4-7-2021

Lecture 03: Hierarchically Low Rank Methods and Applications

David Keyes

King Abdullah University of Science and Technology, david.keyes@kaust.edu.sa

University of Arkansas Department of Mathematical Sciences

46th Spring Lecture Series

David Keyes

Extreme Computing Research Center

King Abdullah University of Science and Technology

5-9 April 2021

Lecture 3

Hierarchically Low Rank Methods and Applications

Motivations for rank-structured linear algebra

- **Mathematical aesthetic**
  - **Rank-structured matrix methods are beautiful – algebraic generalizations of fast multipole methods**
- **Engineering aesthetic**
  - **Data sparsity allows to tune *storage* and *work* to accuracy requirements**
- **Software engineering aesthetic**
  - **"Cool stuff" finds new roles: direct and randomized floating point kernels, tree-traversal from FMM, task-based programming, etc.**
- **Computer architecture dictates**
  - **Emerging architectures are "met on their terms": limited fast memory per core, SIMT instructions, etc.**
- **Application opportunities (see following)**

Reduce memory footprint and operation complexity via data sparsity

- Replace dense blocks with reduced rank representations, whether "born dense" or as arising during matrix operations
  - use high accuracy (high rank) to build "exact" solvers
  - use low accuracy (low rank) to build preconditioners
- Consider hardware parameters in tuning block sizes and maximum rank parameters
  - e.g., cache sizes, warp sizes
- Use randomized SVD, ACA, etc., to form low-rank blocks
  - flop-intensive GEMM-based flat algorithms
- Implement in "batches" of leaf blocks
  - uniform tiles for TLR and groups of tiles at each level of a flattened tree in the case of HLR

Lecture 2

Complexities of rank-structured factorization

For a square dense matrix of $O(N)$:

- **"Straight"** LU **or** LDL^{T}
  - **Operations** $O(N^3)$
  - **Storage** $O(N^2)$
- **Tile low-rank** (Amestoy, Buttari, L'Excellent & Mary, *SISC*, 2016)*
  - **Operations** $O(k^{0.5} N^2)$
  - **Storage** $O(k^{0.5} N^{1.5})$
  - **for uniform blocks with size chosen optimally for max rank** ***k*** **of any compressed block, bounded number of uncompressed blocks per row**
- **Hierarchically low-rank** (Grasedyck & Hackbusch, *Computing*, 2003)
  - **Operations** $O(k^2 N \log^2 N)$
  - **Storage** $O(k N)$
  - **for strong admissibility, where** ***k*** **is max rank of any compressed block**

* **First reported** $O(k^{0.5} N^{2.5})$**, then later** $O(k^{0.5} N^2)$ **for variant that reorders updates and recompression**

Also relevant to sparse problems

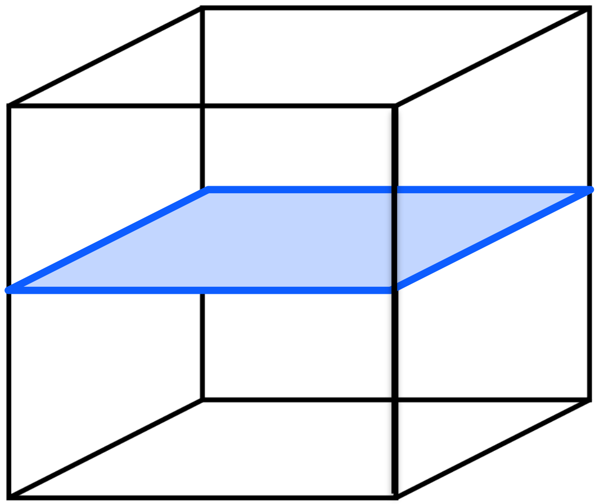
Classical factorizations fill in with elimination

For 3D Poisson solver on a cube with $O(N)$ degrees of freedom:

- **Classical nested dissection generally requires $\boxed{O(N^2)}$ operations**
- **Tile low-rank can yield $\boxed{O(N^{4/3})}$**

  (Amestoy, Buttari, L'Excellent & Mary, *SISC*, 2016)
- **Hierarchically low-rank methods can yield $\boxed{O(N)}$**

  (Bebendorf & Hackbusch, *Numer. Math.*, 2003)
- **Gains come from low-rank treatment of the resulting Schur complements**



Volume

$O(N) = O(n^3)$

Last Surface separator

$O(n^2) = O(N^{2/3})$

Dense elimination

$O((n^2)^3) = O(N^2)$

TLR elimination

$O((n^2)^2) = O(N^{4/3})$

A key question: what is k ?

In the complexity estimates, k is k_{max}, the maximum of the reduced rank over *all* blocks; for block accuracy $\|A - A_k\|_2 < \epsilon$, k scales like $\log(1/\epsilon)^{\gamma}$

For the purpose of *batching* many identical subproblems on GPUs, so that each thread in a warp executes the *same* step of a task on a *different* block of the matrix, without conditionals or masking, we want groups of common k

So k_{max} determines computational complexity of many implementations

k is straightforward to determine operationally for representation of a *given* block: if $A = U\Sigma V^{\top}$ and

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^{\top} \quad \text{then} \quad \|A - A_k\|_2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^{\top} \right\|_2 = \sigma_{k+1}$$

However, it is not straightforward to derive *a priori* the block-by-block accuracy that insures downstream *overall* accuracy

Recall (lecture 2) a wide range of k in TLR for the exponential kernel, not a rare type of kernel decay

Rank distribution challenges with 3D exponential kernels

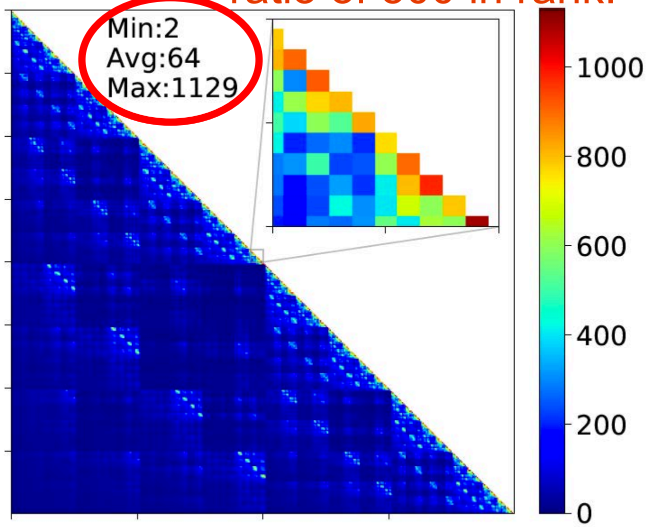
The simple exponential kernel:

$$C(r;\ell) = exp(-\frac{r}{\ell})$$

is suited for rough correlations such as the variation of wind speed or temperature with altitude, and leads to wide rank disparities

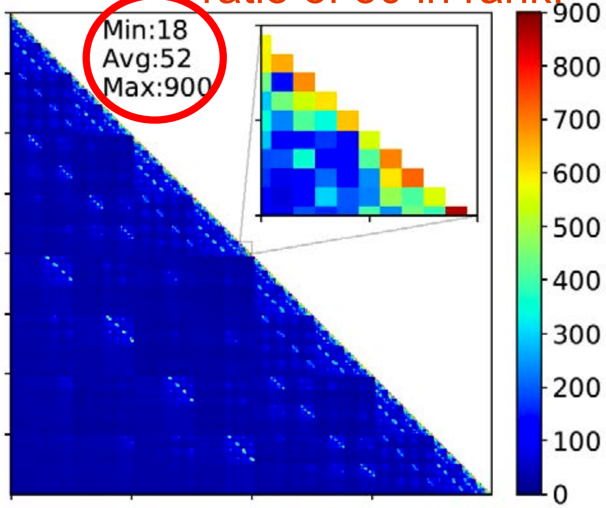
final ranks following Cholesky factorization

ratio of 500 in rank!



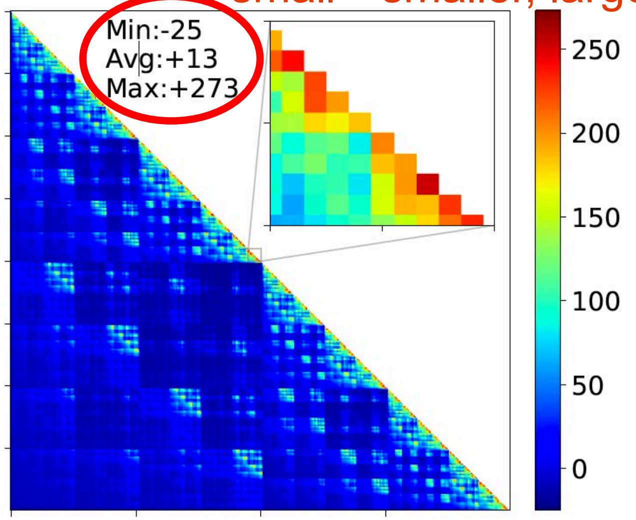
initial ranks following Morton ordering of 3D field with N=1.08M and B=2700 for matrix of 400 x 400 tiles

ratio of 50 in rank!



rank variations before and after factorization

small→smaller, large→larger



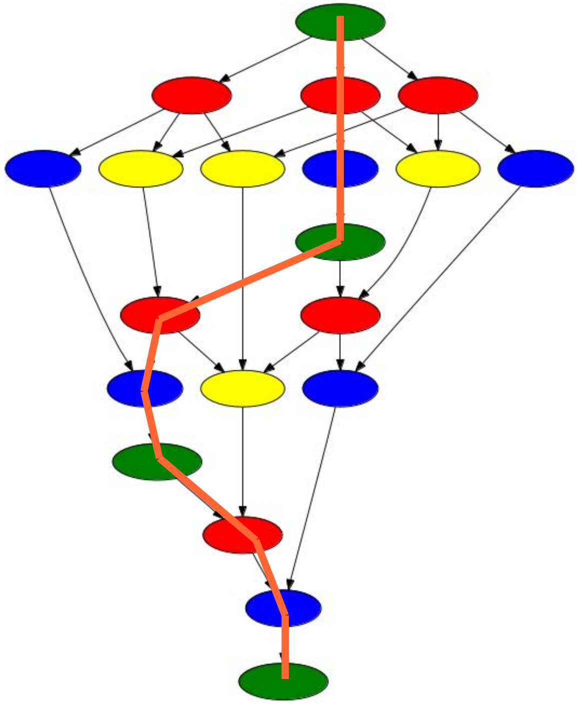
Cao, Pei, Akbudak, Bosilca, Ltaief, K. & Dongarra, *Leveraging PaRSEC Runtime Support to Tackle Challenging 3D Data-sparse Matrix Problems*. IPDPS (IEEE), 2021

TLR is a compromise between optimality and implementation complexity

TLR plugs directly into classical tile algorithms with subroutine overloading

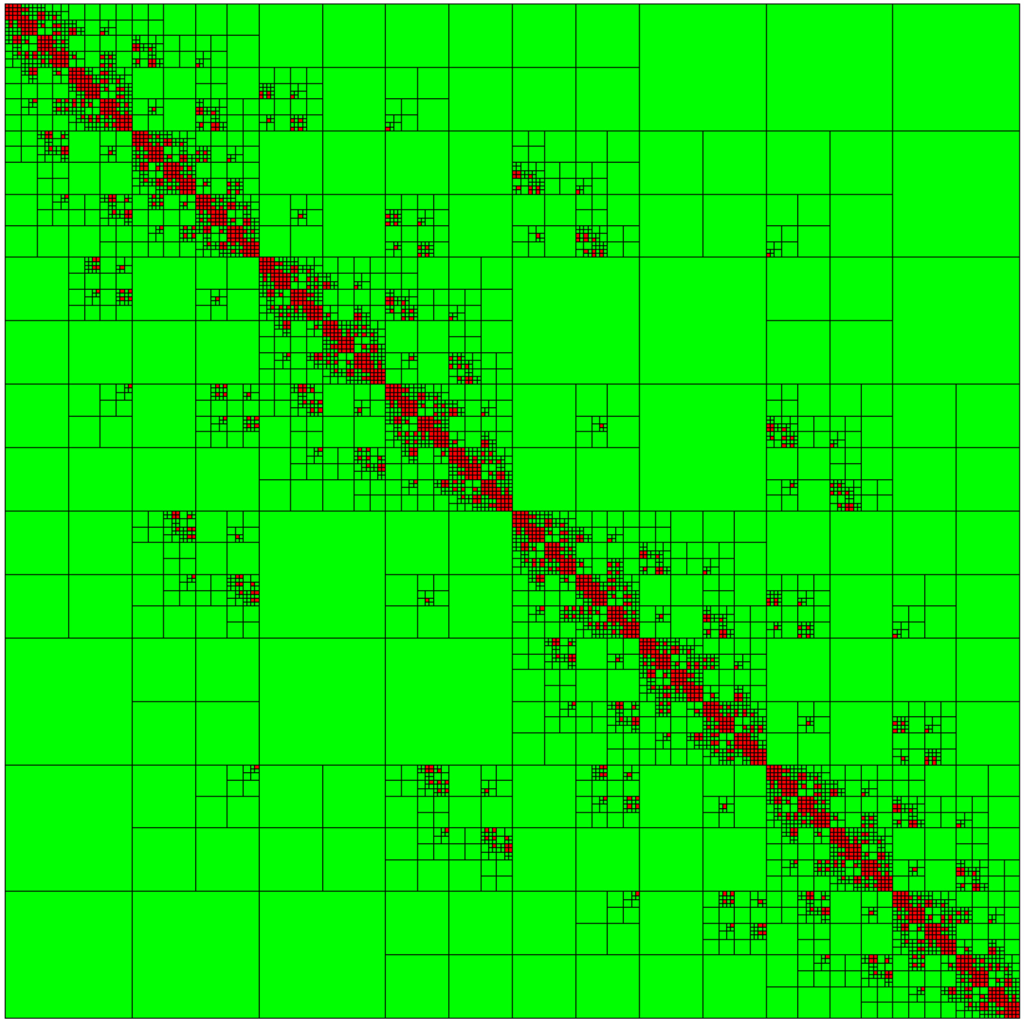
- **dense-dense block operations replaced with dense-TLR or TLR-TLR**
- **in lecture 4, we will follow the same pattern with precision overloading**

```
T = N / B  # B block size; T tiles per dim
for k = 1 to T do
   POTRF(D(k,k))
   for i = k+1 to T do
     TRSM(V(i,k), D(k,k))
   for j = k+1 to T do
         SYRK(D(j,j), U(j,k), V(j,k))
      for i = j+1 to T do
         GEMM(U(i,k), V(i,k), U(j,k),
                 V(j,k), U(i,j), V(i,j), acc)
```



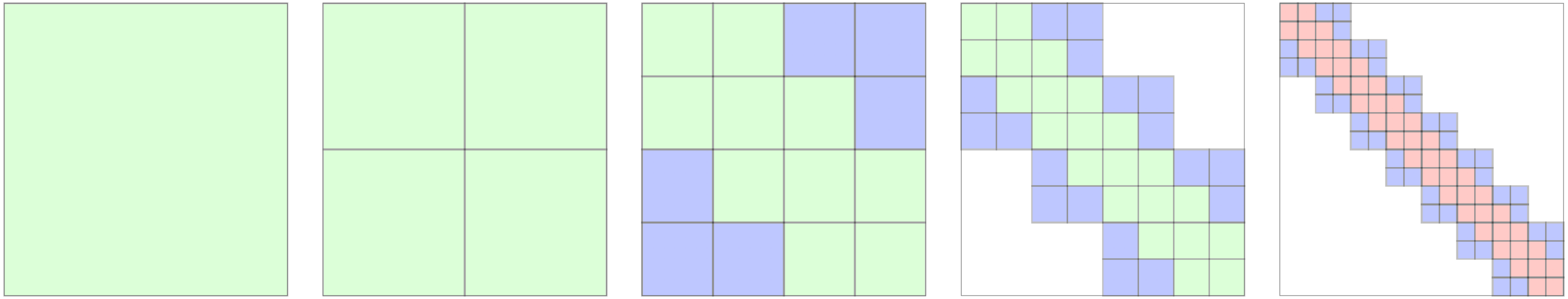
```
T = N / B  # B block size; T tiles per dim
for k = 1 to T do
   POTRF(D(k,k))
   for i = k+1 to T do
     TRSM(V(i,k), D(k,k))
      for j = k+1 to T
         LR_SYRK(D(j,j), U(j,k), V(j,k))
      for i = j+1 to T do
         LR_GEMM(U(i,k), V(i,k), U(j,k),
                 V(j,k), U(i,j), V(i,j), acc)
```

Hierarchical Low Rank (HLR) methods enrich the range of compromises



- Red blocks dense; green blocks low rank
- In this example, many red blocks appear far off the diagonal
- Many structure flavors exist
- Hybrids between them also exist
  - e.g., low-rank blocks from the outer decomposition can be replaced with a TLR matrix instead of an SVD
  - smaller SVDs occur within the TLR subblocks (nested hybrid)
  - could also, in principle with operator overloading, have side-by-side hybrids
- Ultimately, need to balance between *optimal memory and compute complexity* and *complexity of implementation*,
  - considering load balance and ability to exploit SIMD/SIMT instructions and efficient implementations of BLAS

Conceptualization of $\mathcal{H}$-matrix construction



Step 0 Step 1 Step 2 Step 3 Step 4

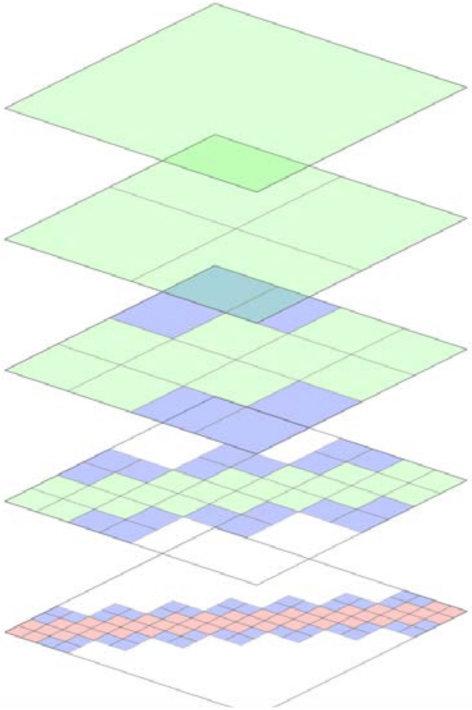
Specify two parameters:

- Block size *b* acceptably small to handle densely
- Rank *k* acceptably small to represent a block (depends on *b*)

Until each block is acceptably small:

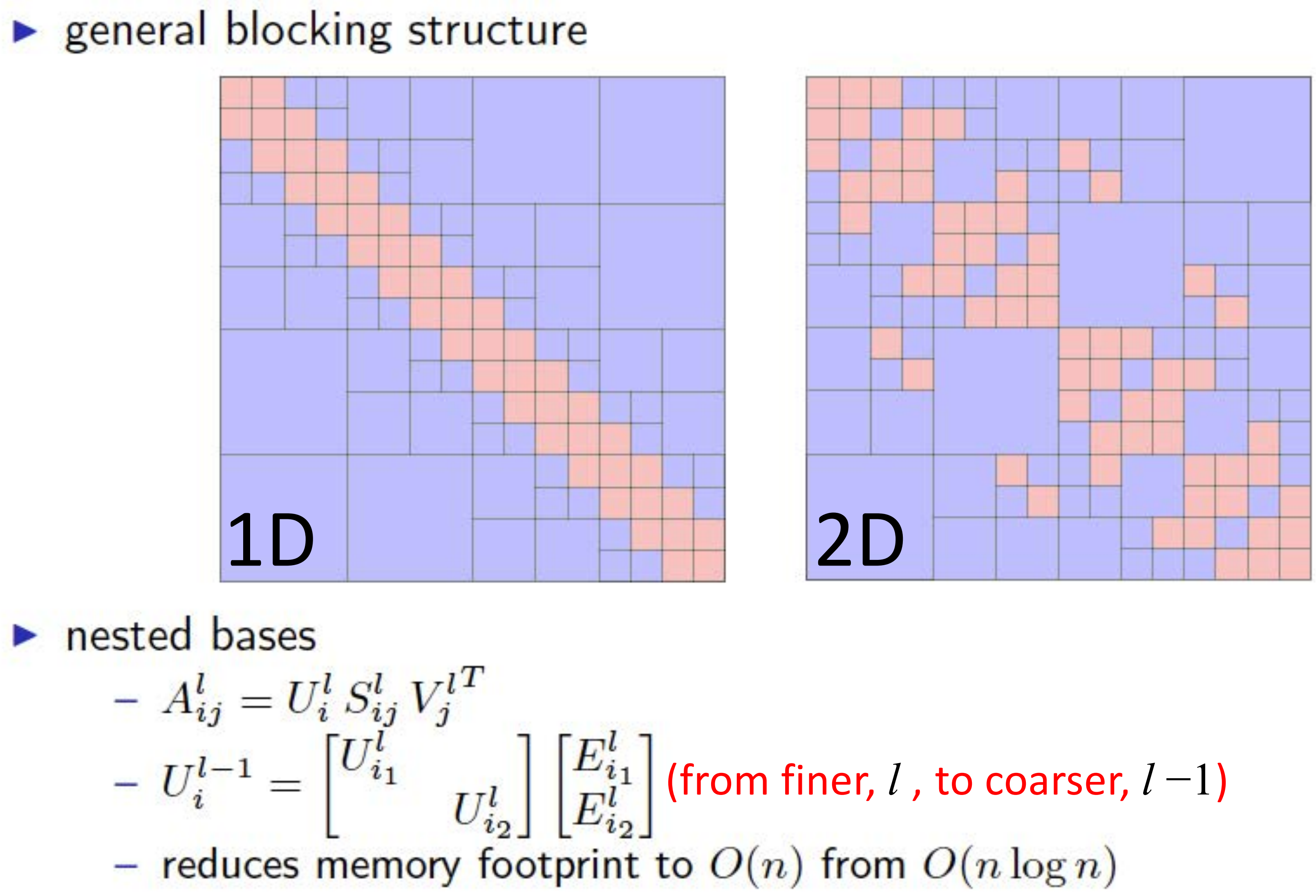
- Compress to req acc
- Is rank acceptably small?
- If not, subdivide block and recur

Take union of leaf blocks



(not an *efficient* algorithm – better in practice to compute tree structure in advance)

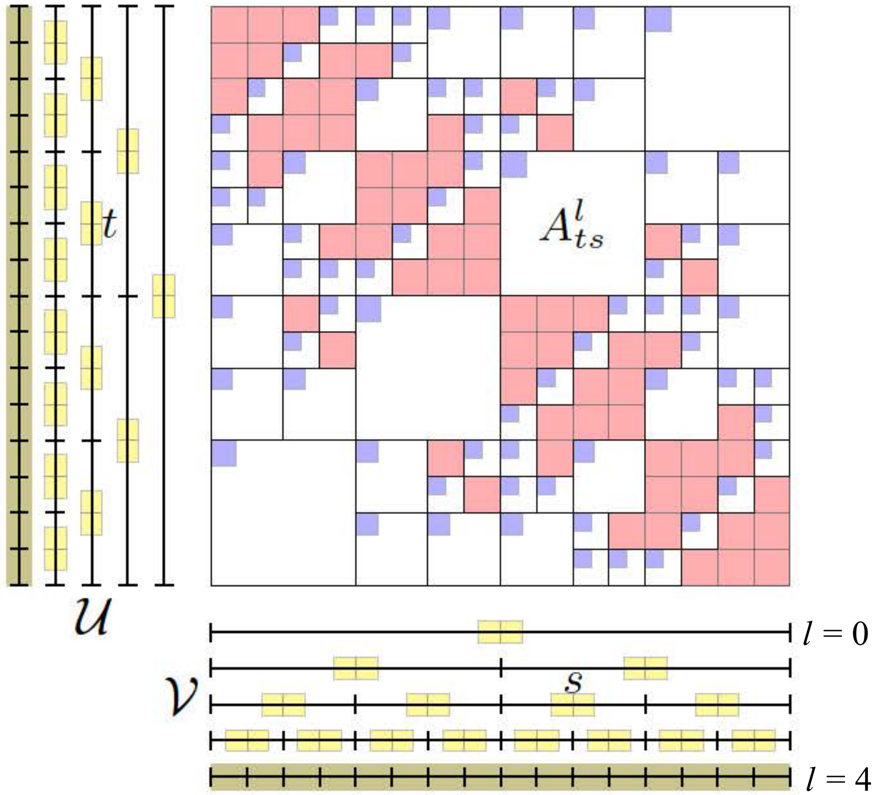
$\mathcal{H}^2$ hierarchical matrix representation



- general blocking structure
- nested bases
  - $A^l_{ij} = U^l_i \, S^l_{ij} \, V^{l\,T}_j$
  - $U^{l-1}_i = \begin{bmatrix} U^l_{i_1} & \\ & U^l_{i_2} \end{bmatrix} \begin{bmatrix} E^l_{i_1} \\ E^l_{i_2} \end{bmatrix}$ (from finer, l , to coarser, $l-1$)
  - reduces memory footprint to $O(n)$ from $O(n \log n)$

Boukaram, Turkiyyah & K., *Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression*, ACM TOMS, 2019

Nested bases of $\mathcal{H}^2$-matrices



$$U_t = \begin{bmatrix} U_{t_1} & \\ & U_{t_2} \end{bmatrix} \begin{bmatrix} E_{t_1} \\ E_{t_2} \end{bmatrix}$$

finest leaf is at 4th level (1/16th)

- Representation is a triplet of trees. Every block is of the form USV^T, and bases are nested.

Ambartsumyan, Boukaram, Bui, Ghattas, K., Stadler, Turkiyyah & Zampini, *Hierarchical Matrix Approximations of Hessians Arising in Inverse Problems Governed by PDEs*, SISC, 2020

Zoology of $\mathcal{H}$-matrices (not comprehensive)

	Flat bases	Nested bases
Weak admissibility	**HODLR** [1]	**HSS** [2]
Strong admissibility	$\mathcal{H}$ [3]	$\mathcal{H}^2$ [4]

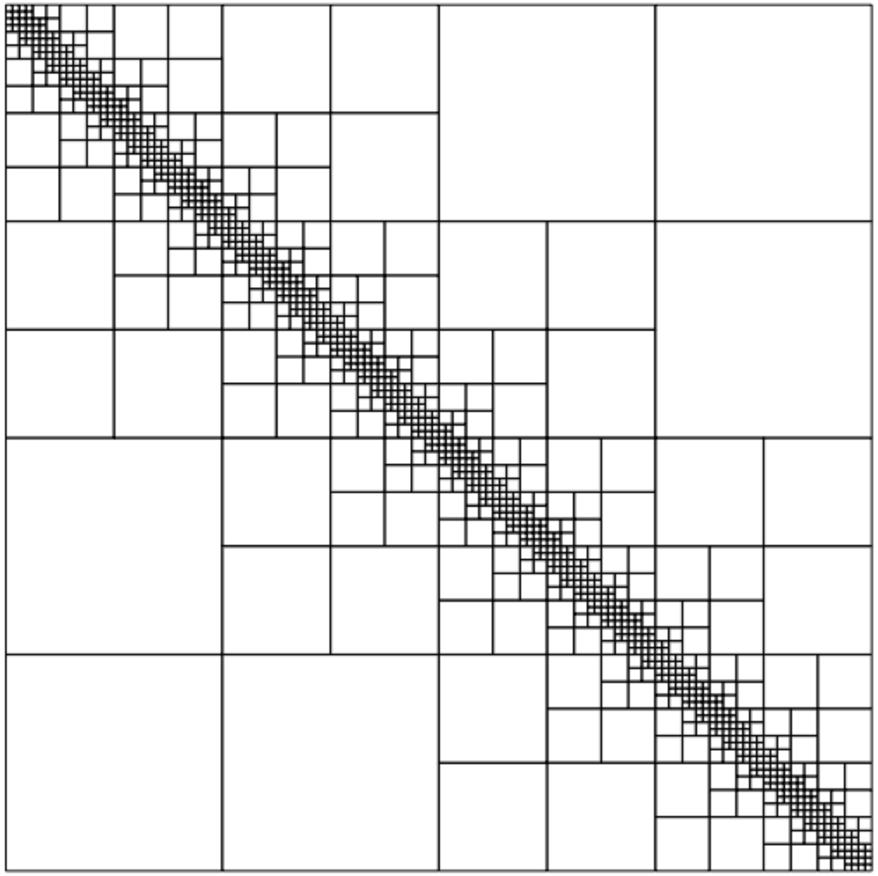
[1] Ambikasaran and Darve, *Journal of Scientific Computing*, 2013

[2] Xia, Chandrasekaran, Gu, and Li, *Numerical Linear Algebra with Applications*, 2010

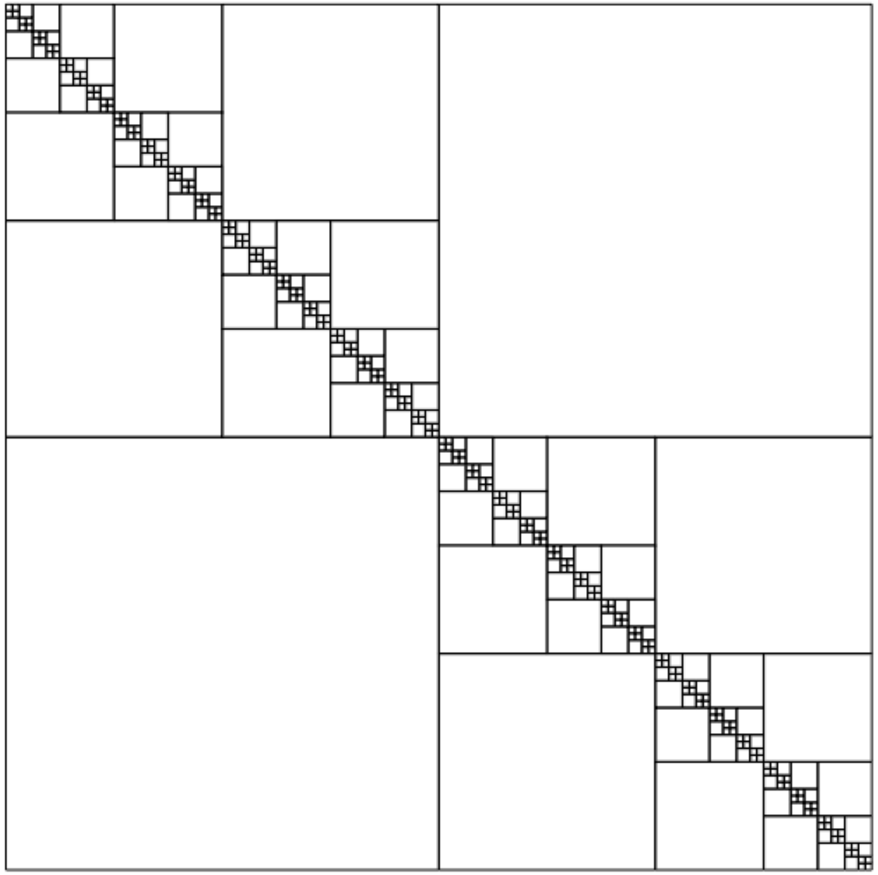
[3] Hackbusch, *Hierarchical Matrices: Algorithms and Analysis,* Springer, 2015

[4] Börm, *Linear Algebra and its Applications*, 2007

Recursive structures of "standard (strong)" vs. "weak" admissibility



strong admissibility in 1D



weak admissibility in 1D

See Bebendorf, Grasedyck, Hackbusch, Kriemann, Khoromskij, *et al.*, 2003-04

Originally motivated by boundary integral operators

Some software leveraging data sparsity

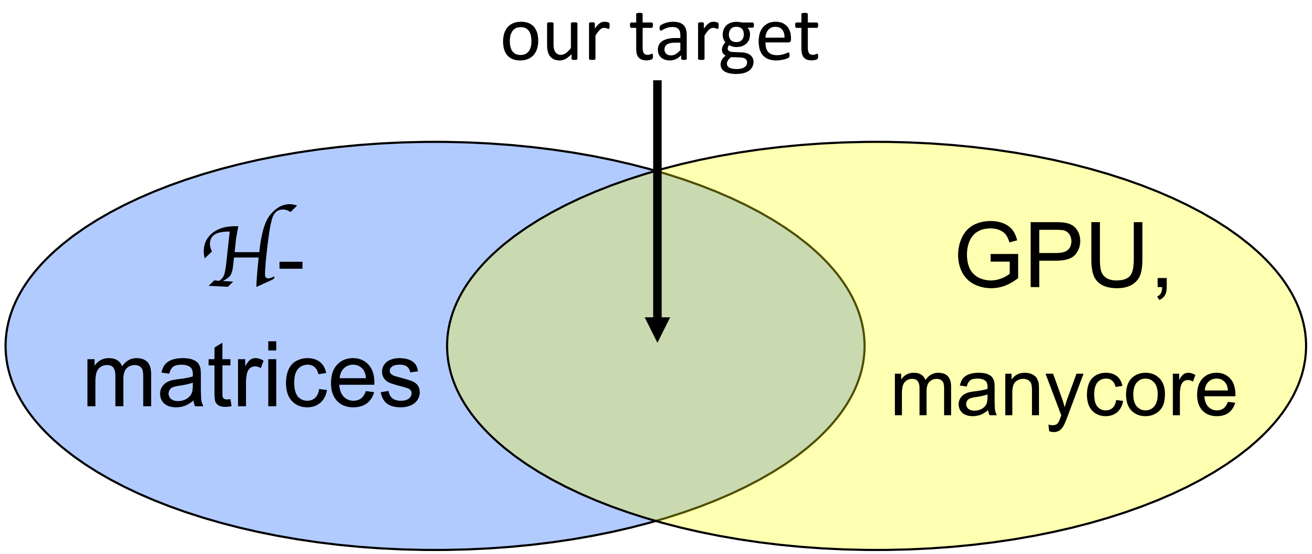
Package	Technique(s)	Format	Author (year, 2005→)
ACR	Cyclic reduction	$\mathcal{H}$	Chavez, Turkiyyah & K. (2017)
AHMED	$\mathcal{H}^{-1}$ and $\mathcal{H}$-LU	$\mathcal{H}$	Bebendorf (2005)
ASKIT	$\mathcal{H}$-LU	HODLR	Yu, March, Xiao & Biros (2016)
BLR PaStiX	Supernodal	BLR	Pichon & Faverge (2017)
CE	$\mathcal{H}^2$-LU	$\mathcal{H}^2$	Sushnikova & Oseledets (2016)
DMHIF	Multifrontal	ID	Li & Ying (2016)
DMHM	Newton-Schulz	$\mathcal{H}$	Li, Poulson & Ying (2014)
GOFMM	Geometry-oblivious Compression, MV	HODLR	Yu, Reiz & Biros (2018)
H2Lib	$\mathcal{H}^{-1}$ and $\mathcal{H}$-LU	$\mathcal{H}^2$	Christophersen & Börm (2015)
H2Pack	Proxy points compression, MV	$\mathcal{H}^2$	Huang, Xing & Chow (2020)
HLib	$\mathcal{H}^{-1}$ and $\mathcal{H}$-LU	$\mathcal{H}$	Börm, Grasedyck & Hackbusch (2004)
HLibPro	$\mathcal{H}^{-1}$ and $\mathcal{H}$-LU	$\mathcal{H}$	Kriemann & Hackbusch (2013)
hm-toolbox	numerous	HSS, HODLR	Massei, Robol & Kressner (2020)
LoRaSp	$\mathcal{H}^2$-LU	$\mathcal{H}^2$	Pouransari, Coulier & Darve (2013)
MF-HODLR	Multifrontal	HODLR	Aminfar & Darve (2016)
MUMPS-BLR	Multifrontal	BLR	Amestoy & Mary (2016)
Structured CHOLMOD	Supernodal	BLR	Chadwick & Bindel (2015)
STRUMPACK	$\mathcal{H}$-LU, Preconditioning	HSS/BLR/HODLR	Ghysels, Li, Liu & Claus (2020)

Lecture 2

Hierarchical algorithms and extreme scale

Must address the *tension* between

- **highly uniform vector, matrix, and general SIMT operations – *prefer regularity and predictability***
- **hierarchical algorithms with tree-like data structures and scale recurrence – *possess irregularity and adaptability***



No miracles will appear in this talk ☹

Introducing H2Opus

an $\mathcal{H}^2$ matrix computation library

High-level operations

- Hierarchical matrix-matrix multiplication (and re-compression)
- Generation of approximate matrix inverse, via Newton-Schulz iteration
- Formation of Schur complements, via randomized sampling

Core utilities

- H-matrix construction from matrix-vector sampling (HARA)
- Low-rank updates (HLRU)
- Matrix compression (Hcompress)
- Basis orthogonalization (Horthog)
- Matrix-vector multiplication (HGEMV)
- Matrix construction from kernel function (Hconstruct)
- Generation of matrix structure from admissibility condition

Batched LA for GPUs

- MAGMA
- cuBLAS, KBLAS

Manycore LA for CPUs

- OpenMP
- Intel MKL

H2Opus features (1)

- Performance-oriented library for $\mathcal{H}^2$ computations
- Runs on GPUs
  - includes batched QR, RRQR, and SVD dense linear algebra routines
- Can also run on CPU-only machines
  - using shared-memory parallelism (via OpenMP and MKL)
- Can represent weak and strong admissiblity matrix structures
  - using geometric KD-tree partitioning and (user-controllable) admissibility condition
- Constructs kernel matrices from user-specified kernel function
  - for example, covariances matrices from Matérn kernels
- Optimized matrix-vector multiplication (“algebraic FMM”)
  - single-vector mult achieving ~80% of peak bandwidth on GPUs
  - multiple vector mults leverage fast GEMMs on GPUs

H2Opus features (2)

- Generation of Hierarchical Orthogonal Bases
  - via batched QR operations in upsweep through bases trees
  - followed by expressing matrix blocks in the new orthogonal Q basis
- Hierarchical Matrix Compression
  - ranks increase during algebraic manipulation
  - perform RRQR / SVD operations followed by truncation of bases to desired accuracy
  - followed by expressing matrix blocks in the new bases
- Low Rank Updates
  - allows globally low rank updates to be compressed into a hierarchical matrix
  - “local” updates (that only affect a portion of the matrix) are also supported

H2Opus features (3)

- Matrix generation from matrix-vector sampling
  - generates hierarchical matrix from a “black-box” operator accessible via mat-vec products
  - generalizes highly successful randomized algorithms (for generating globally low-rank approximations of large dense matrices) to the hierarchical setting
  - formulated as a sequence of low-rank updates at the various matrix levels
- Matrix-matrix multiplication operation
  - takes advantage of the high-performance of multi-vector sampling
- Approximate inverse computation
  - via Newton-Schulz iteration and its higher-order variants
  - can converge very quickly when warm-started from a nearby solution
    - Hessian of previous iteration in optimization
    - Jacobian of previous iteration in nonlinear solver
- Schur complements
  - and other algebraic expressions that can be sampled

On quality software

“Quality software *renormalizes* the difficulty of doing computation.”

Peter Lax, 2005 Abel Prize Winner

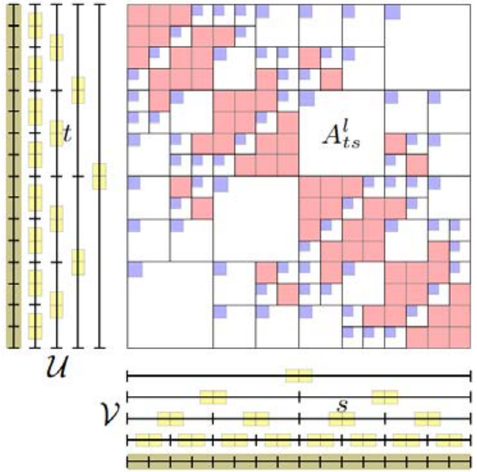
Hierarchical MatVec by tree-traversal

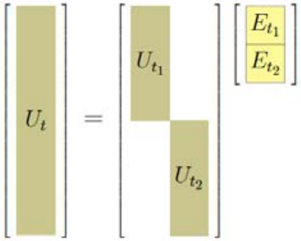
- Representation is a triplet of trees. Every block is of the form USV^T, and bases are nested.
- Informally the matrix is:

$$A_{\mathcal{H}^2} = D + \mathcal{U} \cdot \mathcal{S} \cdot \mathcal{V}^T$$

- We can perform fast, asymptotically optimal, matrix-vector prod.

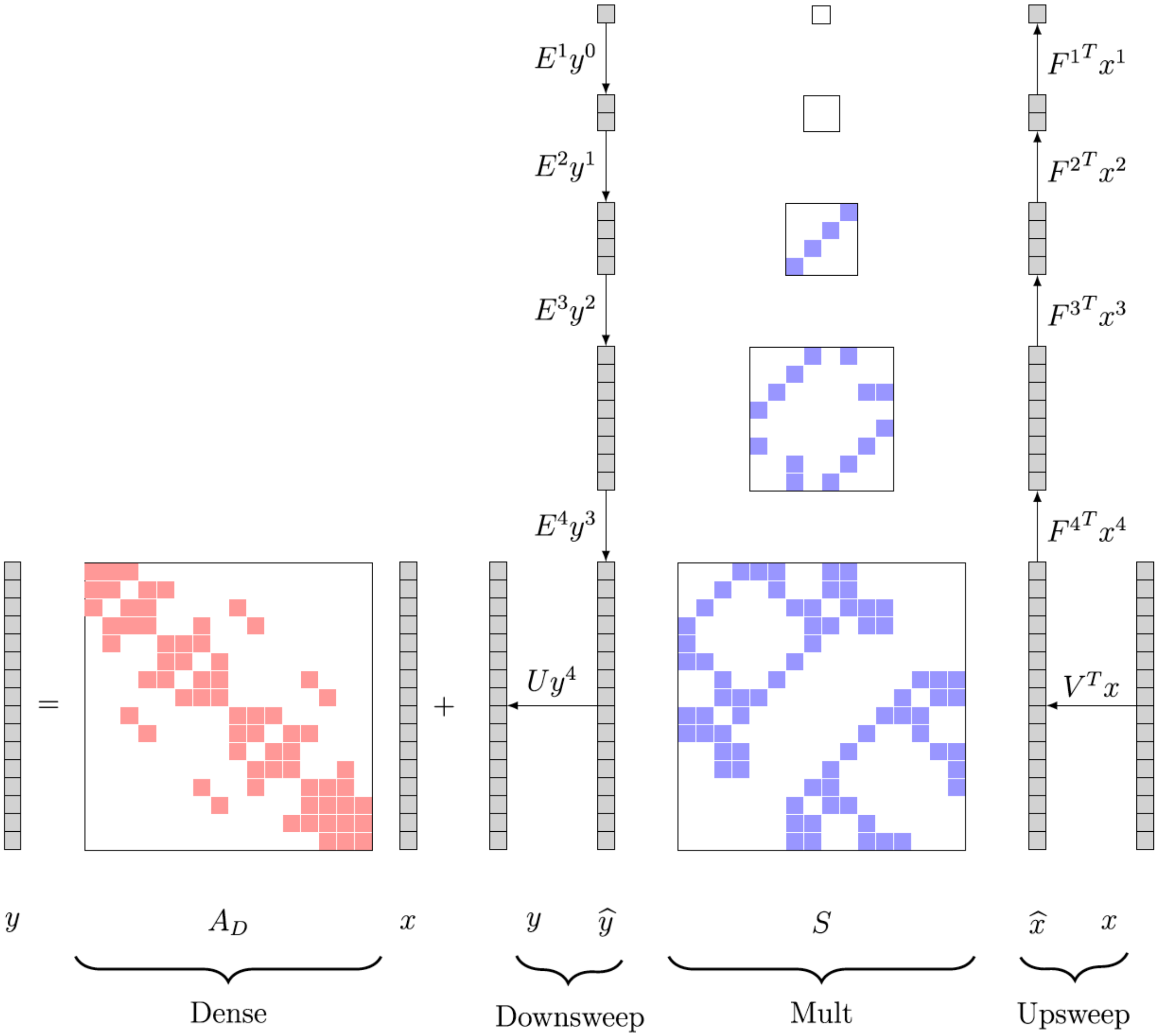
$$A_{\mathcal{H}^2} x = Dx + \mathcal{U} \cdot \mathcal{S} \cdot \mathcal{V}^T \cdot x = Dx + \mathcal{U} \cdot (\mathcal{S} \cdot (\mathcal{V}^T \cdot x)))$$





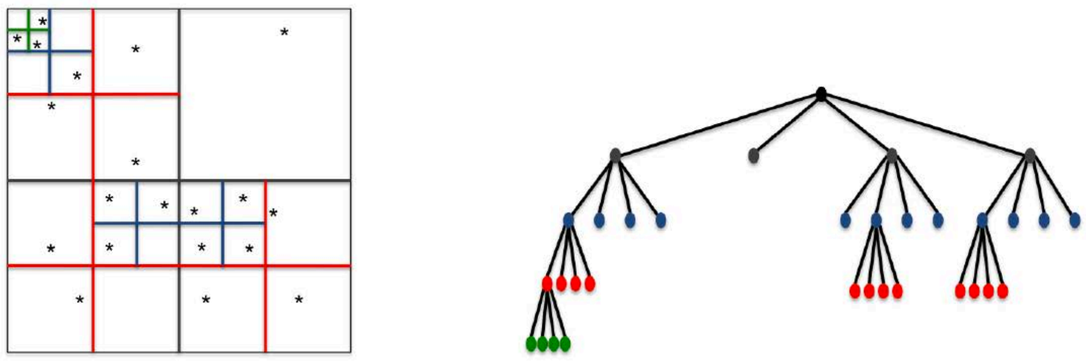
Ambartsumyan, Boukaram, Bui, Ghattas, K., Stadler, Turkiyyah & Zampini, *Hierarchical Matrix Approximations of Hessians Arising in Inverse Problems Governed by PDEs*, SISC, 2020

Tree-based mat-vec



Boukaram, Turkiyyah & K., *Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression*, ACM TOMS, 2019

Where have you seen this before?



(a) 2-D view

(b) Tree view

Figure 1: Hierarchical decomposition

FMM!

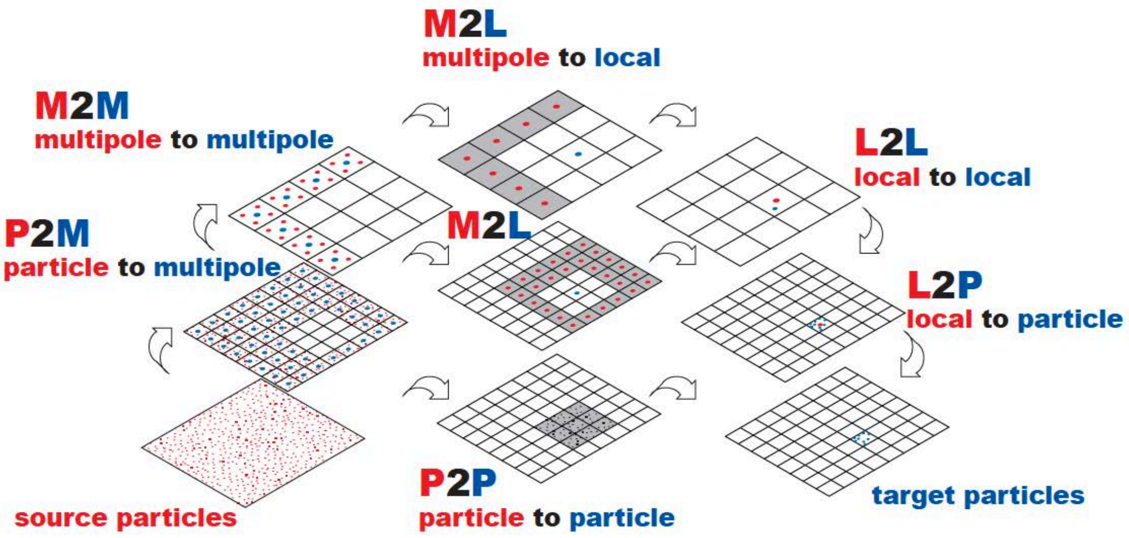
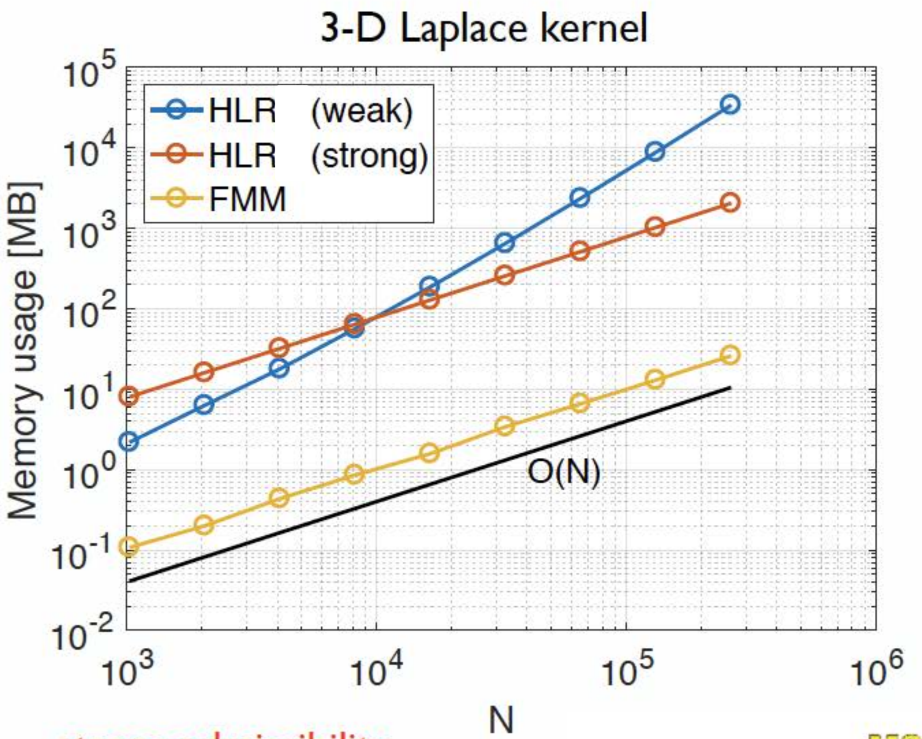


Figure 2: Data-flow of FMM calculation. Data dependency is between red and blue points.

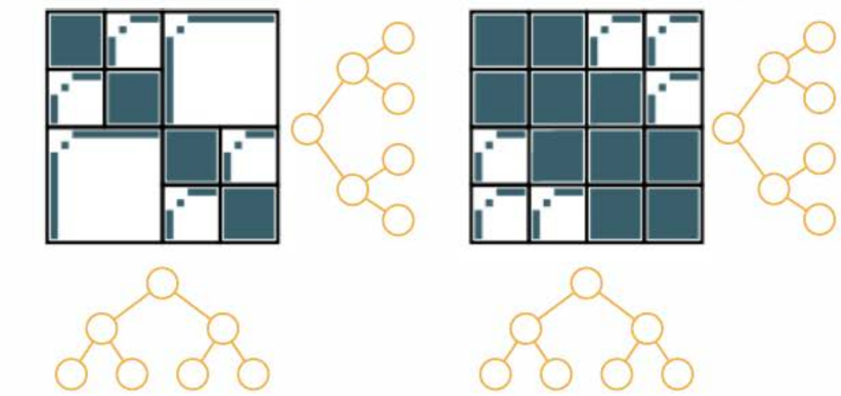
Ibeid, Yokota, and K., *A Performance Model for the Communication in Fast Multipole Methods on HPC Platforms,* IJHPCA (2016)

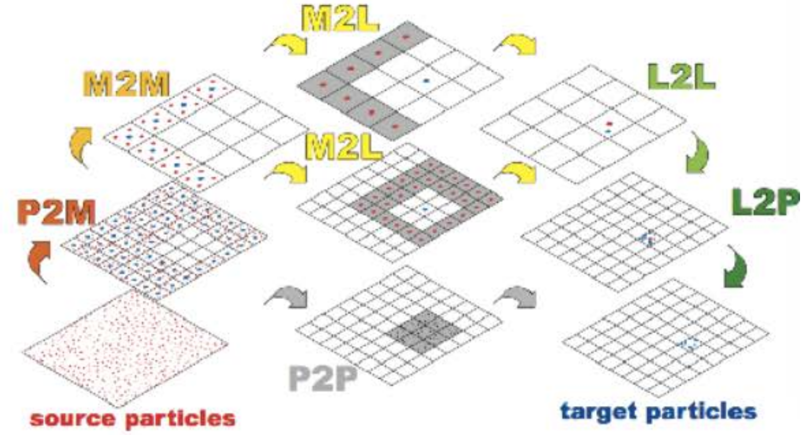
Memory complexity of FMM vs HLR



weak admissibility

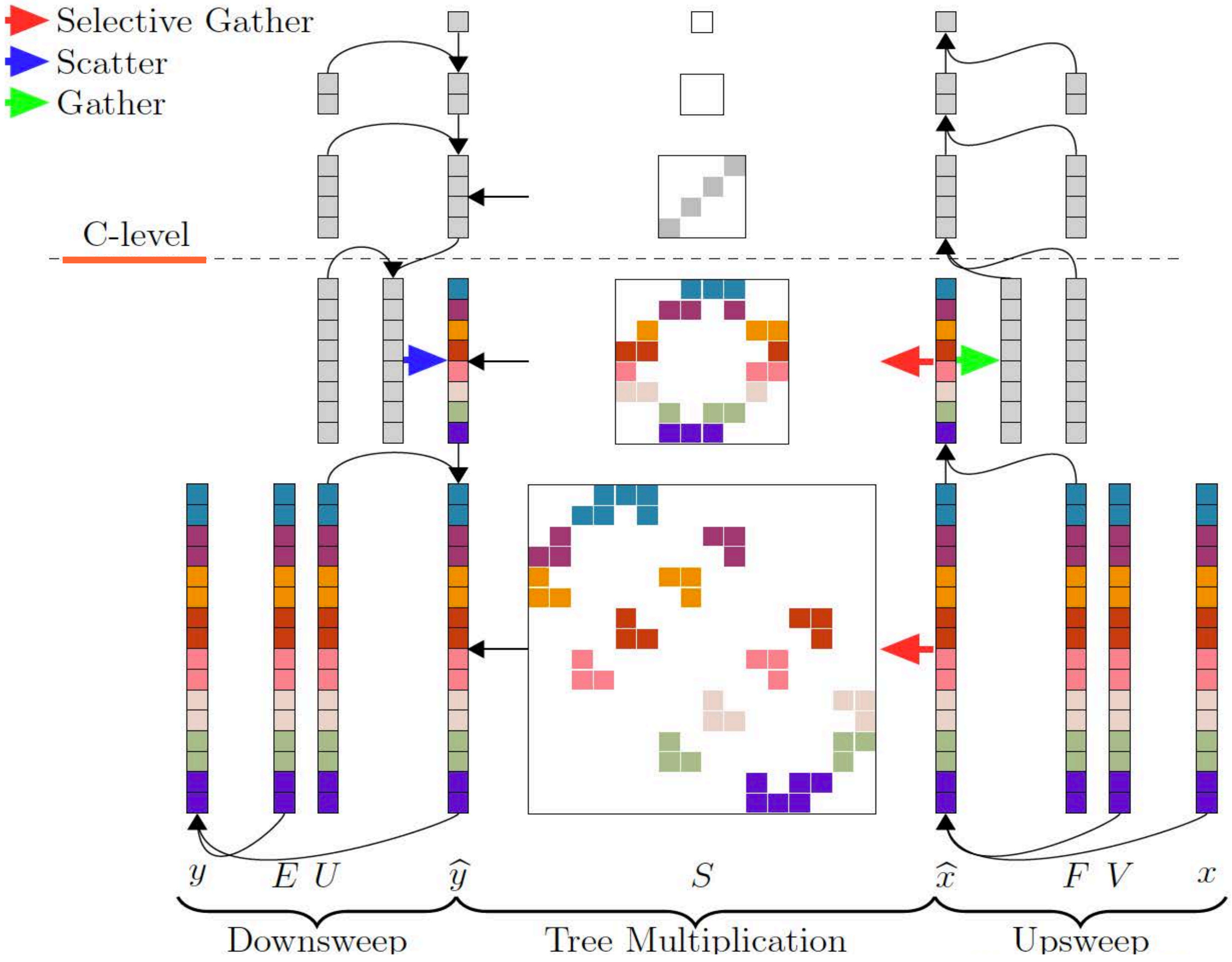
strong admissibility





c/o R. Yokota (Tokyo Tech/KAUST)

Distributed memory mat-vec



> "Hierarchical algorithms do not repeat themselves, but they do rhyme."
>
> (with apologies to Mark Twain)

Illustration on 8 nodes (one color per node)

- **a combination of tricks from FMM and MG ☺**

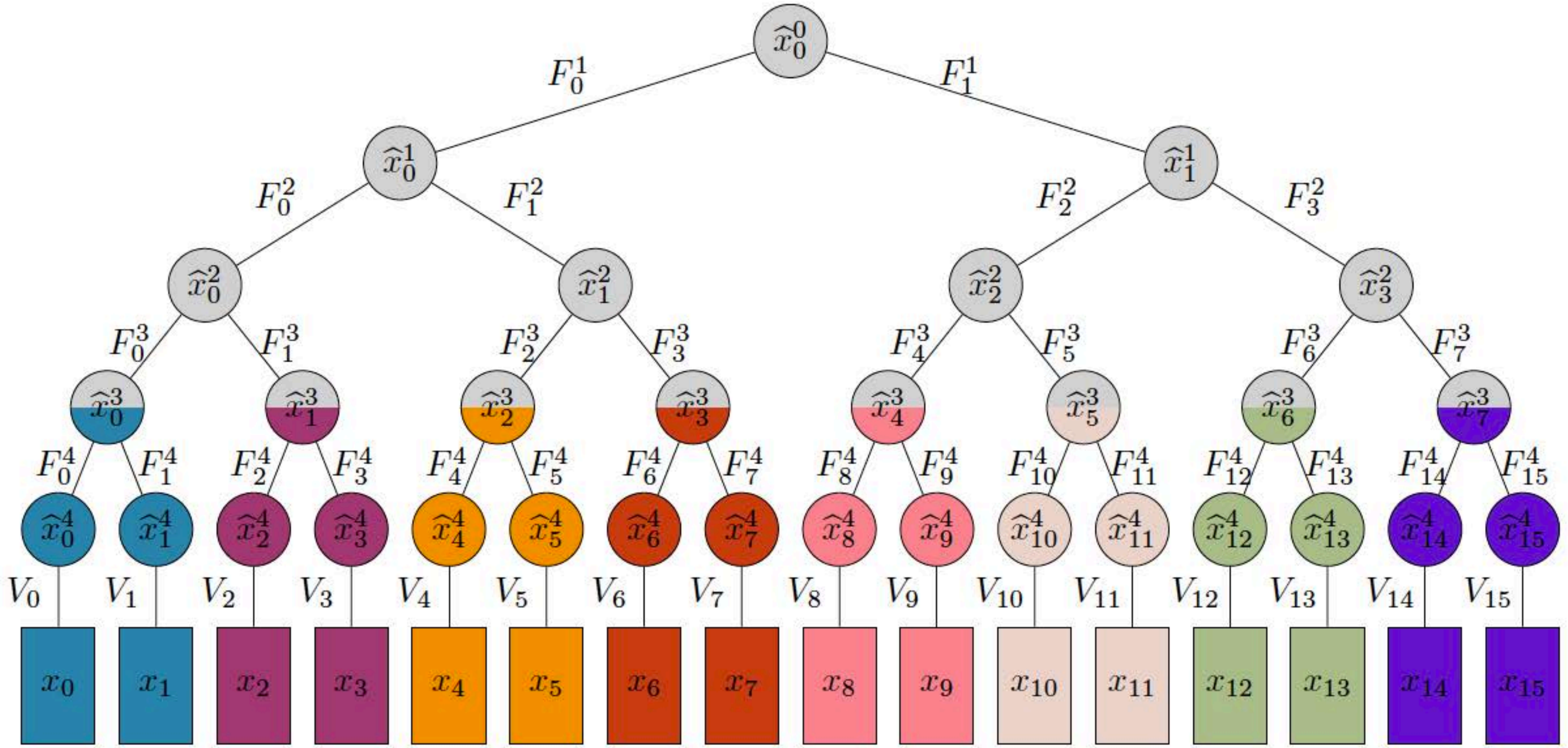
Boukaram, *Hierarchical Matrix Operations on GPUs*, PhD thesis, 2020 [online @ library.kaust.edu.sa]

Applications caveat

- HLR and TLR methods are being applied beyond the rigorous guidance we expect of more traditional linear algebraic methods, as engineered into software
  - e.g., how to choose blockwise tolerances when fitting ranks to satisfy global tolerances for various uses?
- Apologies in advance for examples in this presentation
- Not unlike some compromises that are accepted to increase opportunities for parallelism in full-rank methods
  - e.g., limiting domain of pivoting
- Good news: interesting opportunities for theorists

Distributed upsweep

- Root of basis tree is stored on one GPU
- Branches below a C-level are distributed
- Upsweep proceeds concurrently on the branches
- A gather allows the upsweep to then continue on the root tree



Boukaram, *Hierarchical Matrix Operations on GPUs*, PhD thesis, 2020 [online @ library.kaust.edu.sa]

Lecture 2

Large dense symmetric systems arise as covariance matrices in spatial statistics

- Climate and weather applications have many measurements located regularly or irregularly in a region; prediction is needed at other locations
- Modeled as realization of Gaussian or Matérn spatial random field, with parameters to be fit
- Leads to evaluating the log-likelihood function involving a large dense (but data sparse) covariance

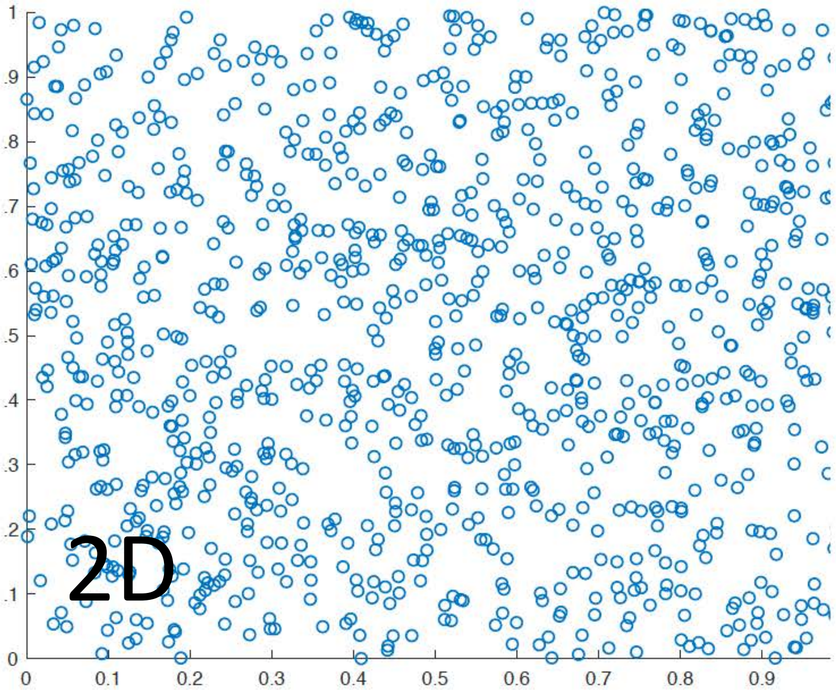
$$\ell(\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{Z}^T \underset{\text{inverse}}{\Sigma^{-1}}(\boldsymbol{\theta})\mathbf{Z} - \frac{1}{2}\log \underset{\text{determinant}}{|\Sigma(\boldsymbol{\theta})|}$$

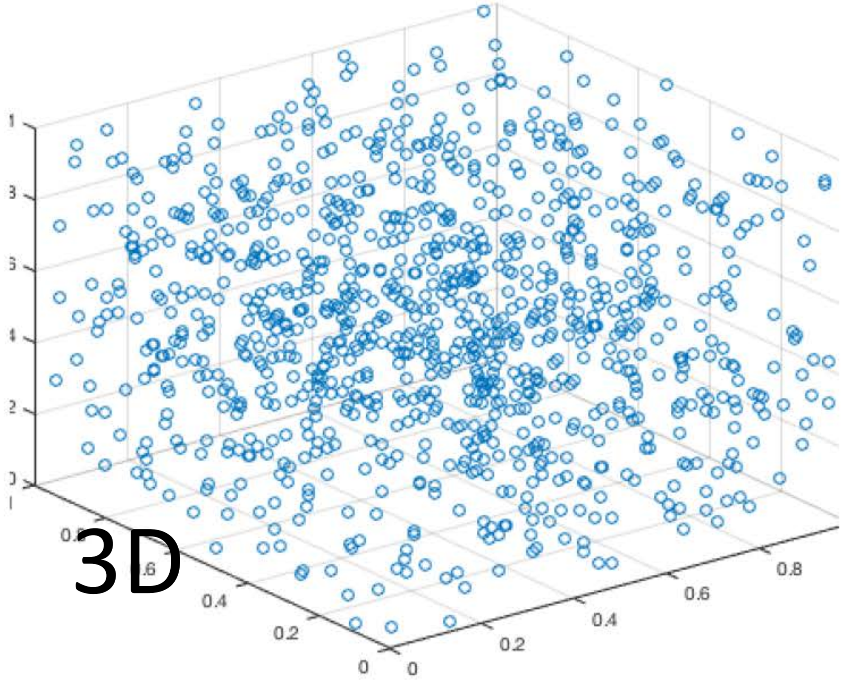
- Solve Σ^{-1} and determinant | Σ | with covariance matrix performed with HLR

Geospatial statistics applications

Synthetic test matrix: random coordinate generation within the unit square or unit cube with Matérn kernel decay, each pair of points connected by

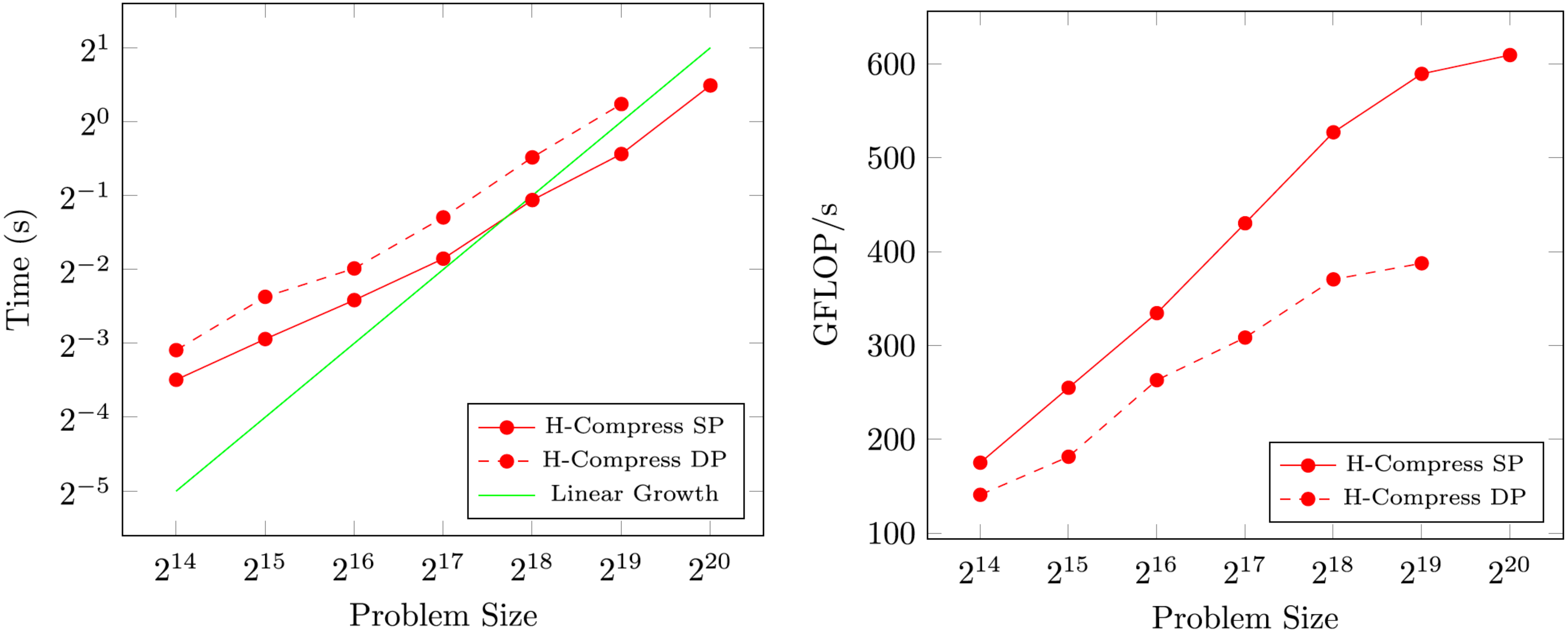
- **linear exp to square exp decay,** $a_{ij} \sim \exp(-c|x_i - x_j|^p)$, $p = 1,2$





Compression

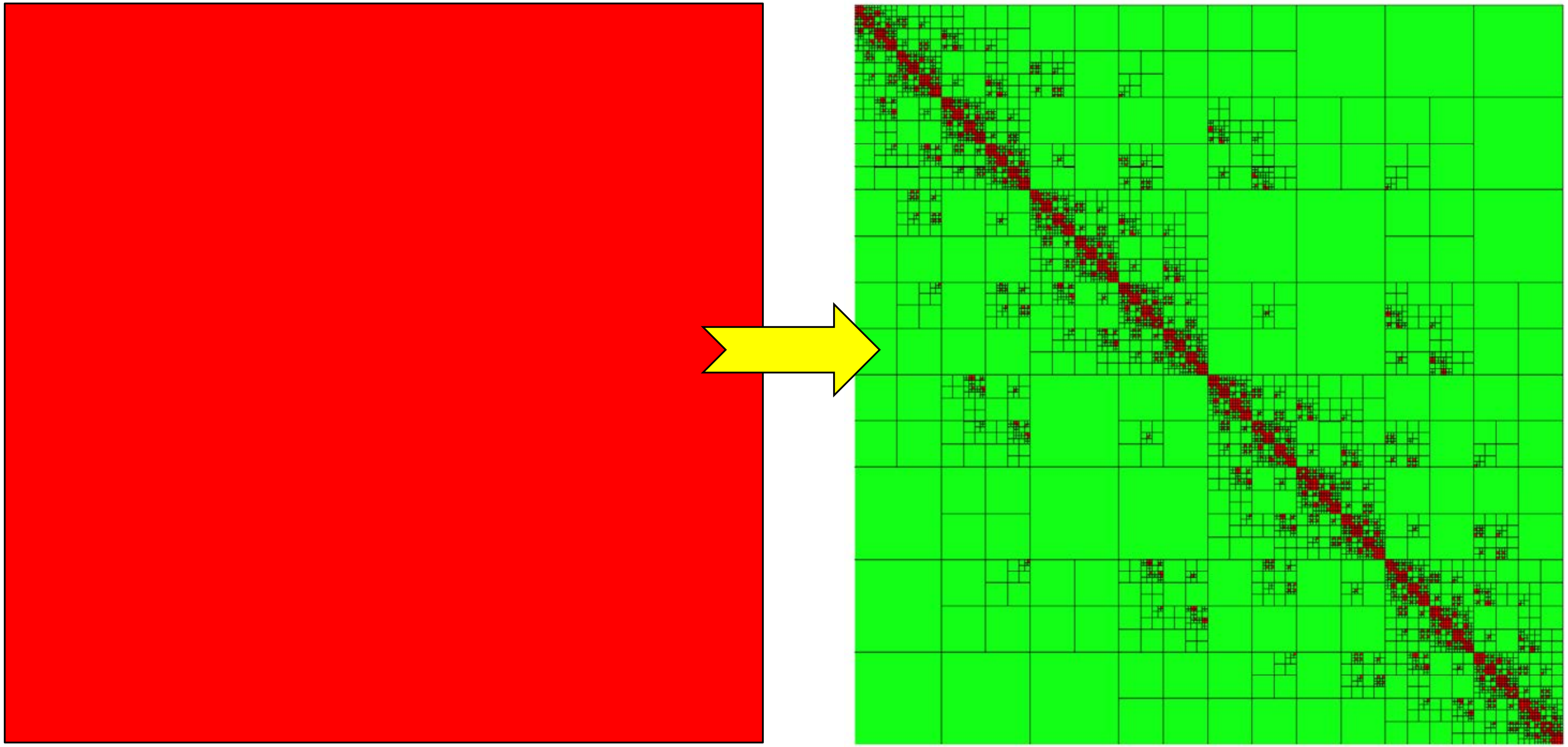
NB: log scale



- 3D covariance matrices from spatial statistics
- running on P100 GPU
- accuracy 10^{-3} computed as $\|Ax - A^{\mathcal{H}}x\| / \|Ax\|$
- leaf size $m = 64$

Boukaram, Turkiyyah & K., *Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression*, ACM TOMS, 2019

Can we avoid forming and storing original?



Original operator, size $O(n^2)$

Compressed, size $O(kn)$

Yes, we can form and compress “on the fly”

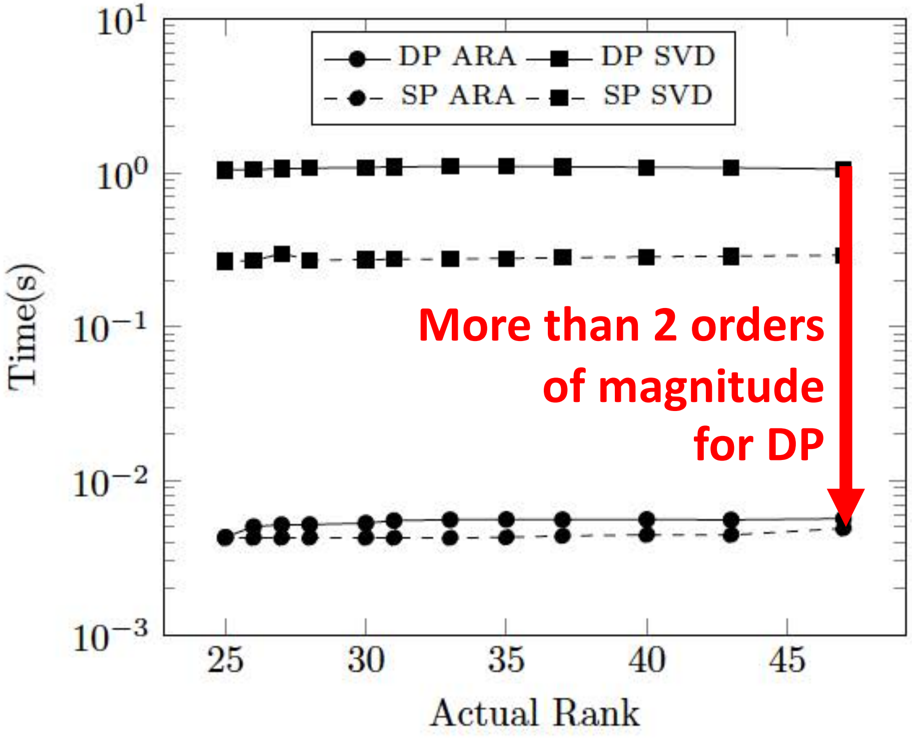
- block by block from dense
- from functional matvecs

Adaptive Randomized Approximations (ARA)

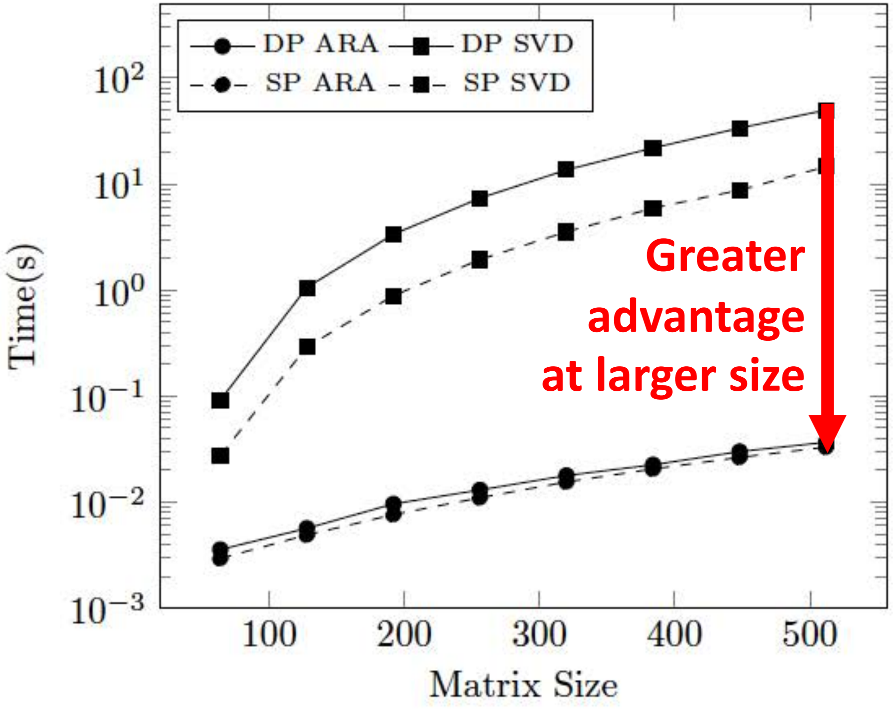
- allow fast construction of low rank approximations
- rely on sampling the matrix through mat-vec operations
  - can be done on multiple vectors simultaneously
  - for increased arithmetic intensity
- applicable to dense matrices and, with hierarchical extensions, to $\mathcal{H}$ matrices
- particularly effective on GPUs
  - can leverage high-performing GEMM routines

Comparison with Jacobi (Givens) SVD

NB: log scale



More than 2 orders of magnitude for DP



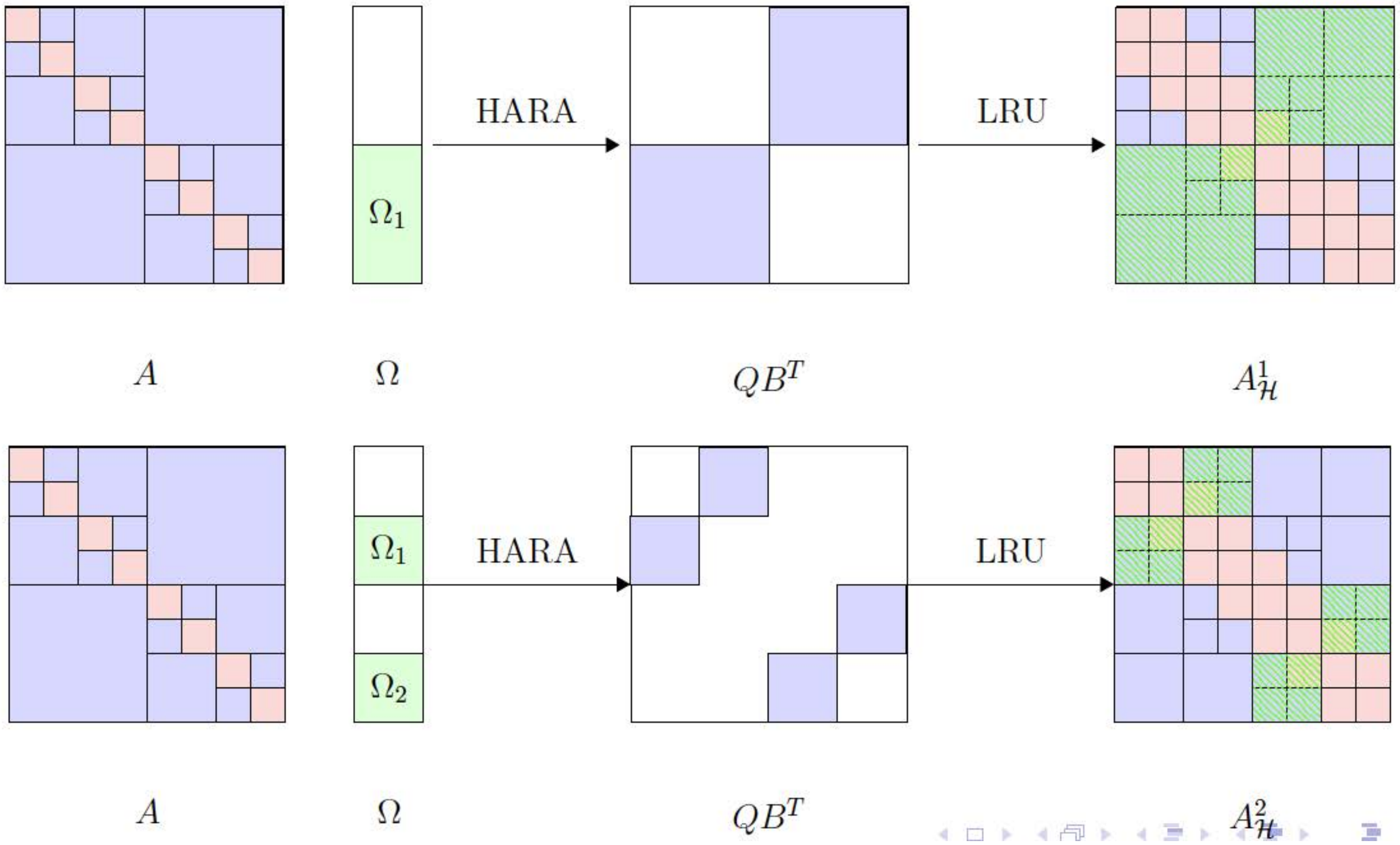
Greater advantage at larger size

- batch of 1000 matrices in single and double precision
- varying rank for fixed size (128)
- varying size for fixed rank (47)

Boukaram, Turkiyyah & Keyes, *Randomized GPU Algorithms for the Construction of Hierarchical Matrices from MatVec Operations*, SISC, 2019

Hierarchical Adaptive Randomized Approximation (HARA)

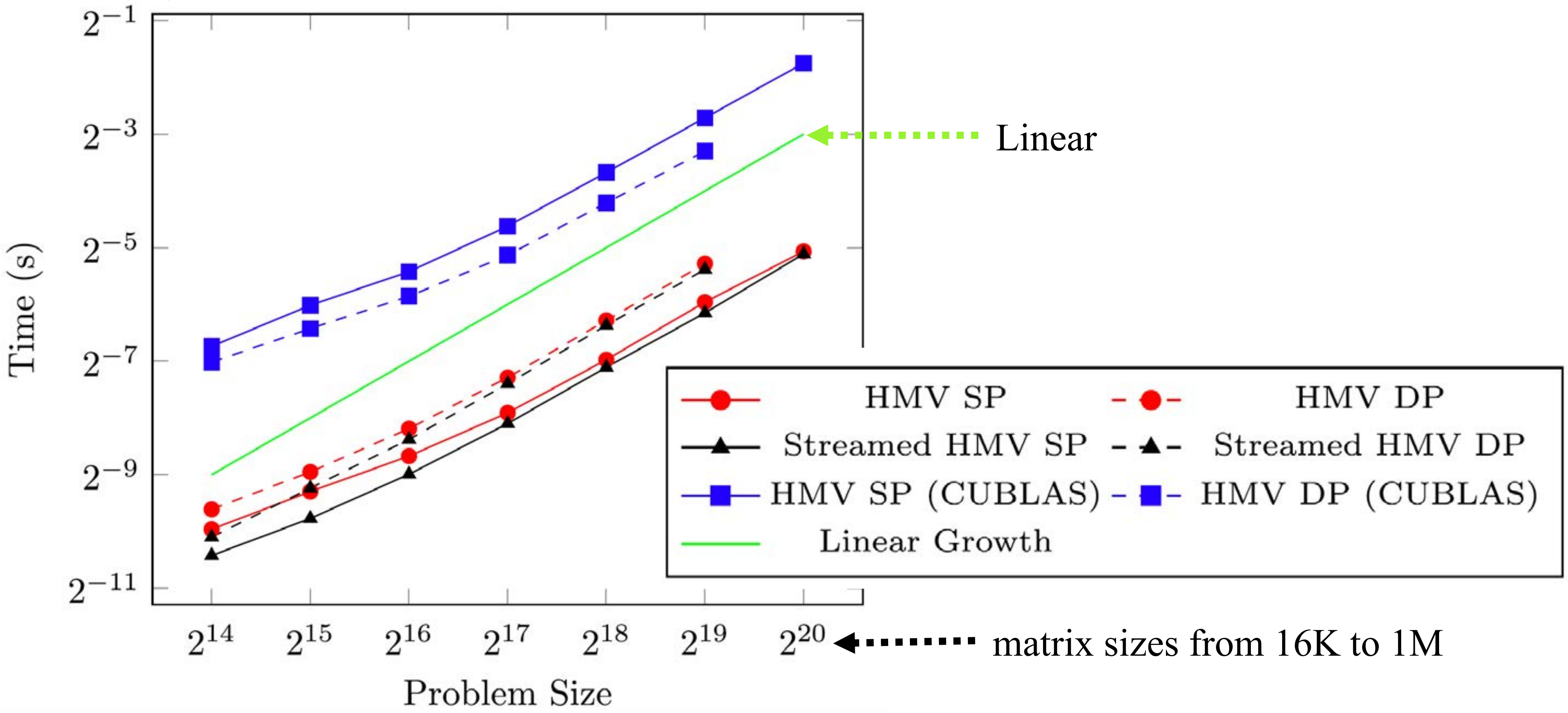
- samples blocks of the matrix and accumulates the local low rank updates into an $\mathcal{H}$-matrix that is recompressed



Boukaram, Turkiyyah & K., *Randomized GPU Algorithms for the Construction of Hierarchical Matrices from MatVec Operations*, SISC, 2019

Hierarchical MatVec execution time

NB: log scale



Linear

matrix sizes from 16K to 1M

- 3D covariance matrices from spatial statistics
- running on P100 GPU
- accuracy 10^{-3} computed as $\|Ax - A^{\mathcal{H}}x\| / \|Ax\|$
- leaf size $m = 64$

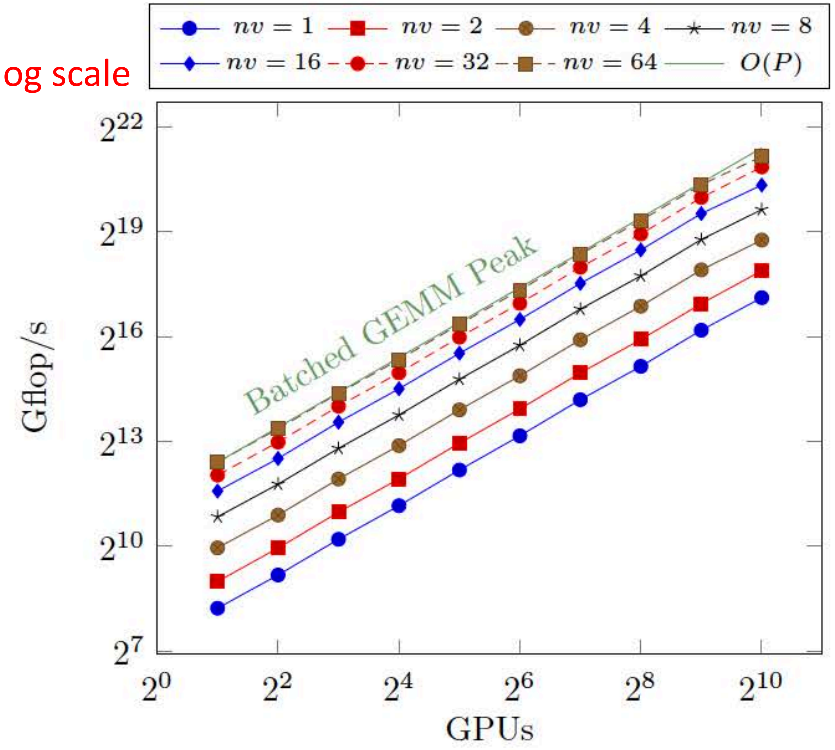
Boukaram, Turkiyyah & K., *Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression*, ACM TOMS (2019)

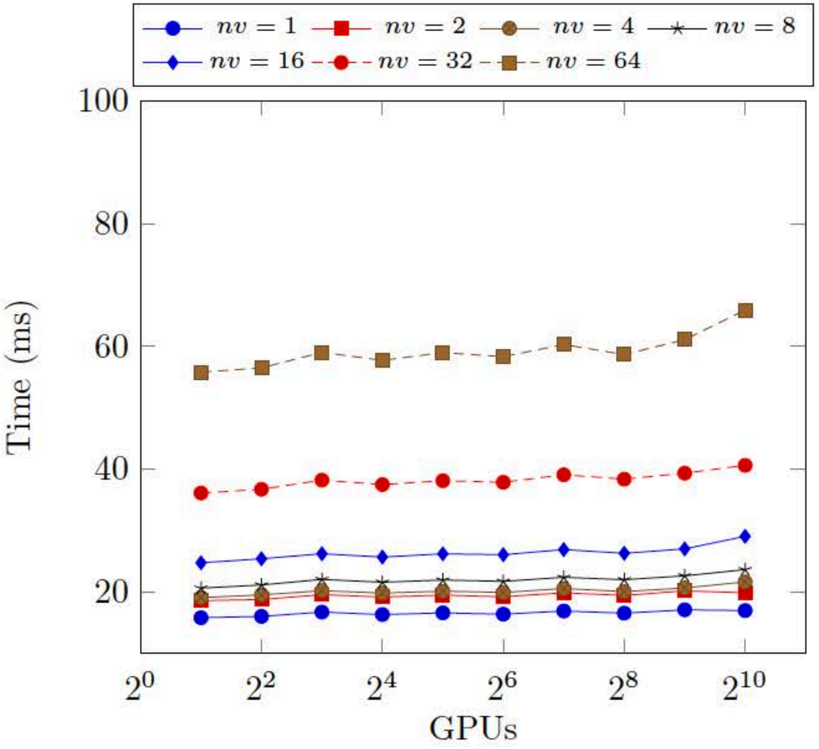
HGEMV on Summit (1024 GPUs)

(the distributed release of H2Opus will follow at github shortly)

- Spatial statistics application in 2D
- $N = 2^{19}$ points per GPU
- Approximated to an accuracy $\epsilon = 10^{-7}$

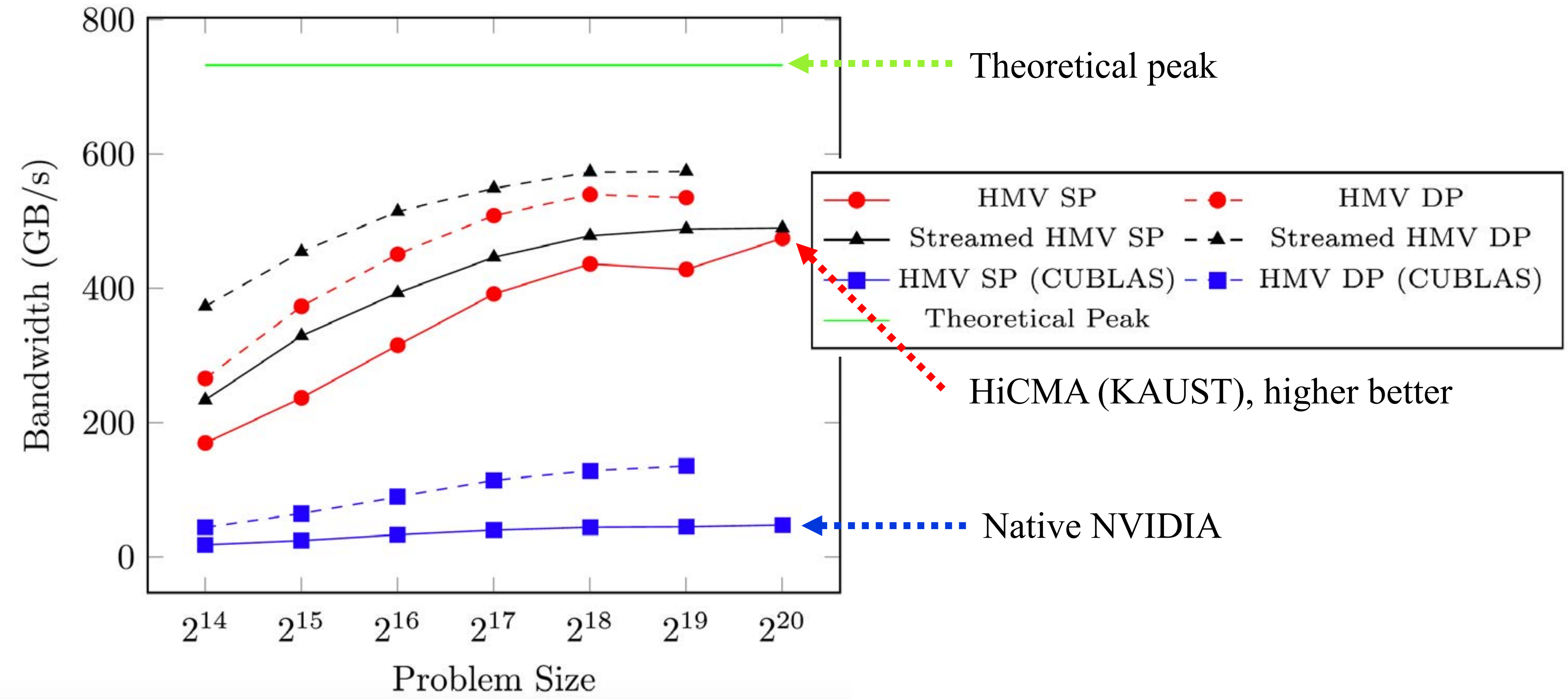
NB: log scale





Boukaram, *Hierarchical Matrix Operations on GPUs*, PhD thesis, 2020 [online @ library.kaust.edu.sa]

Hierarchical MatVec bandwidth



Theoretical peak

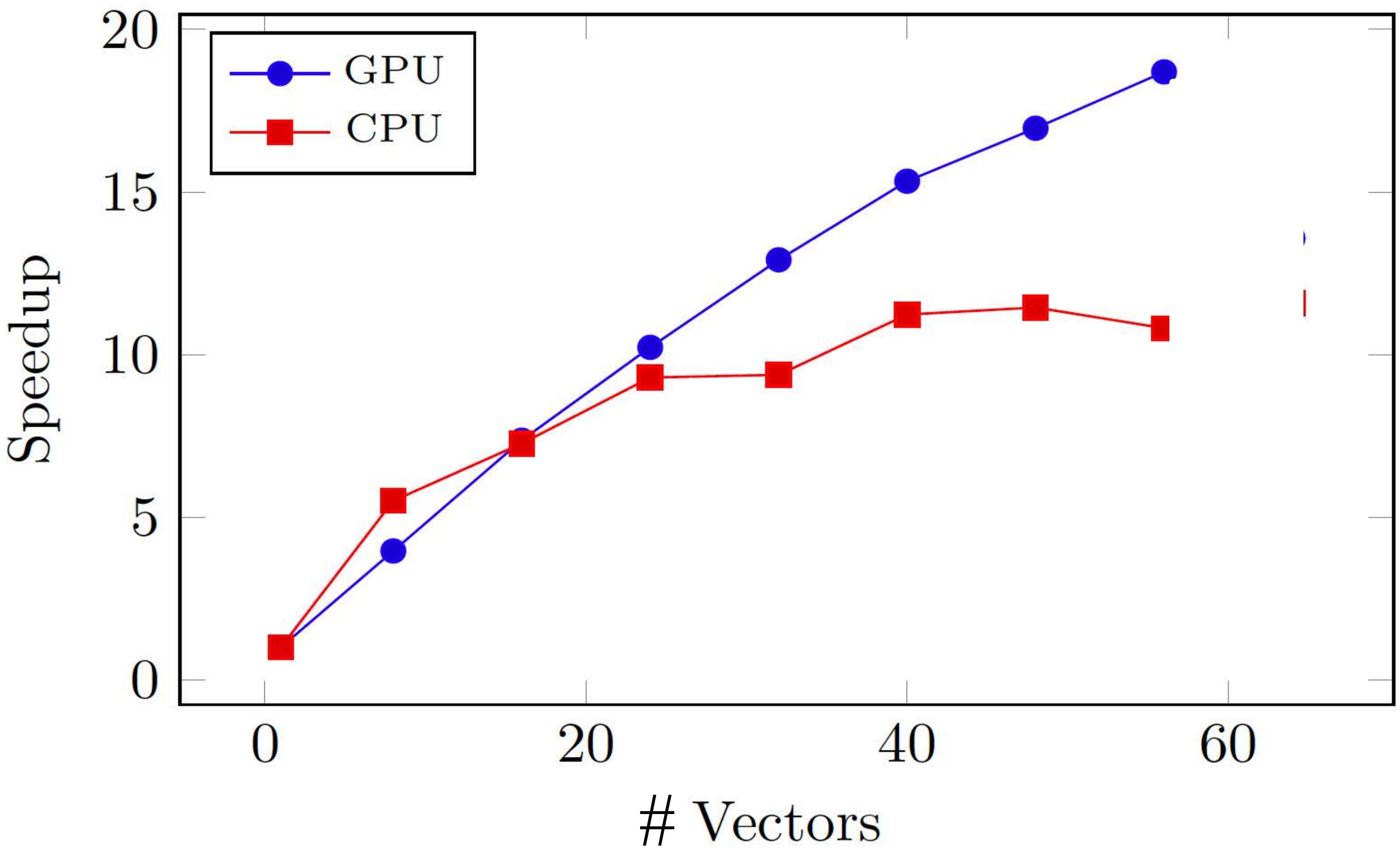
HiCMA (KAUST), higher better

Native NVIDIA

- 3D covariance matrices from spatial statistics
- running on P100 GPU
- accuracy 10^{-3} computed as $\|Ax - A^{\mathcal{H}}x\| / \|Ax\|$
- leaf size $m = 64$

Boukaram, Turkiyyah & K., *Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression*, ACM TOMS, 2019

MatVecs with multiple vectors



- Speedup over single-vector MatVec
- Single precision, size 2^{19} (524,288)
- GPU version obtains 90% of GEMM

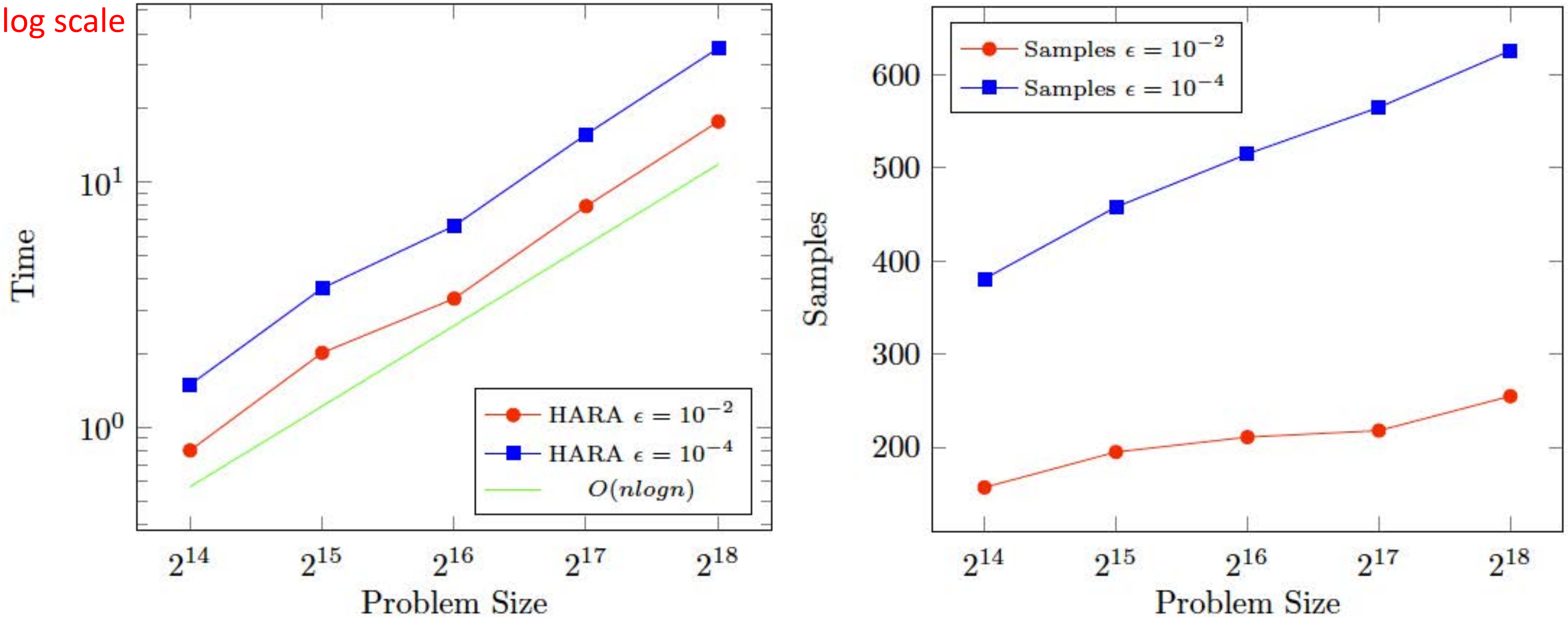
K., Ltaief & Turkiyyah, *Hierarchical Algorithms on Hierarchical Architectures*, Phil Trans Roy Soc Ser A, 2020

$\mathcal{H}$ matrix-$\mathcal{H}$ matrix multiplication

- can be cast as the problem of constructing an $\mathcal{H}$-matrix from matvec operations
- we can do HGEMV operations efficiently on GPUs
  - HGEMV on multiple vectors is even more efficient
- HARA construction of product also performed efficiently on the GPU

Fast matvecs ⇒ fast approx inversions with Newton-Schulz

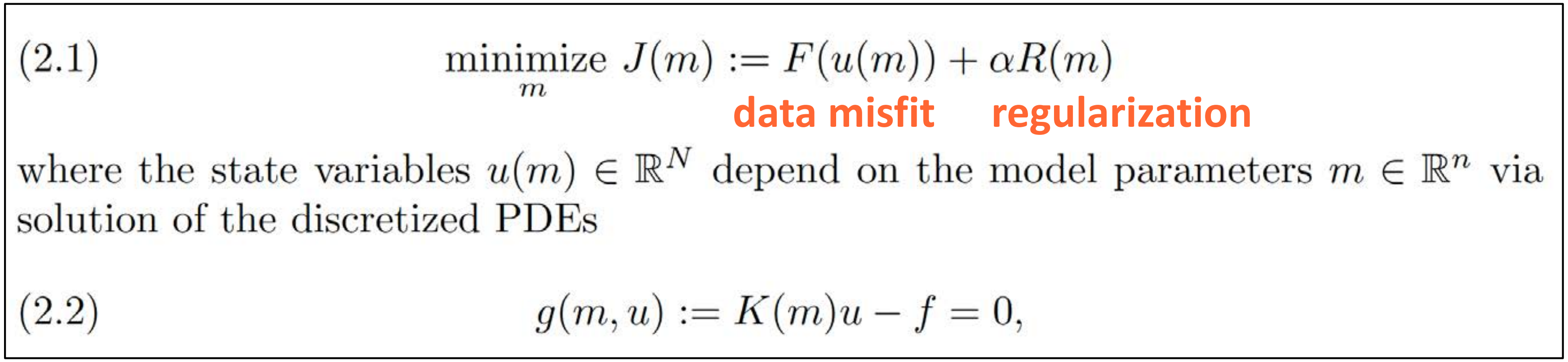
NB: log scale



Boukaram, Turkiyyah & K., *Randomized GPU Algorithms for the Construction of Hierarchical Matrices from MatVec Operations*, SISC, 2019

A prime target for HLR linear algebra: PDE-constrained optimization

- **Dense Hessian matrices arise from**
  - **second variation of data misfit functional in deterministic inverse problems**



$$\underset{m}{\text{minimize}}\ J(m) := \underbrace{F(u(m))}_{\text{data misfit}} + \underbrace{\alpha R(m)}_{\text{regularization}} \tag{2.1}$$

where the state variables $u(m) \in \mathbb{R}^N$ depend on the model parameters $m \in \mathbb{R}^n$ via solution of the discretized PDEs

$$g(m, u) := K(m)u - f = 0, \tag{2.2}$$

  - **covariance in Bayesian inversion for quantifying uncertainties in stochastic inverse problems**

Prime target for HLR linear algebra: PDE-constrained optimization

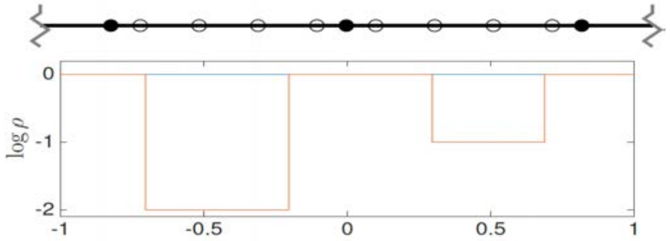
- Historical choices
  - abandon prospects for dimension-independent convergence rates in inverse problems by avoiding Hessians
    - a path to nowhere, given future problem scales
  - use *globally* low-rank Hessian approximation
    - valid for limited information …
    - … not where inverse problems want to be, with their many sources and many sensors
- Hierarchical low rank valid in informed regime

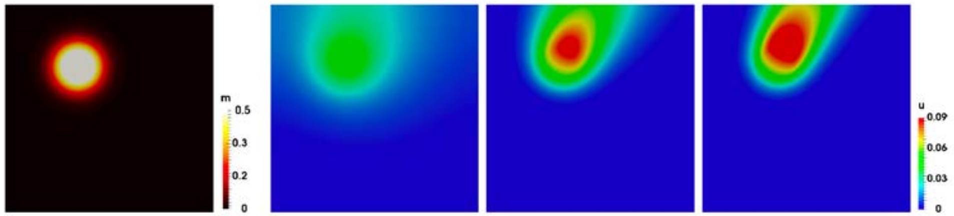
A prime target for HLR linear algebra: PDE-constrained optimization

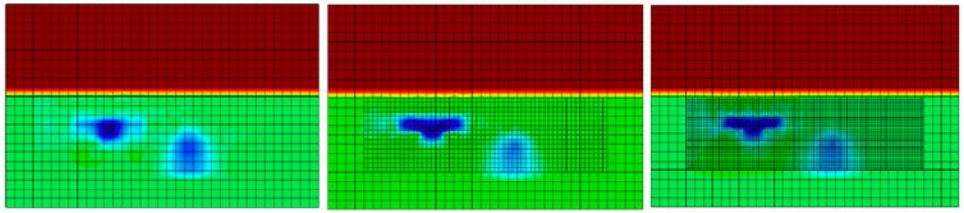
- **Desired operations (all *in situ*, please!)**
  - **matrix-vector multiplication**
  - **matrix-matrix multiplication**
    - **for inversion, roots, etc. using Newton-Schulz iteration**
  - **low-rank Hessian updating**
  - **“born data-sparse” Hessian matrix construction**
  - **recompression of matrix products**
  - **Frobenius and *p*-norms of matrices**

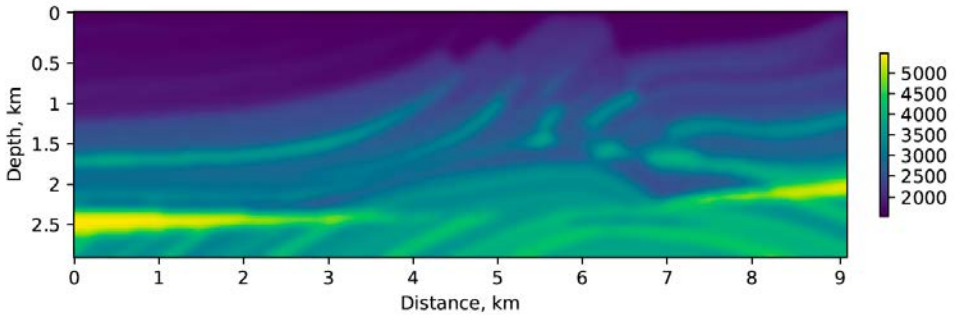
Optimization examples

- **1D transient diffusion**
  - **invert for coefficient**
- **2D stationary advection-diffusion**
  - **invert for source**
- **2D time-domain electromagnetism in diffusive limit**
  - **invert for coefficient**
- **2D frequency-domain wave equation**
  - **invert for coefficient**





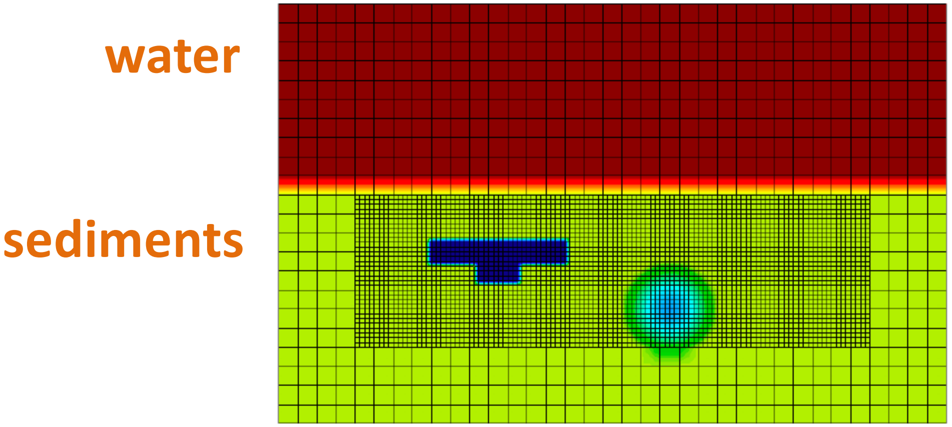




Ambartsumyan, Boukaram, Bui, Ghattas, K., Stadler, Turkiyyah & Zampini, *Hierarchical Matrix Approximations of Hessians Arising in Inverse Problems Governed by PDEs*, SISC, 2020

Inversion example: transient electromagnetic inversion

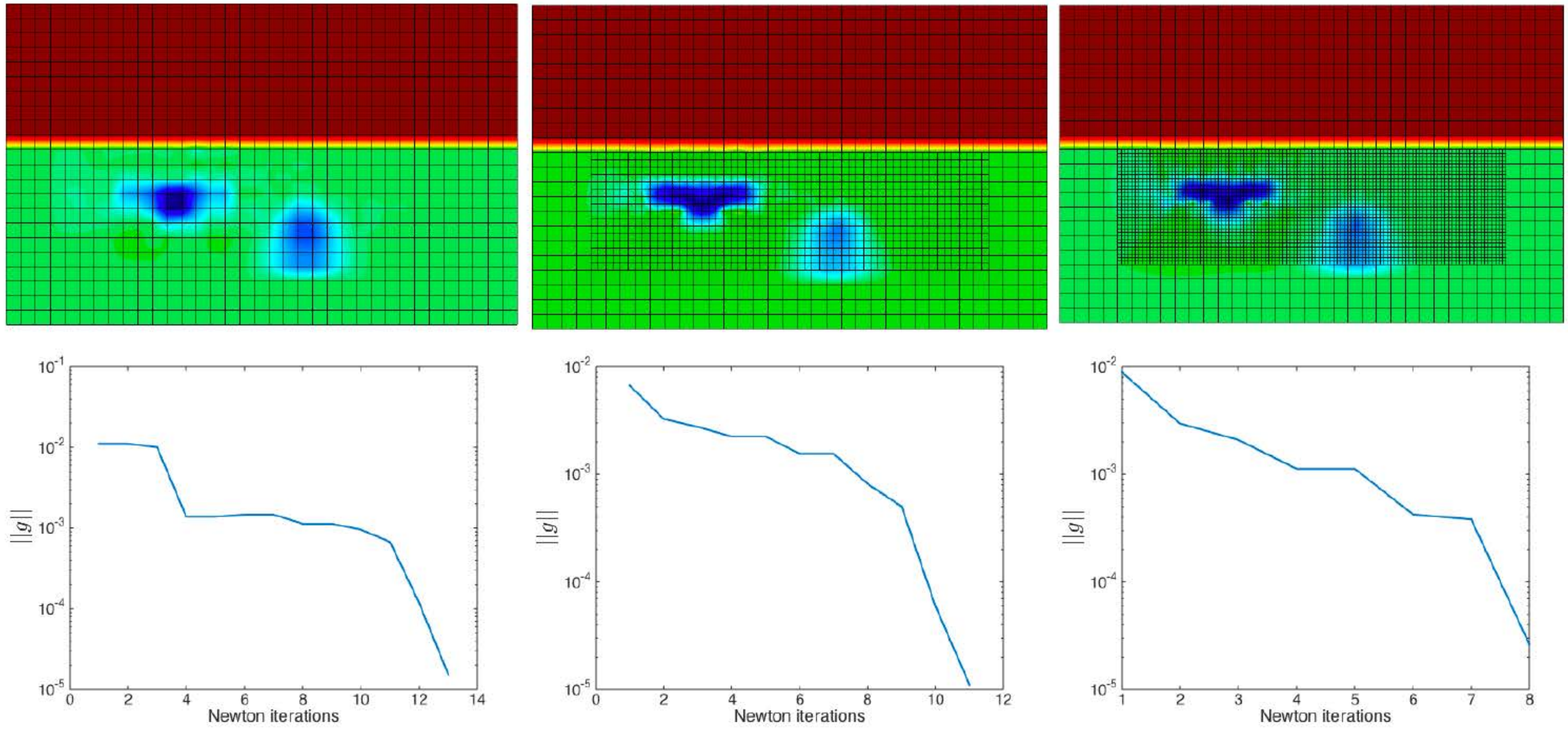
- Important geophysical sensing modality; of interest to Aramco
- Governing equations are Maxwell's equations in the diffusive limit.
- Electrical conductivity of oil is much lower than that of water, sediments, and salt bodies
- Time-domain inversion is the next frontier in this area
- We have developed a custom HPC code (using MFEM, PETSc) and specialized solvers for the simulations
- Below is an example with water, sediments, salt dome, and a T-shape anomaly to recover



Stefano Zampini, KAUST

MFEM, PETSc developer

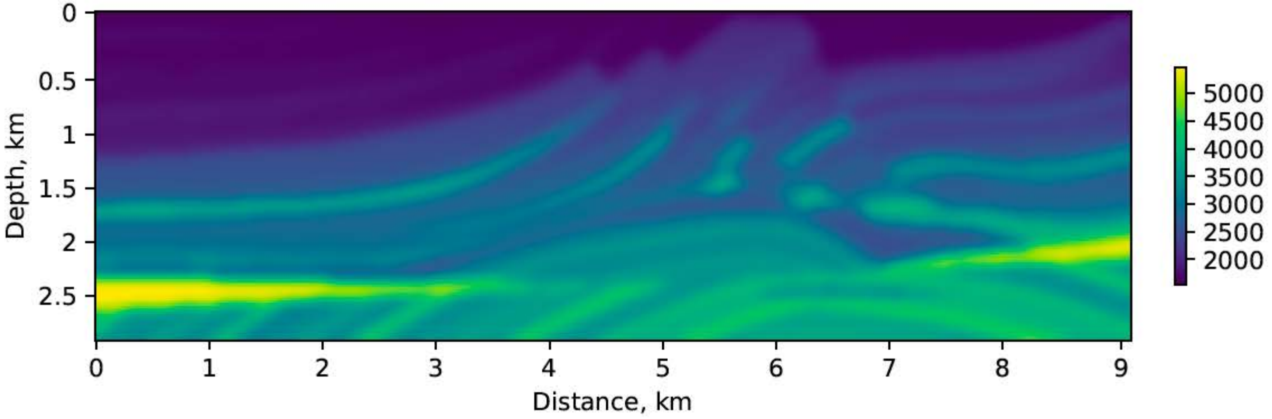
Inversion example: transient electromagnetic inversion

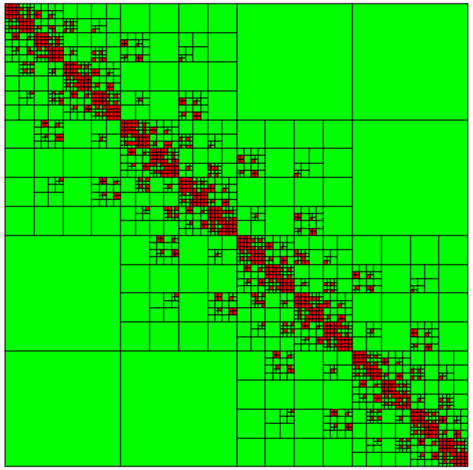


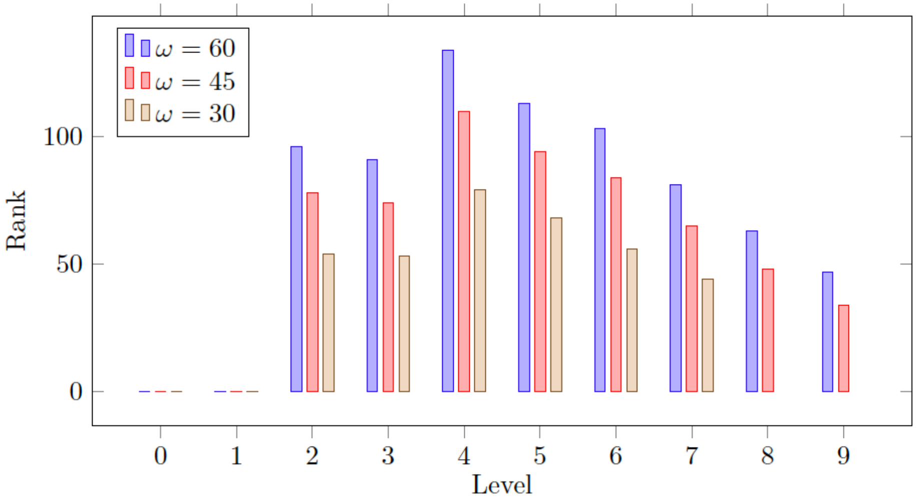
Inversion History: Newton solutions for 3 mesh continuation steps (top), and norm of the gradient as a function of the Newton step (bottom).

Inversion insight: frequency rank dependence

- frequency domain model problem of the most common sensing modality in geophysical exploration
- Marmousi model excited with different angular frequencies using a single supersource input
- local ranks of Hessians grow better than linearly with frequency

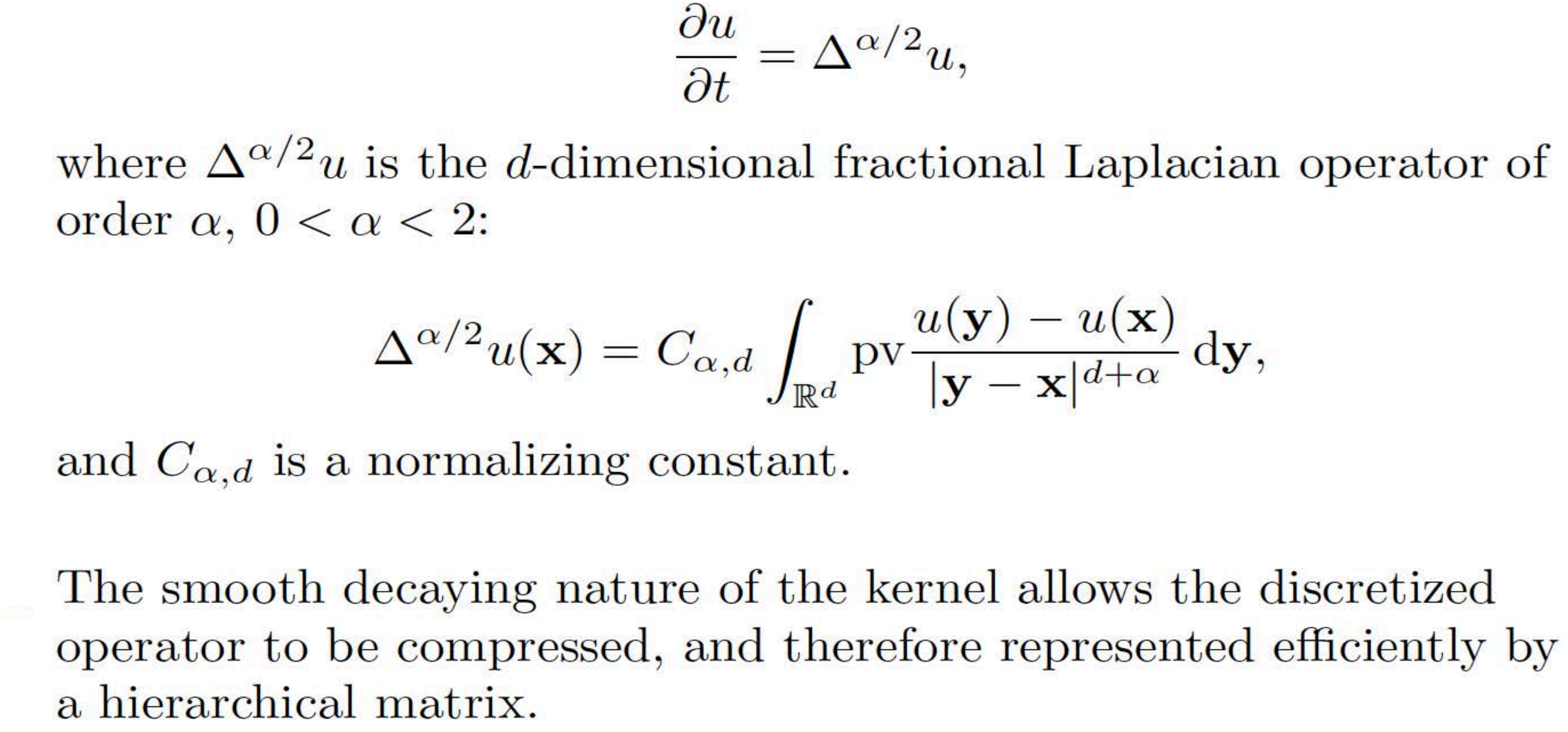






Fractional derivative application

- **1st-order in time, fractional Laplacian in 1D, 2D or 3D space**
  - **literature is mostly 1D, since operator is dense**
  - **this example: github.com/ecrc/h2opus/fractional_diffusion/**
- **Hot topic for diffusive flux models that are sub-Fickian (e.g., porous media, foamy media)**



$$\frac{\partial u}{\partial t} = \Delta^{\alpha/2} u,$$

where $\Delta^{\alpha/2} u$ is the d-dimensional fractional Laplacian operator of order α, $0 < \alpha < 2$:

$$\Delta^{\alpha/2} u(\mathbf{x}) = C_{\alpha,d} \int_{\mathbb{R}^d} \mathrm{pv} \frac{u(\mathbf{y}) - u(\mathbf{x})}{|\mathbf{y} - \mathbf{x}|^{d+\alpha}} \, \mathrm{d}\mathbf{y},$$

and $C_{\alpha,d}$ is a normalizing constant.

The smooth decaying nature of the kernel allows the discretized operator to be compressed, and therefore represented efficiently by a hierarchical matrix.

Smooth particle discretization

- Uses a smooth particle method, discretizing the d-dimensional spatial domain using a finite set of N particles.
- The i-th particle is defined by its position $\mathbf{x}_i$, volume V_i, and strength u_i, and

$$u(\mathbf{x}) = \sum_i V_i u_i \eta_\delta(\mathbf{x} - \mathbf{x}_i), \qquad \eta_\delta(\mathbf{x}) = \frac{1}{\delta^d} \frac{1}{\pi^{\frac{d}{2}}} \exp\left(-|\mathbf{x}|^2\right),$$

  where η_δ is a smoothed radial kernel of unit mass with a smoothing parameter δ.
- The particle positions $\mathbf{x}$ are separately evolving as Lagrangian tracers (and may be transported by the underlying medium in the presence of advective terms).
- This allows the discretized fractional diffusion equations to be written as: $\boxed{\dot{\mathbf{U}} = A_{N\times N}\mathbf{U}_{N\times 1}}$, where all A_{ij} entries are nonzero.

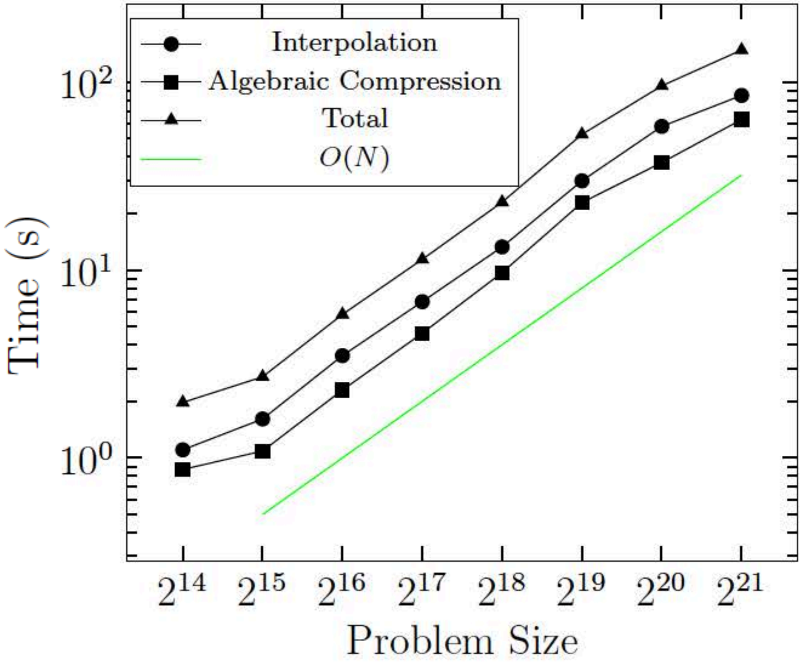
Key operation in explicit integration is a dense mat-vec

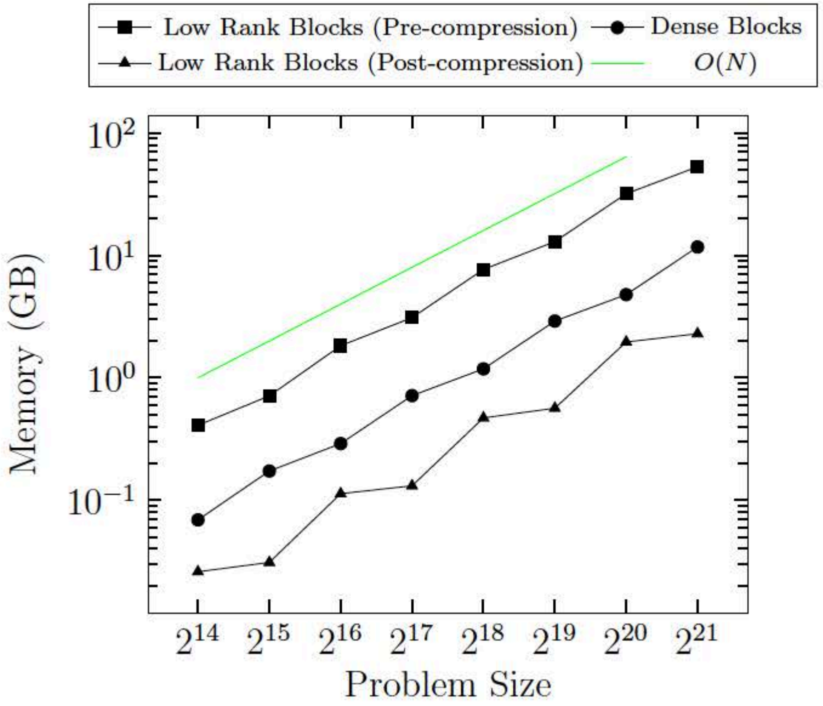
(A would be a sparse Laplacian for Fickian diffusion)

Fractional diffusion application

- construction time and resulting memory footprint for $\alpha = 1.5$
- target approximation accuracy $\epsilon = 10^{-5}$
- problems of various sizes up to $N \sim 2\text{M}$
- execution hardware: 12-core workstation
- both compression steps have linear complexity

NB: log scale

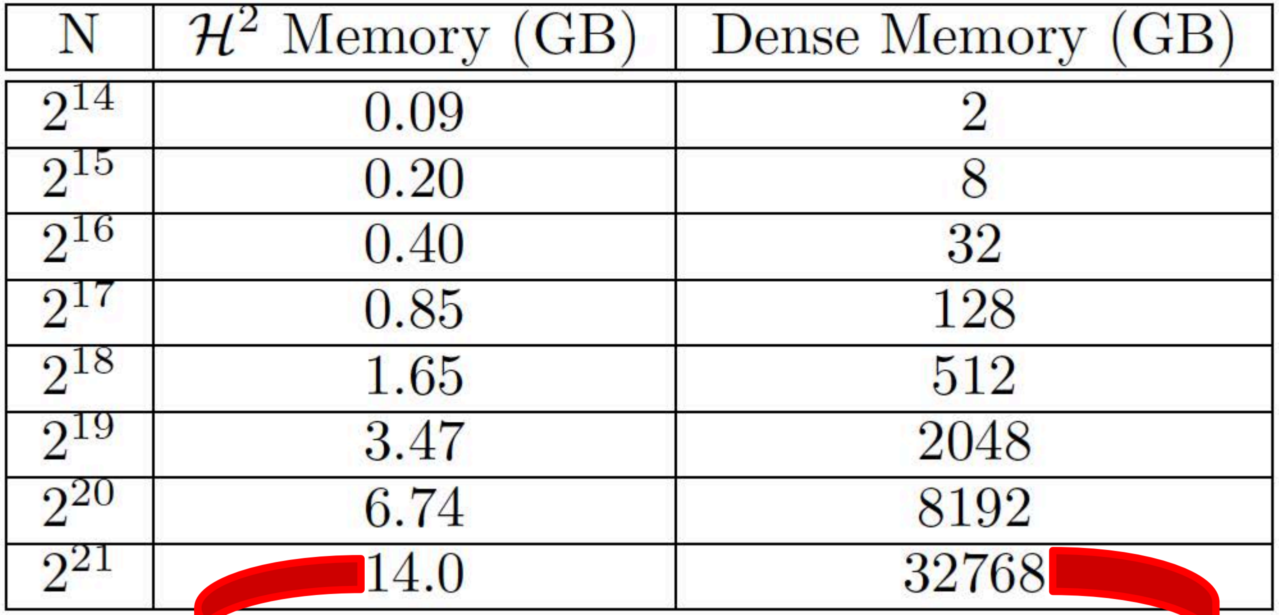




Boukaram, Lucchesi, Turkiyyah, Le Maitre, Knio & K., *Hierarchical Matrix Approximation for Space-fractional Diffusion Equations*, Comp Meths Appl Mech Eng, 2020

Fractional diffusion application

- Overall savings in memory of the hierarchical matrix (dense blocks and compressed blocks) computed to an overall accuracy of $\epsilon = 10^{-5}$
- Comparison of the hierarchical matrix representation and the (hypothetical) dense representation of the discretized operator
- Substantial reduction in memory, of $O(N)$ vs $O(N^2)$



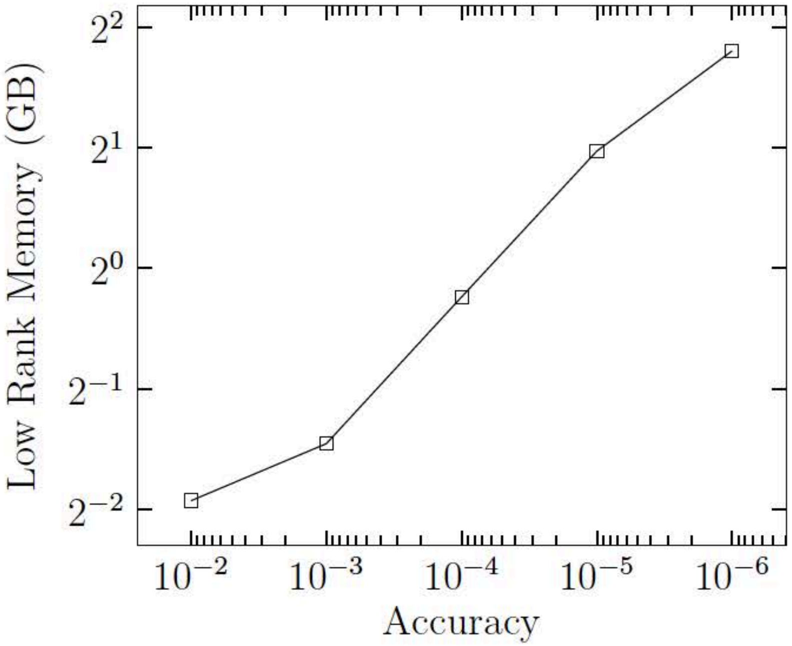
N	$\mathcal{H}^2$ Memory (GB)	Dense Memory (GB)
2^{14}	0.09	2
2^{15}	0.20	8
2^{16}	0.40	32
2^{17}	0.85	128
2^{18}	1.65	512
2^{19}	3.47	2048
2^{20}	6.74	8192
2^{21}	14.0	32768

footprint savings of 2,339x
for 2M DOFs

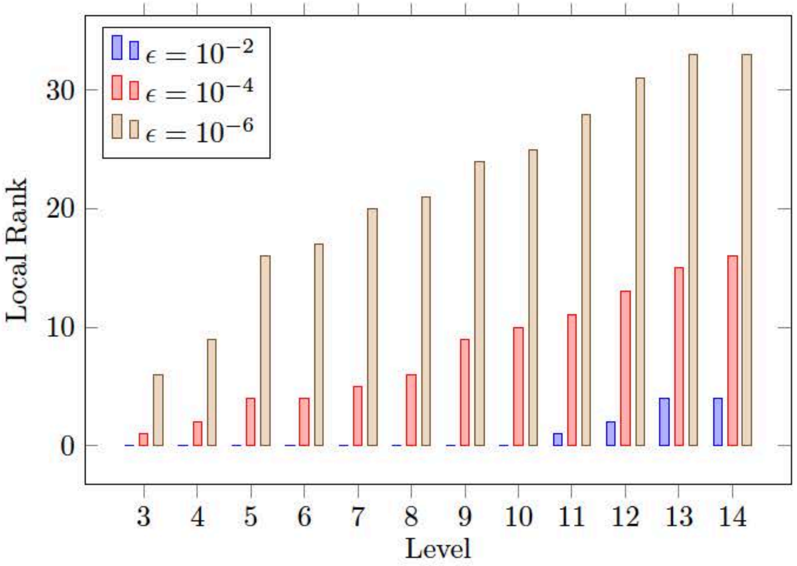
Rank variation with accuracy

- looser accuracy ϵ allows more reduction in the ranks of matrix blocks (local ranks) and in the resulting memory footprint
- shown are the memory footprints of the low rank blocks for a range of target accuracies for a problem of size $N = 2^{20}$
- also shown are the maximum local block ranks for every level of the corresponding hierarchical matrix

NB: log scale



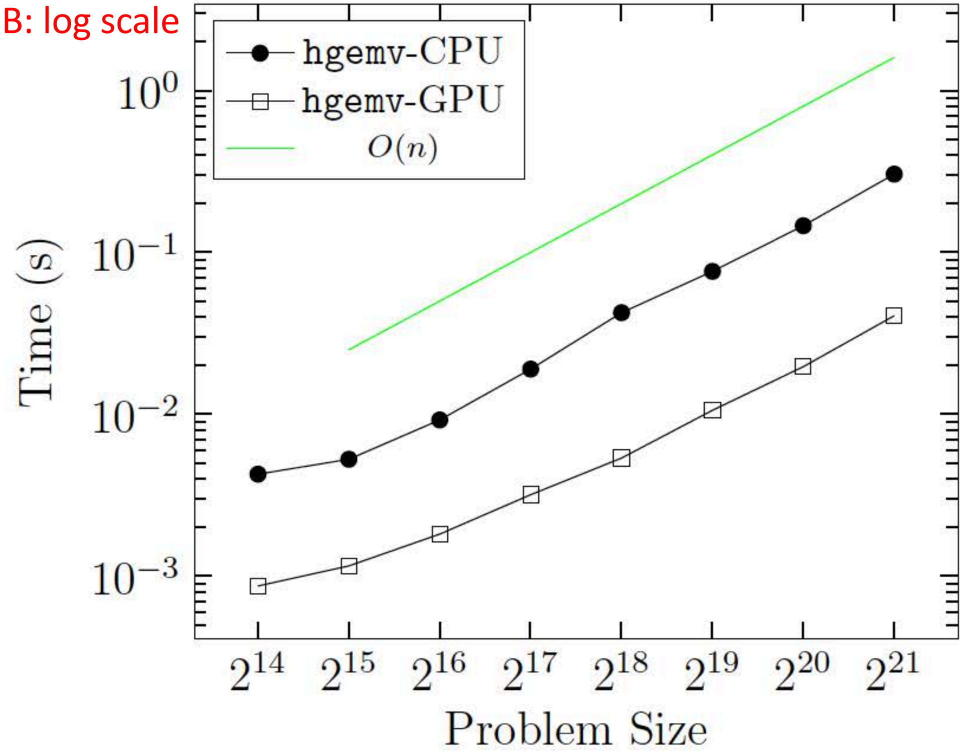
rank $\sim |\log \epsilon|^c$



Scaling of mat-vec

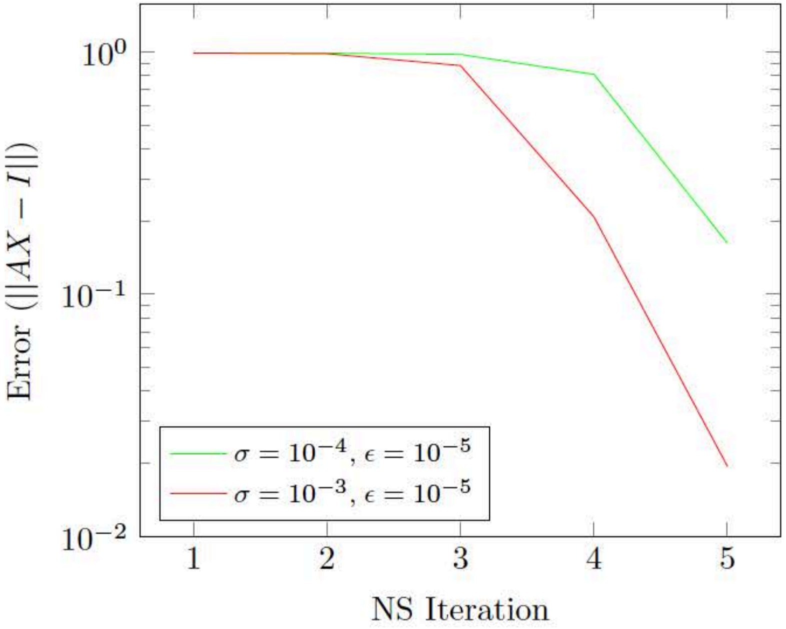
- mat-vec is the core operation in an explicit time integration scheme
- CPU (12-core) and GPU (Nvidia P100) results are shown ($\epsilon = 10^{-5}$)
- GPU is about 8x faster asymptotically
- linear complexity observed, as expected

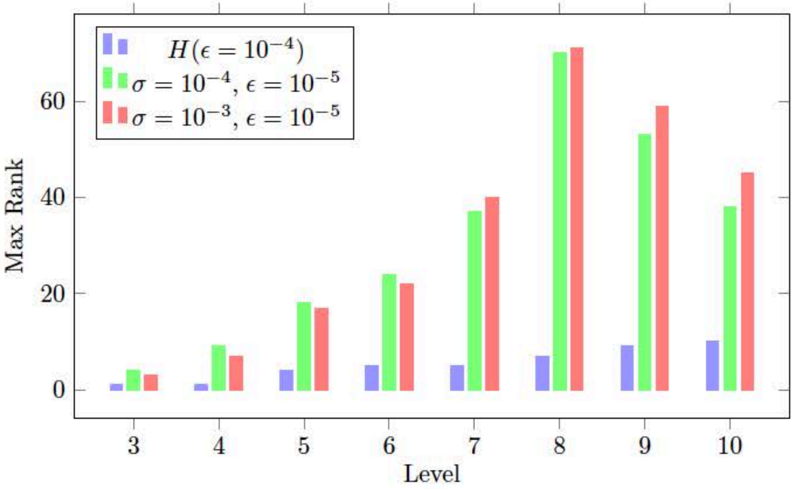
NB: log scale



Newton-Schulz for inverse

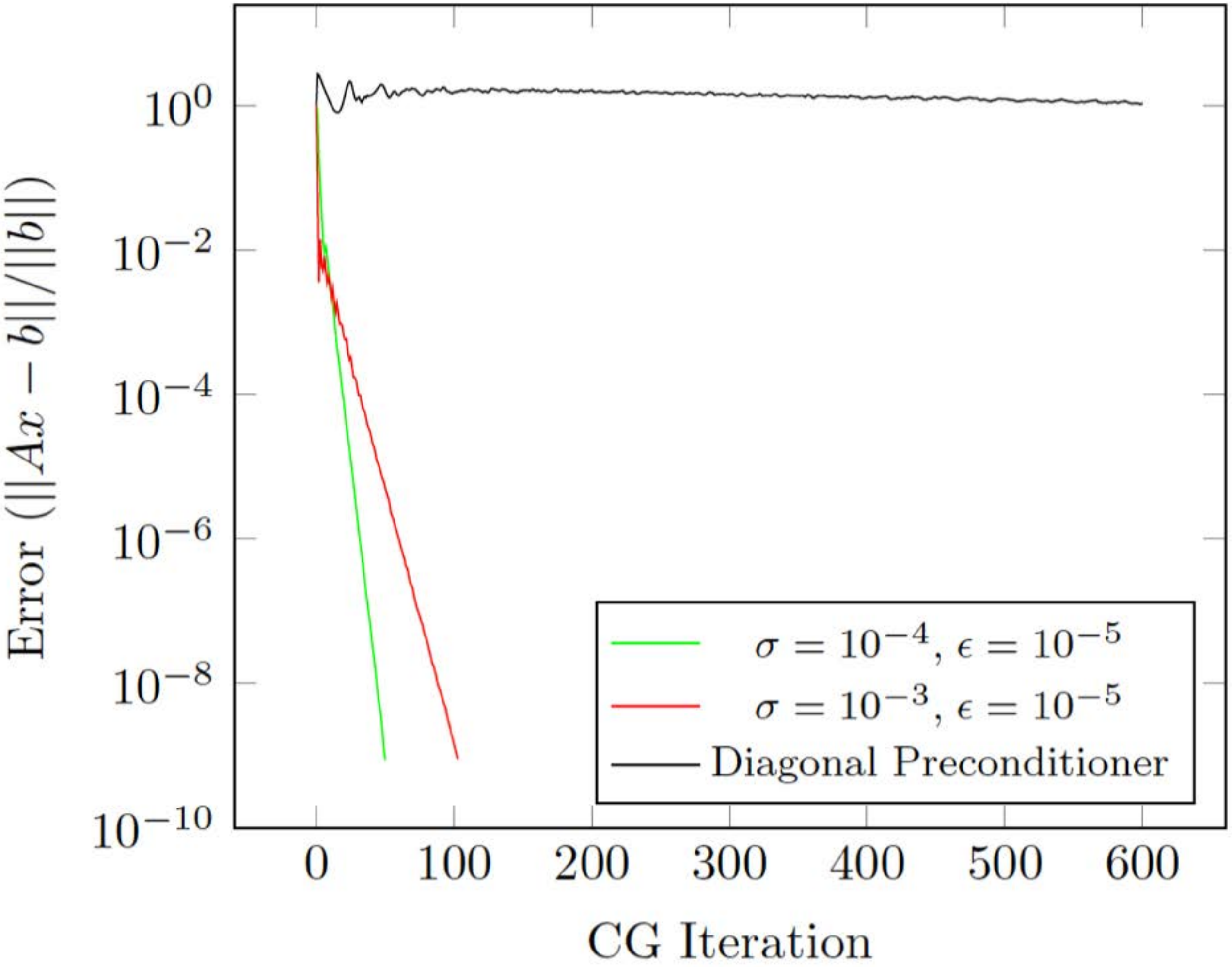
- Matrix inverse approximations may be computed via Newton-Schulz iteration: $X_{p+1} = (2I - X_p A)X_p$
- or its (more arithmetically intensive) higher-order variants: $X_{p+1} = X_p \left(I + R_p + \cdots + R_p^{l-1}\right) = X_p \sum_{i=0}^{v-1} R_p^i$
- Sample results for inverting $A + \sigma I$ with an order 16 iteration. $N = 2^{16}$. Each iterate constructed to an approximation ϵ.





Newton-Schulz as CG preconditioner

- PCG iterations ($\alpha = 1.5, N = 2^{16}$)



Lecture 1

Conclusions, recapped

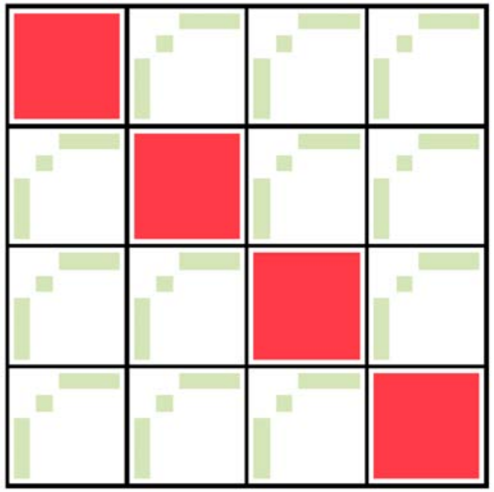
- With controllable trade-offs, many linear algebra operations adapt well to high performance on emerging architectures through
  - higher residence on the memory hierarchy
  - greater SIMT/SIMD-style concurrency
  - reduced synchronization and communication
- Rank-structured matrices, based on uniform tiles or hierarchical subdivision play a major role
- Rank-structured matrix software is here for shared-memory, distributed-memory, and GPU environments
- Many applications are benefiting
  - by orders of magnitude in memory footprint & runtime

Some ripe directions

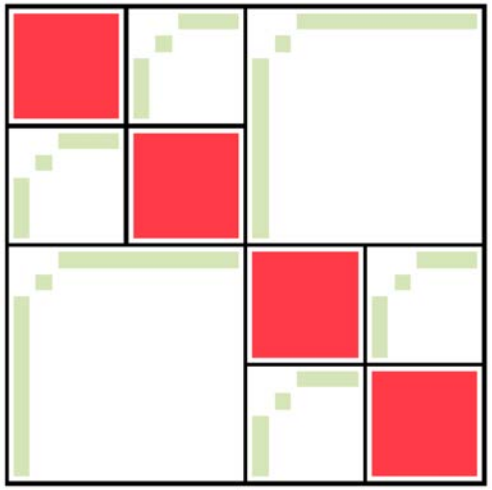
- **Research (heuristics or theory)**
  - Guidance for tuning accuracy of individual block replacements with ultimate global purpose(s) for rank-structured matrix in view
  - Orderings of DOFs from multidimensional problems that minimize overall TLR and HLR memory footprint
  - Point blockings and point-block scalings for multicomponent problems
- **Development (software)**
  - Generalization to variable rank size at leaves of the HLR tree (currently allocated for a max rank *per level,* not as restrictive as max rank for all compressed blocks)
  - Optimizing parallelization of distributed implementation above "C-level" (in analogy to multigrid)

Iconographic conclusion

I ♥



yesterday

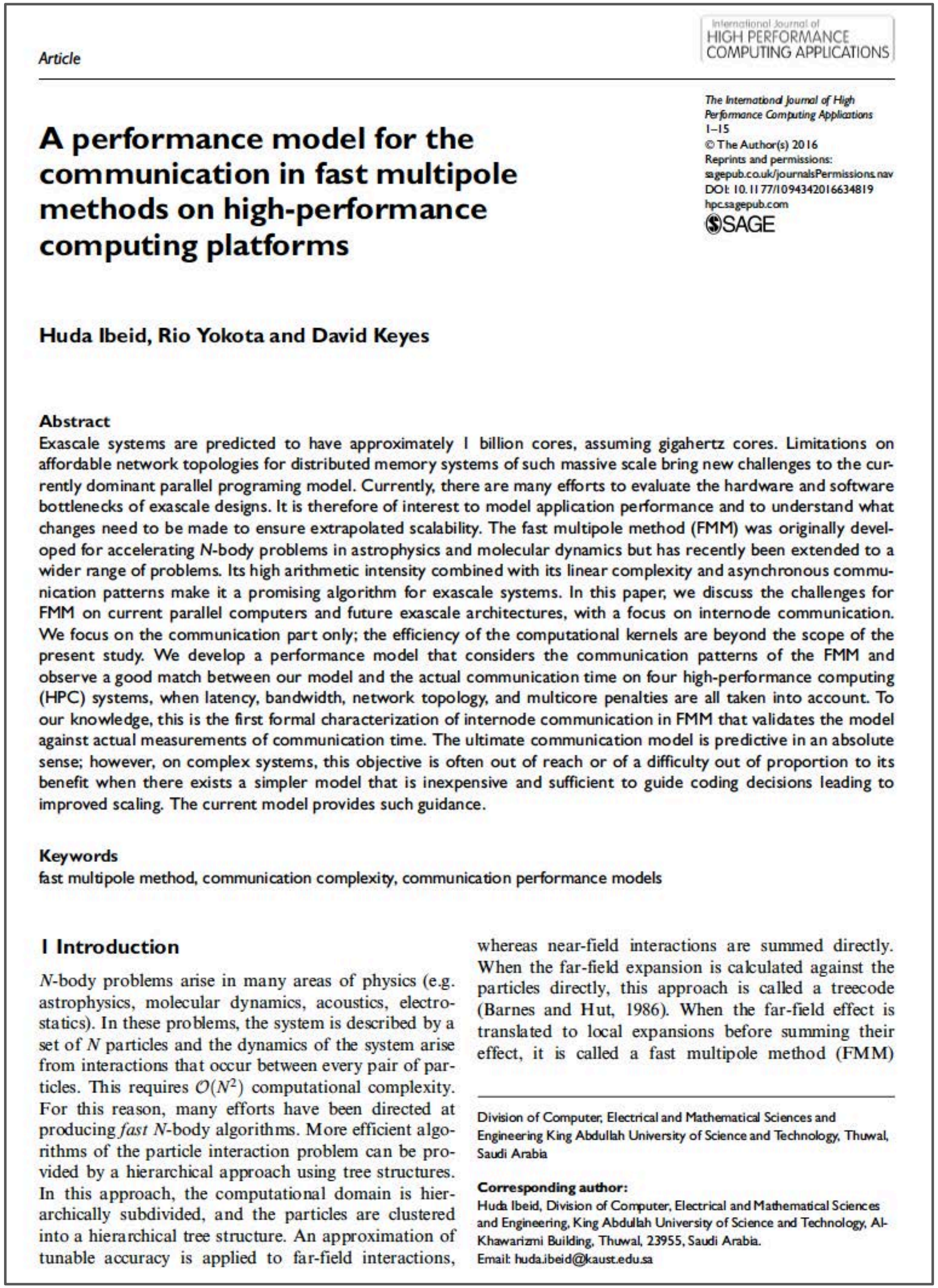


today

Reference

Ibeid, Yokota & K.

Int. J. High Perf. Comput. Applics. (2016)



Article

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–15
© The Author(s) 2016
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342016634819
hpc.sagepub.com
SAGE

A performance model for the communication in fast multipole methods on high-performance computing platforms

Huda Ibeid, Rio Yokota and David Keyes

Abstract

Exascale systems are predicted to have approximately 1 billion cores, assuming gigahertz cores. Limitations on affordable network topologies for distributed memory systems of such massive scale bring new challenges to the currently dominant parallel programing model. Currently, there are many efforts to evaluate the hardware and software bottlenecks of exascale designs. It is therefore of interest to model application performance and to understand what changes need to be made to ensure extrapolated scalability. The fast multipole method (FMM) was originally developed for accelerating *N*-body problems in astrophysics and molecular dynamics but has recently been extended to a wider range of problems. Its high arithmetic intensity combined with its linear complexity and asynchronous communication patterns make it a promising algorithm for exascale systems. In this paper, we discuss the challenges for FMM on current parallel computers and future exascale architectures, with a focus on internode communication. We focus on the communication part only; the efficiency of the computational kernels are beyond the scope of the present study. We develop a performance model that considers the communication patterns of the FMM and observe a good match between our model and the actual communication time on four high-performance computing (HPC) systems, when latency, bandwidth, network topology, and multicore penalties are all taken into account. To our knowledge, this is the first formal characterization of internode communication in FMM that validates the model against actual measurements of communication time. The ultimate communication model is predictive in an absolute sense; however, on complex systems, this objective is often out of reach or of a difficulty out of proportion to its benefit when there exists a simpler model that is inexpensive and sufficient to guide coding decisions leading to improved scaling. The current model provides such guidance.

Keywords

fast multipole method, communication complexity, communication performance models

1 Introduction

N-body problems arise in many areas of physics (e.g. astrophysics, molecular dynamics, acoustics, electrostatics). In these problems, the system is described by a set of *N* particles and the dynamics of the system arise from interactions that occur between every pair of particles. This requires $\mathcal{O}(N^2)$ computational complexity. For this reason, many efforts have been directed at producing *fast N*-body algorithms. More efficient algorithms of the particle interaction problem can be provided by a hierarchical approach using tree structures. In this approach, the computational domain is hierarchically subdivided, and the particles are clustered into a hierarchical tree structure. An approximation of tunable accuracy is applied to far-field interactions, whereas near-field interactions are summed directly. When the far-field expansion is calculated against the particles directly, this approach is called a treecode (Barnes and Hut, 1986). When the far-field effect is translated to local expansions before summing their effect, it is called a fast multipole method (FMM)

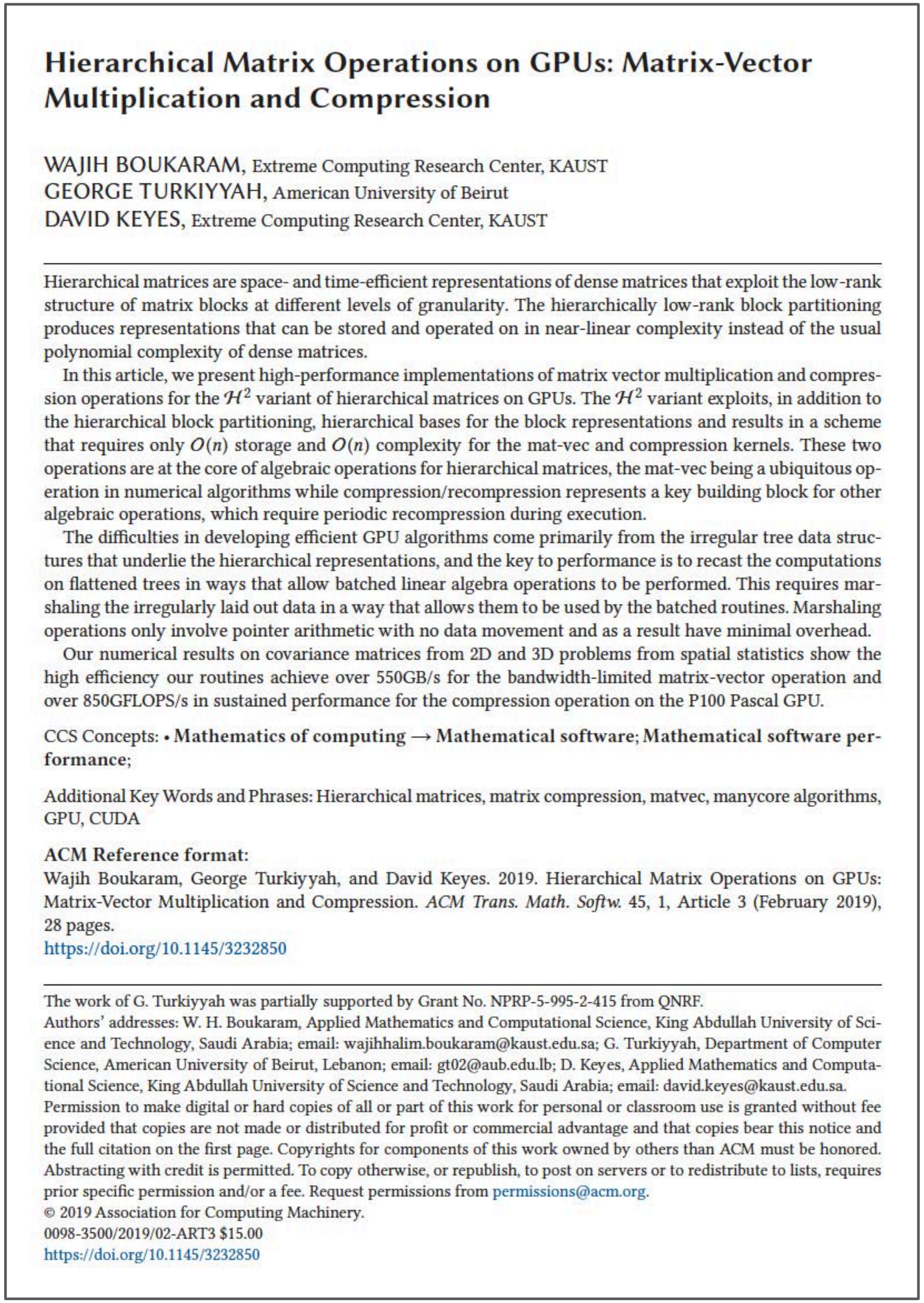
Division of Computer, Electrical and Mathematical Sciences and Engineering King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Corresponding author:
Huda Ibeid, Division of Computer, Electrical and Mathematical Sciences and Engineering, King Abdullah University of Science and Technology, Al-Khawarizmi Building, Thuwal, 23955, Saudi Arabia.
Email: huda.ibeid@kaust.edu.sa

Reference

Boukaram,
Turkiyyah & K.

ACM Trans. Math. Software* *(2019)



Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression

WAJIH BOUKARAM, Extreme Computing Research Center, KAUST
GEORGE TURKIYYAH, American University of Beirut
DAVID KEYES, Extreme Computing Research Center, KAUST

Hierarchical matrices are space- and time-efficient representations of dense matrices that exploit the low-rank structure of matrix blocks at different levels of granularity. The hierarchically low-rank block partitioning produces representations that can be stored and operated on in near-linear complexity instead of the usual polynomial complexity of dense matrices.

In this article, we present high-performance implementations of matrix vector multiplication and compression operations for the $\mathcal{H}^2$ variant of hierarchical matrices on GPUs. The $\mathcal{H}^2$ variant exploits, in addition to the hierarchical block partitioning, hierarchical bases for the block representations and results in a scheme that requires only $O(n)$ storage and $O(n)$ complexity for the mat-vec and compression kernels. These two operations are at the core of algebraic operations for hierarchical matrices, the mat-vec being a ubiquitous operation in numerical algorithms while compression/recompression represents a key building block for other algebraic operations, which require periodic recompression during execution.

The difficulties in developing efficient GPU algorithms come primarily from the irregular tree data structures that underlie the hierarchical representations, and the key to performance is to recast the computations on flattened trees in ways that allow batched linear algebra operations to be performed. This requires marshaling the irregularly laid out data in a way that allows them to be used by the batched routines. Marshaling operations only involve pointer arithmetic with no data movement and as a result have minimal overhead.

Our numerical results on covariance matrices from 2D and 3D problems from spatial statistics show the high efficiency our routines achieve over 550GB/s for the bandwidth-limited matrix-vector operation and over 850GFLOPS/s in sustained performance for the compression operation on the P100 Pascal GPU.

CCS Concepts: • **Mathematics of computing → Mathematical software; Mathematical software performance;**

Additional Key Words and Phrases: Hierarchical matrices, matrix compression, matvec, manycore algorithms, GPU, CUDA

ACM Reference format:
Wajih Boukaram, George Turkiyyah, and David Keyes. 2019. Hierarchical Matrix Operations on GPUs: Matrix-Vector Multiplication and Compression. *ACM Trans. Math. Softw.* 45, 1, Article 3 (February 2019), 28 pages.
https://doi.org/10.1145/3232850

The work of G. Turkiyyah was partially supported by Grant No. NPRP-5-995-2-415 from QNRF.

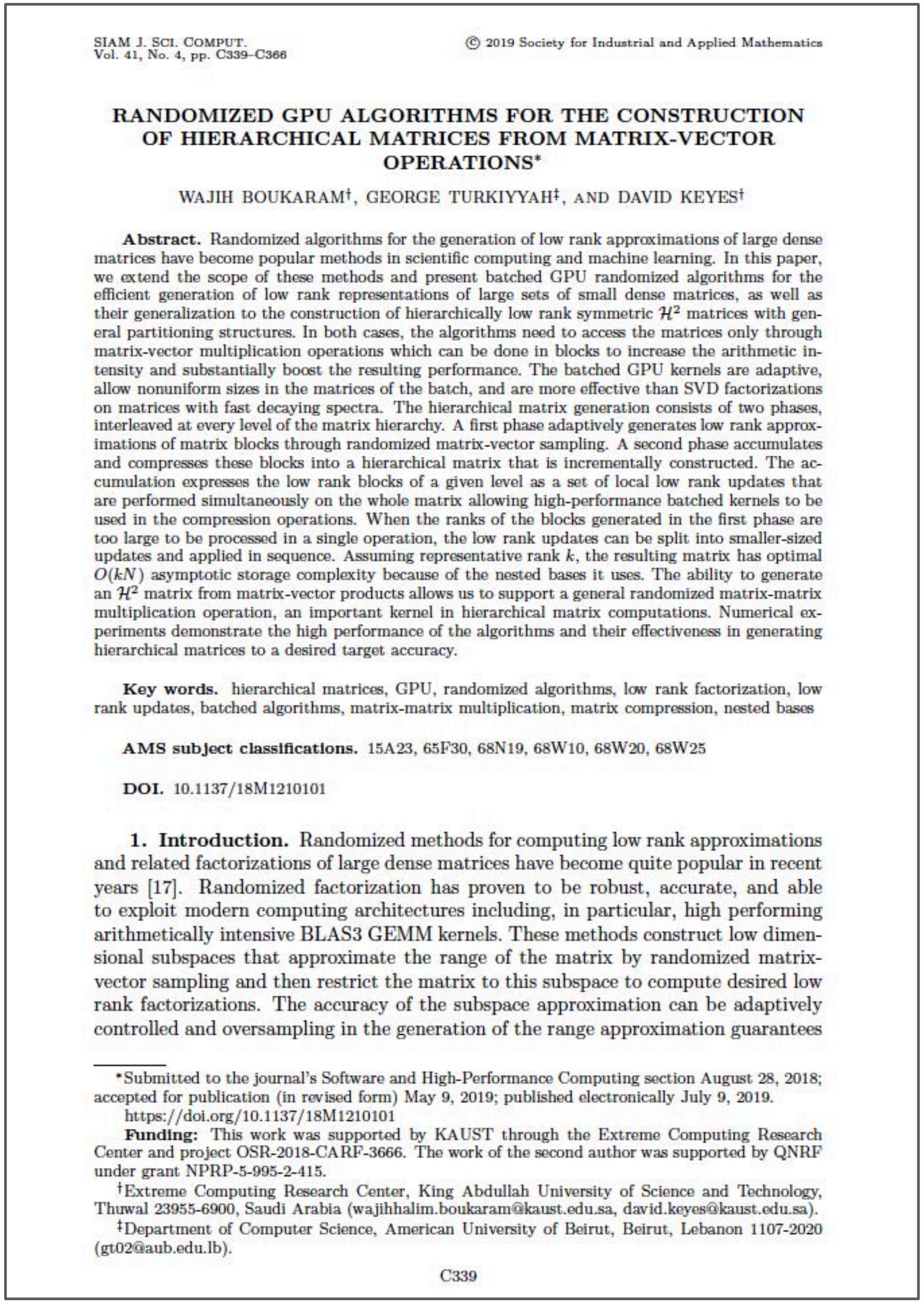
Authors' addresses: W. H. Boukaram, Applied Mathematics and Computational Science, King Abdullah University of Science and Technology, Saudi Arabia; email: wajihhalim.boukaram@kaust.edu.sa; G. Turkiyyah, Department of Computer Science, American University of Beirut, Lebanon; email: gt02@aub.edu.lb; D. Keyes, Applied Mathematics and Computational Science, King Abdullah University of Science and Technology, Saudi Arabia; email: david.keyes@kaust.edu.sa.

0098-3500/2019/02-ART3 $15.00
https://doi.org/10.1145/3232850

Reference

Boukaram, Turkiyyah & K.

SIAM J. Sci. Comput. *(2019)*



SIAM J. SCI. COMPUT.
Vol. 41, No. 4, pp. C339–C366

RANDOMIZED GPU ALGORITHMS FOR THE CONSTRUCTION OF HIERARCHICAL MATRICES FROM MATRIX-VECTOR OPERATIONS*

WAJIH BOUKARAM†, GEORGE TURKIYYAH‡, AND DAVID KEYES†

Abstract. Randomized algorithms for the generation of low rank approximations of large dense matrices have become popular methods in scientific computing and machine learning. In this paper, we extend the scope of these methods and present batched GPU randomized algorithms for the efficient generation of low rank representations of large sets of small dense matrices, as well as their generalization to the construction of hierarchically low rank symmetric $\mathcal{H}^2$ matrices with general partitioning structures. In both cases, the algorithms need to access the matrices only through matrix-vector multiplication operations which can be done in blocks to increase the arithmetic intensity and substantially boost the resulting performance. The batched GPU kernels are adaptive, allow nonuniform sizes in the matrices of the batch, and are more effective than SVD factorizations on matrices with fast decaying spectra. The hierarchical matrix generation consists of two phases, interleaved at every level of the matrix hierarchy. A first phase adaptively generates low rank approximations of matrix blocks through randomized matrix-vector sampling. A second phase accumulates and compresses these blocks into a hierarchical matrix that is incrementally constructed. The accumulation expresses the low rank blocks of a given level as a set of local low rank updates that are performed simultaneously on the whole matrix allowing high-performance batched kernels to be used in the compression operations. When the ranks of the blocks generated in the first phase are too large to be processed in a single operation, the low rank updates can be split into smaller-sized updates and applied in sequence. Assuming representative rank k, the resulting matrix has optimal $O(kN)$ asymptotic storage complexity because of the nested bases it uses. The ability to generate an $\mathcal{H}^2$ matrix from matrix-vector products allows us to support a general randomized matrix-matrix multiplication operation, an important kernel in hierarchical matrix computations. Numerical experiments demonstrate the high performance of the algorithms and their effectiveness in generating hierarchical matrices to a desired target accuracy.

Key words. hierarchical matrices, GPU, randomized algorithms, low rank factorization, low rank updates, batched algorithms, matrix-matrix multiplication, matrix compression, nested bases

AMS subject classifications. 15A23, 65F30, 68N19, 68W10, 68W20, 68W25

DOI. 10.1137/18M1210101

1. Introduction. Randomized methods for computing low rank approximations and related factorizations of large dense matrices have become quite popular in recent years [17]. Randomized factorization has proven to be robust, accurate, and able to exploit modern computing architectures including, in particular, high performing arithmetically intensive BLAS3 GEMM kernels. These methods construct low dimensional subspaces that approximate the range of the matrix by randomized matrix-vector sampling and then restrict the matrix to this subspace to compute desired low rank factorizations. The accuracy of the subspace approximation can be adaptively controlled and oversampling in the generation of the range approximation guarantees

*Submitted to the journal's Software and High-Performance Computing section August 28, 2018; accepted for publication (in revised form) May 9, 2019; published electronically July 9, 2019.
https://doi.org/10.1137/18M1210101

Funding: This work was supported by KAUST through the Extreme Computing Research Center and project OSR-2018-CARF-3666. The work of the second author was supported by QNRF under grant NPRP-5-995-2-415.

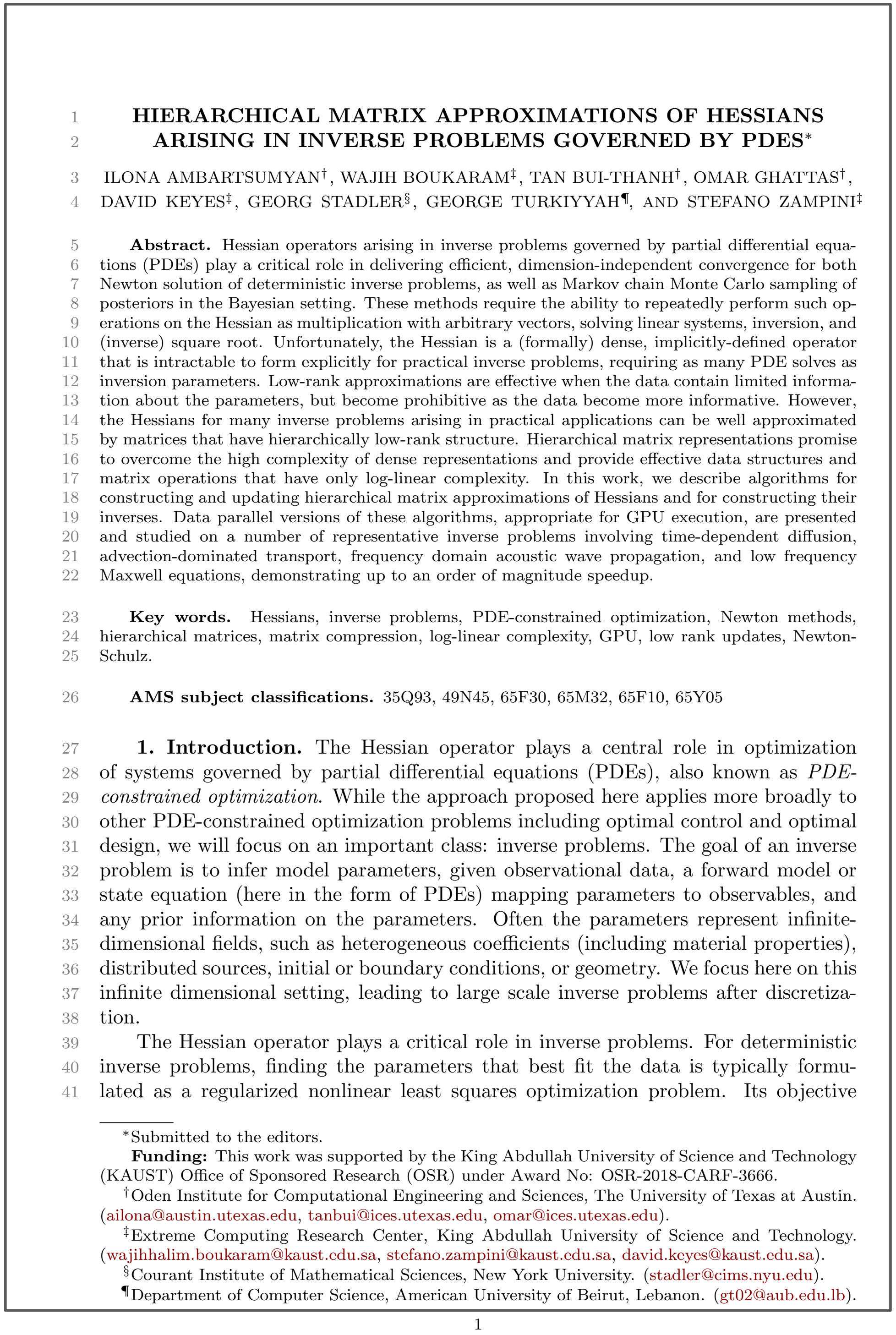
†Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia (wajihhalim.boukaram@kaust.edu.sa, david.keyes@kaust.edu.sa).

‡Department of Computer Science, American University of Beirut, Beirut, Lebanon 1107-2020 (gt02@aub.edu.lb).

Reference

Ambartsumyan, Boukaram, Bui-Thanh, Ghattas, K., Stadler, Turkiyyah & Zampini

SIAM Journal of Scientific Computing (2020)



HIERARCHICAL MATRIX APPROXIMATIONS OF HESSIANS ARISING IN INVERSE PROBLEMS GOVERNED BY PDES*

ILONA AMBARTSUMYAN†, WAJIH BOUKARAM‡, TAN BUI-THANH†, OMAR GHATTAS†, DAVID KEYES‡, GEORG STADLER§, GEORGE TURKIYYAH¶, AND STEFANO ZAMPINI‡

Abstract. Hessian operators arising in inverse problems governed by partial differential equations (PDEs) play a critical role in delivering efficient, dimension-independent convergence for both Newton solution of deterministic inverse problems, as well as Markov chain Monte Carlo sampling of posteriors in the Bayesian setting. These methods require the ability to repeatedly perform such operations on the Hessian as multiplication with arbitrary vectors, solving linear systems, inversion, and (inverse) square root. Unfortunately, the Hessian is a (formally) dense, implicitly-defined operator that is intractable to form explicitly for practical inverse problems, requiring as many PDE solves as inversion parameters. Low-rank approximations are effective when the data contain limited information about the parameters, but become prohibitive as the data become more informative. However, the Hessians for many inverse problems arising in practical applications can be well approximated by matrices that have hierarchically low-rank structure. Hierarchical matrix representations promise to overcome the high complexity of dense representations and provide effective data structures and matrix operations that have only log-linear complexity. In this work, we describe algorithms for constructing and updating hierarchical matrix approximations of Hessians and for constructing their inverses. Data parallel versions of these algorithms, appropriate for GPU execution, are presented and studied on a number of representative inverse problems involving time-dependent diffusion, advection-dominated transport, frequency domain acoustic wave propagation, and low frequency Maxwell equations, demonstrating up to an order of magnitude speedup.

Key words. Hessians, inverse problems, PDE-constrained optimization, Newton methods, hierarchical matrices, matrix compression, log-linear complexity, GPU, low rank updates, Newton-Schulz.

AMS subject classifications. 35Q93, 49N45, 65F30, 65M32, 65F10, 65Y05

1. Introduction. The Hessian operator plays a central role in optimization of systems governed by partial differential equations (PDEs), also known as *PDE-constrained optimization*. While the approach proposed here applies more broadly to other PDE-constrained optimization problems including optimal control and optimal design, we will focus on an important class: inverse problems. The goal of an inverse problem is to infer model parameters, given observational data, a forward model or state equation (here in the form of PDEs) mapping parameters to observables, and any prior information on the parameters. Often the parameters represent infinite-dimensional fields, such as heterogeneous coefficients (including material properties), distributed sources, initial or boundary conditions, or geometry. We focus here on this infinite dimensional setting, leading to large scale inverse problems after discretization.

The Hessian operator plays a critical role in inverse problems. For deterministic inverse problems, finding the parameters that best fit the data is typically formulated as a regularized nonlinear least squares optimization problem. Its objective

*Submitted to the editors.

Funding: This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research (OSR) under Award No: OSR-2018-CARF-3666.

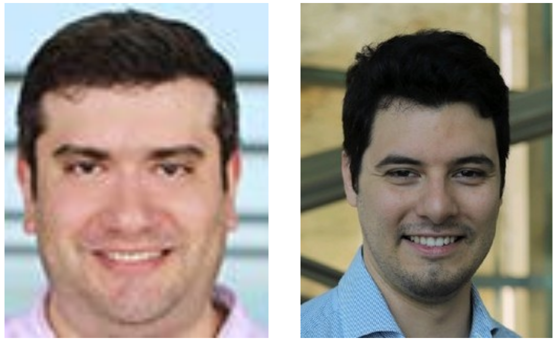
†Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin. (ailona@austin.utexas.edu, tanbui@ices.utexas.edu, omar@ices.utexas.edu).

‡Extreme Computing Research Center, King Abdullah University of Science and Technology. (wajihhalim.boukaram@kaust.edu.sa, stefano.zampini@kaust.edu.sa, david.keyes@kaust.edu.sa).

§Courant Institute of Mathematical Sciences, New York University. (stadler@cims.nyu.edu).

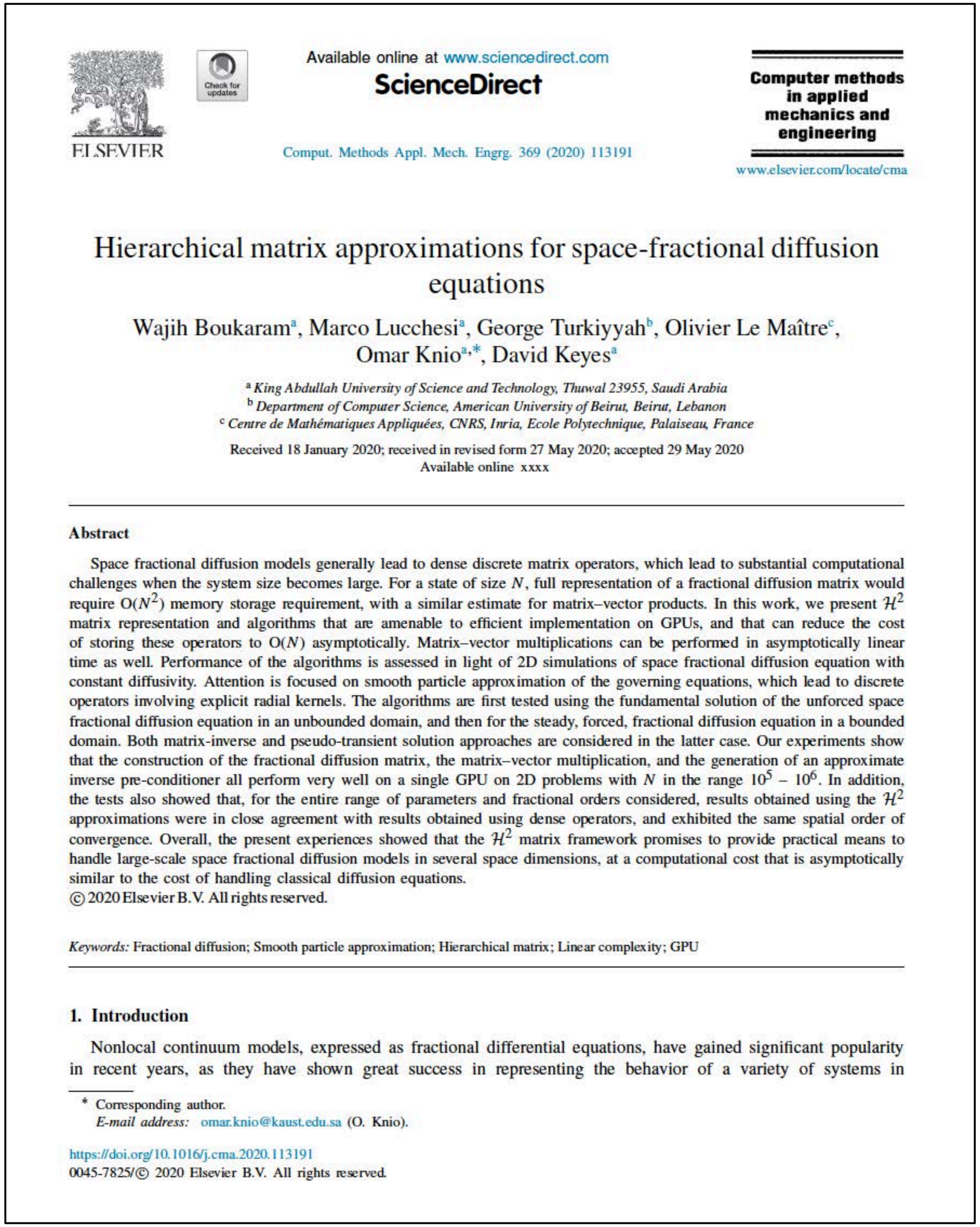
¶Department of Computer Science, American University of Beirut, Lebanon. (gt02@aub.edu.lb).

Reference



<u>Boukaram</u>, <u>Lucchesi</u>, Turkiyyah, Le Maitre, Knio & K.

Computer Methods in Applied Mechanics and Engineering 369:113191 (2020)



ELSEVIER

Check for updates

Available online at www.sciencedirect.com

ScienceDirect

Comput. Methods Appl. Mech. Engrg. 369 (2020) 113191

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Hierarchical matrix approximations for space-fractional diffusion equations

Wajih Boukaram[a], Marco Lucchesi[a], George Turkiyyah[b], Olivier Le Maître[c], Omar Knio[a,*], David Keyes[a]

[a] *King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia*
[b] *Department of Computer Science, American University of Beirut, Beirut, Lebanon*
[c] *Centre de Mathématiques Appliquées, CNRS, Inria, Ecole Polytechnique, Palaiseau, France*

Received 18 January 2020; received in revised form 27 May 2020; accepted 29 May 2020
Available online xxxx

Abstract

Space fractional diffusion models generally lead to dense discrete matrix operators, which lead to substantial computational challenges when the system size becomes large. For a state of size N, full representation of a fractional diffusion matrix would require $O(N^2)$ memory storage requirement, with a similar estimate for matrix–vector products. In this work, we present $\mathcal{H}^2$ matrix representation and algorithms that are amenable to efficient implementation on GPUs, and that can reduce the cost of storing these operators to $O(N)$ asymptotically. Matrix–vector multiplications can be performed in asymptotically linear time as well. Performance of the algorithms is assessed in light of 2D simulations of space fractional diffusion equation with constant diffusivity. Attention is focused on smooth particle approximation of the governing equations, which lead to discrete operators involving explicit radial kernels. The algorithms are first tested using the fundamental solution of the unforced space fractional diffusion equation in an unbounded domain, and then for the steady, forced, fractional diffusion equation in a bounded domain. Both matrix-inverse and pseudo-transient solution approaches are considered in the latter case. Our experiments show that the construction of the fractional diffusion matrix, the matrix–vector multiplication, and the generation of an approximate inverse pre-conditioner all perform very well on a single GPU on 2D problems with N in the range $10^5 - 10^6$. In addition, the tests also showed that, for the entire range of parameters and fractional orders considered, results obtained using the $\mathcal{H}^2$ approximations were in close agreement with results obtained using dense operators, and exhibited the same spatial order of convergence. Overall, the present experiences showed that the $\mathcal{H}^2$ matrix framework promises to provide practical means to handle large-scale space fractional diffusion models in several space dimensions, at a computational cost that is asymptotically similar to the cost of handling classical diffusion equations.

Keywords: Fractional diffusion; Smooth particle approximation; Hierarchical matrix; Linear complexity; GPU

1. Introduction

Nonlocal continuum models, expressed as fractional differential equations, have gained significant popularity in recent years, as they have shown great success in representing the behavior of a variety of systems in

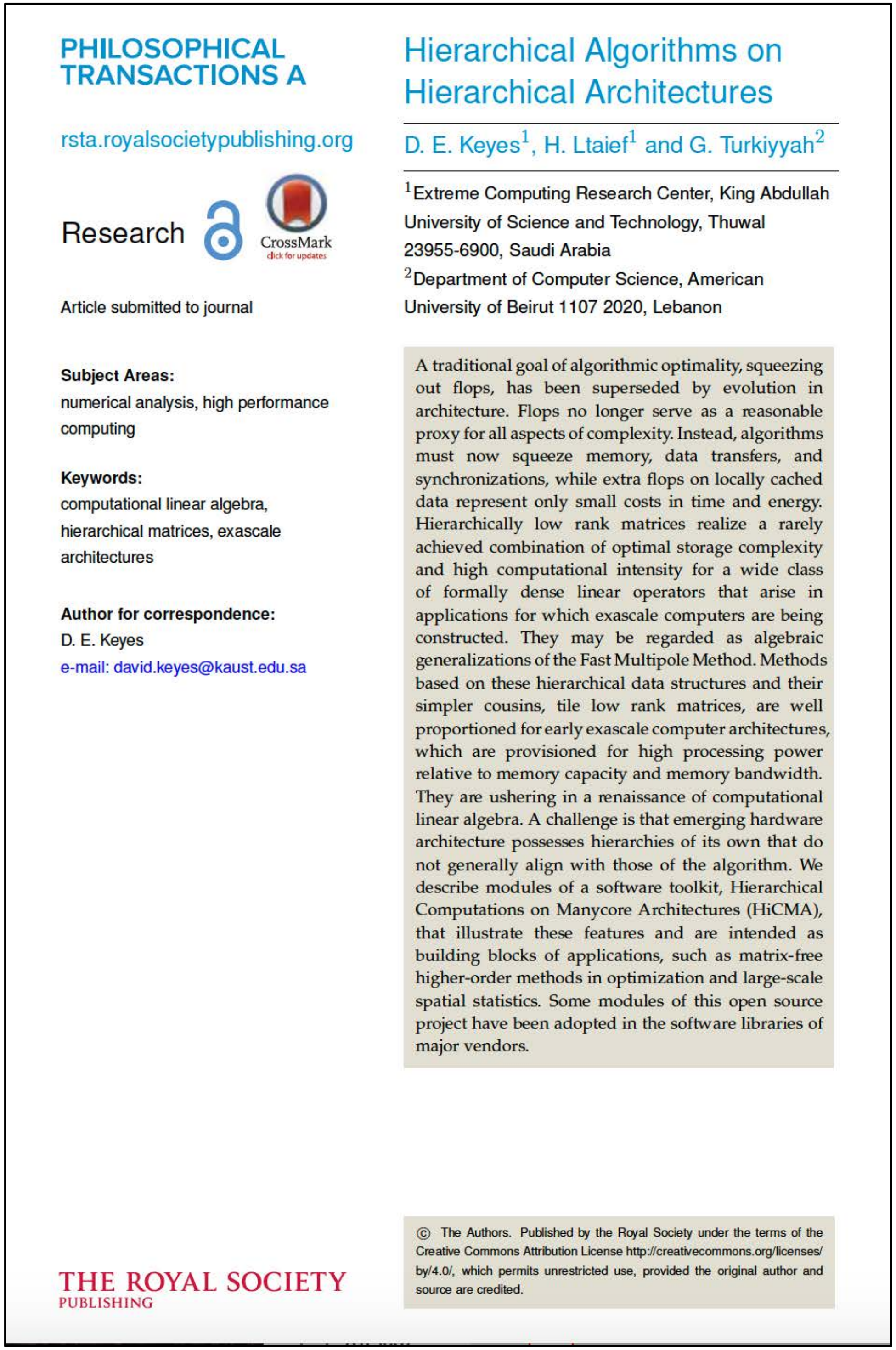
[*] Corresponding author.
E-mail address: omar.knio@kaust.edu.sa (O. Knio).

https://doi.org/10.1016/j.cma.2020.113191
0045-7825/© 2020 Elsevier B.V. All rights reserved.

Reference

K., Ltaief & Turkiyyah

Philosophical Transactions of the Royal Society Series A 378:20190055 (2020, open access)



PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

CrossMark
click for updates

Article submitted to journal

Subject Areas:
numerical analysis, high performance computing

Keywords:
computational linear algebra, hierarchical matrices, exascale architectures

Author for correspondence:
D. E. Keyes
e-mail: david.keyes@kaust.edu.sa

Hierarchical Algorithms on Hierarchical Architectures

D. E. Keyes[1], H. Ltaief[1] and G. Turkiyyah[2]

[1]Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia
[2]Department of Computer Science, American University of Beirut 1107 2020, Lebanon

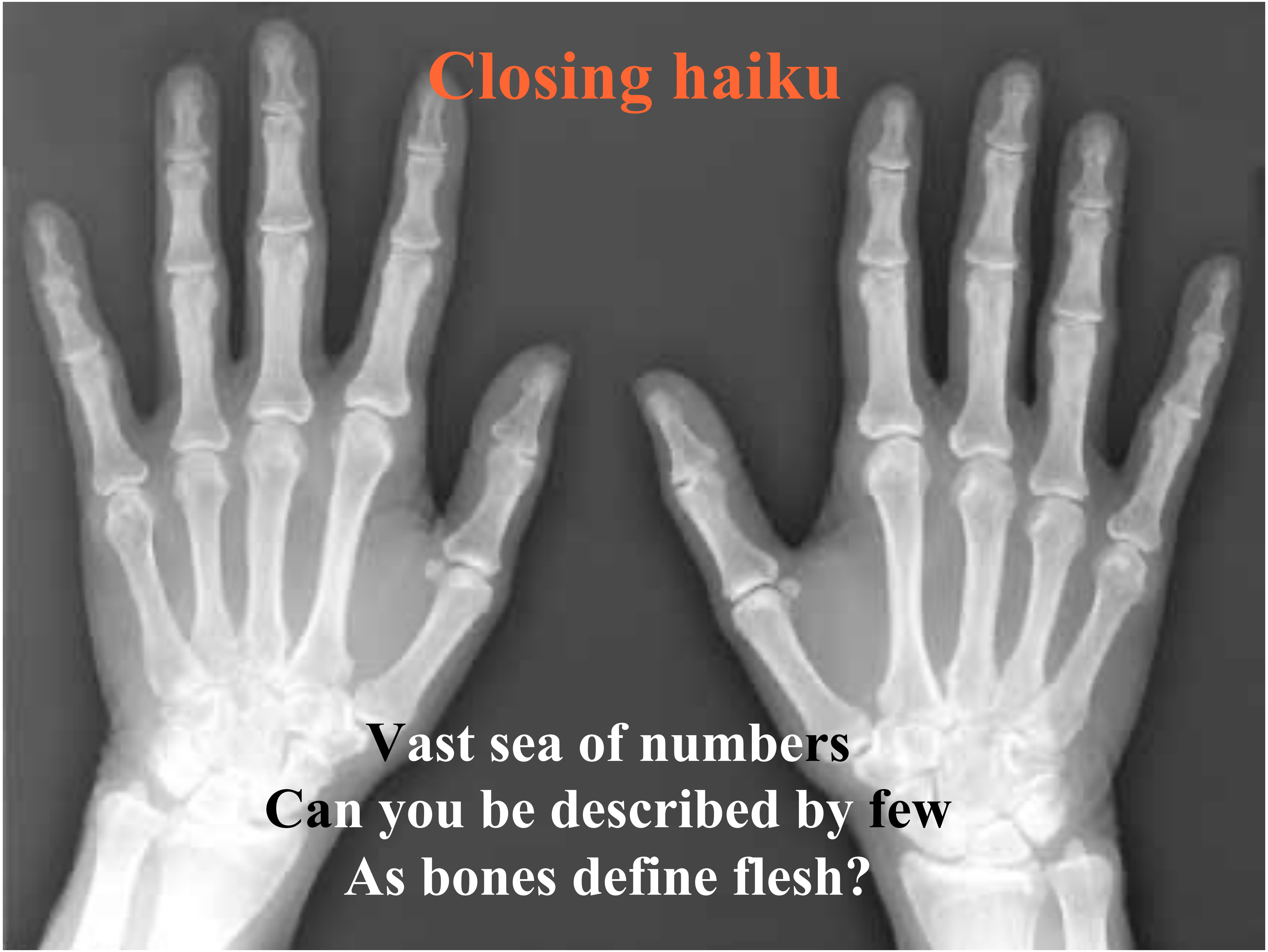
A traditional goal of algorithmic optimality, squeezing out flops, has been superseded by evolution in architecture. Flops no longer serve as a reasonable proxy for all aspects of complexity. Instead, algorithms must now squeeze memory, data transfers, and synchronizations, while extra flops on locally cached data represent only small costs in time and energy. Hierarchically low rank matrices realize a rarely achieved combination of optimal storage complexity and high computational intensity for a wide class of formally dense linear operators that arise in applications for which exascale computers are being constructed. They may be regarded as algebraic generalizations of the Fast Multipole Method. Methods based on these hierarchical data structures and their simpler cousins, tile low rank matrices, are well proportioned for early exascale computer architectures, which are provisioned for high processing power relative to memory capacity and memory bandwidth. They are ushering in a renaissance of computational linear algebra. A challenge is that emerging hardware architecture possesses hierarchies of its own that do not generally align with those of the algorithm. We describe modules of a software toolkit, Hierarchical Computations on Manycore Architectures (HiCMA), that illustrate these features and are intended as building blocks of applications, such as matrix-free higher-order methods in optimization and large-scale spatial statistics. Some modules of this open source project have been adopted in the software libraries of major vendors.

THE ROYAL SOCIETY
PUBLISHING

Very special thanks to…

George Turkiyyah
Visiting Professor
American University of Beirut

Closing haiku



Vast sea of numbers
Can you be described by few
As bones define flesh?

Thank you!

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

شكرا

david.keyes@kaust.edu.sa