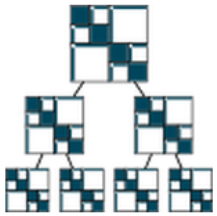
STRUMPACK: STRUctured Matrix PACKage

Accelerating sparse direct solvers with rank-structured matrices and randomization

X. Sherry Li, Pieter Ghysels, François-Henry Rouet
Lawrence Berkeley National Laboratory
Artem Napov
Université Libre de Bruxelles

CIMI HPC Fast Solvers, June 25, 2015



STRUMPACK
STRUctured Matrices PACKage

STRUMPACK - STRUctured Matrices PACKage - is a package for computations with sparse and dense structured matrices, i.e., matrices that exhibit some kind of low-rank property. STRUMPACK uses Hierarchical Semi-Separable representations (HSS) and Randomized Sampling techniques.

STRUMPACK has currently two main components:

- A shared-memory (OpenMP), sparse direct solver.
  The solver can be used as a direct solver or as a preconditioner for any sparse linear system.
  Version 0.9 was released on February 9th, 2015 under the BSD-LBNL license.
  The package (source code and documentation) can be found here: **strumpack-sparse 0.9**
- A distributed-memory (MPI), dense matrix computation package.
  It offers the following features:
  - Compression of a dense matrix into its HSS form.
  - Fast matrix-vector products using the HSS form.
  - Fast solution of linear systems using the HSS form.

  Version 0.9.0 was released on December 19th, 2014 under the BSD-LBNL license.
  The package (source code and documentation) can be found here: **STRUMPACK-Dense 0.9.0**

Publications:

- **F.-H. Rouet, Xiaoye S. Li, P. Ghysels.** *A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization.* Submitted to ACM Transactions on Mathematical Software, 2014.

- **P. Ghysels, X.S. Li, F.-H. Rouet, S. Williams, A. Napov** *An efficient multi-core implementation of a novel HSS-structured multifrontal solver using randomized sampling.* Submitted to SIAM SISC Special Issue CSE 2015, 2015. **preprint**

STRUMPACK overview

http://portal.nersc.gov/project/sparse/strumpack/

- C++, OpenMP, MPI
- Support both real & complex datatypes, single & double precision, and 64-bit indexing.
  - via C++ template
- Two components:
  - Dense – applicable to Toeplitz, Cauchy, BEM, integral equations, etc.
  - Sparse – aim at matrices discretized from PDEs.
- Input interfaces
  - Dense matrix in standard format.
  - Matrix-free – user provides matvec multiplication routine, and routine for selecting some matrix entries.
  - Sparse matrix in CSR format.
- Public domain, BSD license.

1. Dense package

- HSS: Hierarchically Semi-Separable matrix format
- Fast HSS matrix construction
  - Randomized sampling + Interpolative Decomposition (ID) (via RRQR)
- Fast factorization of HSS matrices
- Fast HSS-vector products
- Scalable distributed memory MPI code

- Extensible to include other data-sparse representations: $\mathcal{H}$, $\mathcal{H}^2$, HODLR, etc.

LR compression via randomized sampling

1. Pick random matrix $\Omega_{n\times(k+p)}$, p small, e.g. 10
2. Sample matrix $S = A\Omega$, with slight oversampling p
3. Compute $Q =$ ON-basis(S)

LR compression via randomized sampling

1. Pick random matrix $\Omega_{n\times(k+p)}$, p small, e.g. 10
2. Sample matrix $S = A\Omega$, with slight oversampling p
3. Compute $Q =$ ON-basis(S)

- **Accuracy:** with *probability* $\geq 1 - 6 \cdot p^{-p}$,
  $\|A - QQ^*A\| \leq [1 + 11\sqrt{k+p} \cdot \sqrt{\min\{m,n\}}]\sigma_{k+1}$
- **Cost:** $O(kmn)$

LR compression via randomized sampling

1. Pick random matrix $\Omega_{n\times(k+p)}$, p small, e.g. 10
2. Sample matrix $S = A\Omega$, with slight oversampling p
3. Compute $Q =$ ON-basis(S)

- **Accuracy:** with *probability* $\geq 1 - 6 \cdot p^{-p}$,
  $\|A - QQ^*A\| \leq [1 + 11\sqrt{k+p} \cdot \sqrt{\min\{m,n\}}]\sigma_{k+1}$
- **Cost:** $O(kmn)$

- **LR kernels:** RS vs. "direct" methods (RRQR, truncated SVD)
  - All have same asymptotic cost using explicit matrix.
  - RS can be faster when fast matvec available.
  - RS useful when only matvec available (matrix-free).
- **In sparse solver, costs are different ...**

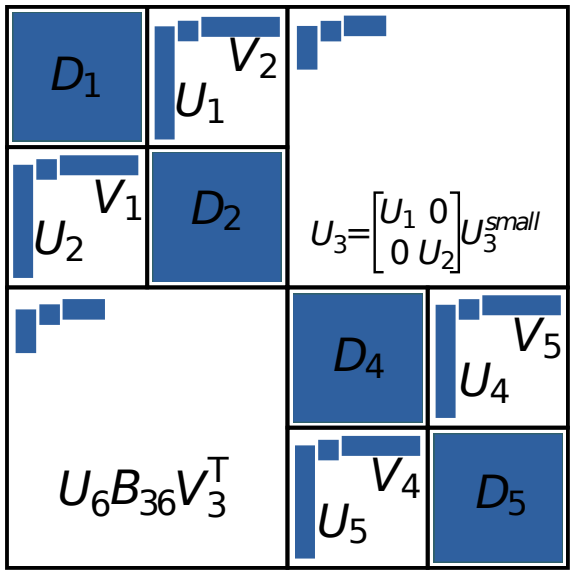
N. Halko,P.G. Martinsson, J.A. Trop, "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decomposition", SIAM Review, Vol.53, pp.217-288, 2011.

Hierarchically Semi-Separable matrices – HSS

- HSS belongs to family of $\mathcal{H}^2$-matrices.
- Dense, but data-sparse with low-rank off-diagonal blocks.
- Hierarchical partitioning; Nested bases.

Hierarchically Semi-Separable matrices – HSS

- HSS belongs to family of $\mathcal{H}^2$-matrices.
- Dense, but data-sparse with low-rank off-diagonal blocks.
- Hierarchical partitioning; Nested bases.



- Off-diagonal blocks are approximated as low-rank

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

- Diagonal blocks are full rank

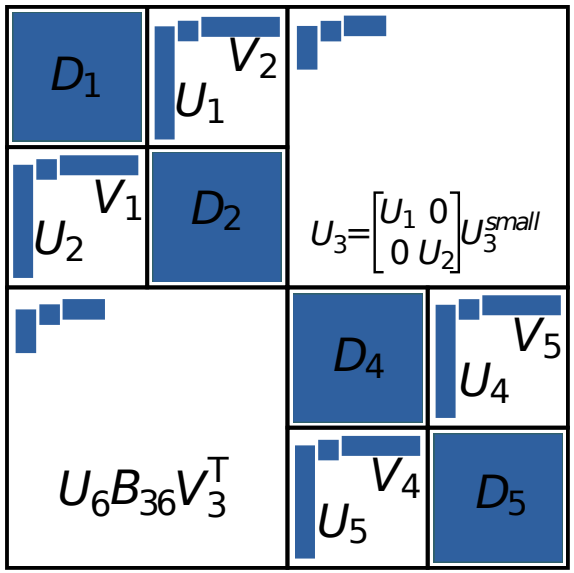
$$D_\tau = A(I_\tau, I_\tau)$$

- Column bases U and row bases V^* are nested

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

Hierarchically Semi-Separable matrices – HSS

- HSS belongs to family of $\mathcal{H}^2$-matrices.
- Dense, but data-sparse with low-rank off-diagonal blocks.
- Hierarchical partitioning; Nested bases.



- Off-diagonal blocks are approximated as low-rank

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

- Diagonal blocks are full rank

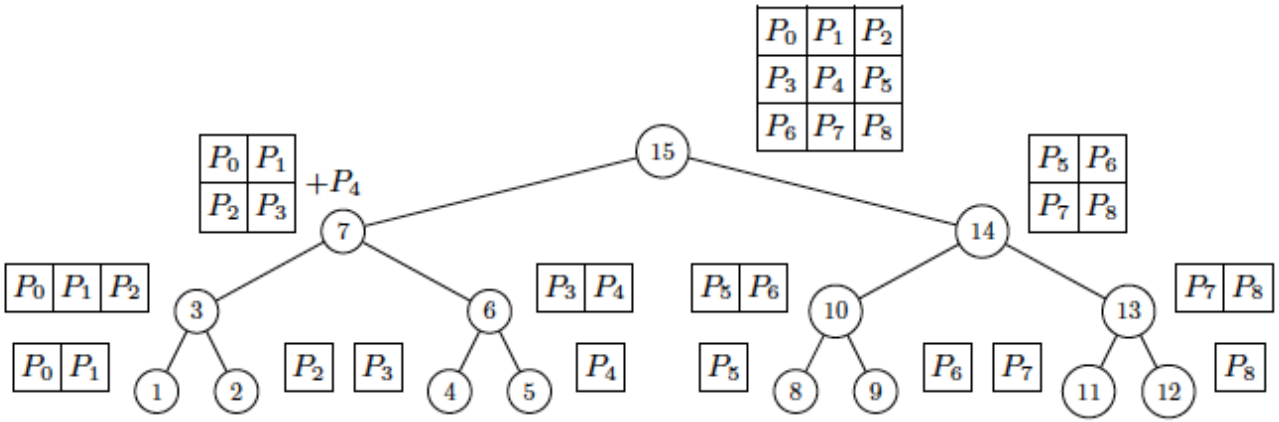
$$D_\tau = A(I_\tau, I_\tau)$$

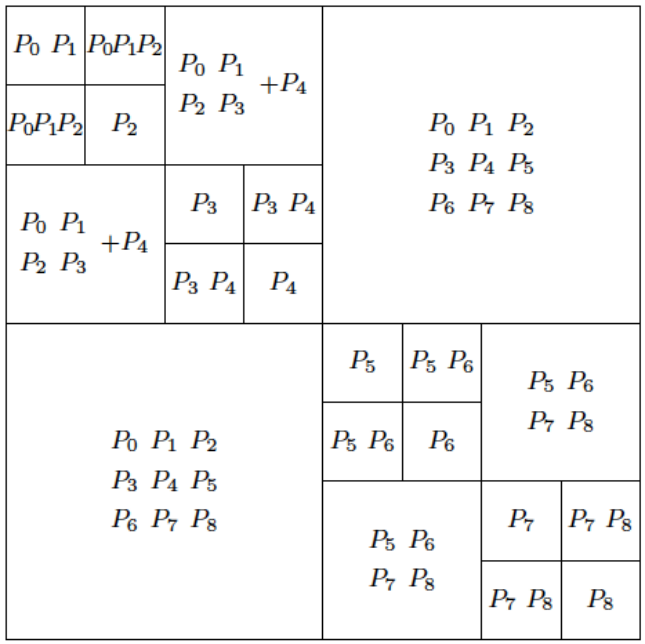
- Column bases U and row bases V^* are nested

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

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

Parallelization based on HSS tree



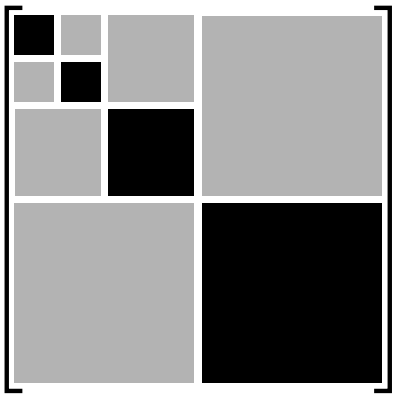


Additional features for software robustness

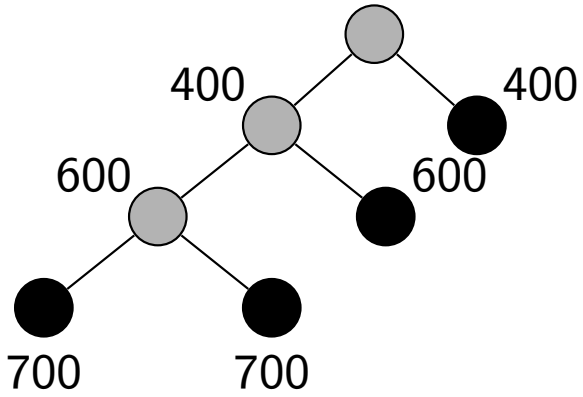
- Adaptive sampling machinery
  - Automatic handling unknown rank patterns: incrementally adjust sample size at any node when rank revealed is too large.
- Matrix-free interface: user provides matvec and element selection routines.
- Non-uniform clustering & partitioning

Additional features for software robustness

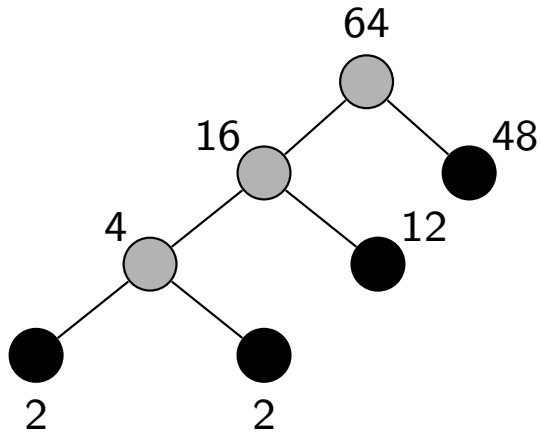
- Adaptive sampling machinery
  - Automatic handling unknown rank patterns: incrementally adjust sample size at any node when rank revealed is too large.
- Matrix-free interface: user provides matvec and element selection routines.
- Non-uniform clustering & partitioning



(d) Matrix structure.



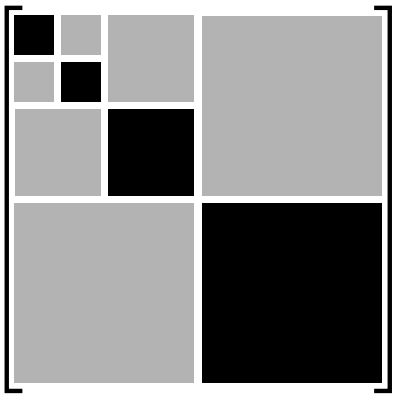
(e) Ranks



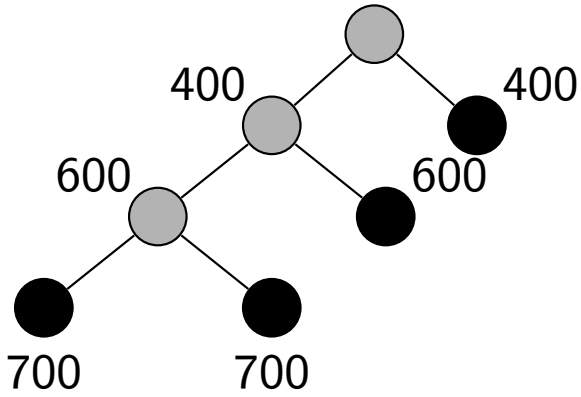
(f) Weighted process mapping

Additional features for software robustness

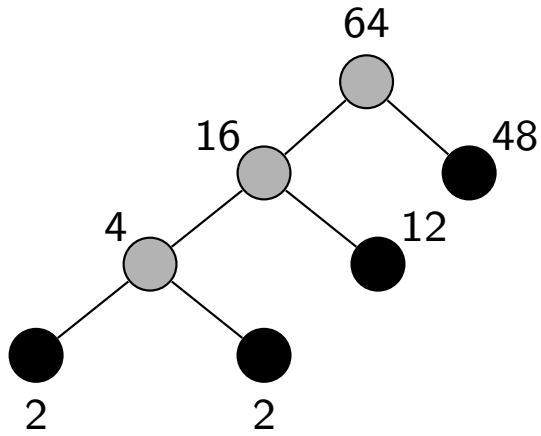
- Adaptive sampling machinery
  - Automatic handling unknown rank patterns: incrementally adjust sample size at any node when rank revealed is too large.
- Matrix-free interface: user provides matvec and element selection routines.
- Non-uniform clustering & partitioning



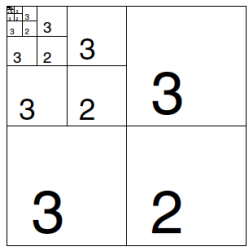
(g) Matrix structure.



(h) Ranks



(i) Weighted process mapping



Hilbert matrix $h_{ij} = \frac{1}{i+j-1}$

HSS compression complexities

We denote r the HSS rank, i.e., the maximum rank found during the different compression steps.
(Note: this is not the maximum rank of the off-diagonal blocks, because of the nested bases.)

- Without randomized sampling: $O(r\,n^2)$.
- With randomized sampling: $O(r^2 n)$ +cost of sampling:
  - Classical matvec: $O(r\,n^2)$.
  - FFT (e.g., Toeplitz matrix): $O(r\,n \log n)$.
  - FMM: $O(r\,n)$.

HSS compression complexities

We denote r the HSS rank, i.e., the maximum rank found during the different compression steps.
(Note: this is not the maximum rank of the off-diagonal blocks, because of the nested bases.)

- Without randomized sampling: $O(r\,n^2)$.
- With randomized sampling: $O(r^2 n)$ +cost of sampling:
  - Classical matvec: $O(r\,n^2)$.
  - FFT (e.g., Toeplitz matrix): $O(r\,n \log n)$.
  - FMM: $O(r\,n)$.

An issue of the randomized sampling approach is the need to access selected elements of the input matrix. Lin, Lu, Ying (2009) and Martinsson (2015) suggest a "peeling" algorithm that requires one matvec per level but no access to selected elements.

Communication analysis

Number of messages and volume of communication on the critical path (for 1 process):

Algorithm	Messages	Words
ScaLAPACK LU	$\mathcal{O}(n \log p)$	$\mathcal{O}\left(n^2 \frac{\log p}{\sqrt{p}}\right)$
Non-randomized HSS compression	$\mathcal{O}(p + r \log^2 p)$	$\mathcal{O}\left(\frac{n^2}{p} + rn + r^2 \log p\right)$
Randomized HSS compression	$\mathcal{O}(p \log p + r \log p + r \log^2 p)$	$\mathcal{O}\left(\frac{n^2}{p} + \frac{rn}{\sqrt{p}} + r^2\right)$
	Redist Sampling Tree	Redist Sampling Tree

- HSS vs LU: fewer messages but potentially more words when r large.
- Randomized vs non-randomized: slightly more messages but fewer words. Both can be reduced in a matrix-free implementation.

Experiments – highly structured matrices (64 MPI tasks)

STRUMPACK-Dense vs. ScaLAPACK

	Matrix	Size	ε	Rank	HSS vs LU Memory overhead	HSS vs LU Time speedup
$a_{i,i} = n^2$ $a_{i,j} = i - j$	SIMPLE TOEPLITZ	80,000	10^{-8}	2	0.1%	**75.2**
			10^{-6}	2	0.1%	68.1
			10^{-4}	3	0.1%	60.5
			10^{-2}	3	0.1%	51.0
$a_{i,i} = \frac{\pi^2}{6}$ $a_{i,j} = \frac{(-1)^{i-j}}{(i-j)^2 d^2}$	QCHEM TOEPLITZ	80,000	10^{-8}	169	1.5%	43.5
			10^{-6}	147	1.4%	**45.5**
			10^{-4}	120	1.0%	40.0
			10^{-2}	30	0.5%	N/A
	HMATRIX	80,000	10^{-8}	787	10.3%	15.3
			10^{-6}	785	10.3%	14.9
			10^{-4}	4	0.2%	**72.9**
			10^{-2}	2	0.2%	69.7

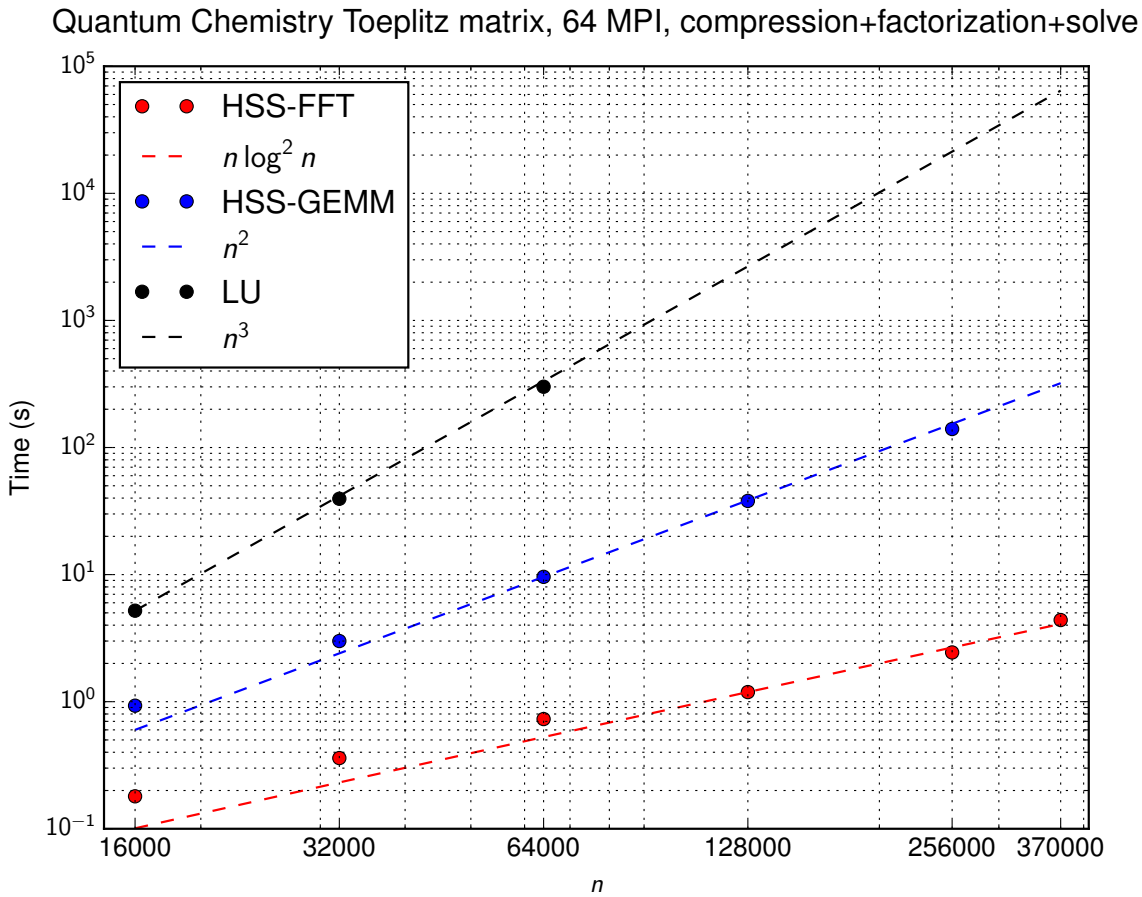
Memory for STRUMPACK-Dense

= explicit input matrix + samples + HSS factors + temp storage

= memory for ScaLAPACK + x% ("overhead")

Matrix-free interface

- QChem Toeplitz: matvec via FFT
- Previous best Toeplitz linear solver (e.g. Levinson): $O(n^2)$



Experiments – less structured matrices (64 MPI tasks)

Matrix	Size	ε	Rank	HSS vs LU Memory overhead	HSS vs LU Time speedup
Schur100	10,000 (single prec.)	10^{-8}	4933	722.6%	0.03
		10^{-6}	840	76.6%	0.5
		10^{-4}	501	40.5%	0.8
		10^{-2}	282	25.6%	**0.9**
BEM Multi Sphere	27,648 (single prec.)	10^{-8}	5995	403.2%	0.07
		10^{-6}	2145	53.5%	0.9
		10^{-4}	1488	38.4%	1.8
		10^{-2}	800	25.2%	**2.9**
BEM Acoustics	10,002	10^{-8}	1433	120.1%	0.3
		10^{-6}	1016	82.4%	0.5
		10^{-4}	793	66.6%	**0.7**
		10^{-2}	379	44.9%	–
Covar30	27,000	10^{-8}	2247	61.1%	0.9
		10^{-6}	1609	47.7%	**1.7**
		10^{-4}	215	14.3%	–
		10^{-2}	3	6.7%	–

Experiments – less structured matrices (64 MPI tasks)

Matrix	Size	ε	Rank	HSS vs LU Memory overhead	HSS vs LU Time speedup
Schur100	10,000 (single prec.)	10^{-8}	4933	722.6%	0.03
		10^{-6}	840	76.6%	0.5
		10^{-4}	501	40.5%	0.8
		10^{-2}	282	25.6%	**0.9**
BEM Multi Sphere	27,648 (single prec.)	10^{-8}	5995	403.2%	0.07
		10^{-6}	2145	53.5%	0.9
		10^{-4}	1488	38.4%	1.8
		10^{-2}	800	25.2%	**2.9**
BEM Acoustics	10,002	10^{-8}	1433	120.1%	0.3
		10^{-6}	1016	82.4%	0.5
		10^{-4}	793	66.6%	**0.7**
		10^{-2}	379	44.9%	–
Covar30	27,000	10^{-8}	2247	61.1%	0.9
		10^{-6}	1609	47.7%	**1.7**
		10^{-4}	215	14.3%	–
		10^{-2}	3	6.7%	–

- Compression threadhold should be set much larger than machine epsilon.
- Iterative refinement.

Experiments – weak scaling

Root node of the multifrontal factorization of a discretized Helmholtz problem (frequency domain, PML boundary, 10Hz).

k (mesh: $k \times k \times k$)	100	200	300	400	500
Matrix size ($=k^2$)	10,000	40,000	90,000	160,000	250,000
MPI tasks	64	256	1,024	4,096	8,192
Maximum rank	313	638	903	1289	1625
Speed-up over ScaLAPACK	**1.8**	**4.0**	**5.4**	**4.8**	**3.9**
Speed-up over Hsolver	**3.8**	**3.7**	**6.0**	**3.3**	**2.0**

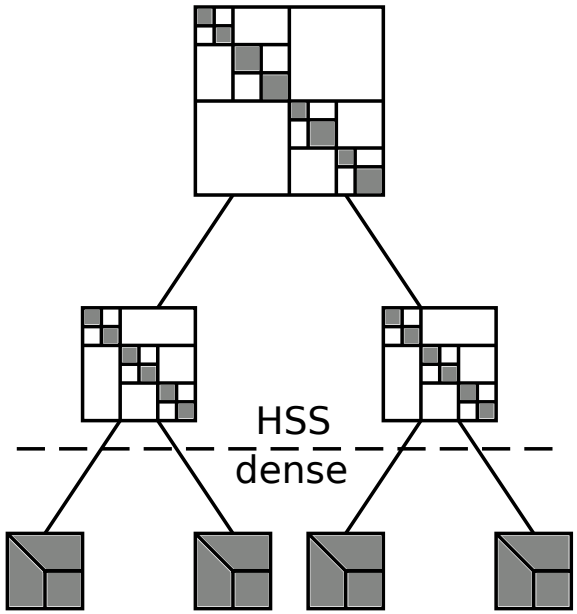
2. Sparse package

- General algebraic sparse linear solver/preconditioner
- Rank-structured sub-matrices: HSS
  - Reduce fill-in, memory usage
  - Speedup computations
- Efficient shared memory code with OpenMP task parallelism

Multifrontal HSS-enabled sparse solver/preconditioner

Approximate frontal matrices with HSS

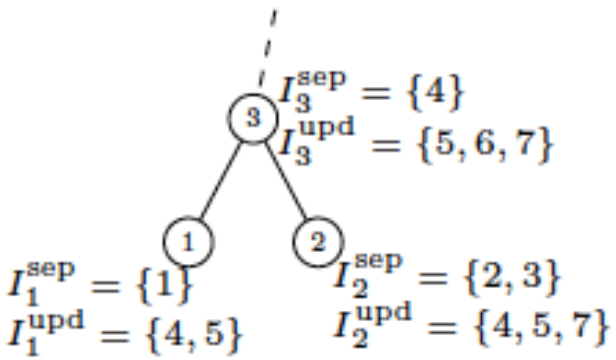
- Fully structured for top ℓ_s levels in the elimination tree, largest frontal matrices
- *ULV* factorization of HSS matrix
- Low-rank Schur complement update

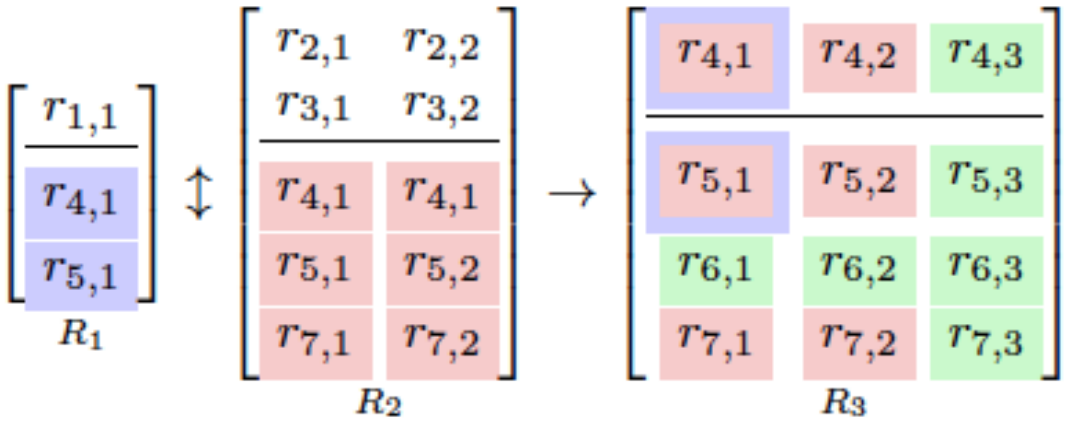


Use multifrontal solver as preconditioner

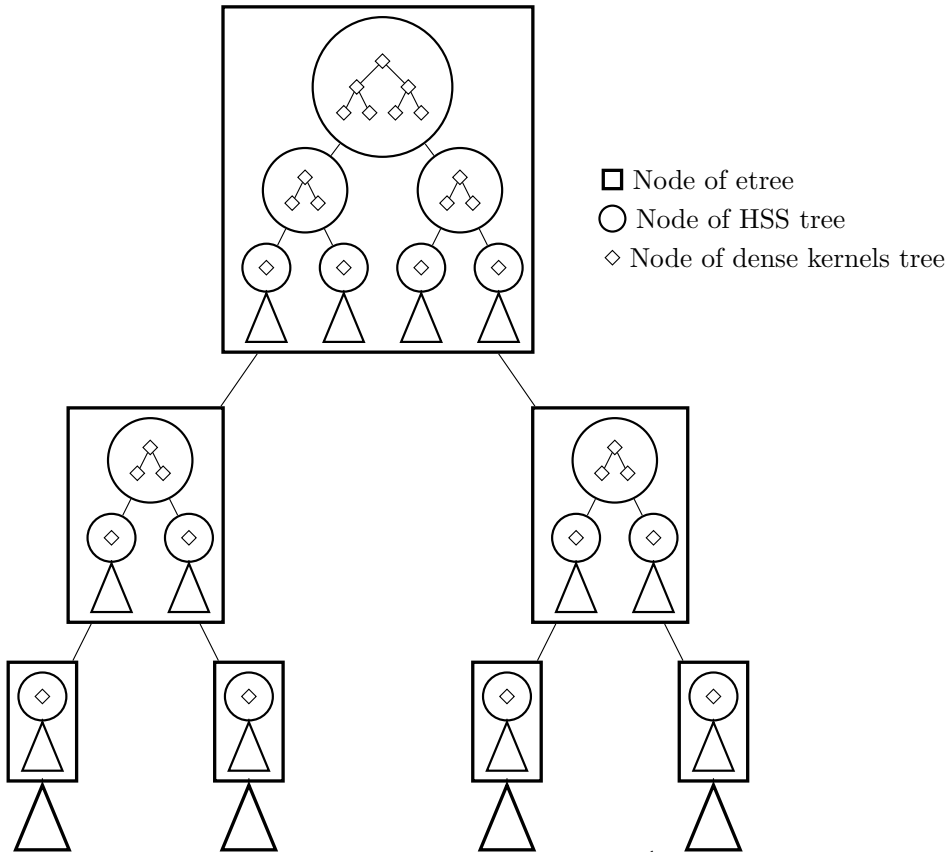
- GMRES outer solver
- Convergence affected by tolerance ε and ℓ_s

Simpler frontal extend-add





Hierarchical parallelism via OpenMP tasking



Solver complexity

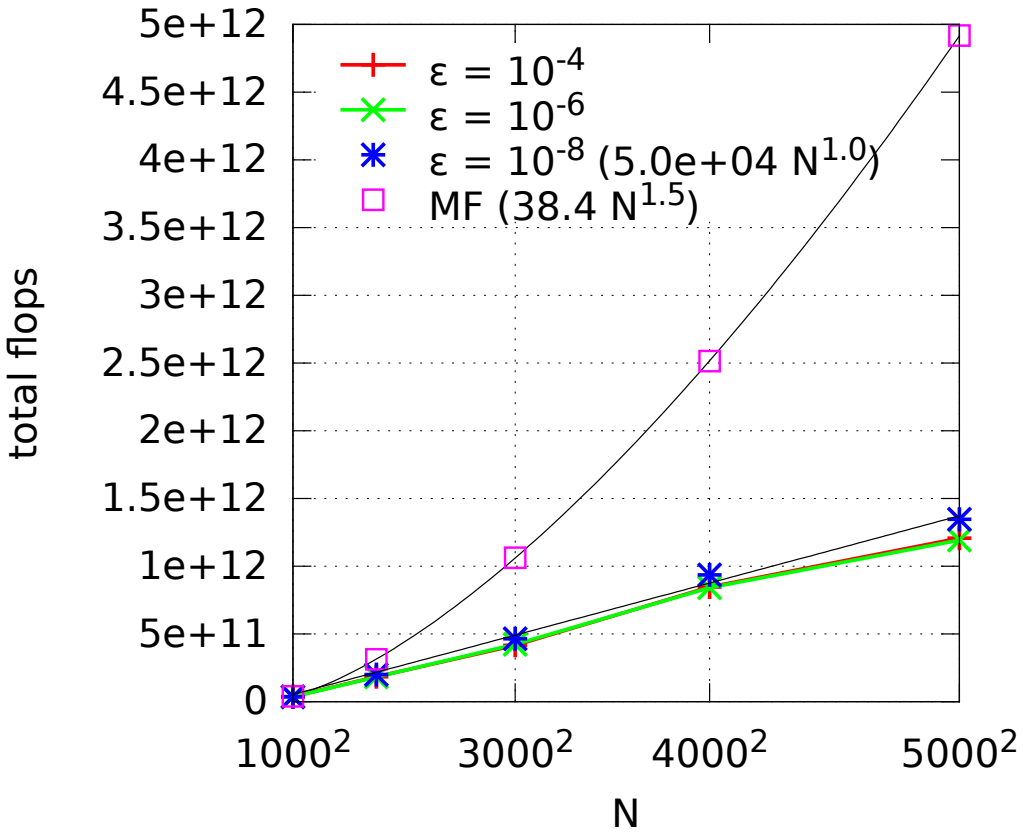
- Solver complexities on regular 2D and 3D meshes: $N = k^d$
- Complexity determined by largest separator
- Choose ℓ_s to balance work on dense and HSS frontal matrices

problem		HSS rank	MF		MF-HSS-RS	
			factor flops	memory	factor flops	memory
2D ($k \times k$)	Poisson	$O(1)$	$O(N^{3/2})$	$O(N \log N)$	$O(N)$	$O(N)$
	Helmholtz	$O(\log k)$				
3D ($k \times k \times k$)	Poisson	$O(k)$	$O(N^2)$	$O(N^{4/3})$	$O(N \log N)$	$O(N)$
	Helmholtz	$O(k)$				

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

S. Chandrasekaran et al. *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*. SIAM Journal on Matrix Analysis and Applications 31.5 (2010): 2261-2290.

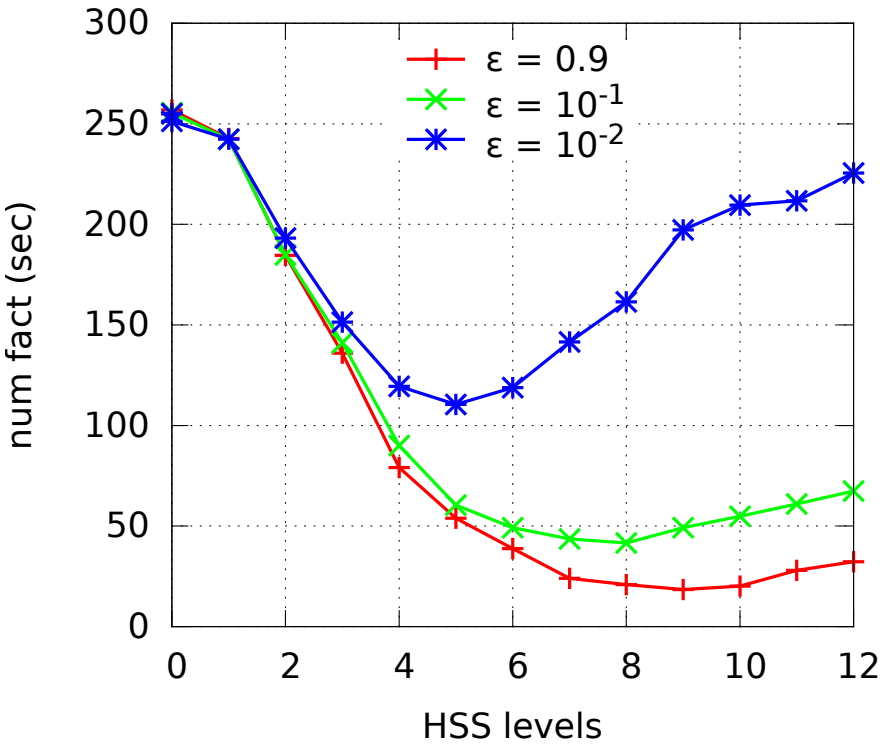
2D Poisson: Flop scaling with problem size

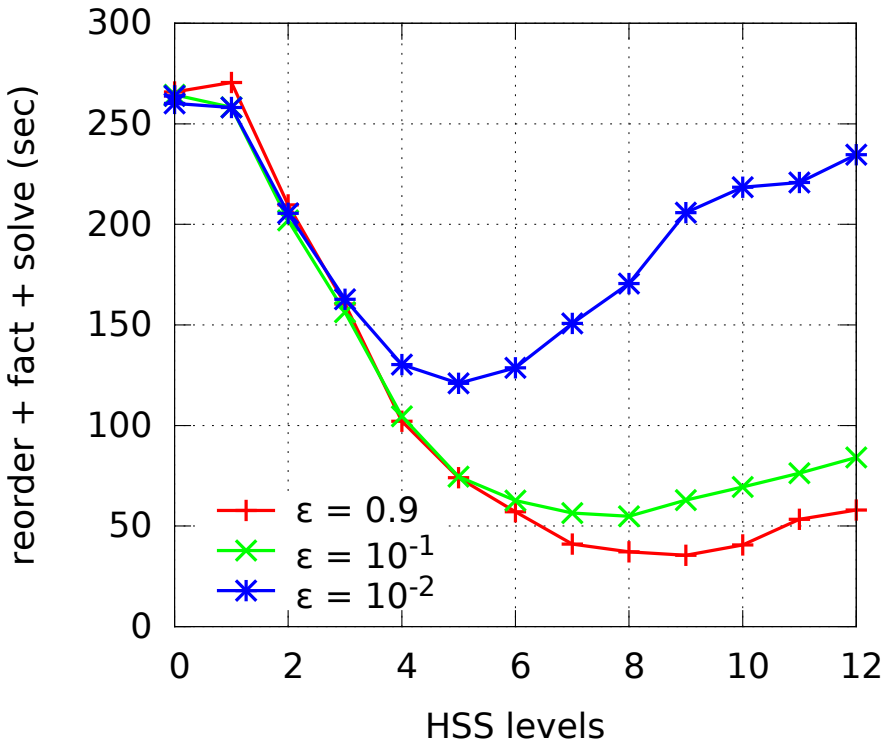


- Linear scaling with N
- Note the much bigger constants

125^3 Poisson equation – 12-core Intel® Ivy Bridge

- Left: Numerical factorization time
- Right: Total solve time (factorization + GMRES solve)
- Different lines for different tolerances in the rank-revealing QR

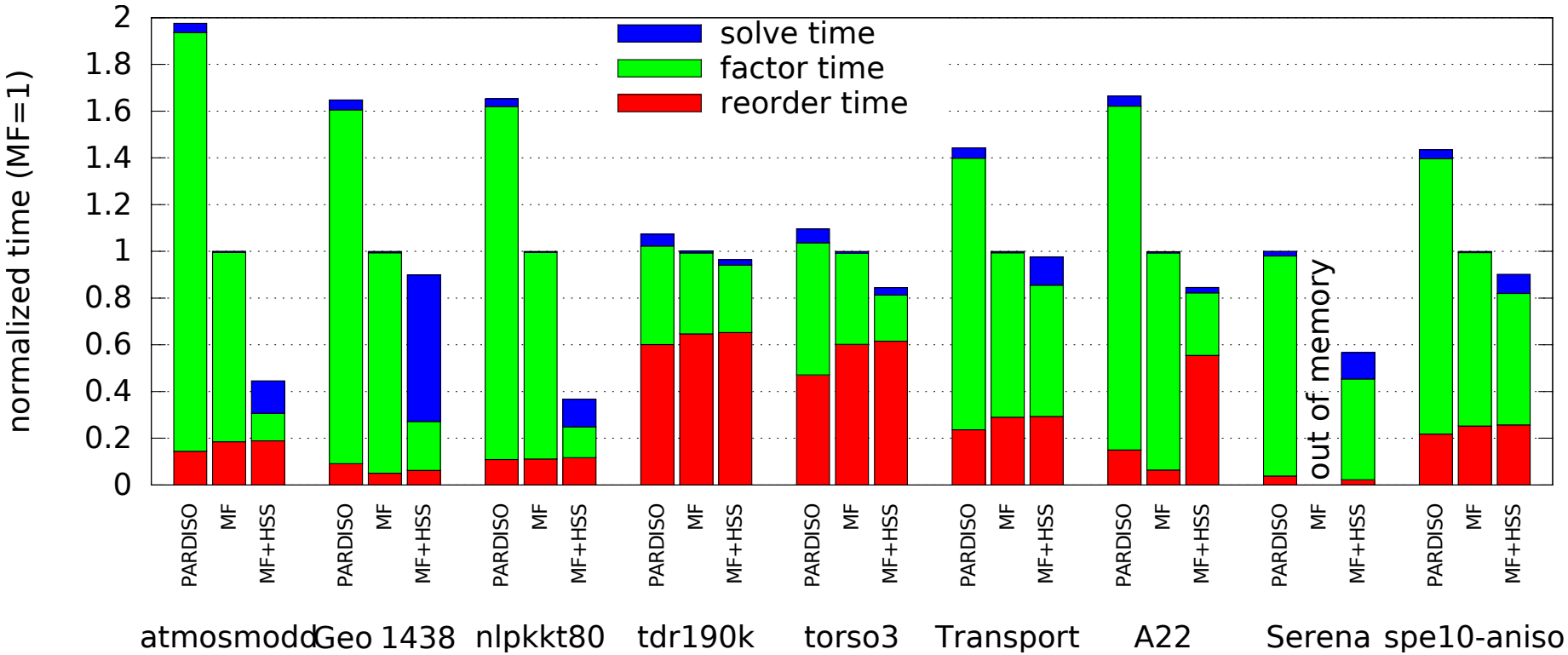




Optimum: $\varepsilon = 0.9$, $\ell_s = 8$ (of 18), 67 GMRES iterations, HSS-rank $= 46$
Pure multifrontal: $\ell_s = 0$

Compare to Intel MKL PARDISO

Matrices from T. Davis collection and SciDAC applications, based on underlying PDE



- HSS enabled solver
  - Extra cost for separator reordering negligible (except for A22)
  - Factorization cost decreases
  - Solve cost (and GMRES iterations) increases
- PARDISO uses same METIS nested dissection reordering

Conclusions and outlook

Software and preprints available at:

http://portal.nersc.gov/project/sparse/strumpack

First steps at developing parallel fast algebraic solvers

- Separation of algebra and analysis
  - Solver is purely algebraic; analysis needed to predicit its performance.
- Showed potential of the fast solver for PDE applications.
- Efficient code for shared memory platforms.
- Scalable MPI code for dense HSS computations.

Outlook

- Developing distributed memory code: MPI+OpenMP+?
  - Larger problems will yield larger gains
  - Is MPI+X the right tool?
- More accurate and/or faster rank-revealing (RR) factorization
  - Strong RRQR, Communication-avoiding RRQR, RRLU, ACA, ...
- Better reordering and hierarchical partitioning
  - Goal: preserve data-sparsity (lower HSS-rank)

Thank you!