Accelerating Direct Linear Solvers with Algorithmic and Hardware Advances

Xiaoye Sherry Li

xsli@lbl.gov

Lawrence Berkeley National Laboratory

SIAM Conference on Applied Linear Algebra, Oct. 26-30, 2015

Acknowledgements

Pieter Ghysels	LBNL
Xing Liu	IBM Watson Research Center
Artem Napov	Université Libre de Bruxelles
François-Henry Rouet	LBNL
Piyush Sao	Georgia Tech
Richard Vuduc	Georgia Tech
Sam Williams	LBNL
Rio Yokota	Tokyo Institute of Technology

The work is partially supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the US Department of Energy under contract no. DE-AC02-05CH11231.

Themes

Many research areas for exascale computing:

- Algorithms with lower arithmetic complexity, lower communication complexity

Themes

Many research areas for exascale computing:

- Algorithms with lower arithmetic complexity, lower communication complexity
  Multilevel algorithms
  - Multigrid
  - Fast Multipole Method (FMM)
  - Hierarchical matrices – algebraic generalization of FMM, (hopefully) applicable to broader classes of problems

Themes

Many research areas for exascale computing:

- Algorithms with lower arithmetic complexity, lower communication complexity
  Multilevel algorithms
  - Multigrid
  - Fast Multipole Method (FMM)
  - Hierarchical matrices – algebraic generalization of FMM, (hopefully) applicable to broader classes of problems

- Parallel algorithms and codes for machines with million-way parallelism, hierarchical organization
  - Distributed memory
  - Manycore nodes: 100s of lightweight cores, accelerators, co-processors

Reduce complexity with hierarchical matrix algorithms

Exploit STRUCTURES

1. Sparsity structure: defined by $\{0,1\}$ structure (Graphs).
   LU factorizaion $\sim O(N^2)$ flops, for many 3D discretized PDEs (loosely speaking).

Exploit STRUCTURES

1. Sparsity structure: defined by $\{0,1\}$ structure (Graphs).
   LU factorizaion $\sim O(N^2)$ flops, for many 3D discretized PDEs (loosely speaking).

   Software:

   - SPARSPAK (1981, George and Liu)
   - YSMP (1982, Eisenstat)
   - MA27 multifrontal (1983, Duff and Reid, MA37, MA38, MA47, MA57 etc. in HSL)
   - MUMPS, PaStiX, SuperLU, UMFPACK, WSMP, ...

Exploit STRUCTURES – low rankness

2. On top of (1), can find Data-sparse structure in dense (sub)matrices (approximation)
   $O(N)$ or $O(N\ polylog(N))$ flops for compression, factorization.

   Hierarchical matrices: $\mathcal{H}$- & $\mathcal{H}^2$-matrix (1999, Hackbusch et al.) and their subclasses.

   [Bebendorf, Bini, Börm, Chandrasekaran, Darve, Dewilde, Grasedyck, Gu, Le Borne, Martinsson, Tygert, Van Barel, van der Veen, Vandebril, Xia, ...]

Exploit STRUCTURES – low rankness

2. On top of (1), can find Data-sparse structure in dense (sub)matrices (approximation)
   $O(N)$ or $O(N\ polylog(N))$ flops for compression, factorization.

   Hierarchical matrices: $\mathcal{H}$- & $\mathcal{H}^2$-matrix (1999, Hackbusch et al.) and their subclasses.

   [Bebendorf, Bini, Börm, Chandrasekaran, Darve, Dewilde, Grasedyck, Gu, Le Borne, Martinsson, Tygert, Van Barel, van der Veen, Vandebril, Xia, ...]

   Software:

   - HLib (2004, Börm and Grasedyck)
   - HLIBPro (2004, Kriemann)
   - HODLR (2013, Ambikasaran and Darve)
   - MUMPS (2015, Amestoy et al.)
   - STRUMPACK (2015, LBNL)

Hierarchical matrix approximation

- Same mathematical foundation as FMM (Greengard and Rokhlin), put in matrix form:
  - Diagonal block ("near field") represented exactly
  - Off-diagonal block ("far field") approximated via low-rank format

FMM separability of Green's function	Algebraic low rank off-diagonal
$G(x,y) \approx \sum_{\ell=1}^{r} f_\ell(x) g_\ell(y)$	$A = \left[\begin{array}{c\|c} D_1 & U_1 B_1 V_2^T \\ \hline U_2 B_2 V_1^T & D_2 \end{array}\right]$
$x \in X, y \in Y$	

- Algebraic power: matrix multiplication, factorization, inversion, tensors, ...

Multilevel is the key to optimal complexity

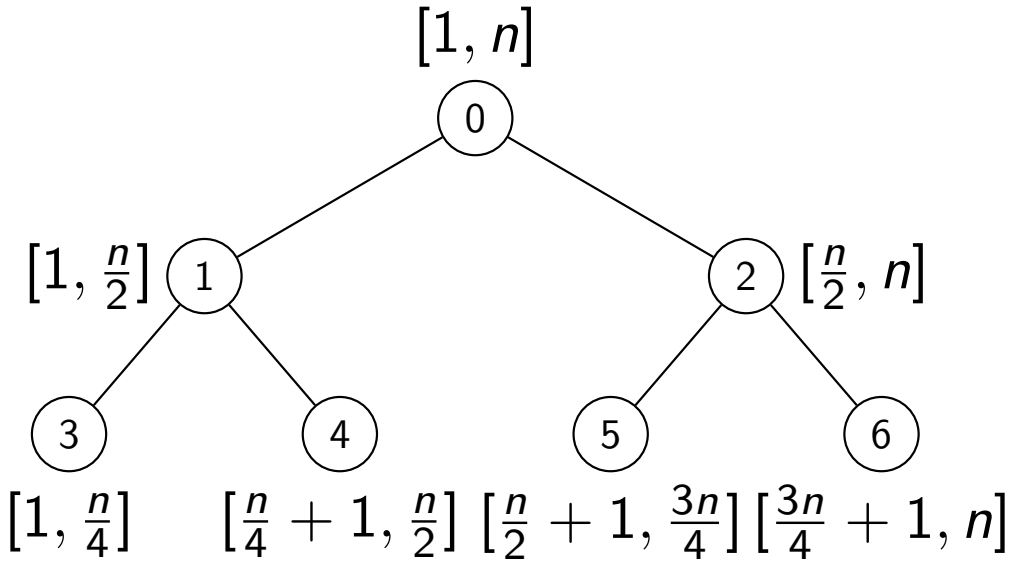
Two hierarchies:

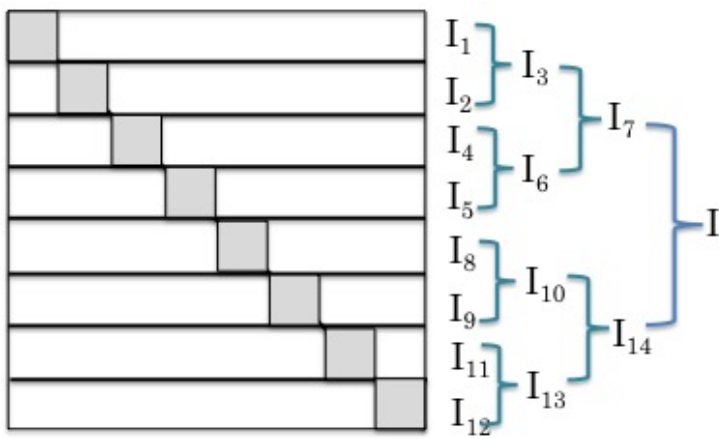
- Hierarchical partitioning using cluster tree
- Hierarchical, nested bases

1. Cluster tree, block cluster tree

Cluster tree $T_{\mathcal{I}}$ defines hierarchical partitioning of the index set $[1, n]$.

- Each node is associated with an interval $\mathcal{I}_\tau$.
- For children ν_1 and ν_2, parent $\mathcal{I}_\tau = \mathcal{I}_{\nu_1} \cup \mathcal{I}_{\nu_2}$, and $\mathcal{I}_{\nu_1} \cap \mathcal{I}_{\nu_2} = \emptyset$

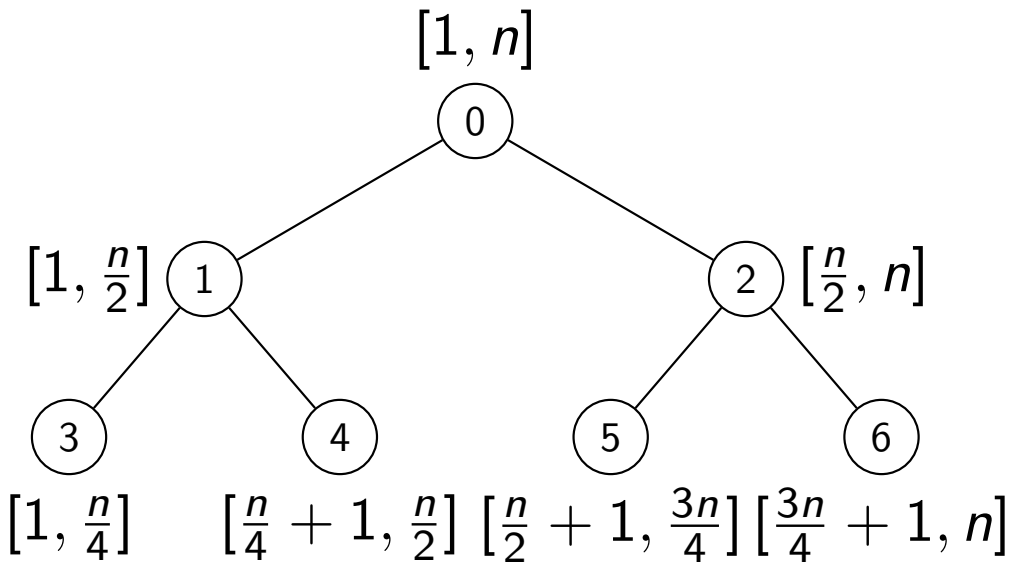


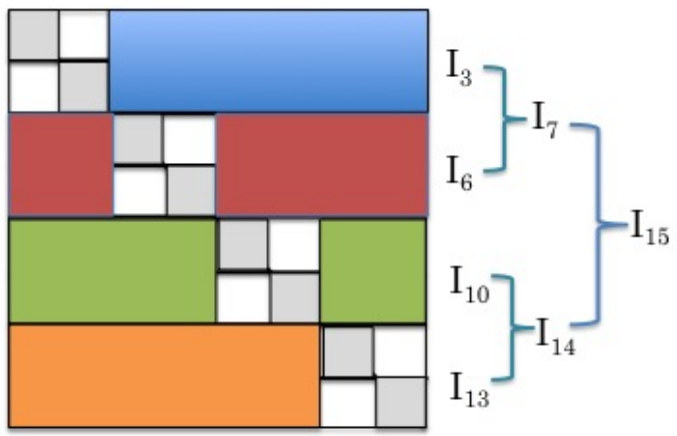


1. Cluster tree, block cluster tree

Cluster tree $T_{\mathcal{I}}$ defines hierarchical partitioning of the index set $[1, n]$.

- Each node is associated with an interval $\mathcal{I}_\tau$.
- For children ν_1 and ν_2, parent $\mathcal{I}_\tau = \mathcal{I}_{\nu_1} \cup \mathcal{I}_{\nu_2}$, and $\mathcal{I}_{\nu_1} \cap \mathcal{I}_{\nu_2} = \emptyset$

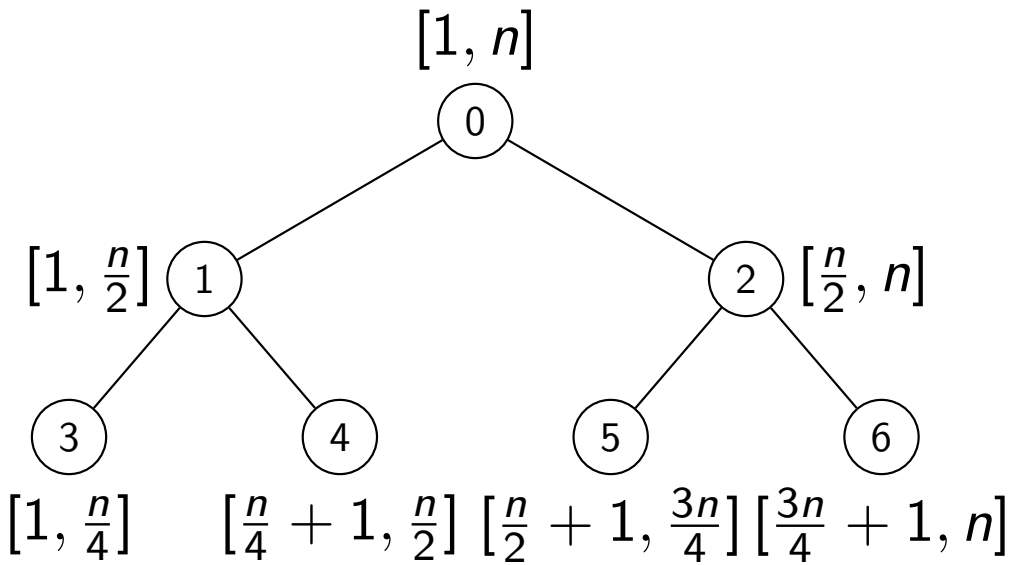


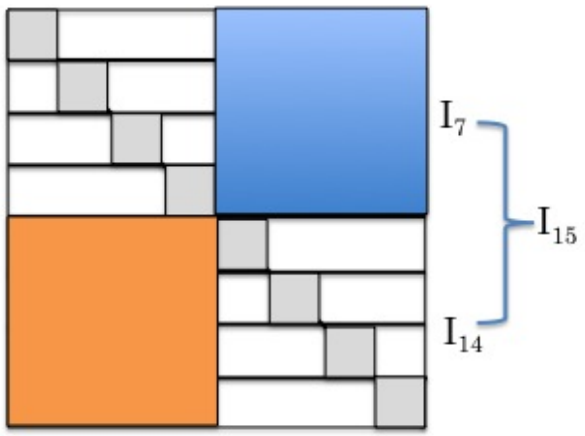


1. Cluster tree, block cluster tree

Cluster tree $T_{\mathcal{I}}$ defines hierarchical partitioning of the index set $[1, n]$.

- Each node is associated with an interval $\mathcal{I}_\tau$.
- For children ν_1 and ν_2, parent $\mathcal{I}_\tau = \mathcal{I}_{\nu_1} \cup \mathcal{I}_{\nu_2}$, and $\mathcal{I}_{\nu_1} \cap \mathcal{I}_{\nu_2} = \emptyset$

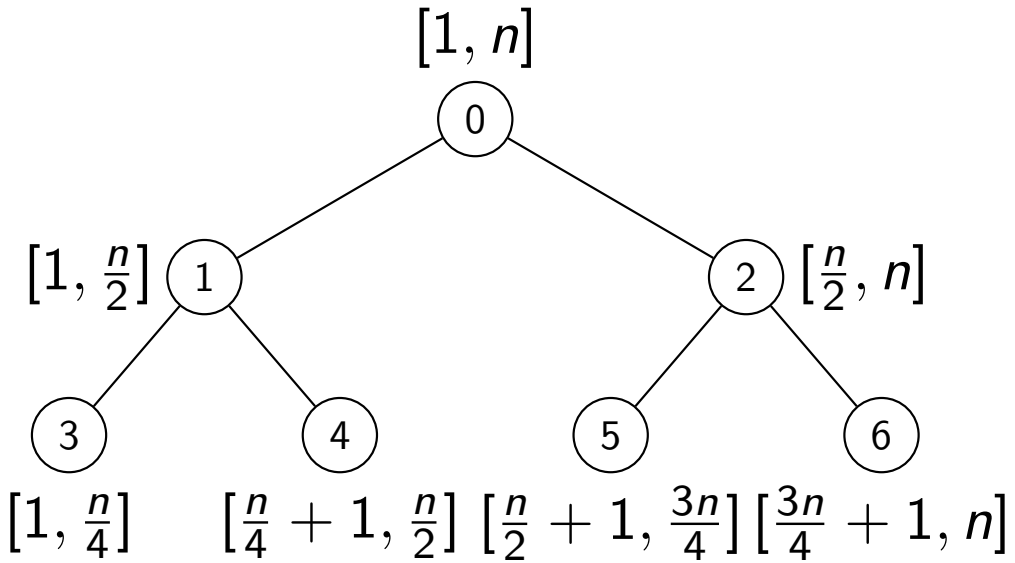


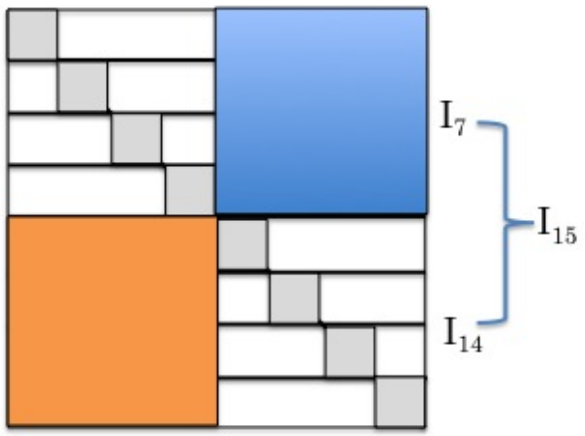


1. Cluster tree, block cluster tree

Cluster tree $T_{\mathcal{I}}$ defines hierarchical partitioning of the index set $[1, n]$.

- Each node is associated with an interval $\mathcal{I}_\tau$.
- For children ν_1 and ν_2, parent $\mathcal{I}_\tau = \mathcal{I}_{\nu_1} \cup \mathcal{I}_{\nu_2}$, and $\mathcal{I}_{\nu_1} \cap \mathcal{I}_{\nu_2} = \emptyset$



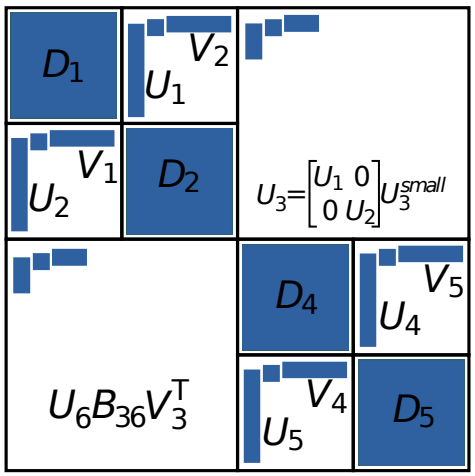


Block cluster tree $T_{\mathcal{I}\times\mathcal{J}}$ defines partitioning of the index set $\mathcal{I} \times \mathcal{J}$, both row- and column-wise.

- Each node corresp. to a matrix block $A(\mathcal{I}_\tau, \mathcal{I}_\sigma)$

2. Nested bases

Example: Hierarchically Semi-Separable matrices (HSS)



- Diagonal blocks are full rank: $D_\tau = A(I_\tau, I_\tau)$
- Off-diagonal blocks as low-rank:

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

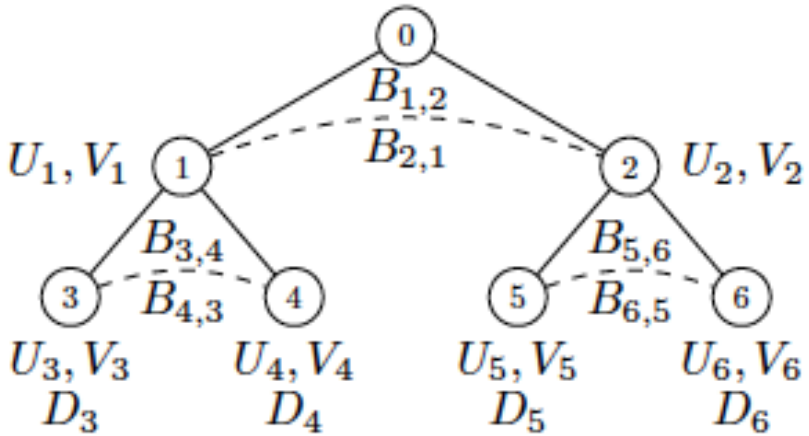
- Column bases U and row bases V^* are nested:

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

HSS matrix – operational view

$$A^{(\ell)} = U^{(\ell)} A^{(\ell-1)} V^{(\ell)^T} + B^{(\ell)}, \qquad \text{levels } \ell = 1, 2, \ldots, L$$

Data structures built on cluster tree (i.e. HSS tree):



- Keep as an unevaluated product & sum
- Operations going up / down the cluster tree

HSS matrix – ULV factorization

ULV-like factored form (U and V^* unitary, L triangular)

$$\Gamma_{1;b\leftrightarrow 2;t}\begin{bmatrix} I & & \\ & \Omega_1 & \\ & & I \\ & & & \Omega_2 \end{bmatrix}\begin{bmatrix} \Gamma_{3;b\leftrightarrow 4;t} & \\ & \Gamma_{5;b\leftrightarrow 6;t} \end{bmatrix}\begin{bmatrix} \Omega_3 & & \\ & \Omega_4 & \\ & & \Omega_5 \\ & & & \Omega_6 \end{bmatrix} A \begin{bmatrix} Q_3^* & & \\ & Q_4^* & \\ & & Q_5^* \\ & & & Q_6^* \end{bmatrix}\begin{bmatrix} \Gamma_{3;b\leftrightarrow 4;t}^T & \\ & \Gamma_{5;b\leftrightarrow 6;t}^T \end{bmatrix}\begin{bmatrix} I & & \\ & Q_1^* & \\ & & I \\ & & & Q_2^* \end{bmatrix}\Gamma_{1;b\leftrightarrow 2;t}^T$$

$$= \left[\begin{array}{|c|c|c|c|c|c|c|}
\hline
L_3 & & & & & & \\
\hline
0 & L_4 & & & & & \\
\hline
(\Omega_1 L_{4,3})_t & (\Omega_1 L_{3,4})_t & L_1 & & & & \\
\hline
\multicolumn{3}{|c|}{0} & L_5 & & & \\
\hline
 & & & 0 & L_6 & & \\
\hline
 & & & (\Omega_2 L_{6,5})_t & (\Omega_2 L_{5,6})_t & L_2 & \\
\hline
(\Omega_1 L_{4,3})_b & (\Omega_1 L_{3,4})_b & W_{1;b} Q_{1;t}^* & \multicolumn{2}{|c|}{B_{1,2} V_2^* \begin{bmatrix} V_5^* Q_{5;t}^* & V_5^* Q_{5;b}^* \\ V_6^* Q_{6;t}^* & V_6^* Q_{6;b}^* \end{bmatrix}\begin{bmatrix} I & \\ & Q_2^* \end{bmatrix}} & & D_0 \\
\hline
\multicolumn{3}{|c|}{B_{2,1} V_1^* \begin{bmatrix} V_3^* Q_{3;t}^* & V_3^* Q_{3;b}^* \\ V_4^* Q_{4;t}^* & V_4^* Q_{4;b}^* \end{bmatrix}\begin{bmatrix} I & \\ & Q_1^* \end{bmatrix}} & (\Omega_2 L_{6,5})_b & (\Omega_2 L_{5,6})_b & W_{2;b} Q_{2;t}^* & \\
\hline
\end{array}\right]$$

Families of $\mathcal{H}$ and $\mathcal{H}^2$ matrices

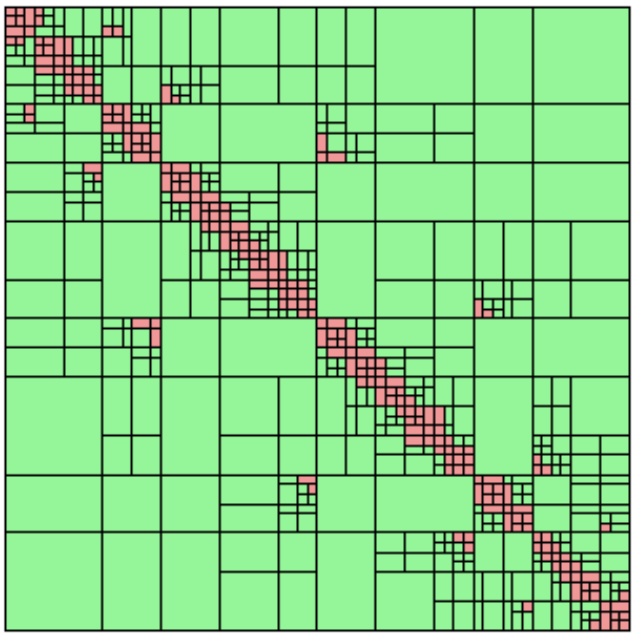
- Admissibile block (τ, σ): $\max\{\mathrm{diam}(\tau), \mathrm{diam}(\sigma)\} \leq \eta \, \mathrm{dist}(\tau, \sigma)$
- Strong admissibility: blocks next to diagonal not compressed, only compress well separated blocks

Families of $\mathcal{H}$ and $\mathcal{H}^2$ matrices

- Admissibile block (τ, σ): $\max\{\text{diam}(\tau), \text{diam}(\sigma)\} \leq \eta \, \text{dist}(\tau, \sigma)$
- Strong admissibility: blocks next to diagonal not compressed, only compress well separated blocks

- $\mathcal{H}$: split a node in a block cluster tree if its block is admissible
- $\mathcal{H}^2$: uniform $\mathcal{H}$ partitioning, with nested bases

[Börn/Grasedyck/Hackbusch]

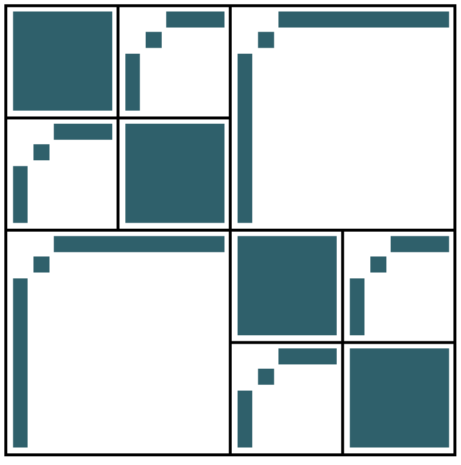


Various data-sparse formats

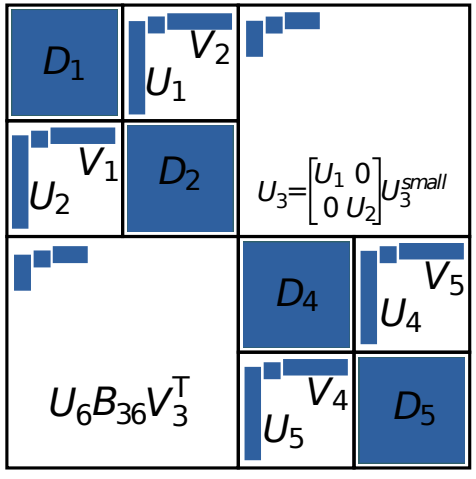
Method	Hier. part.	Nested bases	Admissibility	Family
HODLR	yes	no	weak	$\mathcal{H}$
HSS/HBS	yes	yes	weak	$\mathcal{H}^2$
Barnes-Hut	yes	no	strong	$\mathcal{H}$
FMM	yes	yes	strong	$\mathcal{H}^2$
BLR	no	no	strong	

Various data-sparse formats

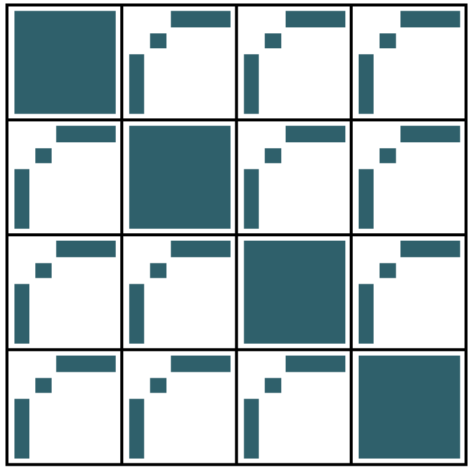
HODLR



HSS

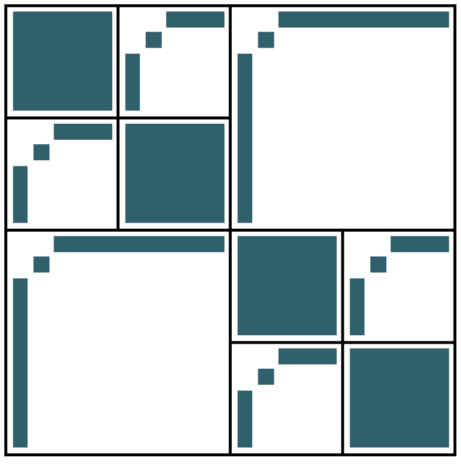


BLR

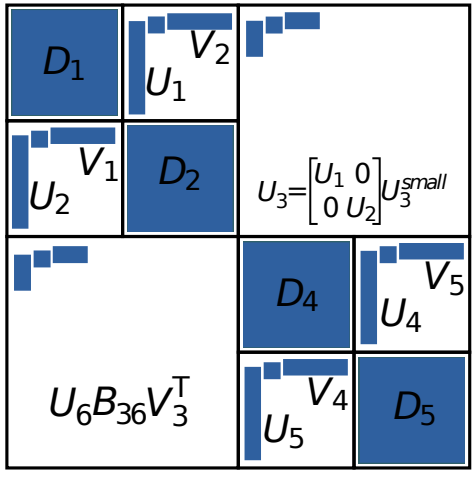


Various data-sparse formats

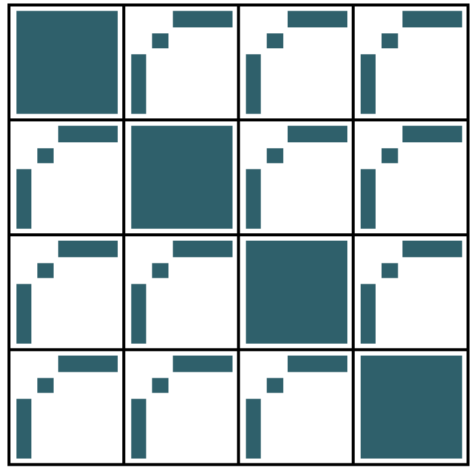
HODLR



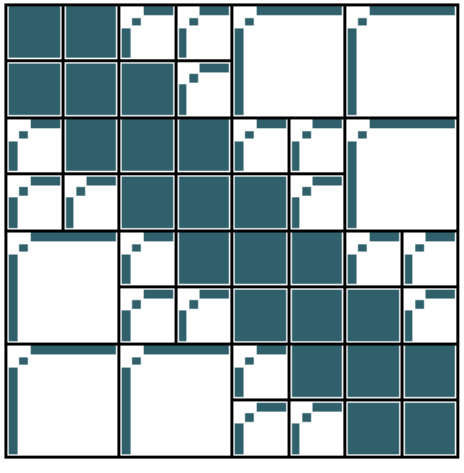
HSS



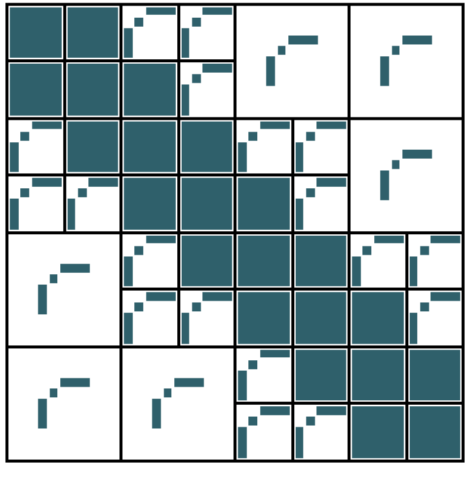
BLR



Barnes-Hut



FMM



Practical comparisons ... two other talks

- Francois-Henry Rouet, MS39, Thursday, 11:45-12:10
  "A Comparison of Different Low-Rank Approximation Techniques"

- Rio Yokota, MS45, Thursday, 3:30-3:55
  "Comparison of FMM and HSS at Large Scale"

Low rank 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 $Q = \text{ON-basis}(S)$

Low rank 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:** $O(kmn)$

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

RS vs. "direct" methods (RRQR, truncated SVD)

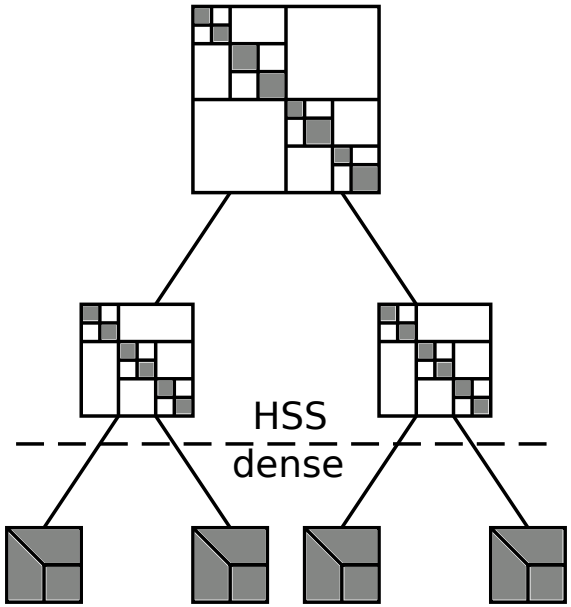
- When using explicit matrix, all have same asymptotic cost.
- RS can be faster when fast matvec available:
  - FMM ($O(kn)$)
  - Structured random matrix, e.g. SRFT (Subsampled Random Fourier Transform) ($O(\log(k)mn)$)
- RS useful when only matvec available (matrix-free).

- **In sparse solver, RS is more appealing ...**

Embedding HSS in multifrontal sparse solver

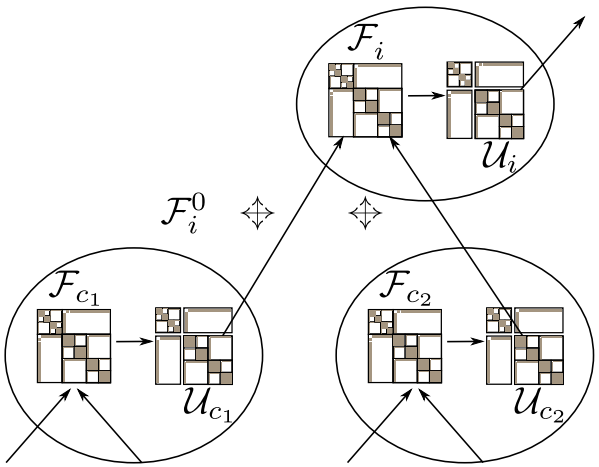
Approximate frontal matrices with HSS

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



Sparse multifrontal: Extend-Add $\mathcal{F}_j = A_j \boxplus \mathcal{U}_{c_1} \boxplus \mathcal{U}_{c_1}$

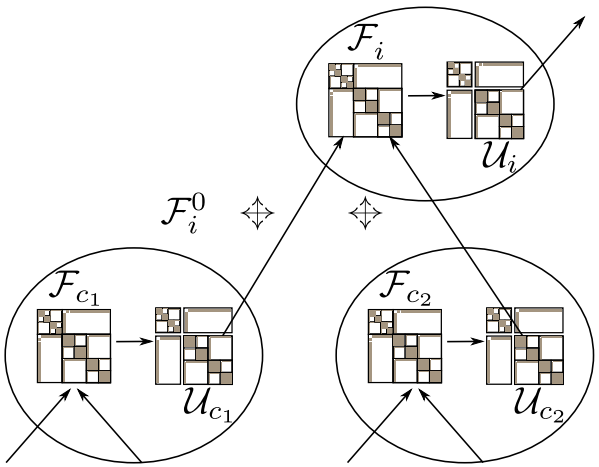
$$\mathcal{U}_{c_1} + \mathcal{U}_{c_2} = \begin{pmatrix} a_1 & b_1 & 0 \\ c_1 & d_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} a_2 & 0 & b_2 \\ 0 & 0 & 0 \\ c_2 & 0 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 & b_2 \\ c_1 & d_1 & 0 \\ c_2 & 0 & d_2 \end{pmatrix}$$



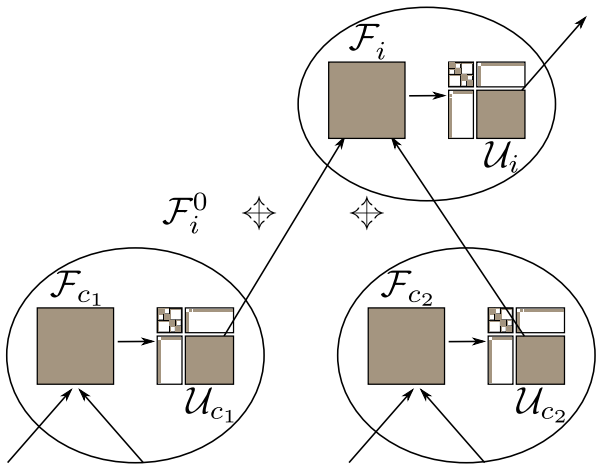
Main obstacle:
extend-add of two HSS structures

Sparse multifrontal: Extend-Add $\mathcal{F}_j = A_j \Leftrightarrow \mathcal{U}_{c_1} \Leftrightarrow \mathcal{U}_{c_1}$

$$\mathcal{U}_{c_1} + \mathcal{U}_{c_2} = \begin{pmatrix} a_1 & b_1 & 0 \\ c_1 & d_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} a_2 & 0 & b_2 \\ 0 & 0 & 0 \\ c_2 & 0 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 & b_2 \\ c_1 & d_1 & 0 \\ c_2 & 0 & d_2 \end{pmatrix}$$



Main obstacle:
extend-add of two HSS structures

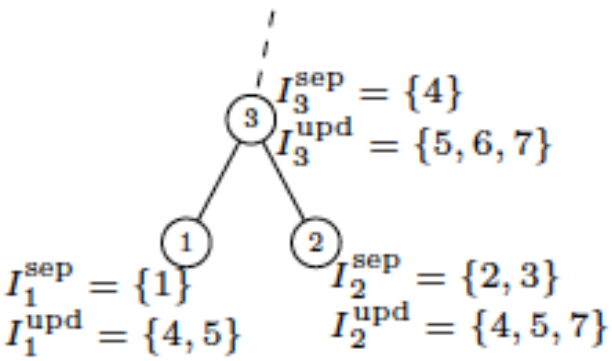


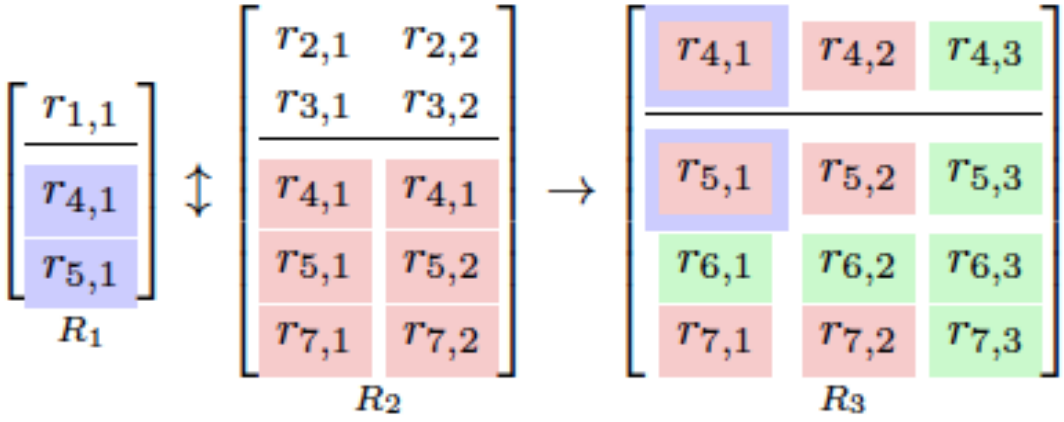
Hsolver [Wang/Rouet/Xia/L. 2013]
Compromise:

- keep update matrix as dense
- extend-add dense

RS simplies frontal extend-add

Skinny extend-merge:





Arithmetic complexities – dense HSS

Let r = HSS rank, i.e., maximum rank found during the different compression steps.

Compression

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

Arithmetic complexities – dense HSS

Let r = HSS rank, i.e., maximum rank found during the different compression steps.

Compression

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

ULV factorization and solve: $O(r\,N)$

Arithmetic complexities – sparse solver w/ HSS embedding

Rank patterns for discretized PDEs on k^d mesh: [Chandrasekaran et al.'10, Enquist/Ying'11, Enquist/Zhao '14]

	2D	3D
Poisson	$O(1)$	$O(k)$
Helmholtz	$O(\log k)$ or $O(k)$	$O(k)$

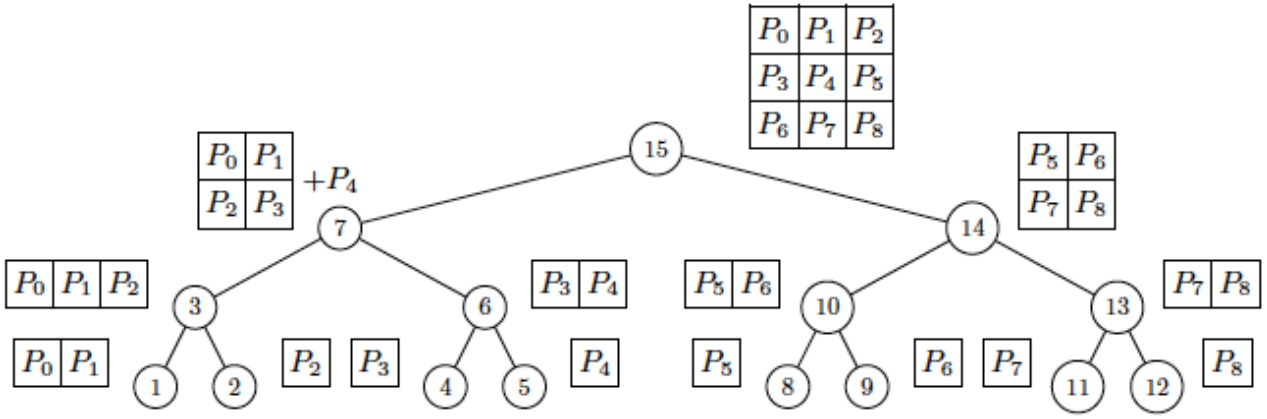
Solver complexities:

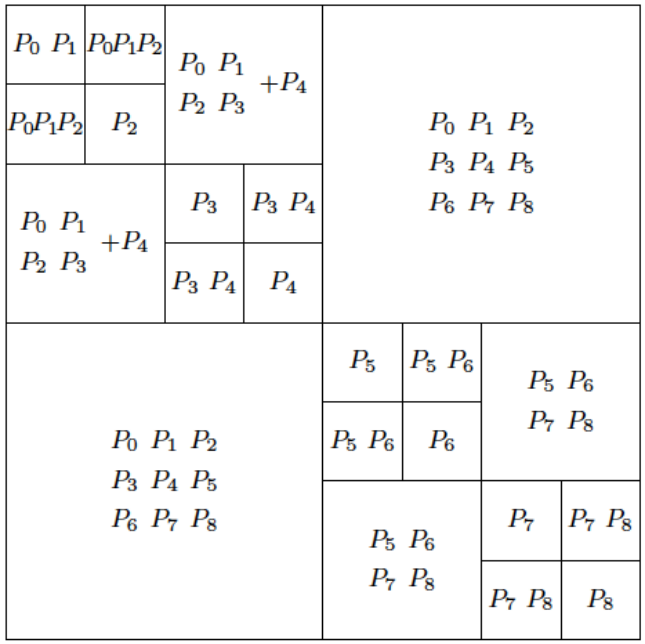
3D Helmholtz cost: [Xia '13] (seems true for many PDEs)

	Mem	Flops
MF-HSS	$O(N \log N)$	$O(N^{4/3} \log N)$
MF-HSS + RS	$O(N)$	$O(N \log N)$

Distributed-memory parallel algorithm

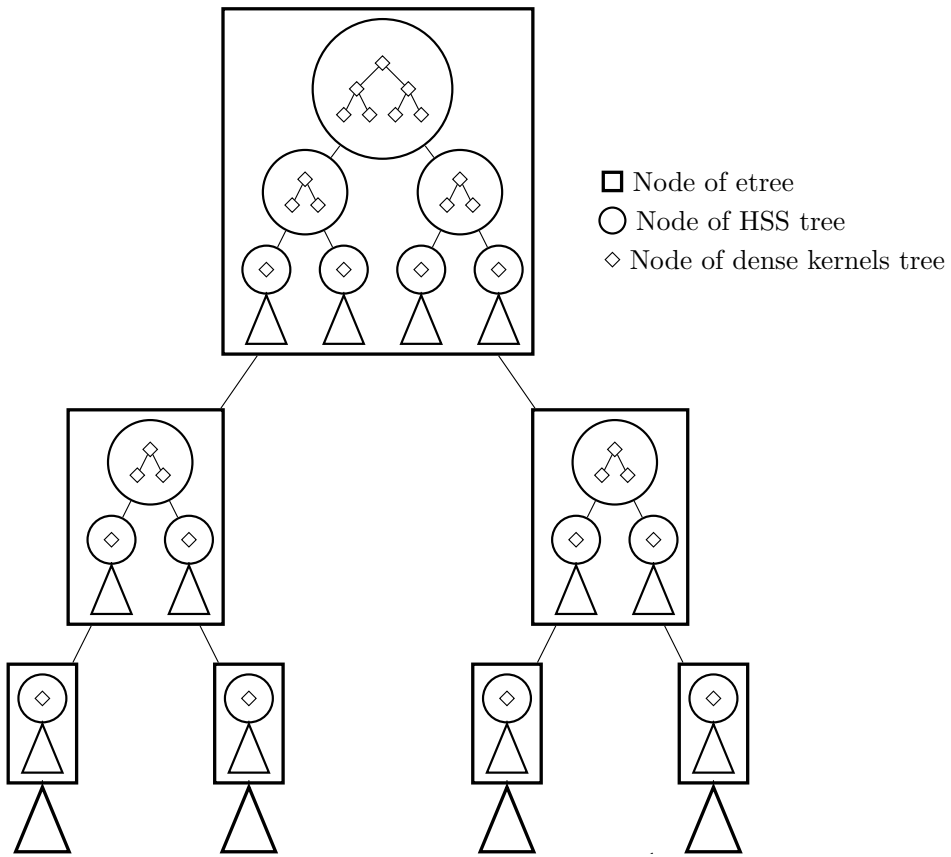
Use HSS tree to help data distribution





Multilevel parallelism in sparse MF-HSS solver

Elimination tree, HSS tree, BLAS tree



Dense – communication analysis

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

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

- Further reduce communication? Lower bound?

STRUMPACK – STRUctured Matrices PACKage

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

- C++, OpenMP, MPI
- Support both real & complex datatypes, single & double precision (via template), and 64-bit indexing.
- Input interfaces:
  - Dense matrix in standard format.
  - Matrix-free
  - Sparse matrix in CSR format.
- Can take user input: cluster tree & block partition
- Functions:
  - HSS construction, HSS-vector product, ULV factorization, Solution.
- Public domain, BSD license.

STRUMPACK – STRUctured Matrices PACKage

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

- C++, OpenMP, MPI
- Support both real & complex datatypes, single & double precision (via template), and 64-bit indexing.
- Input interfaces:
  - Dense matrix in standard format.
  - Matrix-free
  - Sparse matrix in CSR format.
- Can take user input: cluster tree & block partition
- Functions:
  - HSS construction, HSS-vector product, ULV factorization, Solution.
- Public domain, BSD license.

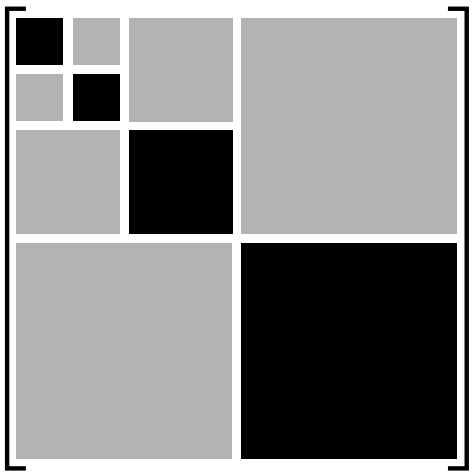
- Extensible to include other data-sparse formats.

Making software robust

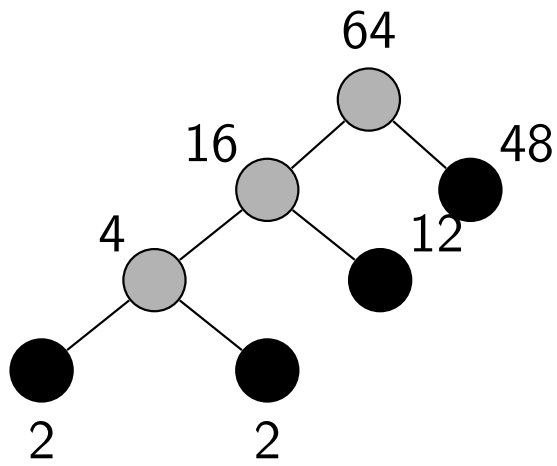
- Adaptive sampling machinery
  - Automatic handling unknown rank patterns: incrementally adjust sample size at any node when rank revealed is too large.
- Non-uniform clustering & partitioning

Making software robust

- Adaptive sampling machinery
  - Automatic handling unknown rank patterns: incrementally adjust sample size at any node when rank revealed is too large.
- Non-uniform clustering & partitioning



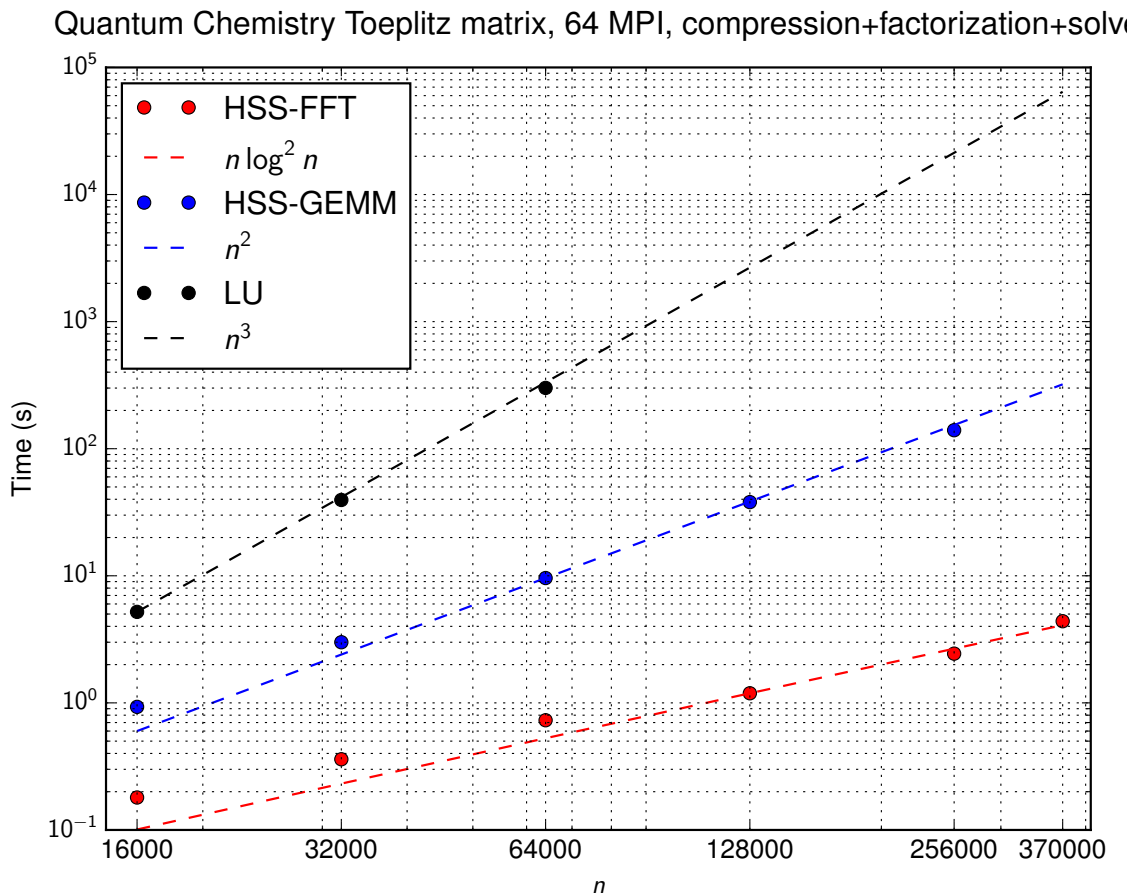
(c) Matrix structure.



(d) Weighted process mapping

Matrix-free interface

- Toeplitz: matvec via FFT
  Quantum chemistry: $a_{i,i} = \frac{\pi^2}{6}$, $a_{i,j} = \frac{(-1)^{i-j}}{(i-j)^2 d^2}$
- Previous best Toeplitz linear solver (e.g. Levinson): $O(n^2)$



Parallel weak scaling

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

k (3D mesh: k^3)	100	200	300	400	500
Matrix size ($=k^2$)	10,000	40,000	90,000	160,000	250,000
# Cores	64	256	1,024	4,096	8,192
Maximum rank	313	638	903	1289	1625
Compression time	2.0	13.0	30.6	60.8	133.6
Speed-up over ScaLAPACK	**1.8**	**4.0**	**5.4**	**4.8**	**3.9**
Flops ratio	**0.6**	**18.8**	**132.7**	**626.1**	**1716.7**

Parallel weak scaling

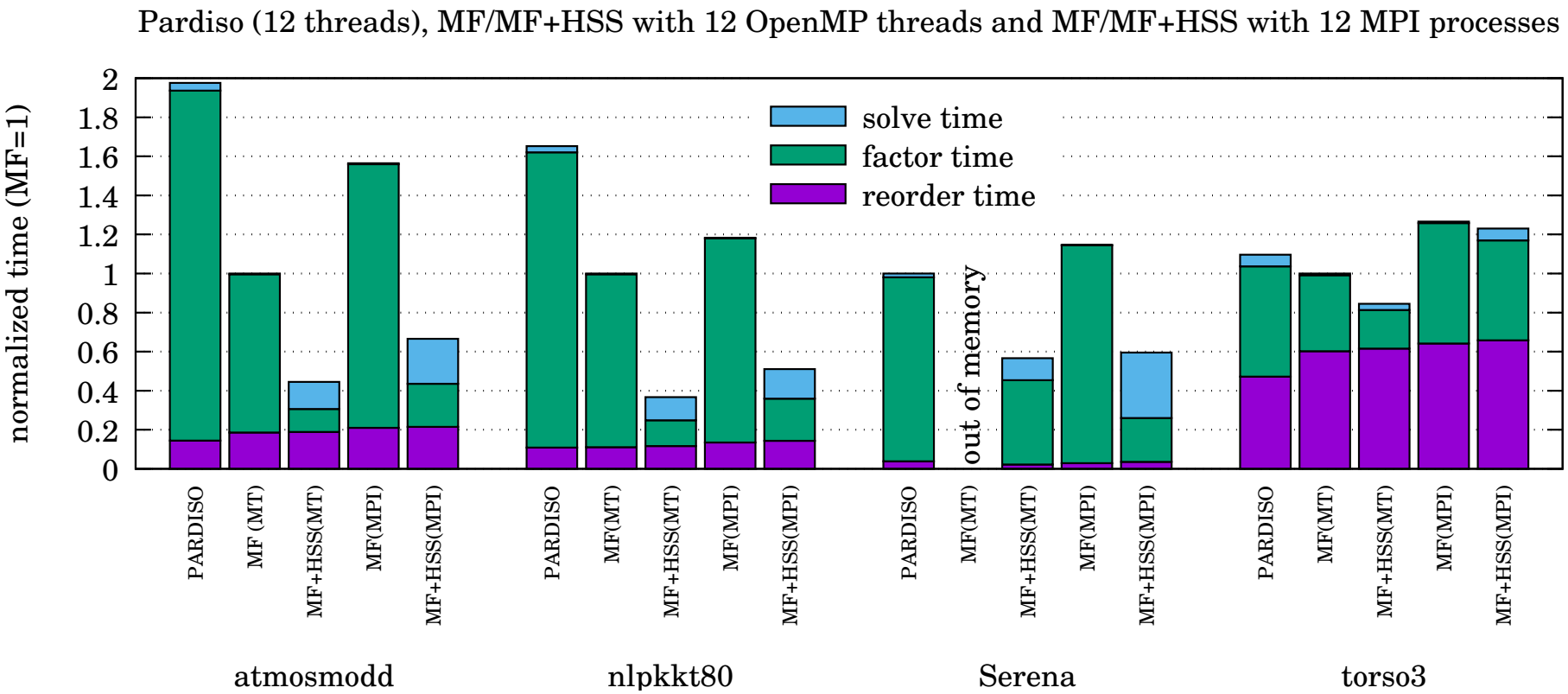
Root node of the multifrontal factorization of a discretized Helmholtz problem (frequency domain, PML boundary, 10Hz).

k (3D mesh: k^3)	100	200	300	400	500
Matrix size ($=k^2$)	10,000	40,000	90,000	160,000	250,000
# Cores	64	256	1,024	4,096	8,192
Maximum rank	313	638	903	1289	1625
Compression time	2.0	13.0	30.6	60.8	133.6
Speed-up over ScaLAPACK	**1.8**	**4.0**	**5.4**	**4.8**	**3.9**
Flops ratio	**0.6**	**18.8**	**132.7**	**626.1**	**1716.7**

Load imbalance

Parallel performance of sparse MF-HSS solvers

Cray XC30, Edison at NERSC



- Compared to PARDISO in Intel MKL library (12 threads)
- OpenMP (12 threads): MF, MF-HSS
- MPI (12 tasks): MF, MF-HSS

Further details ...

- Pieter Ghysels, MS51, Friday, 11:45-12:10
  "A Parallel Multifrontal Solver and Preconditioner Using Hierarchically Semiseparable Structured Matrices"

New parallel algorithms tracking architecture advances

Evolution of parallel machines, programming

- Vector machines, program with vectorization directives
- Shared memory UMA, program with directives or explicit threading
- Distributed memory machines presented major challenges
  - Data distribution, locality
  - Program with explicit messages, e.g., MPI
- Recently, heterogeneous node architectures, more disruptive
  - NUMA, socket / core / vector unit; accelerator / co-processor (e.g., GPU)
  - Memory per core is small
  - Program with mixed MPI & threads & CUDA ...

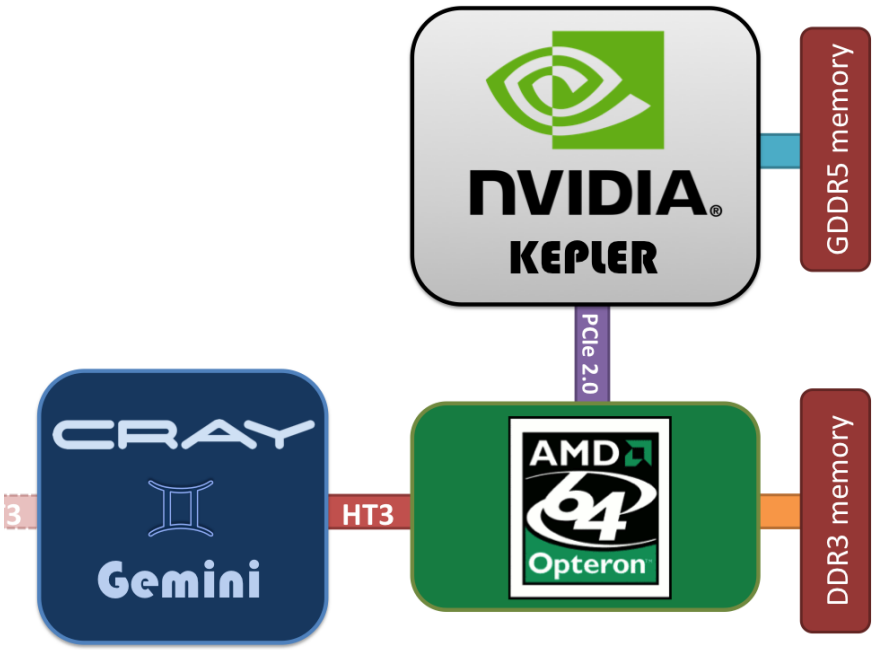
Evolution of parallel machines, programming

- Vector machines, program with vectorization directives
- Shared memory UMA, program with directives or explicit threading
- Distributed memory machines presented major challenges
  - Data distribution, locality
  - Program with explicit messages, e.g., MPI
- Recently, heterogeneous node architectures, more disruptive
  - NUMA, socket / core / vector unit; accelerator / co-processor (e.g., GPU)
  - Memory per core is small
  - Program with mixed MPI & threads & CUDA ...

Mixing task parallelism and data parallelism.

Variety of node architectures

Titan at ORNL:
16-core AMD + K20X GPU

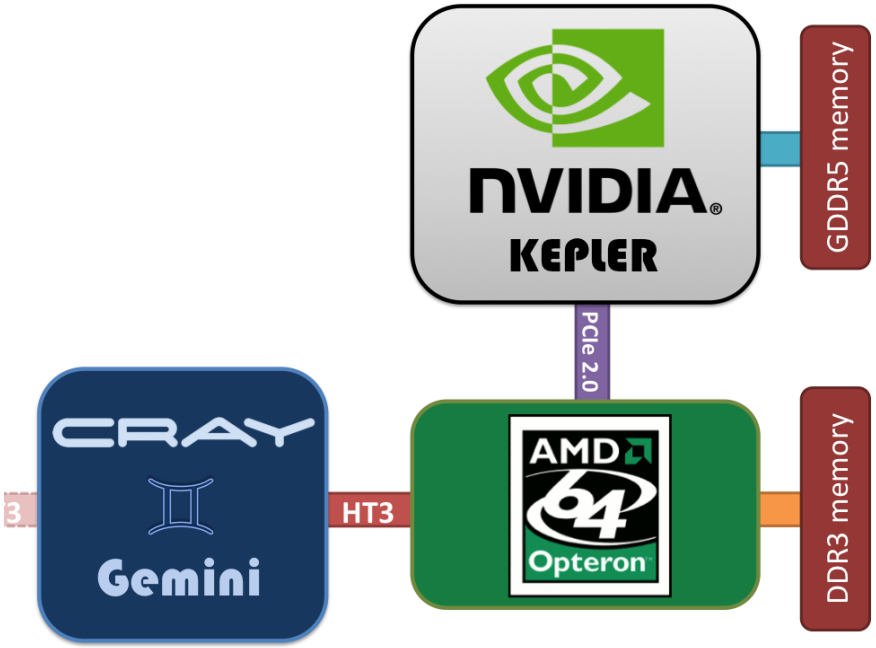


Programming:

- Separate CPU/GPU programs
- Transfer data between them

Variety of node architectures

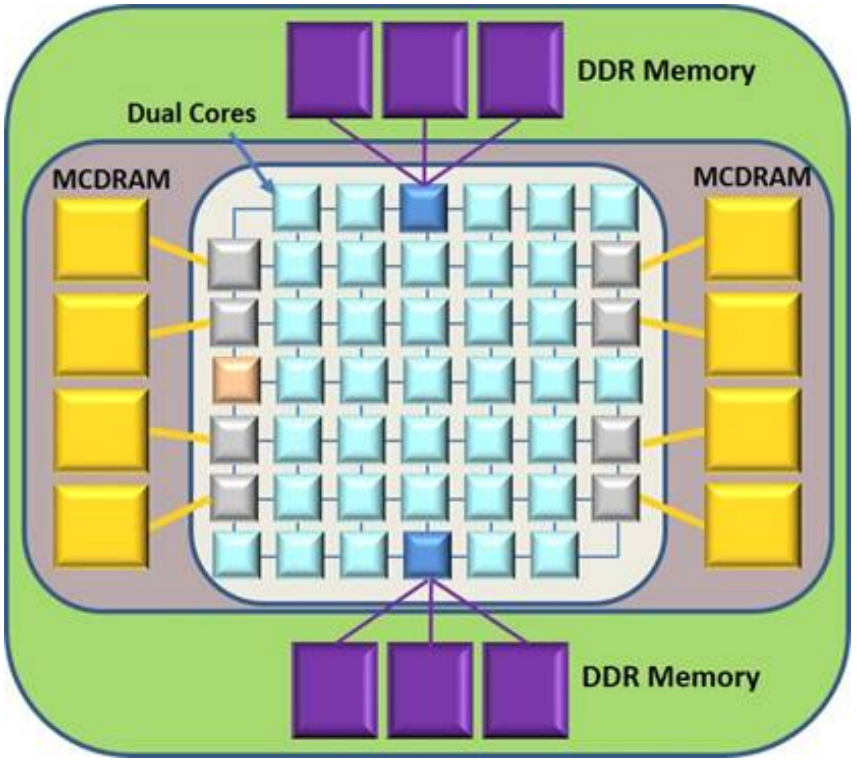
Titan at ORNL:
16-core AMD + K20X GPU



Programming:

- Separate CPU/GPU programs
- Transfer data between them

Intel Xeon Phi KNL (2016)

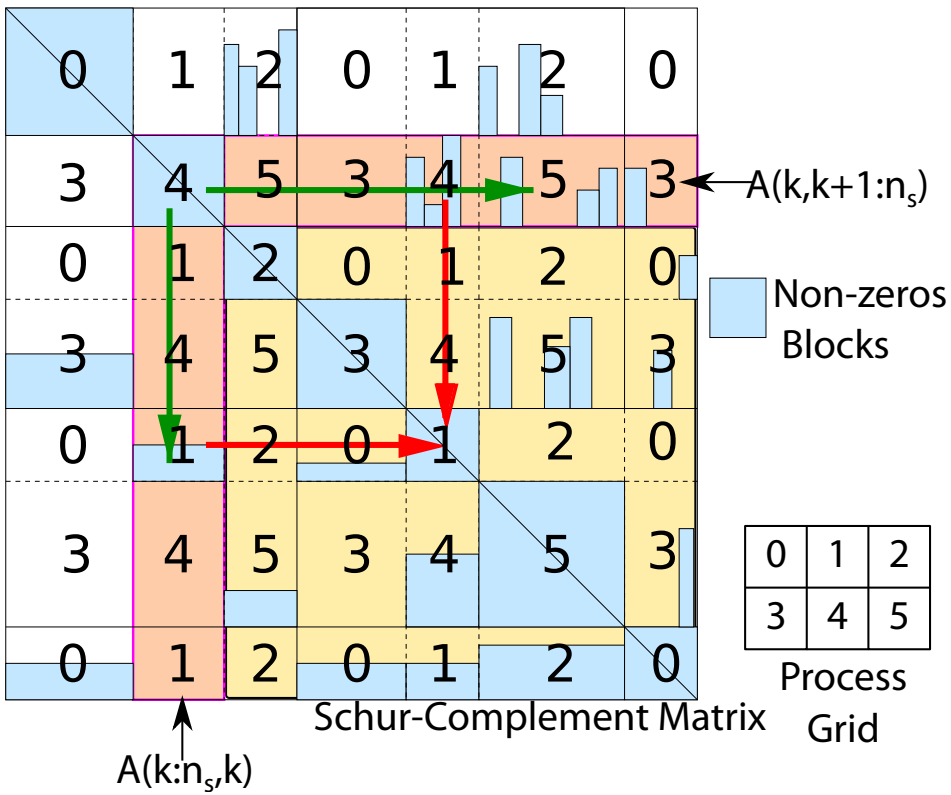


- 72 cores
- 4 threads/core
- 2 512bit vector units/core

SuperLU_DIST

http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

A widely used open-source sparse direct solver library [LBNL/UC Berkeley]



Algorithms

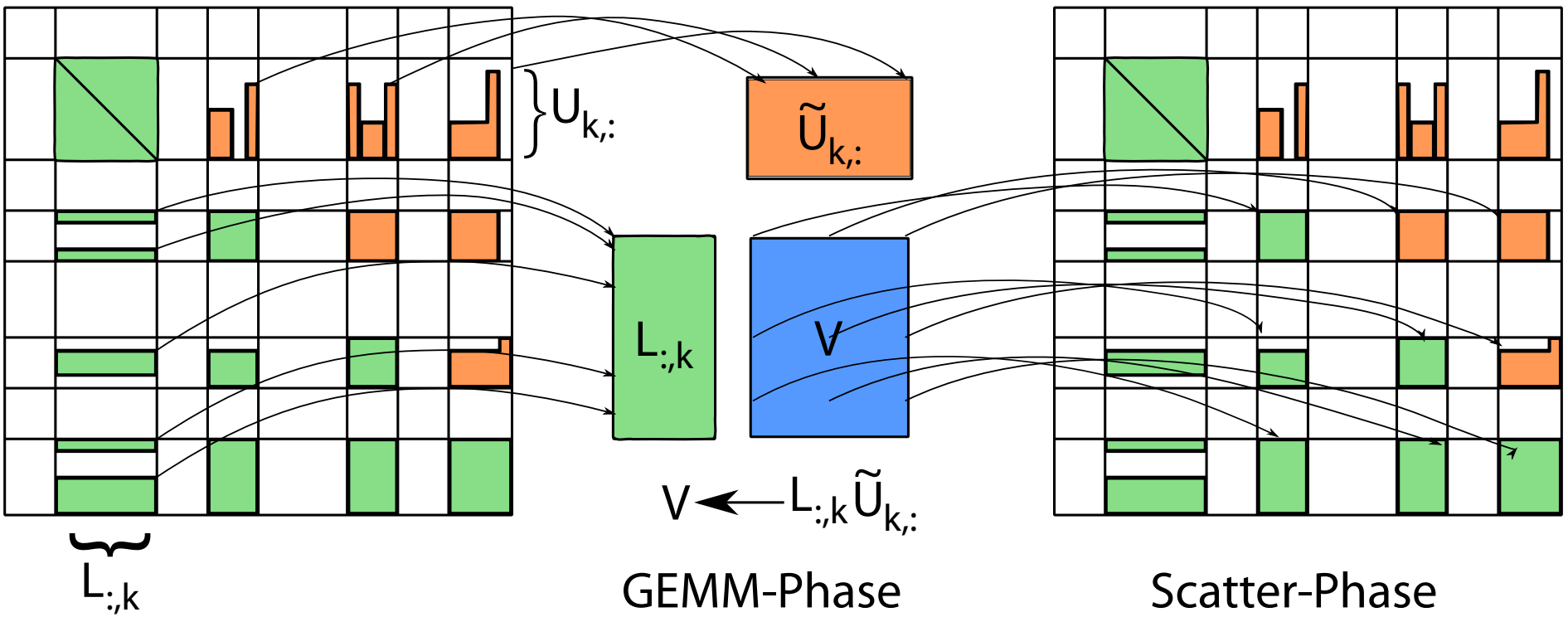
- Right looking
- Static pivoting
- 2D Cyclic data distribution

Two Major computational phases

1. Panel Factorization
2. Schur-Complement Update

Schur-complement udate

- Over 80% factorization time, ample parallelism
- Two operations: GEMM, Gather/Scatter



Design questions for accelerator / co-processor

- Only use accelerator, or use CPU as well?
  Accelerator memory small
  $\rightarrow$ best to use both (offload some computations to GPU)
- What to offload?
  Panel factorization not suitable for fine-grained data-parallel model
  $\rightarrow$ offload only Schur complement update
- Schur complement update: GEMM, and Gather/Scatter?
  - GEMM only compute intensive [Sao/Vuduc/L. 2014]
  - Both GEMM and Gather/Scatter indirect addressing, memory intensive [Sao/Liu/Vuduc/L. 2015]

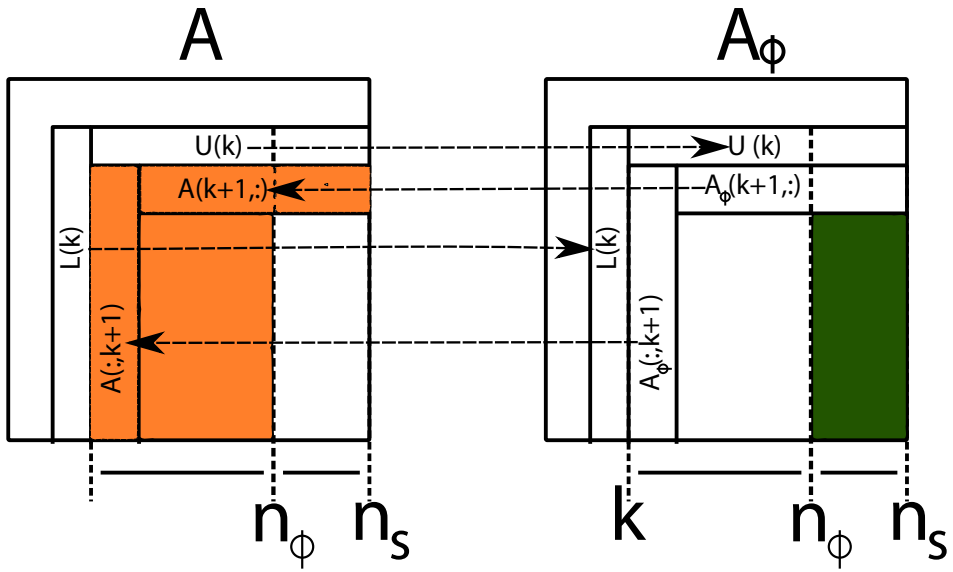
Design questions for accelerator / co-processor

- Only use accelerator, or use CPU as well?
  Accelerator memory small
  $\rightarrow$ best to use both (offload some computations to GPU)
- What to offload?
  Panel factorization not suitable for fine-grained data-parallel model
  $\rightarrow$ offload only Schur complement update
- Schur complement update: GEMM, and Gather/Scatter?
  - GEMM only compute intensive [Sao/Vuduc/L. 2014]
  - Both GEMM and Gather/Scatter indirect addressing, memory intensive [Sao/Liu/Vuduc/L. 2015]

Overlap activities on both CPU/GPU to hide transfer latency over PCIe bus (10-15 microseconds)

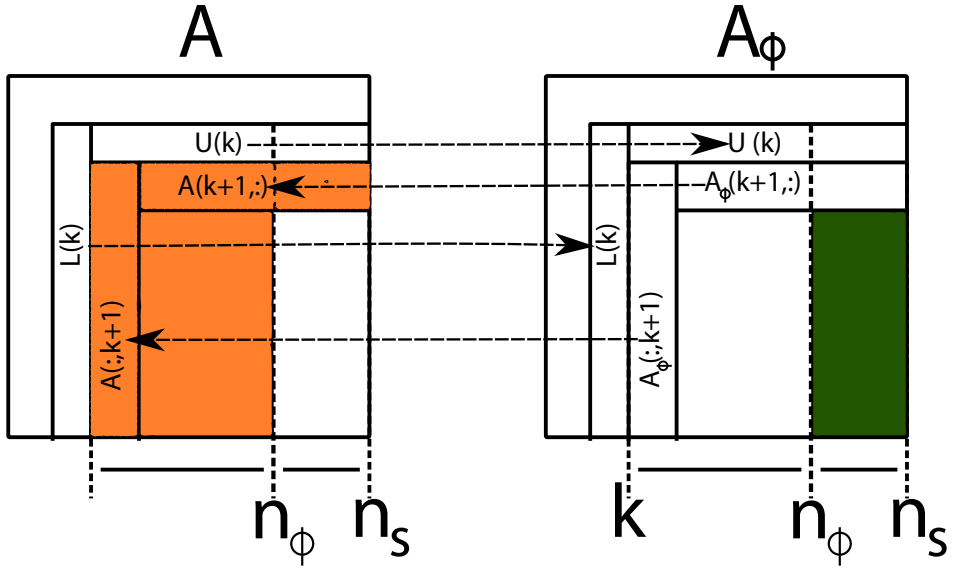
HALO algorithm – Highly Asynchronous Lazy Offload

- Maintain two partial sums of Schur-complement on both CPU and GPU, selectively offload Schur updates
  - A_ϕ has the same sparsity structure as A (2x memory duplicate)
  - Part of Schur update on CPU, part on GPU
- Reduce to-be-factorized panel on CPU, absorbing GPU's panel.



HALO algorithm – Highly Asynchronous Lazy Offload

- Maintain two partial sums of Schur-complement on both CPU and GPU, selectively offload Schur updates
  - A_ϕ has the same sparsity structure as A (2x memory duplicate)
  - Part of Schur update on CPU, part on GPU
- Reduce to-be-factorized panel on CPU, absorbing GPU's panel.



2-3x performance gains on 1000+ nodes GPU cluster Titan at ORNL

2-5x reduction in memory footprint

Further details ...

- Piyush Sao, MS48, Thursday, 4:00-4:25
  "A Sparse Direct Solver for Distributed Memory GPU and Xeon Phi Accelerated Systems"

Further details ...

- Piyush Sao, MS48, Thursday, 4:00-4:25
  "A Sparse Direct Solver for Distributed Memory GPU and Xeon Phi Accelerated Systems"

Other sparse factorization GPU work:

- PARDISO: left-looking sparse LU, offload BLAS [Schenk/Christen/Burkhart, 2008]
- WSMP: multifrontal sparse Cholesky, offload BLAS [George et al., 2011]
- Multifrontal sparse Cholesky, offload frontal matrix computation [Krawezik and Poole 2009, Vuduc et al. 2011, Yu/Wang/Pierce 2011]
- Threading, offload large frontal matrix computation [Lucas et al., 2010]
- Sparse multifrontal QR: offload subtrees of the assembly tree [Yeralan/Davis/Ranka, 2013]

Summary

- Sparse matrices can be made sparser by combining structural sparsity with data sparsity.
  - Good ordering and hierarchical clustering / partitioning
  - Alternative rank-revealing procedures
  - Dynamic load balancing
- Need redesign known algorithms, refactor existing codes for new architectures.

THANK YOU !