 10^{-08}	0.2	0.04	0.5	8.9	22.7	3	65
STRUMPACK 10^{-02}	0.1	0.02	1.0	3.4	7.9	65	10

Simple Toeplitz matrix, $n = 12,500$.

Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	63.0	63.5	-	1192.1	1	-
HODLR 10^{-14}	0.1	0.04	0.2	13.4	13.4	1	4
HODLR 10^{-08}	0.1	0.03	0.2	12.0	12.0	1	2
HODLR 10^{-02}	0.01	0.03	0.6	10.6	10.7	5	1
HLIBPro 10^{-14}	0.07	0.6	0.8	16.3	16.3	1	-
HLIBPro 10^{-08}	0.07	0.5	0.7	16.3	16.3	1	-
HLIBPro 10^{-02}	0.07	0.4	0.8	15.0	13.9	3	-
MUMPS-BLR 10^{-14}	-	8.2	9.3	-	48.9	1	-
MUMPS-BLR 10^{-08}	-	7.5	8.2	-	48.9	1	-
MUMPS-BLR 10^{-02}	-	5.0	6.9	-	35.8	4	-
STRUMPACK 10^{-14}	0.02	0.02	0.05	2.9	7.3	1	2
STRUMPACK 10^{-08}	0.02	0.02	0.05	2.9	7.3	1	2
STRUMPACK 10^{-02}	0.02	0.02	0.1	2.7	7.2	6	2

Results – 3/10

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

Solver	Times (s)			Mem (MB)		Iter	Rank
	Compr.	Facto.	Total	Compr.	Facto.		
LAPACK	-	41.0	41.0	-	865.0	1	-
HODLR 10^{-14}	157.9	288.9	448.2	952.1	2250.1	2	1750
HODLR 10^{-08}	35.0	52.1	89.6	493.8	935.7	9	739
HODLR 10^{-02}	0.01	0.05	NoCV	10.7	10.9	NoCV	12
HLIBPro 10^{-14}	174.4	73.0	247.8	765.0	764.7	1	-
HLIBPro 10^{-08}	95.3	95.6	191.5	567.6	577.5	3	-
HLIBPro 10^{-02}	0.8	2.8	NoCV	46.3	30.8	NoCV	-
MUMPS-BLR 10^{-14}	-	48.0	48.9	-	865.0	2	-
MUMPS-BLR 10^{-08}	-	34.4	35.7	-	737.0	3	-
MUMPS-BLR 10^{-02}	-	5.0	49.6	-	203.3	130	-
STRUMPACK 10^{-14}	213.7	62.7	277.7	614.3	1651.9	2	2661
STRUMPACK 10^{-08}	71.3	24.5	97.8	423.8	945.1	6	1486
STRUMPACK 10^{-02}	1.0	13.7	111.1	216.5	648.8	436	2

2D Laplacian Schur complement, $n = 12,500$ ($12,500 \times 12,500$ mesh).

Solver	Times (s)			Mem (MB)		Iter	Rank
	Compr.	Facto.	Total	Compr.	Facto.		
LAPACK	-	63.0	63.5	-	1192.1	1	-
HODLR 10^{-14}	0.3	0.08	1.2	16.7	16.8	7	8
HODLR 10^{-08}	0.2	0.06	1.2	14.4	14.5	8	6
HODLR 10^{-02}	0.2	0.04	1.4	11.6	11.6	11	4
HLIBPro 10^{-14}	0.07	0.2	1.0	10.9	11.1	7	-
HLIBPro 10^{-08}	0.06	0.1	1.0	10.3	10.5	7	-
HLIBPro 10^{-02}	0.05	0.1	1.0	9.9	10.1	7	-
MUMPS-BLR 10^{-14}	-	8.4	9.3	-	38.1	1	-
MUMPS-BLR 10^{-08}	-	8.9	10.2	-	38.4	2	-
MUMPS-BLR 10^{-02}	-	8.3	10.5	-	38.1	5	-
STRUMPACK 10^{-14}	1.0	0.1	1.3	12.0	29.8	1	18
STRUMPACK 10^{-08}	1.0	0.1	1.3	8.2	28.9	1	13
STRUMPACK 10^{-02}	0.9	0.1	1.6	10.4	28.3	5	8

3D Laplacian Schur complement, $n = 12,100$ ($110 \times 110 \times 110$ mesh).

Solver	Times (s)			Mem (MB)		Iter	Rank
	Compr.	Facto.	Total	Compr.	Facto.		
LAPACK	-	-	55.6	-	1117.0	1	-
HODLR 10^{-14}	42.7	76.0	119.5	605.0	1222.1	2	783
HODLR 10^{-08}	15.6	25.1	43.6	368.7	689.7	12	510
HODLR 10^{-02}	0.9	0.9	13.8	79.2	86.4	101	112
HLIBPro 10^{-14}	46.3	55.7	102.4	554.0	555.7	2	-
HLIBPro 10^{-08}	21.4	40.9	64.2	389.5	392.4	13	-
HLIBPro 10^{-02}	2.6	10.7	23.4	96.4	104.5	83	-
MUMPS-BLR 10^{-14}	-	29.3	30.4	-	665.7	2	-
MUMPS-BLR 10^{-08}	-	16.2	17.7	-	401.0	3	-
MUMPS-BLR 10^{-02}	-	6.1	14.3	-	150.8	20	-
STRUMPACK 10^{-14}	85.0	26.9	112.9	424.7	1106.0	2	1302
STRUMPACK 10^{-08}	49.8	11.9	64.7	309.3	714.9	3	906
STRUMPACK 10^{-02}	19.5	6.8	30.4	209.5	480.1	19	468

Results – 6/10

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

Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	35.0	35.0	-	763.2	1	-
HODLR 10^{-14}	31.5	39.4	71.4	400.0	580.4	2	653
HODLR 10^{-08}	4.2	2.6	7.6	117.7	150.7	7	185
HODLR 10^{-02}	0.1	0.04	0.8	5.8	5.8	9	0
HLIBPro 10^{-14}	111.8	38.7	151.6	544.5	544.6	1	-
HLIBPro 10^{-08}	90.1	21.6	111.9	429.2	429.3	1	-
HLIBPro 10^{-02}	2.3	7.6	10.6	80.0	80.3	8	-
MUMPS-BLR 10^{-14}	-	22.5	23.3	-	508.8	2	-
MUMPS-BLR 10^{-08}	-	8.5	9.6	-	238.9	3	-
MUMPS-BLR 10^{-02}	-	3.8	5.9	-	37.4	7	-
STRUMPACK 10^{-14}	338.0	92.2	432.2	695.4	2404.1	2	3614
STRUMPACK 10^{-08}	51.2	15.6	67.6	277.2	851.2	2	1182
STRUMPACK 10^{-02}	10.0	2.1	14.7	106.3	251.4	6	384

HODLR artificial matrix, $n = 12.500$.

Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	63.0	63.5	-	1192.1	1	-
HODLR 10^{-14}	1.3	2.1	4.4	107.8	109.9	2	188
HODLR 10^{-08}	1.9	2.0	4.2	106.8	108.9	1	187
HODLR 10^{-02}	0.05	1.2	1.6	38.2	38.2	3	1
HLIBPro 10^{-14}	6.1	99.7	106.0	243.8	461.3	1	-
HLIBPro 10^{-08}	6.2	43.0	49.5	237.2	259.3	1	-
HLIBPro 10^{-02}	1.5	0.9	2.9	71.5	71.5	3	-
MUMPS-BLR 10^{-14}	-	62.6	63.3	-	1105.1	1	-
MUMPS-BLR 10^{-08}	-	56.0	57.1	-	939.4	2	-
MUMPS-BLR 10^{-02}	-	5.0	6.1	-	35.8	2	-
STRUMPACK 10^{-14}	19.9	5.4	26.0	182.9	545.7	1	400
STRUMPACK 10^{-08}	16.4	3.8	20.6	153.9	445.6	1	360
STRUMPACK 10^{-02}	1.0	0.5	1.9	37.5	111.9	3	1

Two-electron integrals, C8H18 molecule, $n = 11,664$.

Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	48.7	49.8	-	1038.0	1	-
HODLR 10^{-14}	133.4	123.1	257.5	776.4	1082.7	2	1854
HODLR 10^{-08}	18.1	14.1	32.9	330.8	393.5	3	705
HODLR 10^{-02}	0.1	0.05	6.0	14.7	14.8	58	12
HLIBPro 10^{-14}	100.9	222.3	330.2	764.5	804.7	42	-
HLIBPro 10^{-08}	21.5	95.6	123.0	397.7	409.7	42	-
HLIBPro 10^{-02}	0.3	2.1	7.1	28.7	31.6	43	-
MUMPS-BLR 10^{-14}	-	58.7	59.7	-	1006.8	2	-
MUMPS-BLR 10^{-08}	-	33.7	35.1	-	685.1	3	-
MUMPS-BLR 10^{-02}	-	4.7	29.2	-	55.0	64	-
STRUMPACK 10^{-14}	158.4	33.9	193.9	311.2	1039.8	2	1700
STRUMPACK 10^{-08}	21.8	2.1	23.9	83.6	257.9	3	570
STRUMPACK 10^{-02}	2.1	0.1	19.5	8.4	24.5	179	16

3D FMM matrix (Laplacian kernel), $n = 12,000$.

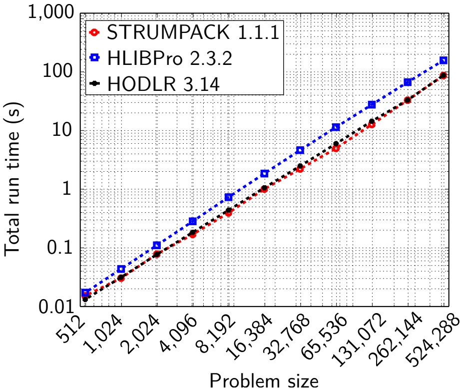
Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	53.0	53.5	-	1098.6	2	-
HODLR 10^{-14}	87.3	101.0	190.1	758.7	1068.1	2	1608
HODLR 10^{-08}	15.1	15.6	33.1	346.6	425.9	12	641
HODLR 10^{-02}	0.1	0.05	6.0	14.7	14.8	58	5
HLIBPro 10^{-14}	64.2	217.6	282.1	730.3	768.6	2	-
HLIBPro 10^{-08}	26.6	104.8	131.9	428.3	467.5	12	-
HLIBPro 10^{-02}	0.9	4.0	NoCV	46.5	43.3	NoCV	-
MUMPS-BLR 10^{-14}	-	62.9	63.9	-	1077.8	2	-
MUMPS-BLR 10^{-08}	-	32.7	34.4	-	708.6	4	-
MUMPS-BLR 10^{-02}	-	8.9	NoCV	-	183.5	NoCV	-
STRUMPACK 10^{-14}	126.4	16.9	144.1	527.4	1590.8	5	1775
STRUMPACK 10^{-08}	52.3	11.2	64.7	269.6	257.9	5	570
STRUMPACK 10^{-02}	1.0	1.5	NoCV	8.4	24.5	NoCV	2

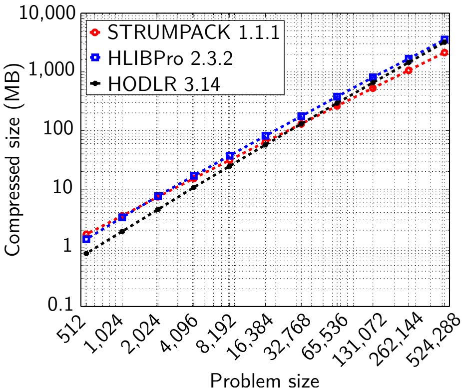
Pendigits Gaussian kernel, $n = 10,992$.

Solver	Times (s) Compr.	Facto.	Total	Mem (MB) Compr.	Facto.	Iter	Rank
LAPACK	-	52.6	53.0	-	921.8	1	-
HODLR 10^{-14}	0.8	0.3	1.5	22.1	23.1	4	51
HODLR 10^{-08}	0.2	0.1	0.8	9.5	9.5	5	22
HODLR 10^{-02}	0.1	0.1	1.0	7.2	7.2	8	3
HLIBPro 10^{-14}	0.09	0.5	1.5	12.2	13.7	7	-
HLIBPro 10^{-08}	0.08	0.4	1.4	10.4	11.4	7	-
HLIBPro 10^{-02}	0.06	0.2	1.2	8.6	9.3	7	-
MUMPS-BLR 10^{-14}	-	10.4	11.1	-	48.3	1	-
MUMPS-BLR 10^{-08}	-	8.1	9.1	-	40.6	2	-
MUMPS-BLR 10^{-02}	-	7.6	9.8	-	38.5	5	-
STRUMPACK 10^{-14}	215.3	30.1	245.8	133.7	501.8	1	2327
STRUMPACK 10^{-08}	8.5	0.08	8.8	9.6	22.7	2	113
STRUMPACK 10^{-02}	1.4	0.05	1.9	3.7	10.9	5	9

A matrix-free problem

Problem: Quantum Chemistry Toeplitz matrix, threshold 10^{-14}.
Benchmark: compression + factorization + GMRES iterations.





- Run times behave similarly, with a $\sim$ 2x slowdown for HLIBPro.
- Memory: HSS pays off for very large problems; $O(n)$ behavior.

Findings

For our test suite:

- 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.

Findings

For our test suite:

- 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.

Remarks:

- HODLR and HLIBPro perform similarly in this setting, but HLIBPro can take advantage of geometry.
- With STRUMPACK/HSS, the compressed matrix is the smallest in most cases, but there is a large increase after factorization.
- HSS + randomized sampling should perform better for sparse problems. Compression of an “independent” dense matrix: $O(rn^2)$, inside a sparse factorization: $O(r^2 n)$ (cheaper sampling).
- MUMPS-BLR limits the worst-case: no huge increase in run time or memory for 10^{-14}. It rejects blocks with large ranks.

Future work

- Parallel experiments, larger scale:
  - HLIBPro: block-wise $\mathcal{H}-$arithmetic doesn't give much parallelism for dense problems. Work in progress at the algorithmic level.
  - HODLR: need to try new code.
  - STRUMPACK-dense ready.
  - MUMPS-BLR ready.
- Sparse problems:
  - HLIBPro ready.
  - HODLR: need to try new code.
  - STRUMPACK-sparse: shared-memory ready, distributed-memory almost done.
  - MUMPS-BLR ready.

End

STRUMPACK-related talks at LA15

- Today, 3:30pm, Rio Yokota, *A comparison of FMM and HSS at scale*, MS45 (fast solvers).
- Tomorrow, 11:45am, Pieter Ghysels, *A parallel multifrontal solver using HSS matrices*, MS51 (preconditioners).

Thank you for your attention!

Any questions?