

David Keyes & the HiCMA Team
Extreme Computing Research Center (ECRC)
King Abdullah University of Science and Technology

9 September 2020



Data-sparse Linear Algebra
for Large-scale Applications
on Emerging Architectures



Happy Dennis Ritchie's birthday

(born 9 September 1941)



Dennis Ritchie

1941-2011

The



Bell Labs
Programming Language
UNIX
(1969-1973)

“C is quirky, flawed, and an enormous success.”

“The number of UNIX installations has grown to 10, with more expected...”

E-NLA themes

We resonate with several series themes to date:

- low-rank approximation (last 4 talks)
- communication reduction (2 talks)
- randomization (2 talks)
- architectural adaptation (implicit in several talks, explicit here)
 - processor and memory heterogeneity
 - exploitation of low precision
 - high-performance ML-oriented SIMD instructions

These working slides (~8.5MB) are at <https://github.com/ecrc/h2opus/doc>
as “ENLA_20200909.pdf”

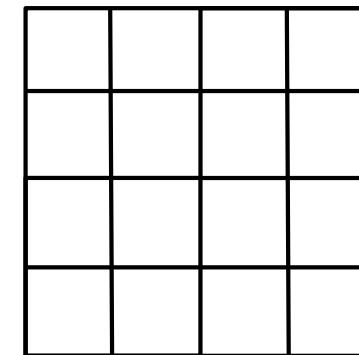
An improved set will be posted at the E-NLA site after the Q&A.

Conclusions, up front

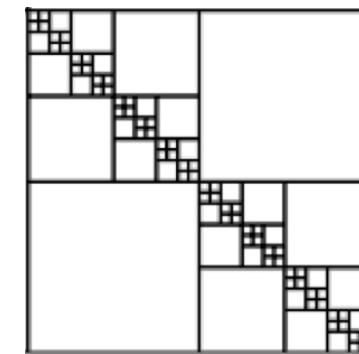
- With controllable trade-offs, many linear algebra operations adapt well to high performance on emerging architectures through
 - higher residence on the memory hierarchy
 - greater SIMD-style concurrency
 - reduced synchronization and communication
- Rank-structured matrices, based on uniform tiles 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

Rank-structured matrices* – 2 types herein

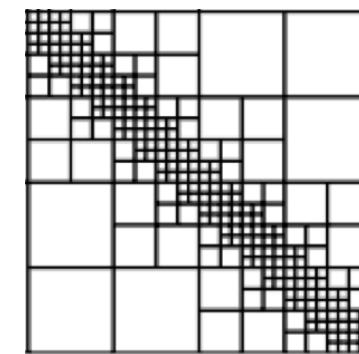
- **Tile low rank (TLR)**
 - all blocks at a single level of subdivision
(could in principle vary in size)
- **Hierarchically low rank (HLR)**
 - blocks at left at various levels upon recursive subdivision
 - weak and strong “admissibility” variants
- **HLR more than two decades old**
 - Hackbusch (1999), Tyrtyshnikov (2000)
 - Fiedler (1993) defined “structure ranks”
- **Prevalent topic in recent SIAM ALA conferences (4 MS at HKBU mtg)**



TLR



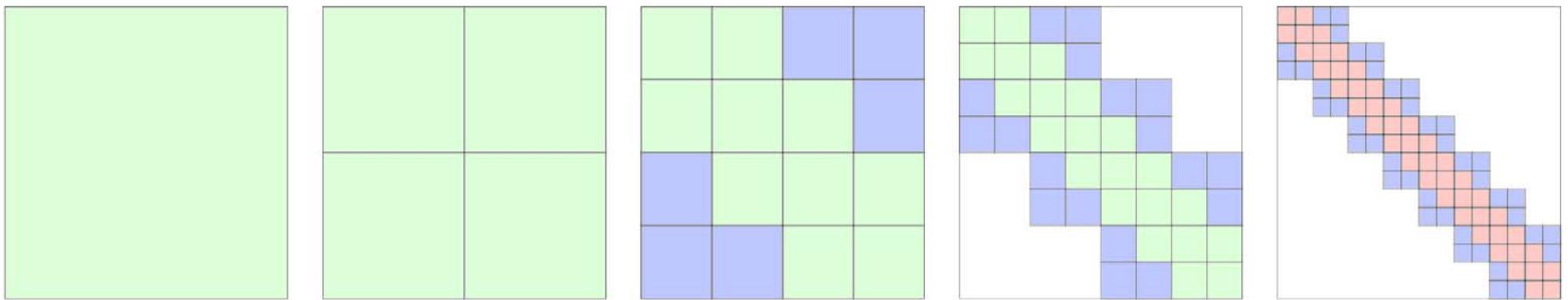
HLR
weakly
admissible



HLR
strongly
admissible

* A rank-structured matrix is a matrix with enough low-rank blocks that it pays to take advantage of them (paraphrasing Wilkinson on sparse matrices)

Conceptualization of \mathcal{H} -matrix construction



Step 0

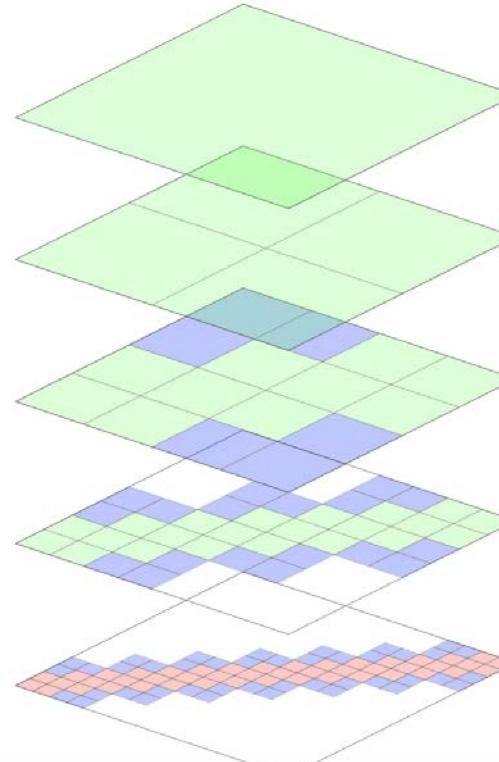
Step 1

Step 3

Step 4

Specify two parameters:

- Block size acceptably small to handle densely
- Rank acceptably small to represent a block



Until each block is acceptably small:

- Is rank acceptably small?
- If not, subdivide block

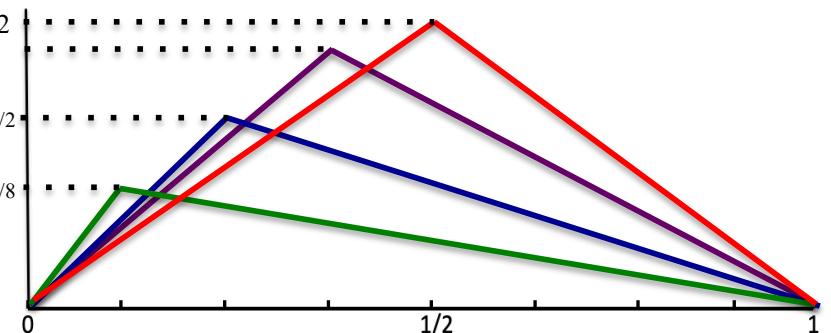
Take union of leaf blocks

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

Example: 1D Laplacian

$$A = \left[\begin{array}{ccc|c} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & \\ \hline & & -1 & \\ & -1 & & \end{array} \right] \leftrightarrow \left[\begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right] \left[\begin{array}{cccc} -1 & 0 & 0 & 0 \end{array} \right]$$

$$A^{-1} = \frac{1}{8} \times \left[\begin{array}{ccccc|cc} 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 12 & 10 & 8 & 6 & 4 & 2 \\ 5 & 10 & 15 & 12 & 9 & 6 & 3 \\ 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 15 & 10 & 5 \\ 2 & 4 & 6 & 8 & 10 & 12 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \right] \leftrightarrow \left[\begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right] \left[\begin{array}{cccc} 4 & 3 & 2 & 1 \end{array} \right]$$



What types of matrices are candidates?

Matrices arising from

- covariances
- integral equations with displacement kernels
- Schur complements within discretizations of elliptic and parabolic PDEs
- Hessians from PDE-constrained optimization
- fractional differential equations
- radial basis functions from unstructured meshing
- kernel matrices from machine learning applications

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 $O(k^{0.5} N^{2.5})$
 - 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

Also relevant to sparse problems

Classical factorizations fill in with elimination

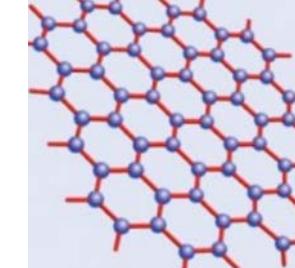
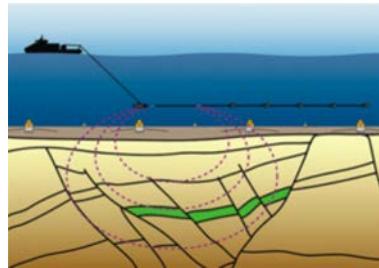
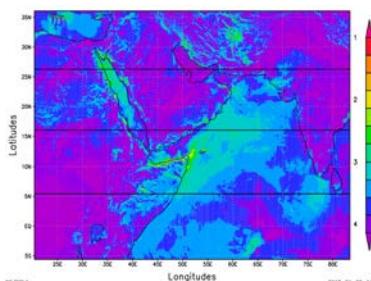
For 3D Poisson solver on a cube with $O(N)$ degrees of freedom:

- Classical nested dissection generally requires $O(N^2)$ operations
- Tile low-rank can yield $O(N^{4/3})$
(Amestoy, Buttari, L'Excellent & Mary, *SISC*, 2016)
- Hierarchically low-rank methods can yield $O(N)$
(Bebendorf & Hackbusch, *Numer. Math.*, 2003)
- Gains come from low-rank treatment of the resulting Schur complements

What kinds of applications?

Applications that possess

- memory capacity constraints (e.g., geospatial statistics, PDE-constrained optimization)
- energy constraints (e.g., remote telescopes)
- real-time constraints (e.g., wireless commun.)
- running time constraints (e.g., chem, materials, genome-wide association studies (GWAS))



Geospatial statistics motivation



“Increasing amounts of data are being produced by remote sensing instruments and numerical models, while techniques to handle millions of observations have historically lagged behind. [...] computational implementations that work with irregular observations are still rare.”

- Dorit Hammerling, NCEP



$1M \times 1M$ dense sym DP matrix requires 4 TB, $N^3 \sim 10^{18}$ Flops

Traditional approaches:

- Global low rank
- Zero outer diagonals

Better approaches:

- Hierarchical low rank
- Reduced precision outer diagonals

Overall motivations

- **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 roles: direct and randomized floating point kernels, tree-traversal from FMM, task-based programming, etc.
- **Computer architecture requirement**
 - Emerging architectures are met on their terms: limited fast memory per core, SIMD instructions, etc.
- **Application opportunities (as cited)**

Some “universals” of exascale computing

Architectural imperatives

- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-style batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages

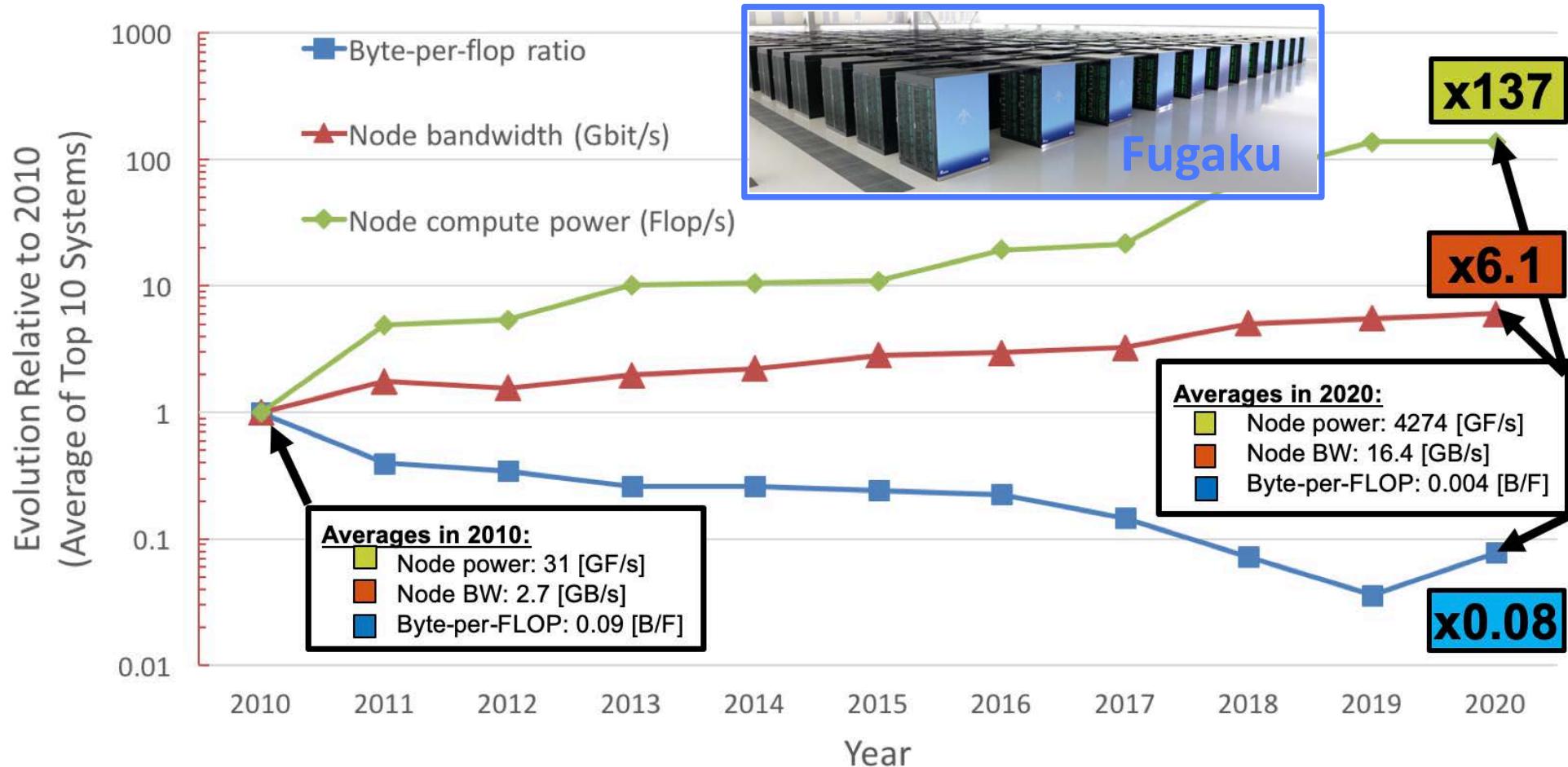
Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

HPL Top 10 bandwidth trends, 2010-2020



Keren Bergman's lab at Columbia has been tracking architectural trends in memory and networking interconnects for two decades. This slide is updated for Fugaku.

The last three #1 systems

TaihuLight (Nov 2017) B/F = 0.004

Summit (June 2018) B/F = 0.0005

Fugaku (June 2020) B/F = 0.303

Algorithmic opportunity

To achieve the potential of emerging architectures, we need implementations of

- linear solvers
- least squares solvers
- eigensolvers
- singular value solvers

that

- offer tunable accuracy-time-space tradeoffs
- exploit hierarchy of precisions
- *may* require more flops but complete earlier, thanks to more concurrency

Two universes of NLA exist side-by-side



* Global indices *

```
do i {  
    do j {  
        for (i,j) in S do op  
    }  
}
```

* Local indices *

for matrix blocks (k,l)

```
do i {  
    do j {  
        for (i,j) in  $S_{k,l}$  do op  
    }  
}
```

Algorithms were once flat (Cholesky, 1910)

Sur la résolution numérique
des systèmes d'équations linéaires

A. Cholesky

La solution des problèmes dépendant de données expérimentales, qui peuvent dans certains cas être soumises à des conditions, et auxquelles on applique la méthode des méthodes casuelles est toujours subordonnée au calcul numérique des racines d'un système d'équations linéaires. C'est le cas de la recherche des lois physiques, c'est aussi le cas de la compensation de réseaux géodésiques. Il est donc intéressant de rechercher un moyen simple et aussi simple que possible d'effectuer la résolution numérique d'un système d'équations linéaires.

Le procédé que nous allons indiquer s'applique aux systèmes d'équations symétriques auxquelles, contrairement à ce qu'il se passe dans les casuels, nous renonçons tout d'abord que la résolution d'un système de n équations linéaires à n inconnues peut très facilement se ramener à la résolution d'un système de n équations linéaires symétriques à n inconnues.

Considérons en effet le système suivant

I

$$\begin{cases} A_1^1 Y_1 + A_2^1 Y_2 + A_3^1 Y_3 + \dots + A_n^1 Y_n + C_1 = 0 \\ A_2^1 Y_1 + A_2^2 Y_2 + A_3^2 Y_3 + \dots + A_n^2 Y_n + C_2 = 0 \\ \vdots \\ A_n^1 Y_1 + A_n^2 Y_2 + \dots + A_n^n Y_n + C_n = 0 \end{cases}$$

Effectuons la transformation linéaire représentée par le système :

II

$$\begin{cases} Y_1 = A_1^1 \lambda_1 + A_2^1 \lambda_2 + \dots + A_n^1 \lambda_n \\ Y_2 = A_2^1 \lambda_1 + A_2^2 \lambda_2 + \dots + A_n^2 \lambda_n \\ \vdots \\ Y_n = A_n^1 \lambda_1 + A_n^2 \lambda_2 + \dots + A_n^n \lambda_n \end{cases}$$

Le système I d'équations donne n inconnues Y_j à être remplacé par le système III donnant n inconnues λ_j permettant, à l'aide de II, de calculer les valeurs de Y_j .

III

$$\begin{cases} A_1^1 \lambda_1 + A_2^1 \lambda_2 + \dots + A_n^1 \lambda_n + C_1 = 0 \\ A_1^2 \lambda_1 + A_2^2 \lambda_2 + \dots + A_n^2 \lambda_n + C_2 = 0 \\ \vdots \\ A_1^n \lambda_1 + A_2^n \lambda_2 + \dots + A_n^n \lambda_n + C_n = 0 \end{cases}$$

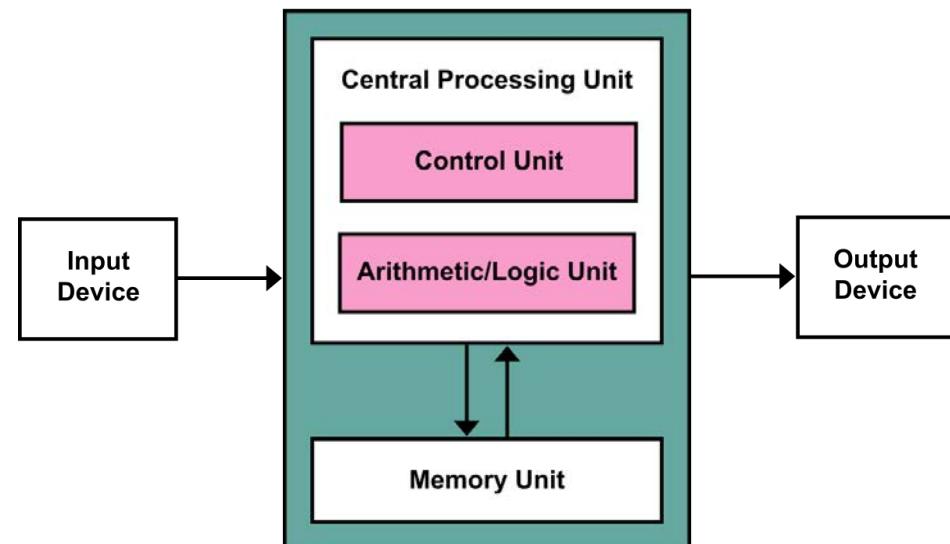
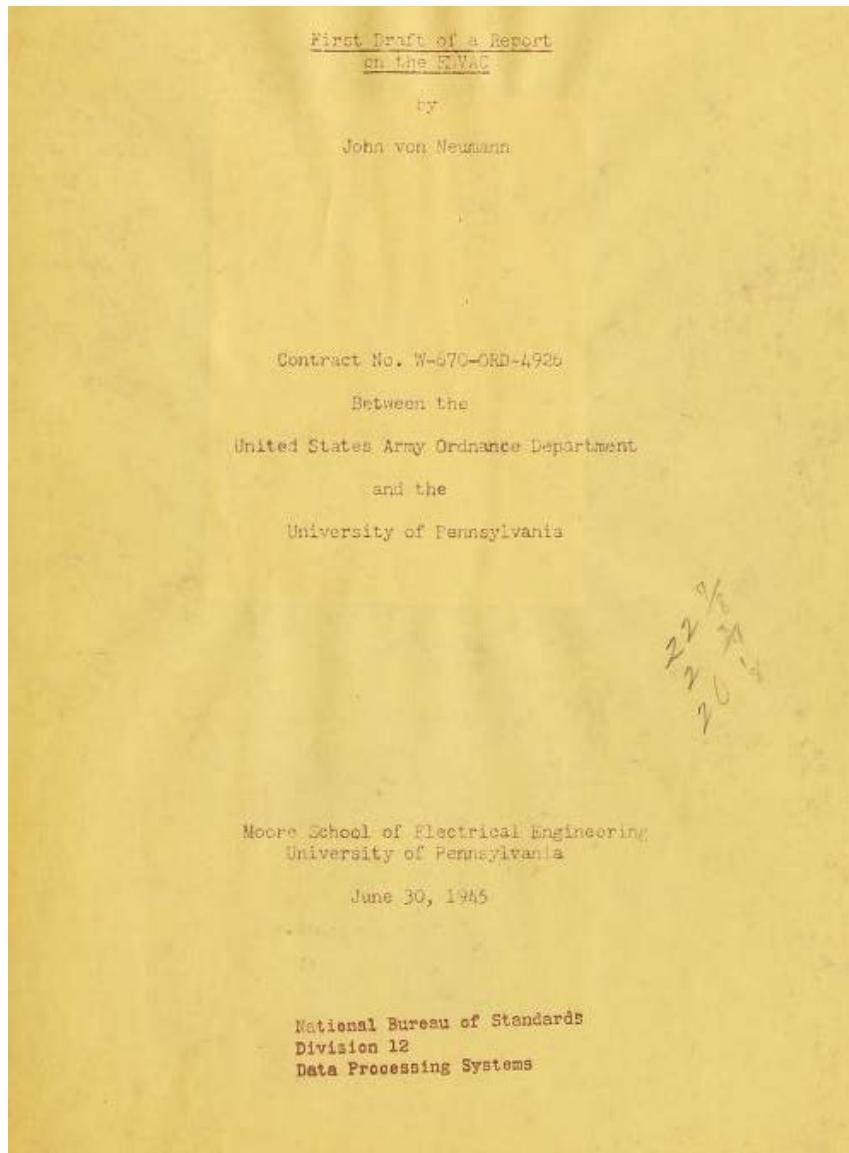

```

 $\ell_{11} = \sqrt{a_{11}}$ ;
for  $j = 2, \dots, n$  do
|    $\ell_{j1} = a_{j1}/\ell_{11}$ ;
end
for  $i = 2, \dots, n-1$  do
|    $\ell_{ii} = (a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2)^{1/2}$ ;
|   for  $j = i+1, \dots, n$  do
|   |    $\ell_{ji} = (a_{ji} - \sum_{k=1}^{i-1} \ell_{jk} \ell_{ik}) / \ell_{ii}$ ;
|   end
|    $\ell_{nn} = (a_{nn} - \sum_{k=1}^{n-1} \ell_{nk}^2)^{1/2}$ ;
end

```

triangular recurrence

Architectures were flat, as well (vN, 1945)



One hierarchy is not so bad...

As humans managing implementation complexity,
we would prefer:

- hierarchical algorithms on flat architectures
or even (suboptimally)
- flat algorithms on hierarchical architectures

... but two independent hierarchies may not match

- need to marshal irregular structures into uniform batches *and/or*
- feed dynamic runtime queues

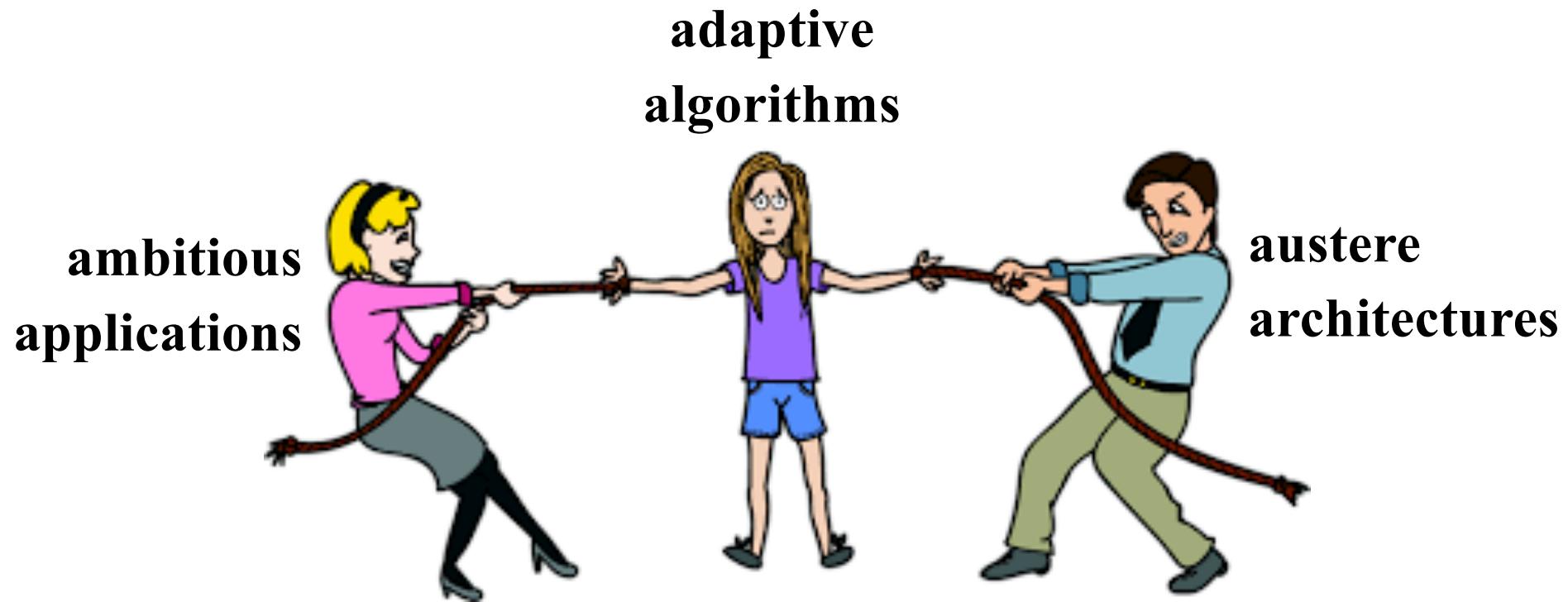
Hierarchies may not perfectly match, but...

We go to exascale with the architectures we have, not with the architectures we want.

- First exascale Gordon Bell Prize awarded on the heterogeneous Summit system at ORNL (currently the #2 ranked system by HPL), with GPUs and Power9 cores
- A 4,000-node subset of Summit sustained 1.88 ExaOp/s of mixed precision on a genomics (GWAS) application for the 2018 Prize
- Majority of these operations are half-precision (16-bit floating point) NVIDIA tensor-core matrix-matrix multiplies, 64 FP operations per instruction

Algorithmic philosophy

Algorithms must span a widening gulf ...

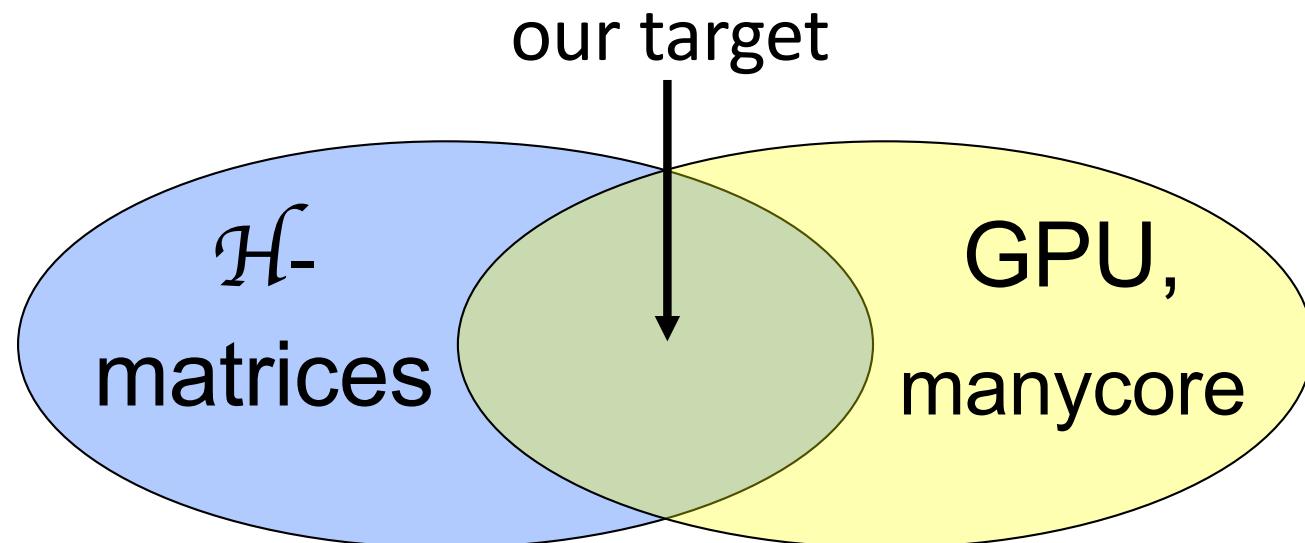


A full employment program
for algorithm developers 😊

Hierarchical algorithms and extreme scale

Must address the *tension* between

- highly uniform vector, matrix, and general SIMD 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 ☺

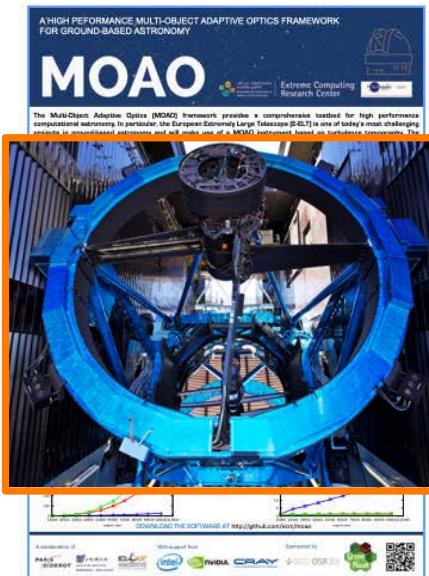
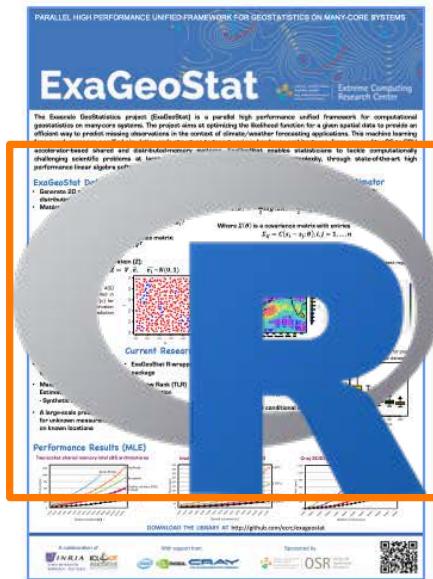
→ Billions of

€ £ \$ ¥

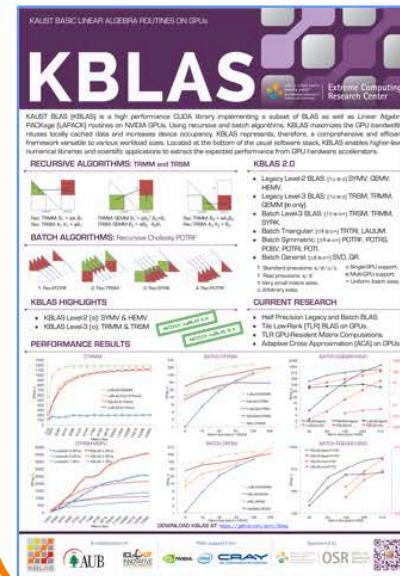
of scientific software worldwide hangs in the balance until our algorithmic infrastructure evolves to span the architecture-applications gap

– meet hierarchy with hierarchy

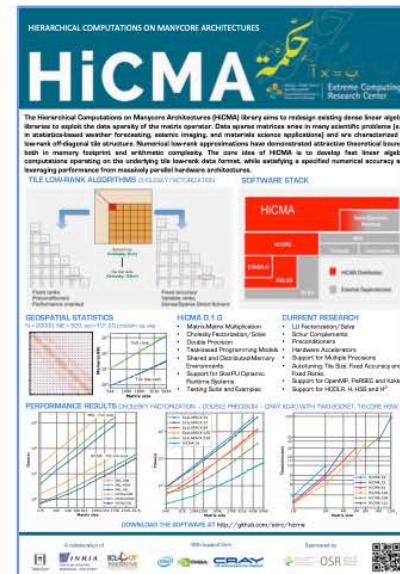
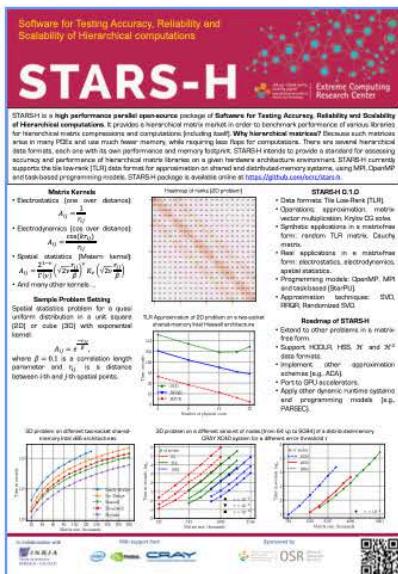
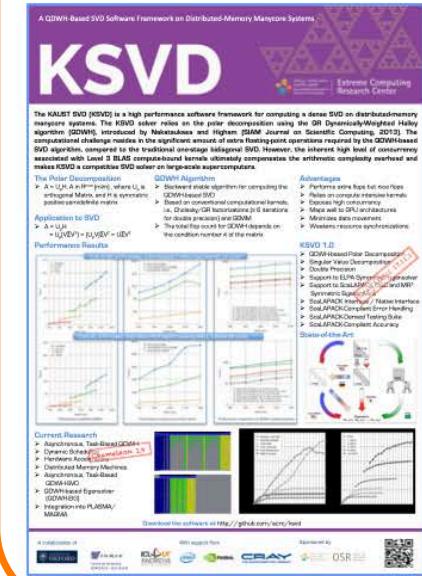
Available at <https://github.com/ecrc/>



in NVIDIA cuBLAS



in Cray LibSci



H2Opus
github.com/ecrc/h2opus/

Today introducing H2Opus

an 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 approximate inverses

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

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]

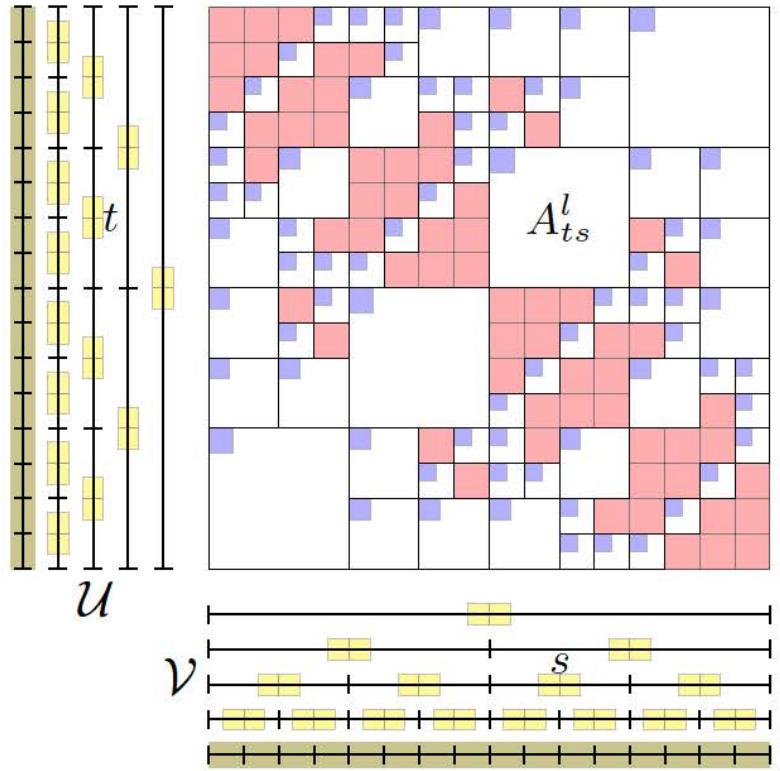
[1] Ambikasaran and Darve, *Journal of Scientific Computing*, 2013

[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

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

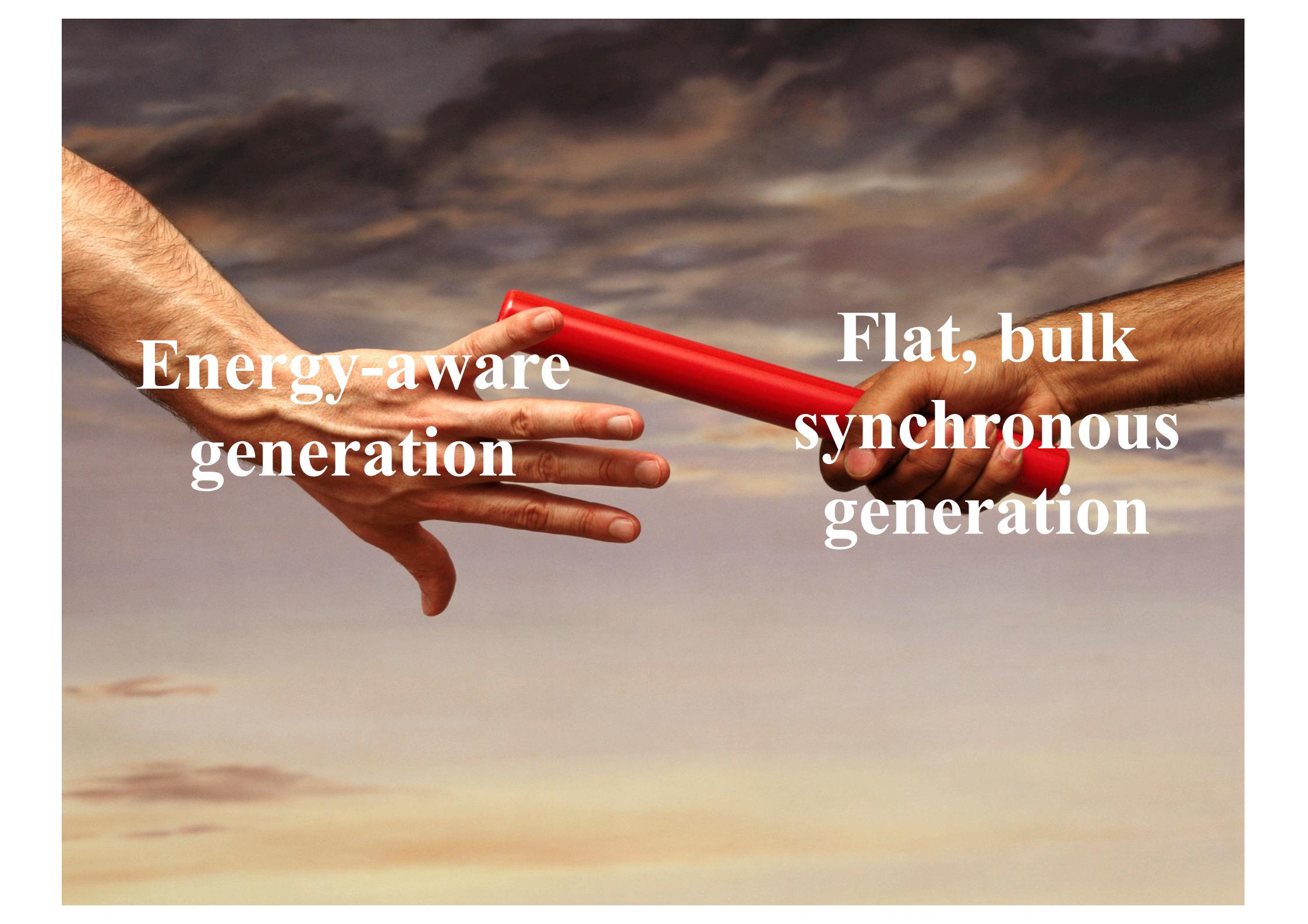


$$\begin{bmatrix} U_t \end{bmatrix} = \begin{bmatrix} U_{t_1} \\ & U_{t_2} \end{bmatrix} \begin{bmatrix} E_{t_1} \\ \vdots \\ E_{t_2} \end{bmatrix}$$

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

Some other LA software leveraging data sparsity

Package	Technique(s)	Format	Author (year)
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)

A photograph of two hands reaching across a cloudy, orange-tinted sky to hold a single red relay baton. The hand on the left is white, and the hand on the right is darker-skinned. The baton is positioned horizontally, symbolizing teamwork and energy transfer.

Energy-aware
generation

Flat, bulk
synchronous
generation

HiCMA design strategies

- 1) Employ dynamic runtime systems based on directed acyclic task graphs (DAGs)
 - e.g., PaRSEC, Quark, StarPU, Charm++, Legion, OmpSs, HPX
 - dynamic scheduling capabilities in OpenMP
- 2) Co-design libraries to diverse architectures while presenting high-level application programmer interface
- 3) Exploit data sparsity of rank-structured type (TLR & HLR)
 - meet the “curse of dimensionality” with the “blessing of low rank”

Most of
the talk
spent
here

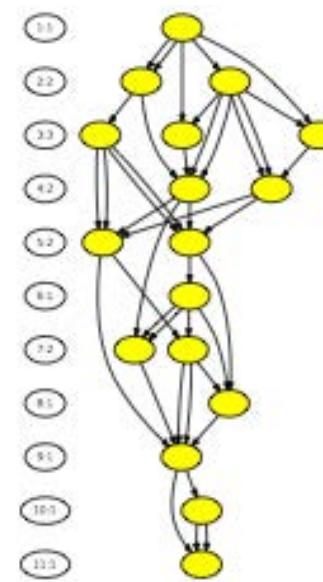
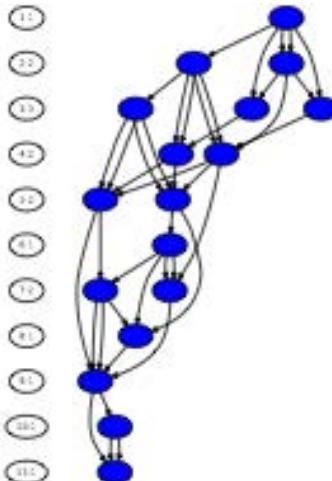
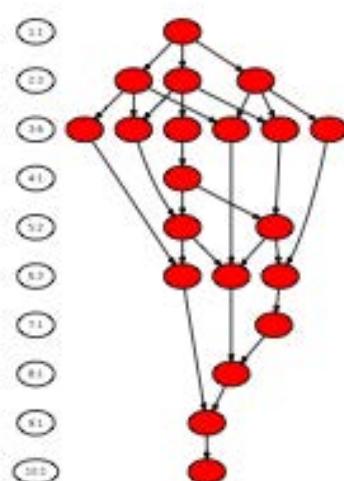
1) Taskification based on DAGs

- **Advantages**
 - ◆ remove artifactual synchronizations in the form of subroutine boundaries
 - ◆ remove artifactual orderings in the form of pre-scheduled loops
 - ◆ expose more concurrency
- **Disadvantages**
 - ◆ pay overhead of managing task graph
 - ◆ potentially lose some memory locality

1) Reduce over-ordering and synchronization through DAGs, ex.: generalized eigensolver

$$Ax = \lambda Bx$$

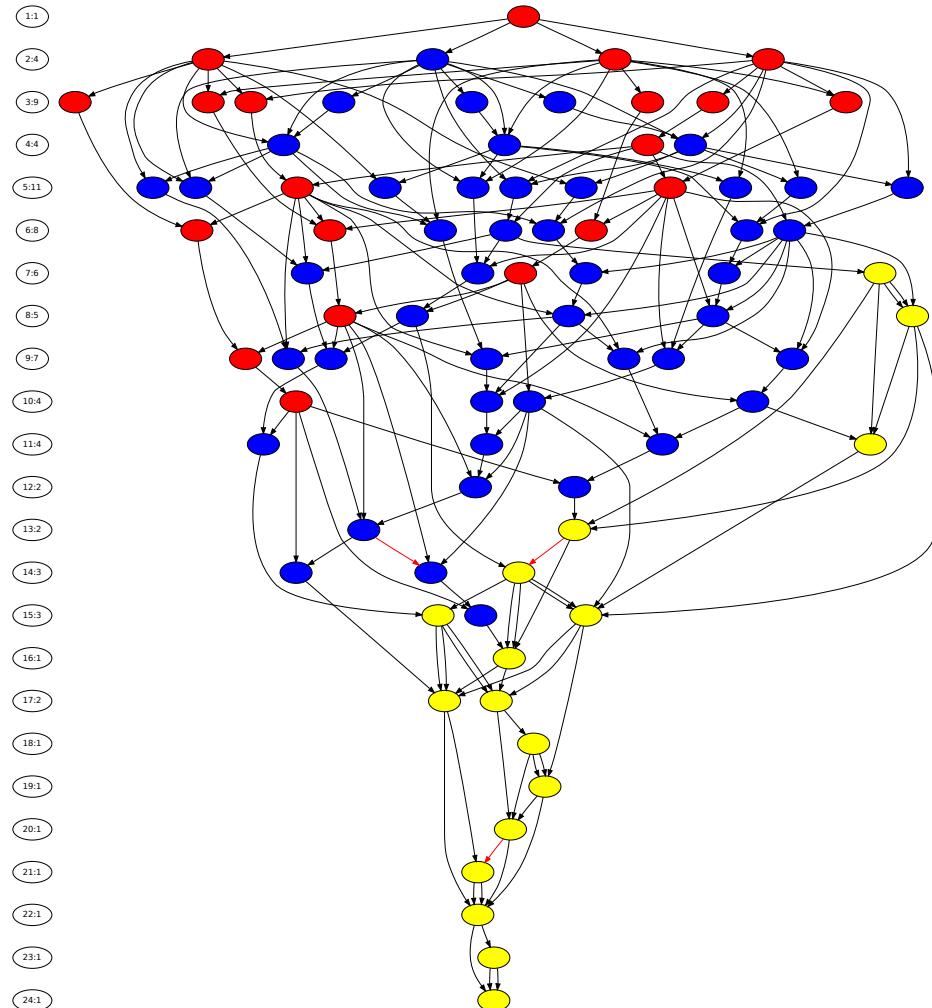
Operation	Explanation	LAPACK routine name
① $B = L \times L^T$	Cholesky factorization	POTRF
② $C = L^{-1} \times A \times L^{-T}$ or HEGST	application of triangular factors	SYGST
③ $T = Q^T \times C \times Q$	tridiagonal reduction	SYEVD or HEEVD
④ $Tx = \lambda x$	QR iteration	STERF



Ltaief, Luszczek, Haidar & Dongarra, *Solving the Generalized Symmetric Eigenvalue Problem using Tile Algorithms on Multicore Architectures*, Adv Parallel Comp, 2012

Loop nests and subroutine calls, with their over-orderings, can be replaced with DAGs

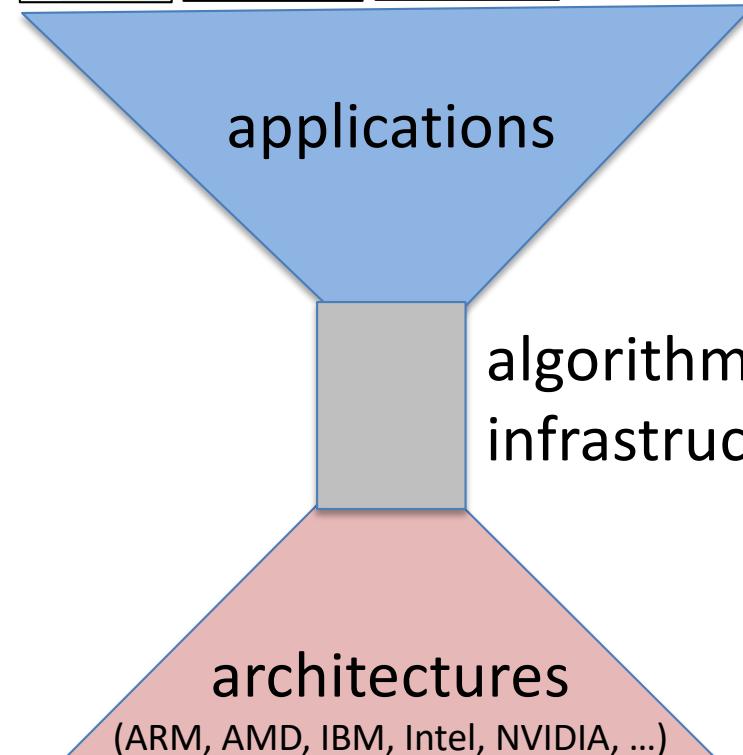
- Diagram shows a **dataflow ordering** of the steps of a 4×4 symmetric generalized eigensolver
- Nodes are tasks, color-coded by type, and edges are data dependencies
- Time is vertically downward
- Wide is good; short is good



2) Co-design to diverse architectures

- **Advantages**
 - ◆ tiling and recursive subdivision create large numbers of small problems that can be marshaled for batched operations on GPUs and MICs
 - amortize call overheads
 - polyalgorithmic approach based on block size
 - ◆ non-temporal stores, coalesced memory accesses, double-buffering, etc. reduce sensitivity to memory
- **Disadvantages**
 - ◆ code is more complex
 - ◆ code is architecture-specific at the bottom

“Hourglass” model for algorithms (borrowed from internet protocols)



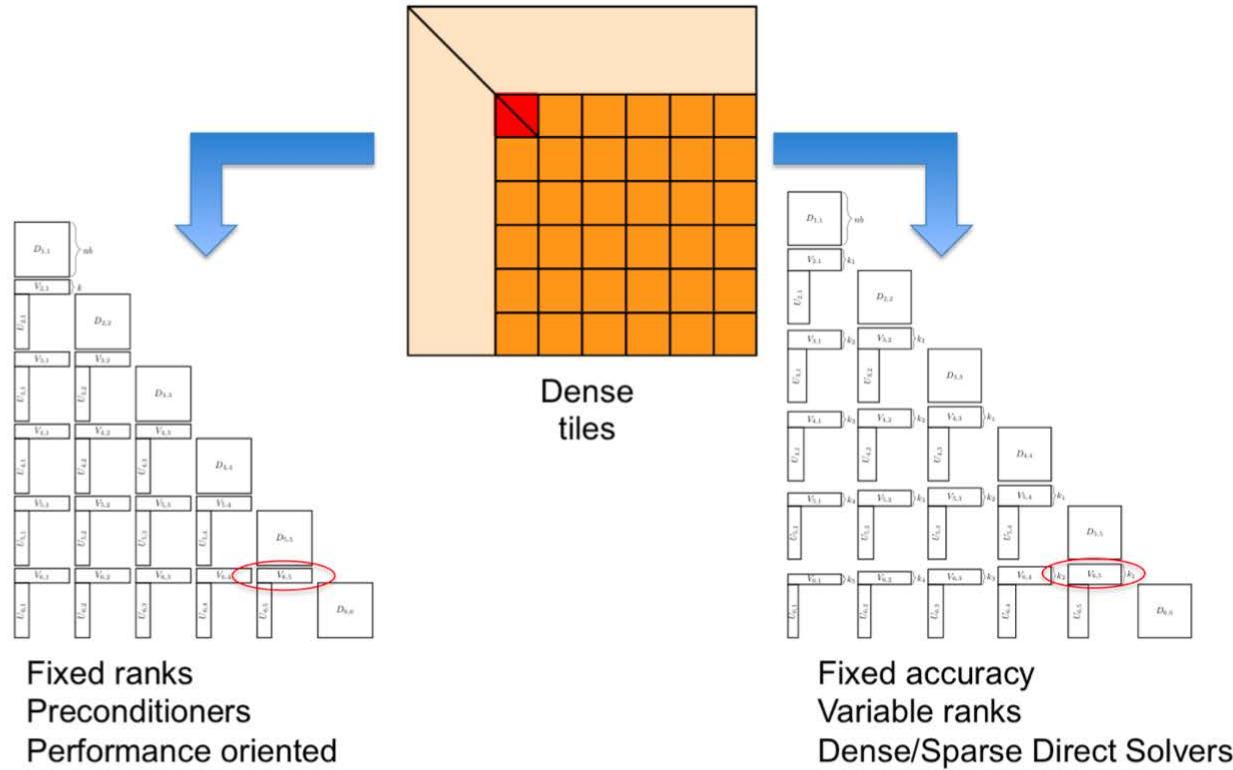
3) Rank-structured operators

- **Advantages**
 - ◆ shrink memory footprints to live higher on the memory hierarchy
 - higher means quick access (\uparrow arithmetic intensity)
 - ◆ reduce operation counts
 - ◆ tune work to accuracy requirements
 - e.g., preconditioner versus solver
- **Disadvantages**
 - ◆ pay cost of (re-)compression
 - ◆ not all operators compress well

Reduce memory footprint and operation complexity with low rank

- 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 (Halko, Martinsson & Tropp, 2009) to form low-rank blocks
 - a flop-intensive GEMM-based flat algorithm
- Implement in “batches” of leaf blocks
 - flattening trees in the case of HLR

Tile Low Rank (TLR) is a compromise between optimality and complexity



T. Mary, PhD Dissertation, Block Low-Rank multifrontal solvers: complexity, performance, and scalability, 2017.

C. Weisberger, PhD Dissertation, Improving multifrontal solvers by means of algebraic Block Low-Rank representations, 2013.



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 admissibility 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 (via randomized sampling)**
 - 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

Programmed pause

Questions?

We resume with examples of TLR and HLR

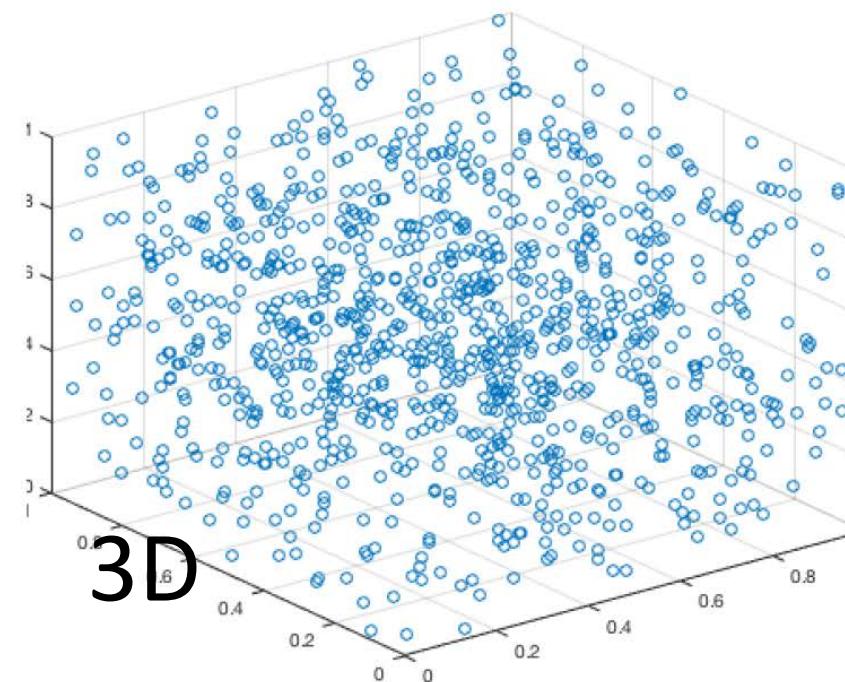
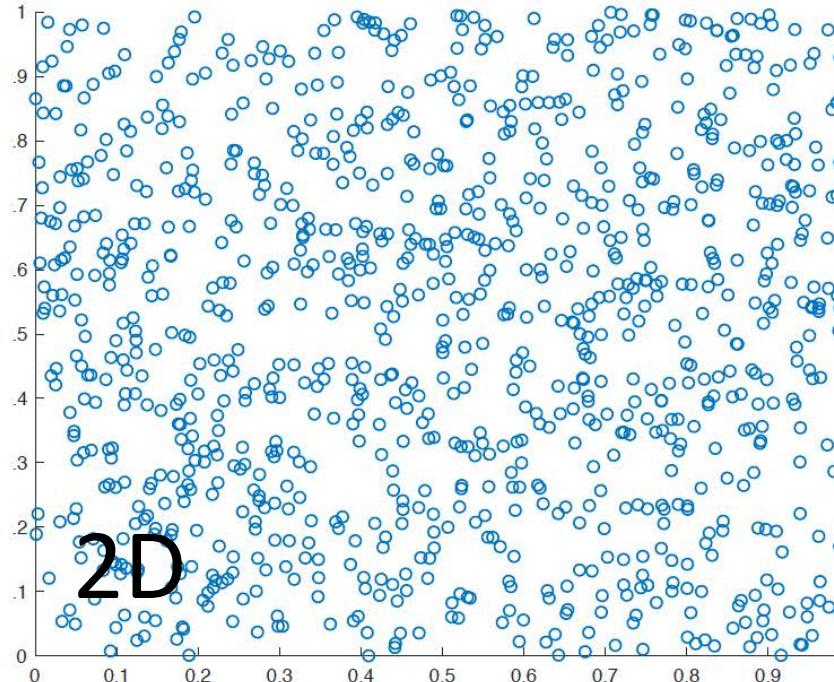
Caveat

- TLR and HLR 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

Geospatial statistics model problems

Input correlation matrix: random coordinate generation within the unit square or unit cube with Matérn kernel decay

- e.g., linear exp to square exp decay, $a_{ij} \sim \exp(-c|x_i - x_j|^2)$



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

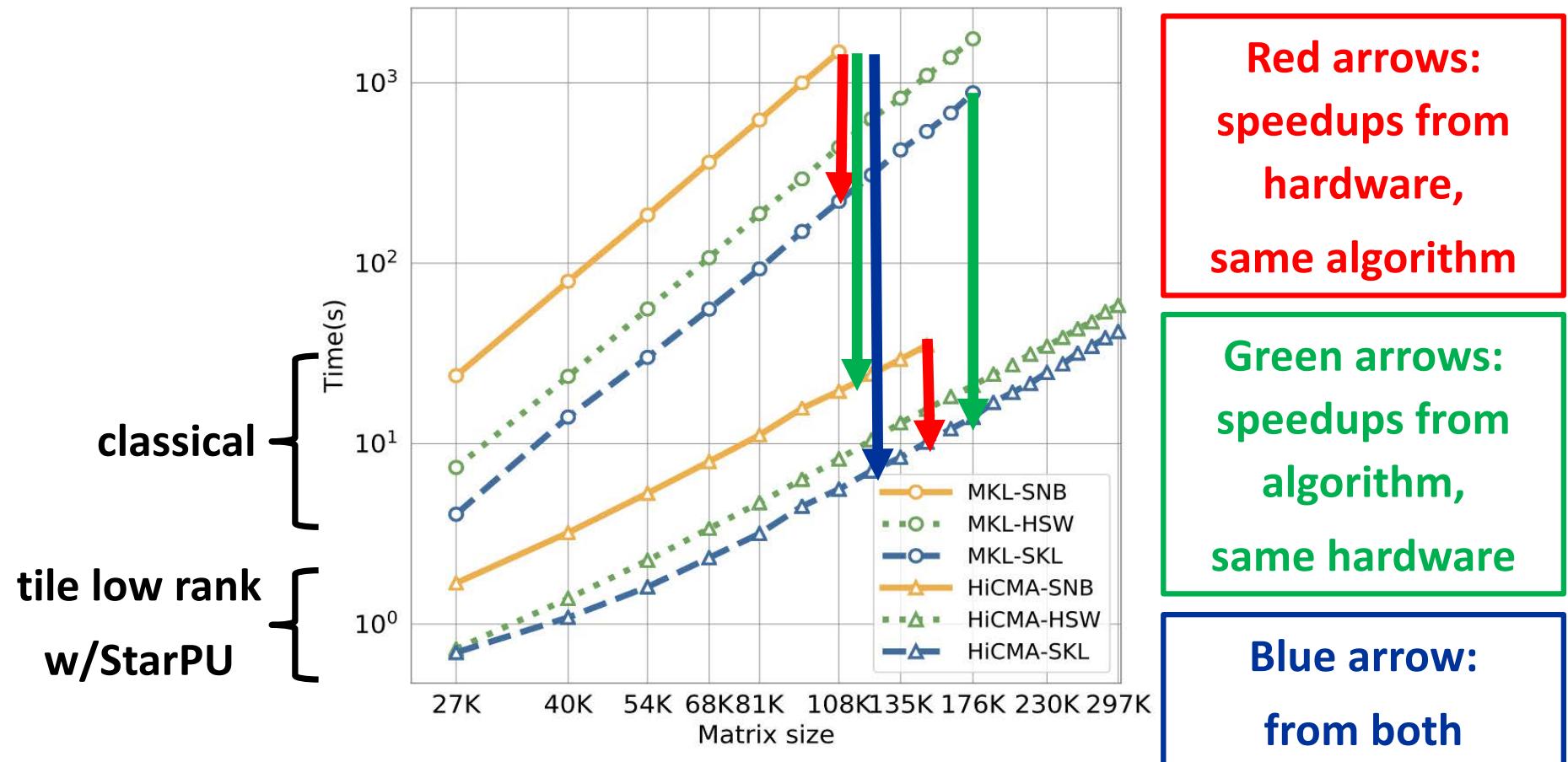
$$\ell(\boldsymbol{\theta}) = -\frac{1}{2} \mathbf{Z}^T \underset{\text{inverse}}{\Sigma^{-1}}(\boldsymbol{\theta}) \mathbf{Z} - \frac{1}{2} \log \underset{\text{determinant}}{|\Sigma(\boldsymbol{\theta})|}$$

- Determinant and inverse of Σ depend upon Cholesky, dominated by DPOTRF factorization routine (next slides)

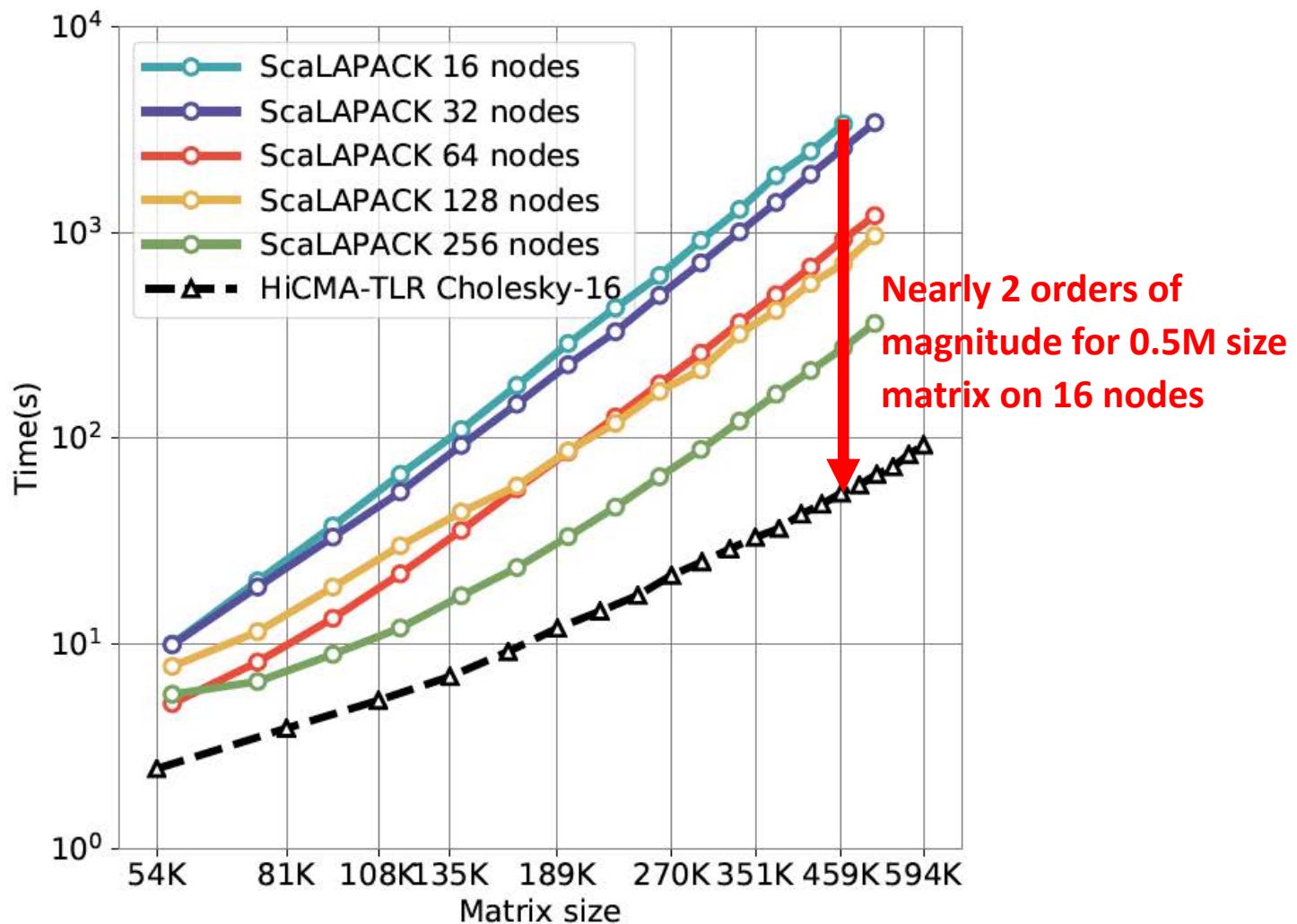
HiCMA TLR vs. Intel MKL on shared memory

Geospatial statistics (Gaussian kernel) to accuracy 1.0e-8

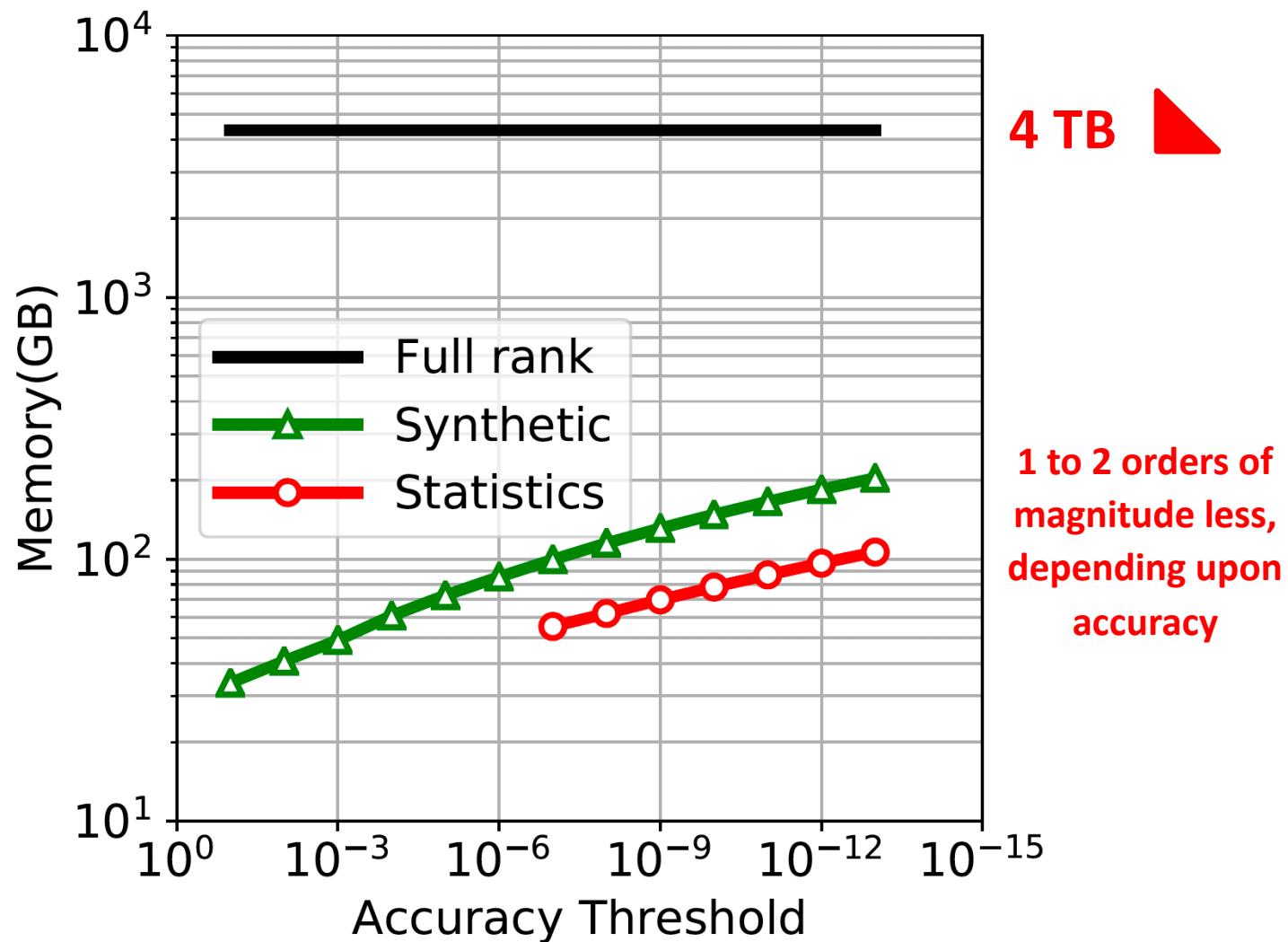
- Three generations of Intel manycore (Sandy Bridge, Haswell, Skylake)
- Two generations of linear algebra (classical dense and tile low rank)



HiCMA vs. ScaLAPACK on distributed memory



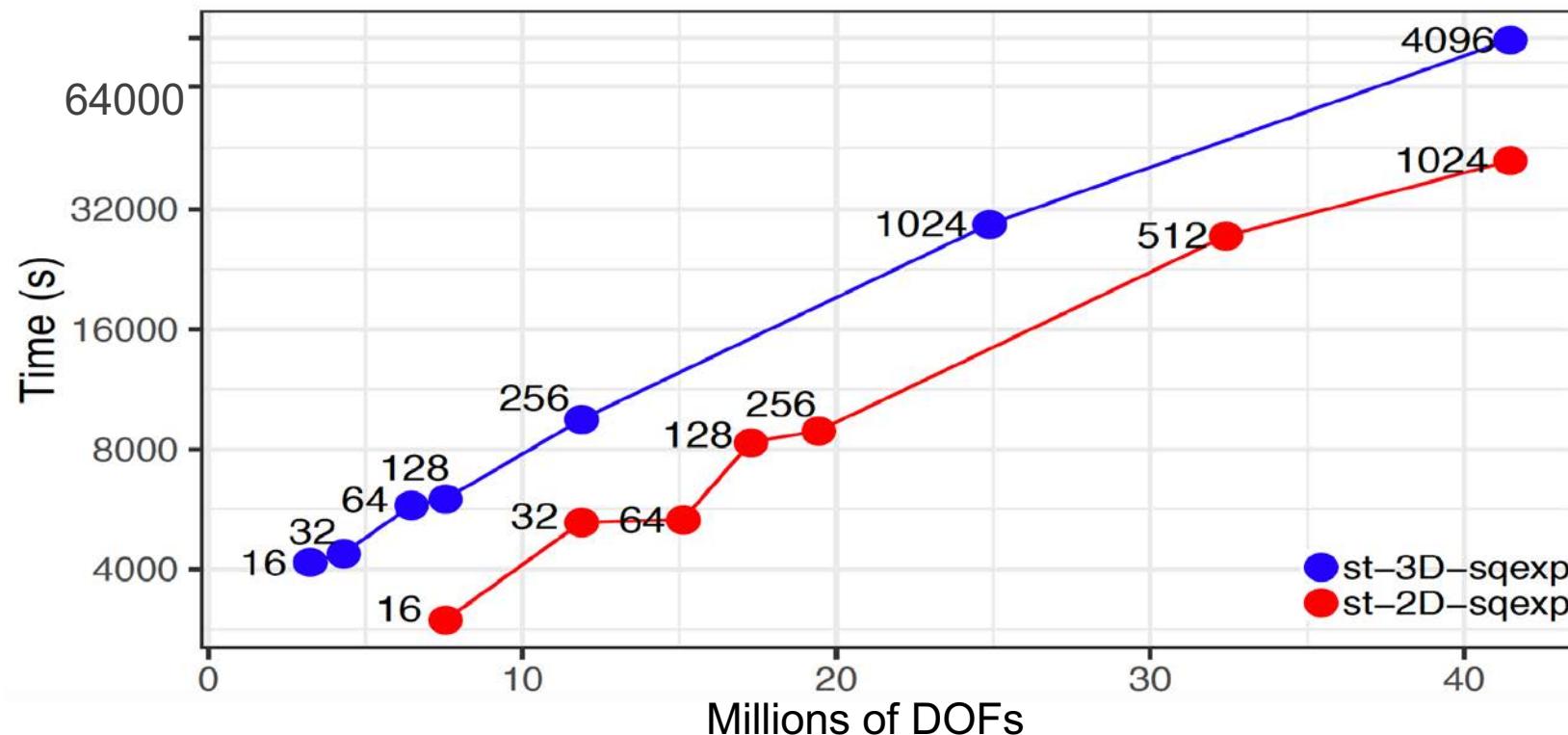
Memory footprint for DP matrix of size 1M



TLR *tour de force*

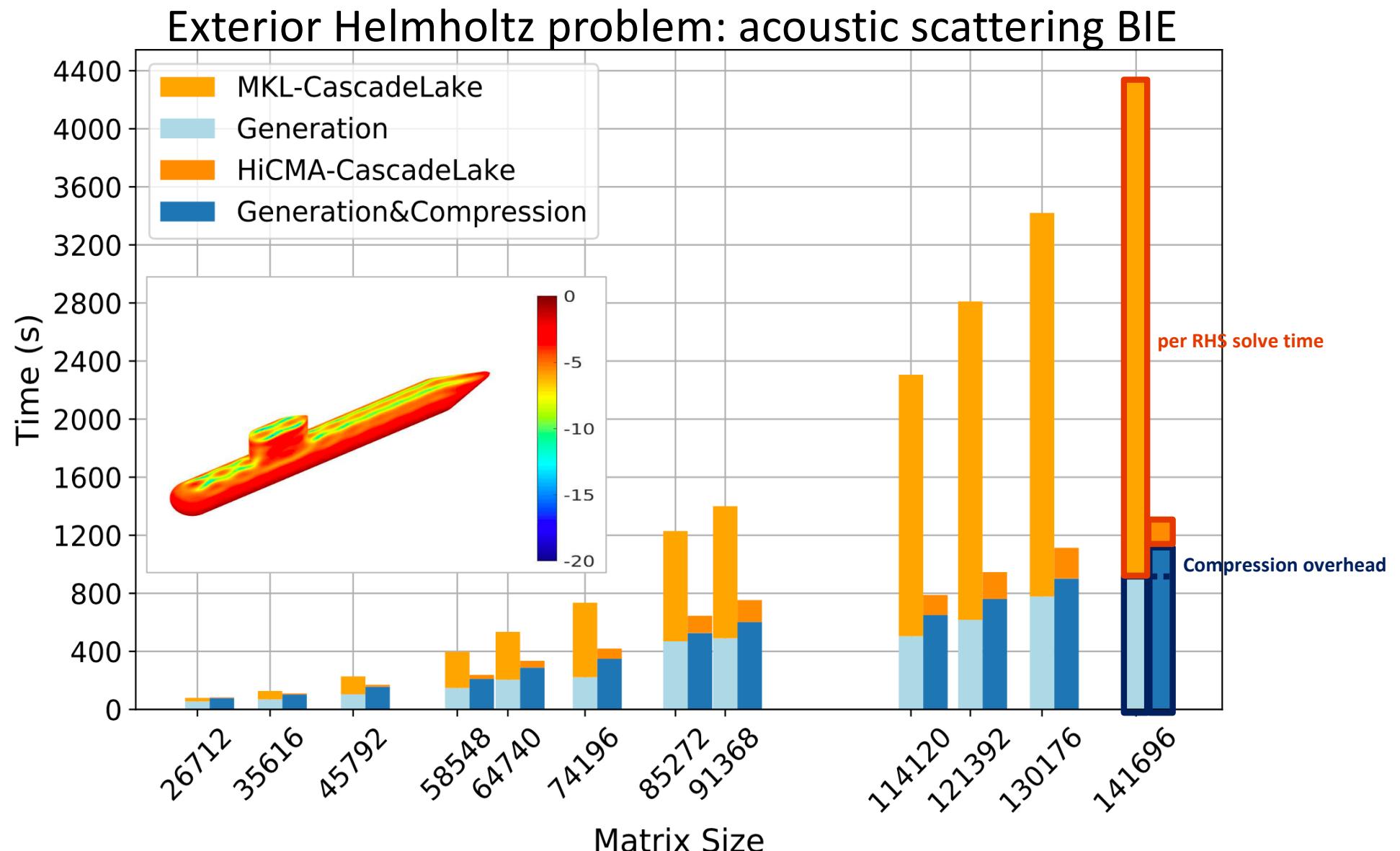
Cholesky factorization of a TLR matrix (DPOTRF) derived from Gaussian covariance of random distributions, up to 42M DOFs, on up to 4096 nodes (131,072 Haswell cores) of a Cray XC40

- would require 14.1 PetaBytes in dense DP
- would require 77 days by ScaLAPACK (at the TLR rate of 3.7 Pflop/s)



Cao, Pei, Akbudak, Mikhalev, Bosilca, Ltaief, K. & Dongarra, *Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications*. PASC '20 (ACM), 2020

Compress (once) on the fly, solve many with HLU



Reference



Al-Harthi, Alomairy,
Akbudak, Chen,
Ltaief, Bagci & K.

*Lecture Notes in
Computer Science*
12151:209
(2020, open access)

Solving Acoustic Boundary Integral Equations Using High Performance Tile Low-Rank LU Factorization

Noha Al-Harthi, Rabab Alomairy^(✉), Kadir Akbudak, Rui Chen, Hatem Ltaief, Hakan Bagci, and David Keyes

Extreme Computing Research Center, Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, Thuwal, Jeddah 23955, Saudi Arabia
{Noha.Harthi,Rabab.Omairy,Kadir.Akbudak,Rui.Chen,Hatem.Ltaief,Hakan.Bagci,David.Keyes}@kaust.edu.sa

Abstract. We design and develop a new high performance implementation of a fast direct LU-based solver using low-rank approximations on massively parallel systems. The LU factorization is the most time-consuming step in solving systems of analyzing acoustic scattering from large objects. This is obtained by discretizing the boundary value problem using a higher-order Nyström method. The inherent data sparsity of the matrix allows for efficient low-rank approximations while still capturing the full solution. In particular, the proposed LU-based Rank (TLR) data compression format is designed for parallel Computations on Manycore Architectures. It reduces the complexity of “classical” dense direct solvers to O(n^2) order. We taskify the underlying boundary integral equation computations. We then employ the StarPU runtime system to orchestrate the scheduling of tasks on distributed-memory systems. The resulting solver is able to compensate for the load imbalance between tasks while mitigating the overhead of data movement. We evaluate our TLR LU-based solver and study the effect of different numerical accuracies. The new solver is compared to the state-of-the-art dense factorization solvers on various parallel systems, for analysis of both synthetic and real geometries.

Keywords: Tile low-rank LU-based solver · Acoustic scattering · Task-based parallelism · Dynamic runtime systems

© The Author(s) 2020
P. Sadayappan et al. (Eds.): ISC High Performance
https://doi.org/10.1007/978-3-030-50743-5_11



The Gauss Centre for Supercomputing (GCS) is pleased to award the **GCS AWARD 2020** to:

to

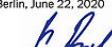
- Ms. Noha Al-Harthi
- Ms. Rabab Alomairy
- Dr. Kadir Akbudak
- Mr. Rui Chen
- Dr. Hatem Ltaief
- Dr. Hakan Bagci
- Dr. David Keyes

of King Abdullah University of Science and Technology (KAUST), Thuwal, KSA

for their outstanding scientific work, submitted for the ISC 2020 research paper session:

SOLVING ACOUSTIC BOUNDARY INTEGRAL EQUATIONS USING HIGH PERFORMANCE TILE LOW-RANK LU FACTORIZATION

Berlin, June 22, 2020

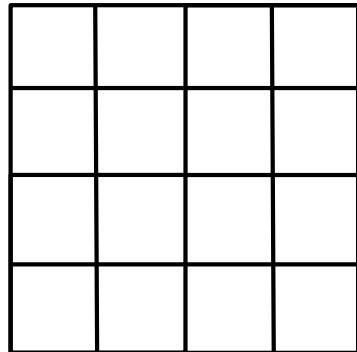
 

Prof. Dr.-Ing. Michael M. Resch
Chairman of the International GCS Award Committee
Vice Chairman of the Board of Directors, GCS

Dr. Claus Axel Müller
Managing Director, GCS

Gauss Centre for Supercomputing (GCS) e.V. | Alexanderplatz 1 | 10178 Berlin (Germany)

So far, Tile Low Rank examples ...



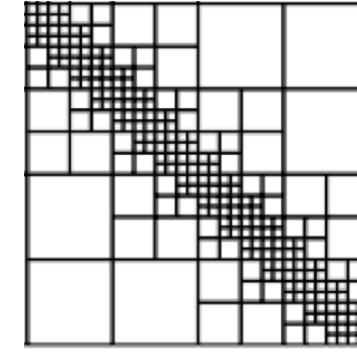
Tile Low Rank

Operations $O(k^{0.5} N^{2.5})$

Storage $O(k^{0.5} N^{1.5})$



(for an accuracy-dependent k)



Hierarchical Low Rank

Operations $O(k^2 N \log^2 N)$

Storage $O(k N)$

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

$$(2.1) \quad \underset{m}{\text{minimize}} \quad J(m) := F(u(m)) + \alpha R(m)$$

data misfit regularization

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

$$(2.2) \quad g(m, u) := K(m)u - f = 0,$$

- ◆ 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

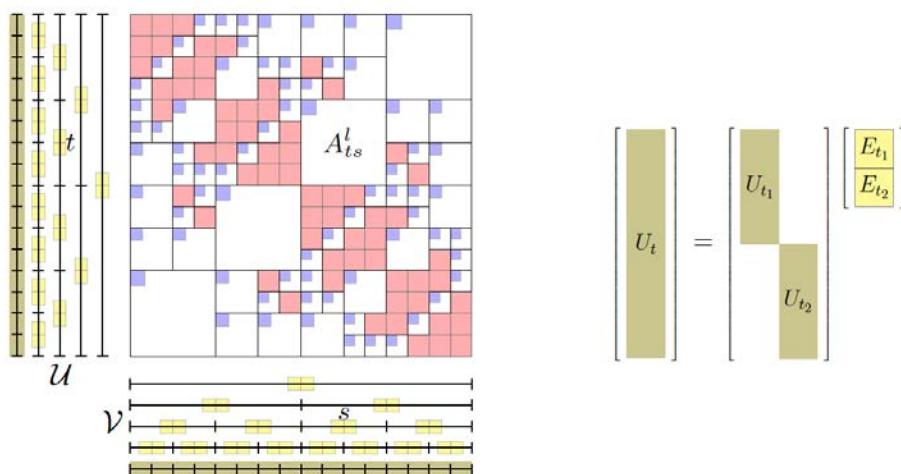
Hierarchical MatVec by tree-traversal

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

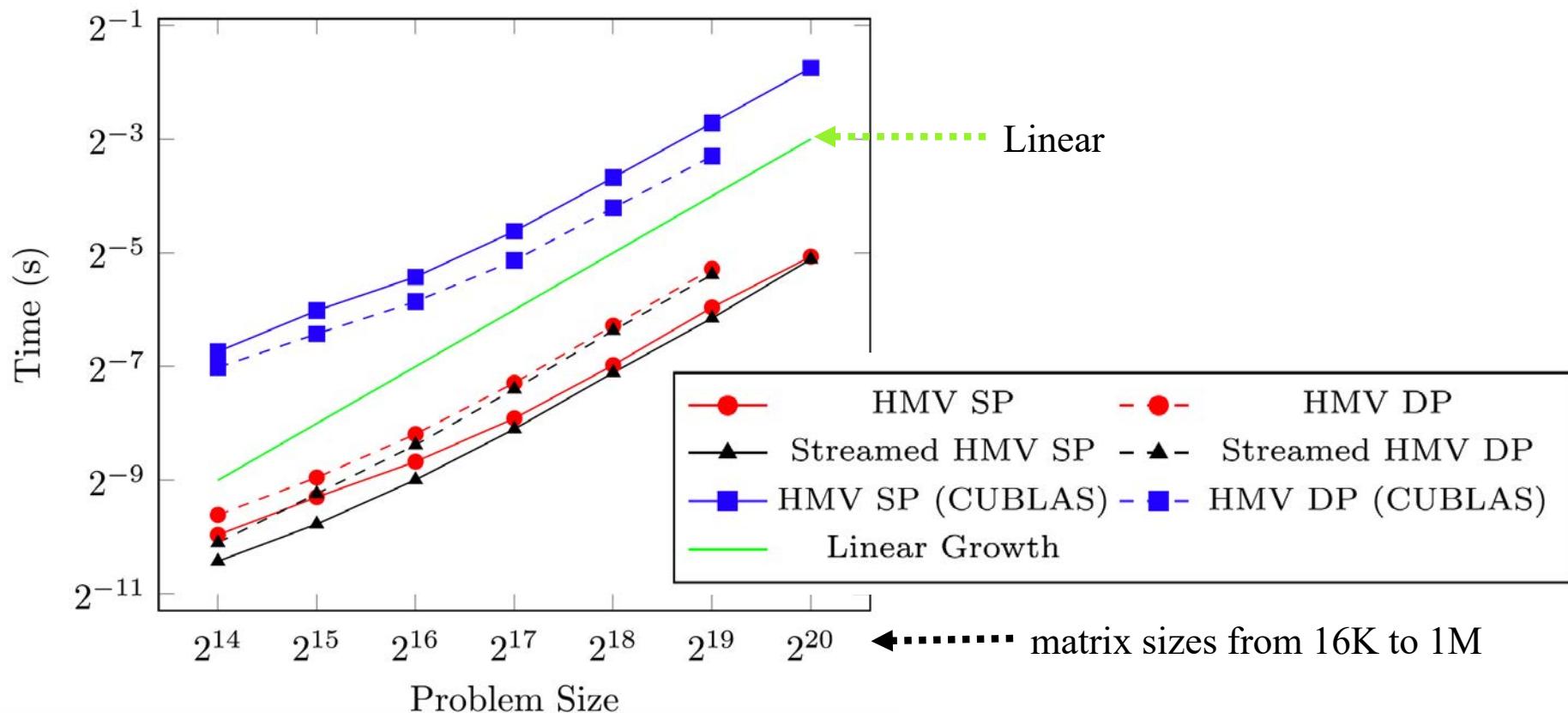
$$A_{\mathcal{H}^2} = D + \mathcal{U} \cdot \mathcal{S} \cdot \mathcal{V}^T$$

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

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

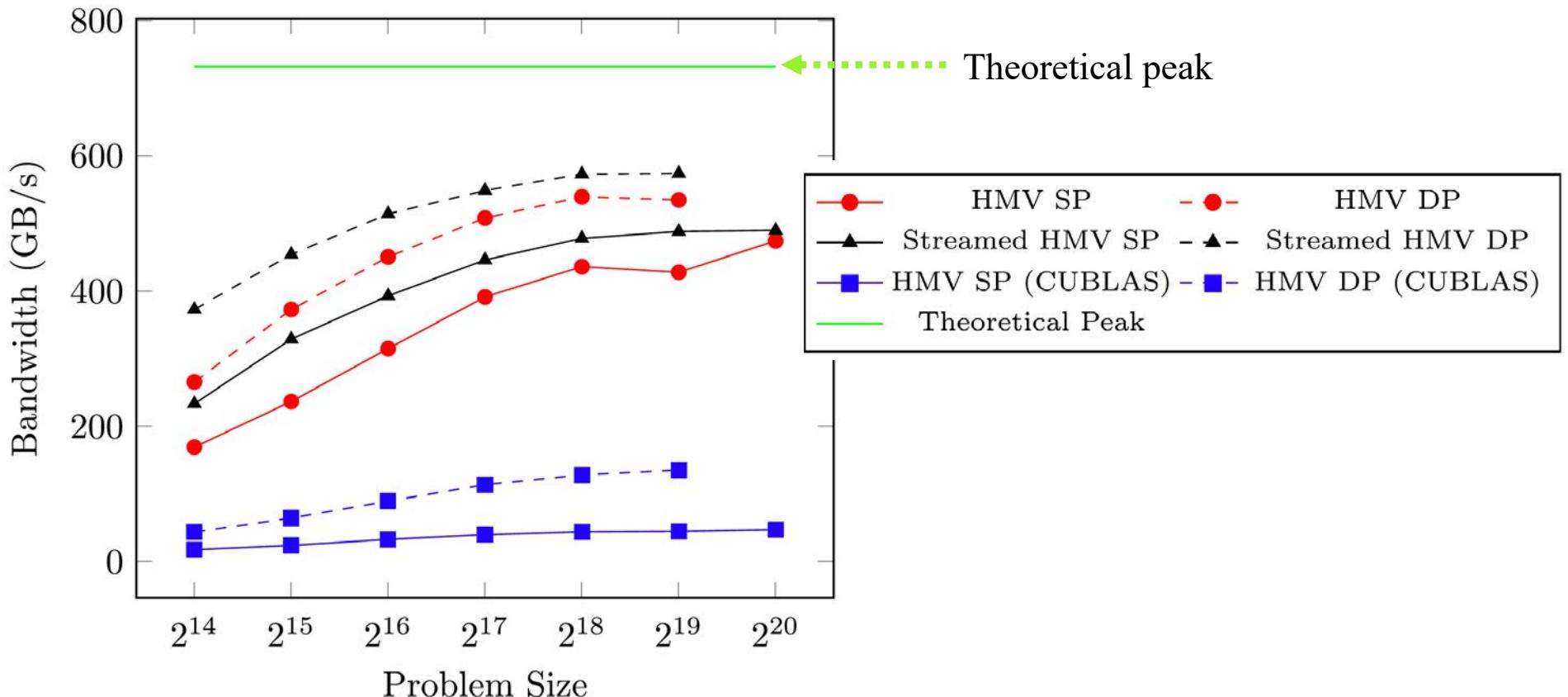


Hierarchical MatVec execution time



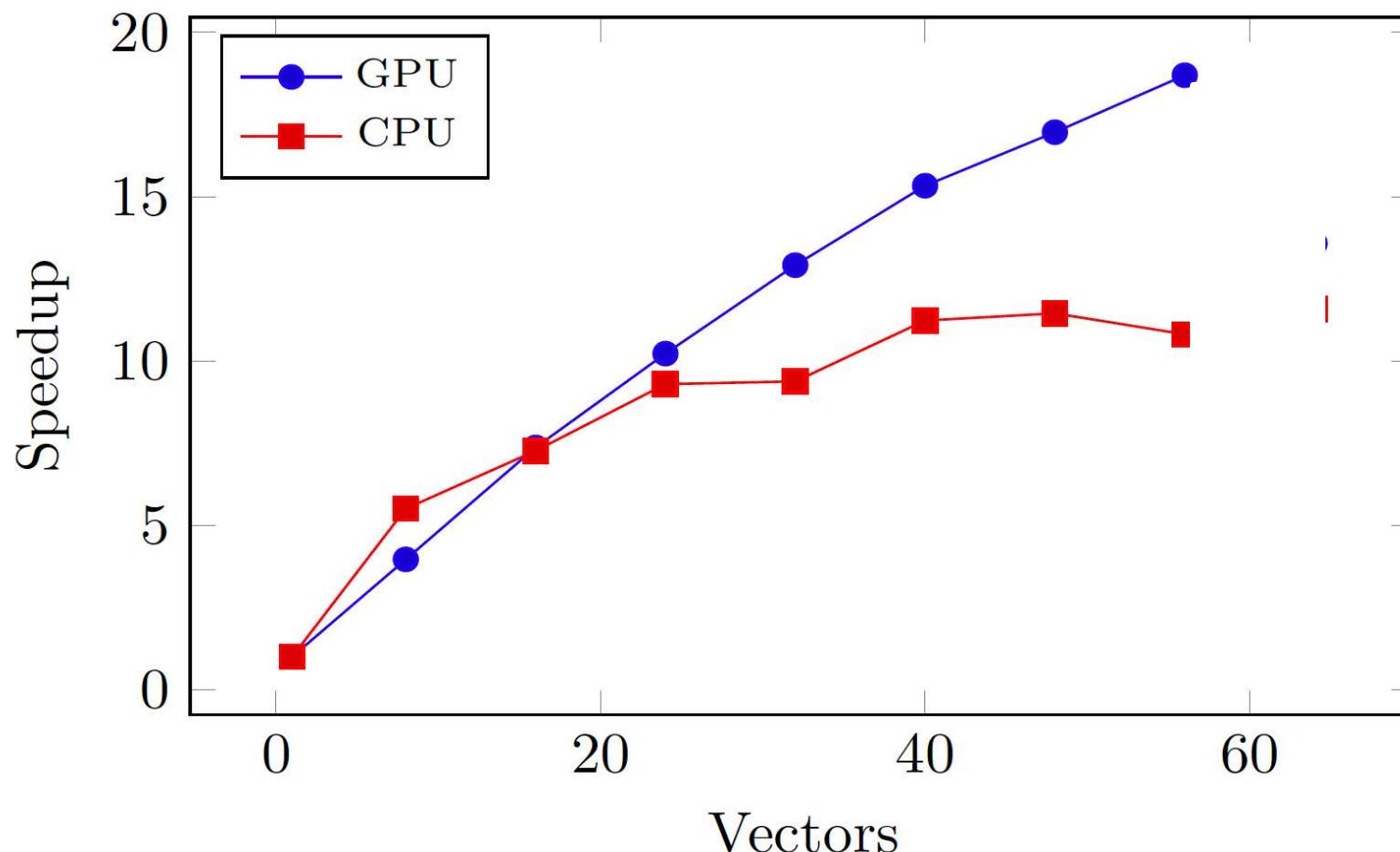
- ▶ 3D covariance matrices from spatial statistics
- ▶ running on P100 GPU
- ▶ accuracy 10^{-3} computed as $\|Ax - A^Hx\|/\|Ax\|$
- ▶ leaf size $m = 64$

Hierarchical MatVec bandwidth



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

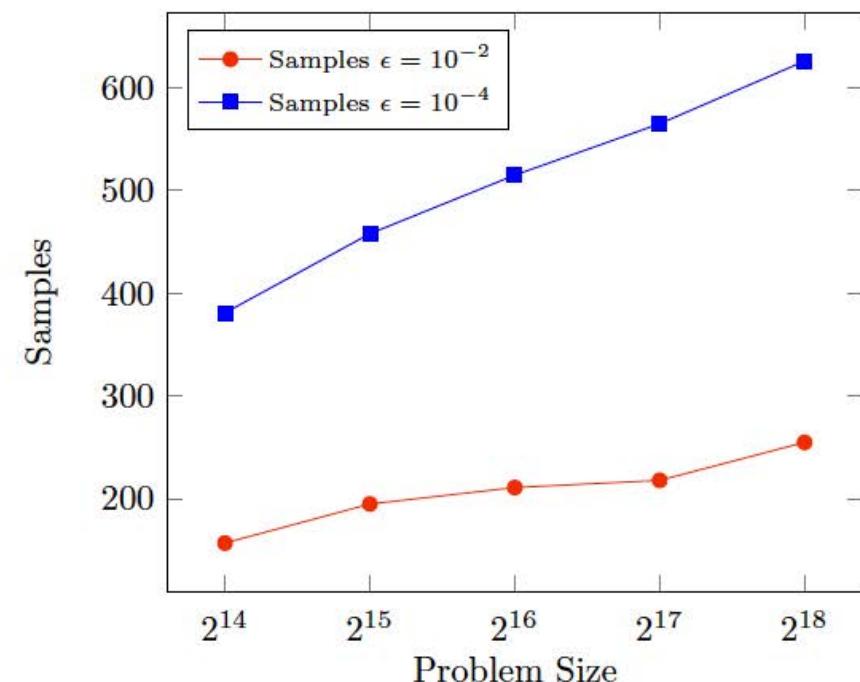
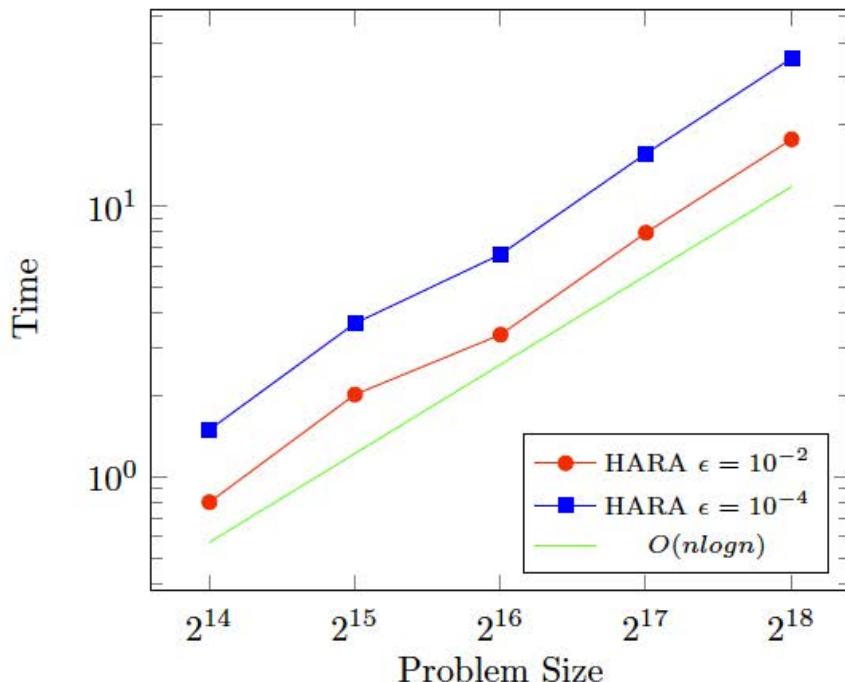
MatVecs with multiple vectors



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

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

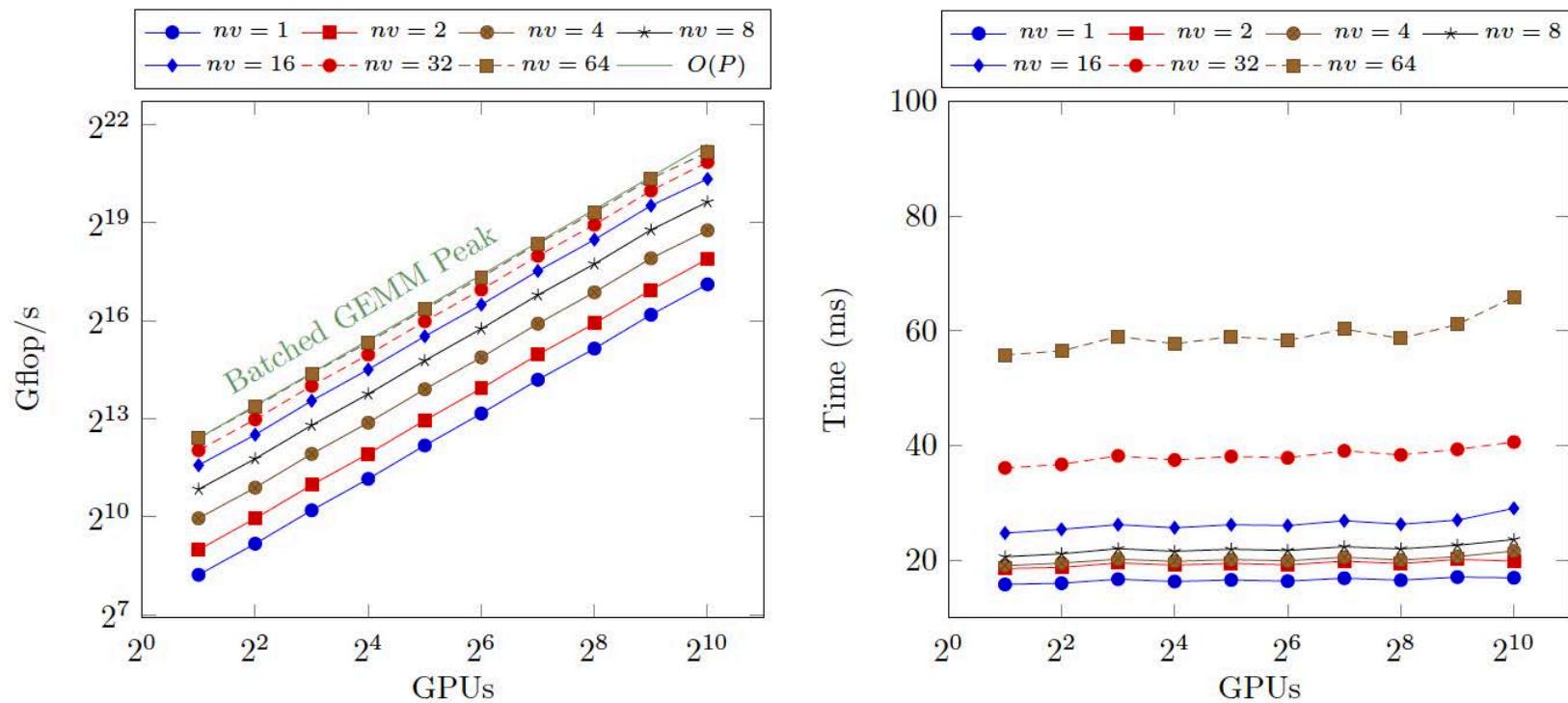
- ▶ can be cast as the problem of constructing an \mathcal{H} -matrix from matvec operations
- ▶ we can do HGEMV operations efficiently on GPUs
 - HGEMV on multiple vectors is even more efficient
- ▶ HARA construction of product also performed efficiently on the GPU
Fast matvecs \Rightarrow fast approx inversions with Newton-Schulz



Hgemv on Summit (1024 GPUs)

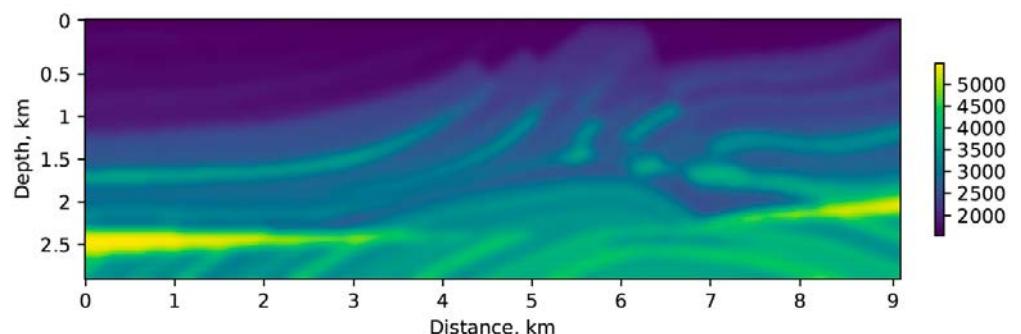
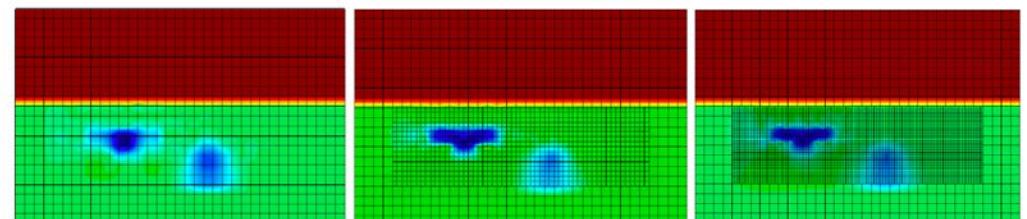
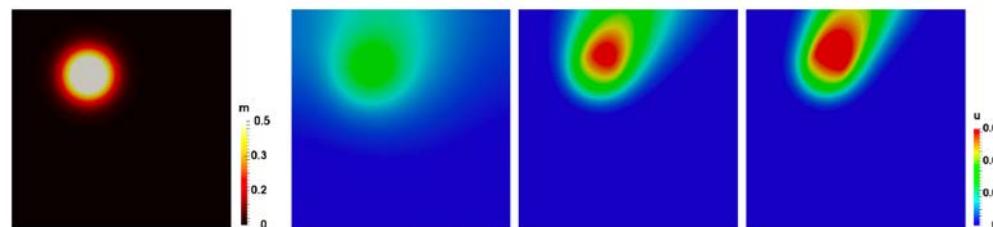
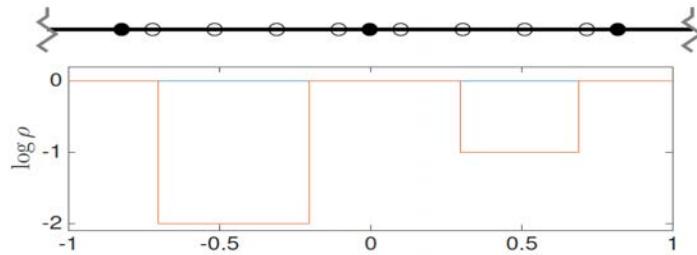
(distributed release of H2Opus will follow shortly)

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



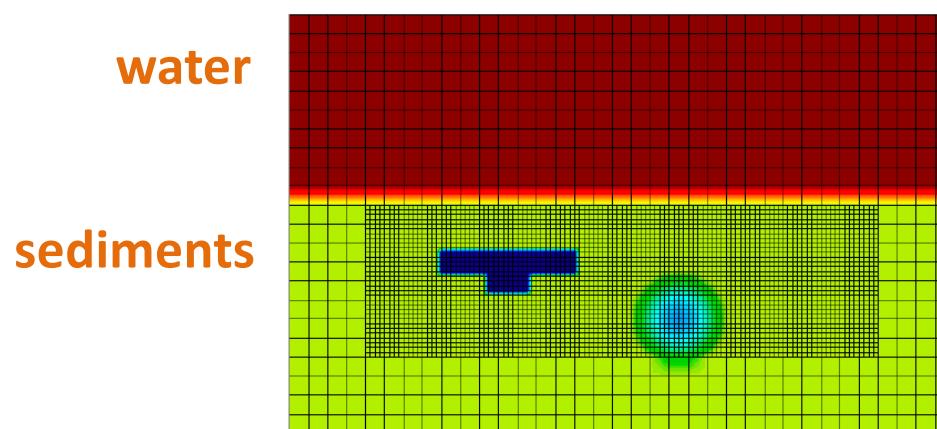
Optimization examples in SISC paper

- **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



Inversion example: transient electromagnetic inversion

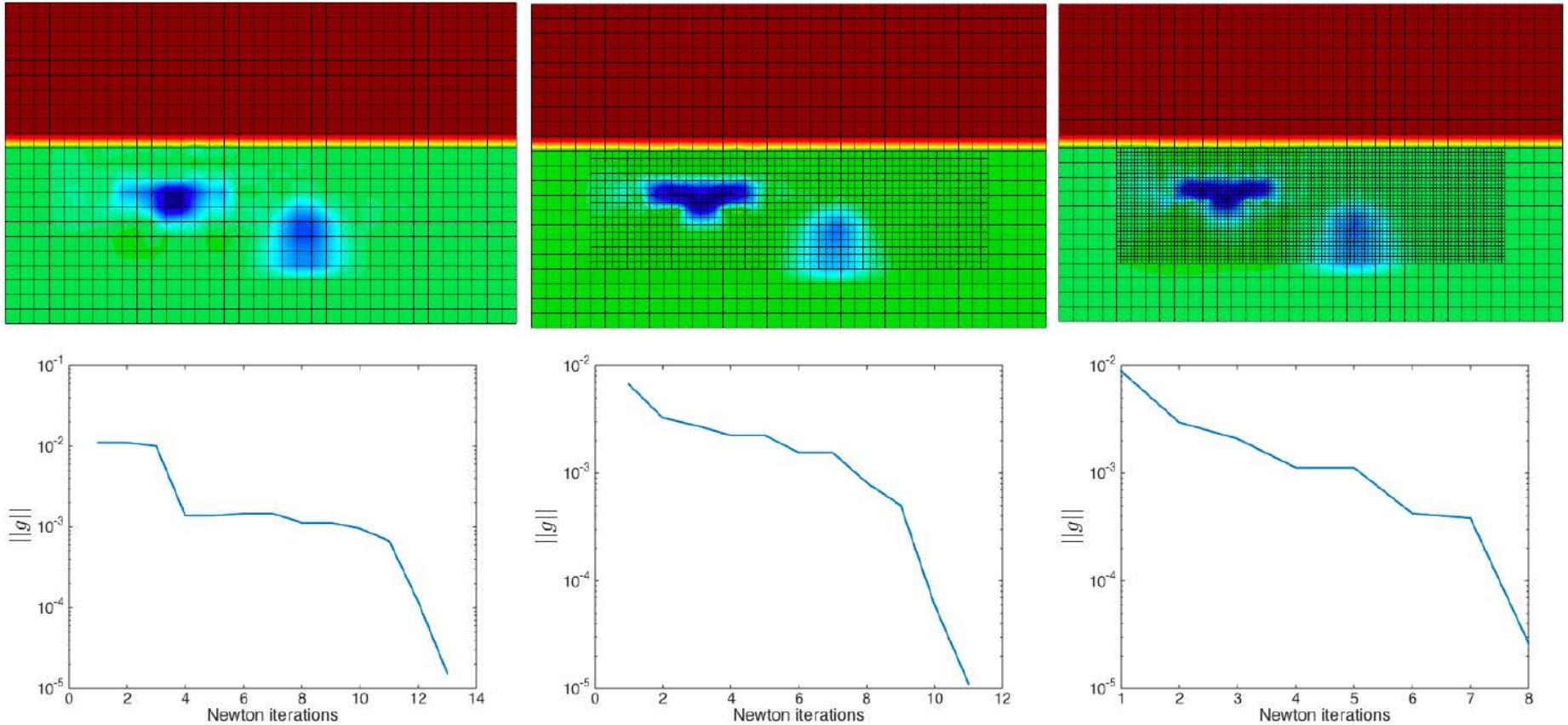
- ▶ Important geophysical sensing modality; of interest to Aramco
- ▶ Governing equations are Maxwell's equations in the diffusive limit.
- ▶ Electrical conductivity of oil is much lower than that of water, sediments, and salt bodies
- ▶ Time-domain inversion is the next frontier in this area
- ▶ We 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

MFEM, PETSc developer

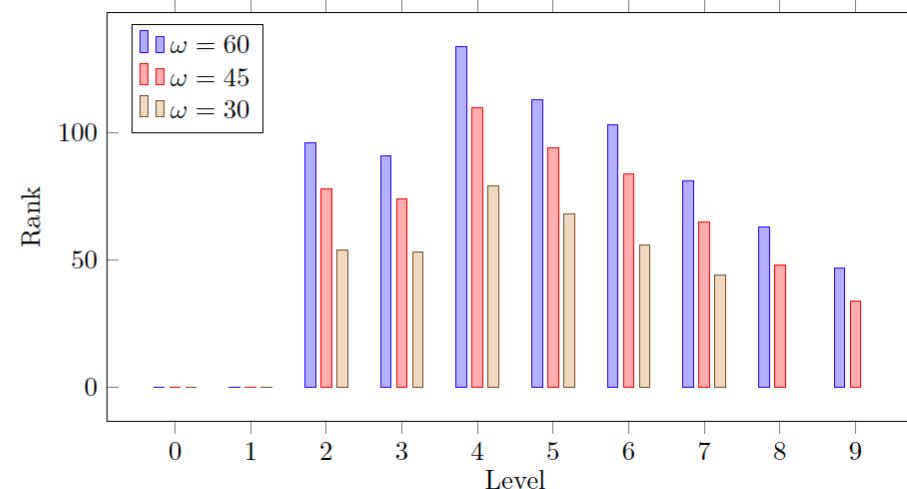
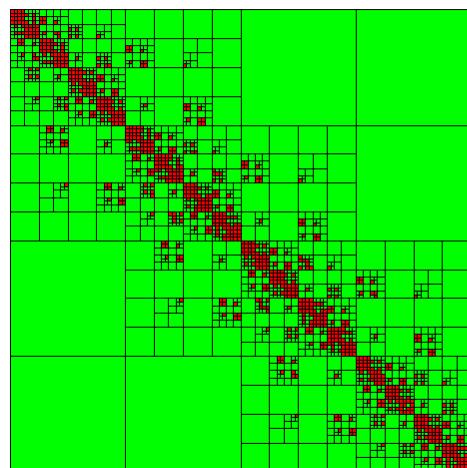
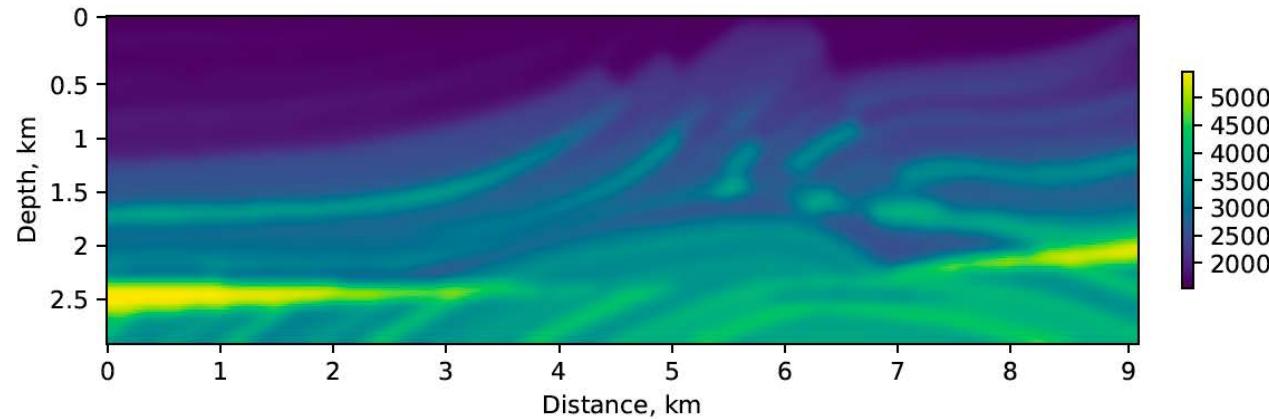
Inversion example: transient electromagnetic inversion



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



Reference

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

*SIAM Journal of
Scientific Computing
(2020, to appear)*

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

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

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

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

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

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

39 The Hessian operator plays a critical role in inverse problems. For deterministic
40 inverse problems, finding the parameters that best fit the data is typically formu-
41 lated 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).

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 , $0 < \alpha < 2$:

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

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

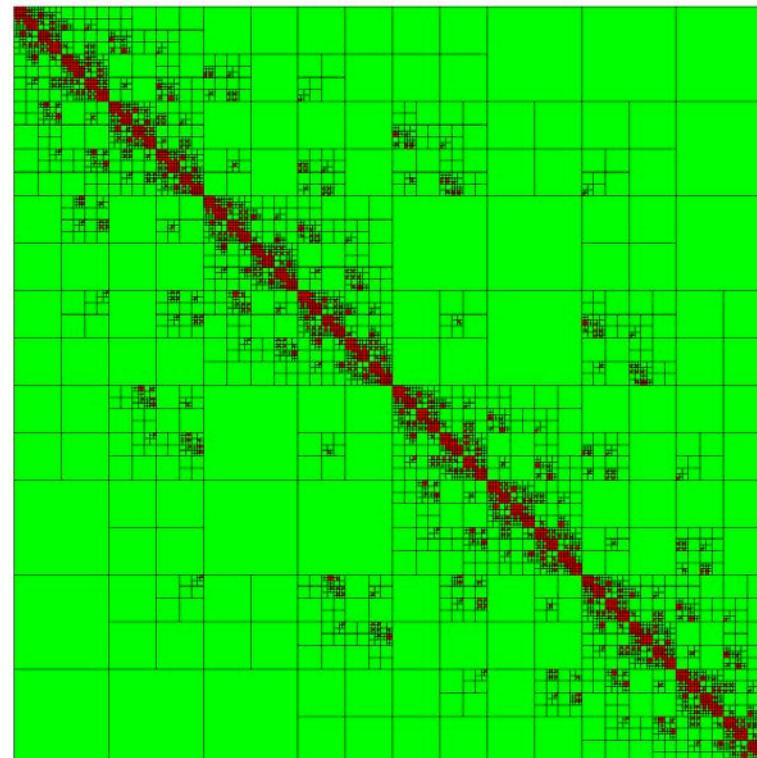
$$u(\mathbf{x}) = \sum_i V_i u_i \eta_\delta(\mathbf{x} - \mathbf{x}_i), \quad \eta_\delta(\mathbf{x}) = \frac{1}{\delta^d} \frac{1}{\pi^{\frac{d}{2}}} \exp(-|\mathbf{x}|^2),$$

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: $\dot{\mathbf{U}} = A_{N \times N} \mathbf{U}_{N \times 1}$, where all A_{ij} entries are nonzero.
key operation in explicit integration is a dense mat-vec

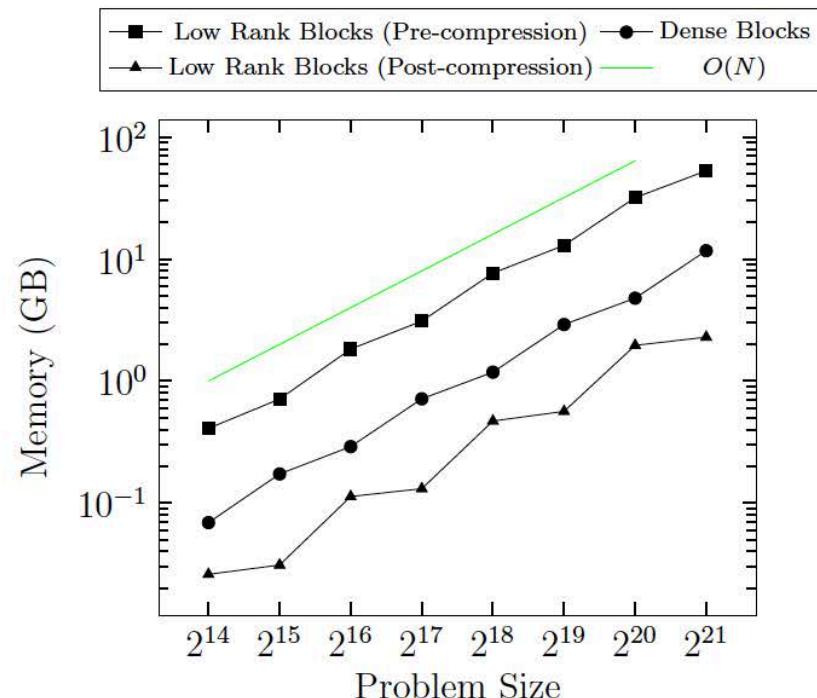
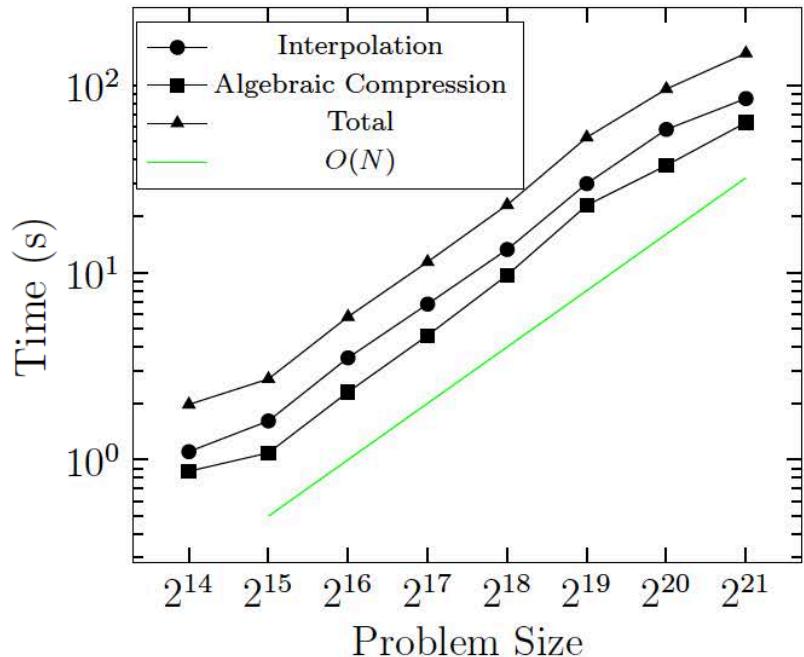
HLR representation

- The nature of the kernel allows blocks of A to have low rank representations for any desired approximation level ϵ
- Sample problem: 2d square region, N particles distributed uniformly, KD-tree binary partitioning produced a clustering, geometric admissibility criterion generates the matrix structure (i.e., blocks that admit low rank representations)
- Resulting matrix structure:



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 $N \sim 2M$
- execution hardware: 12-core workstation
- both compression steps have linear complexity



Fractional diffusion application

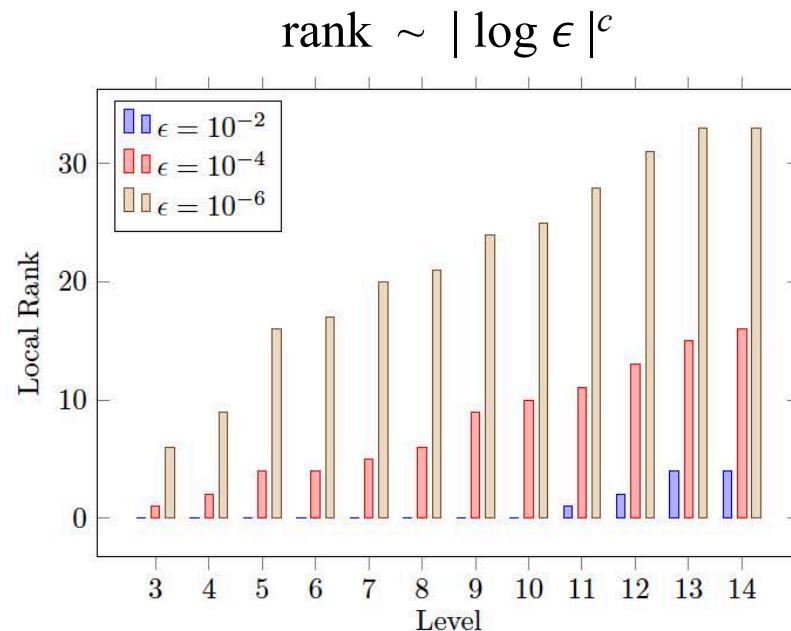
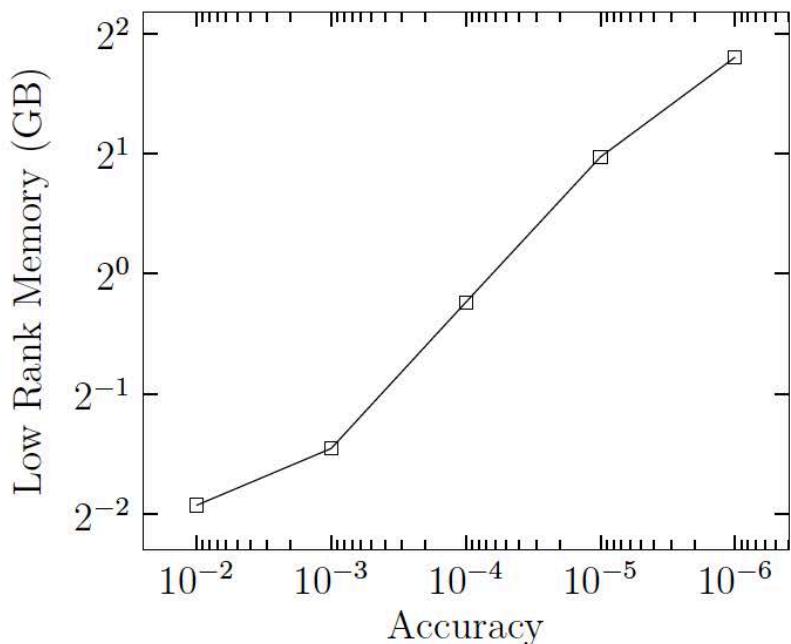
- Overall savings in memory of the hierarchical matrix (dense blocks and compressed blocks) computed to an overall accuracy of $\epsilon = 10^{-5}$
- Comparison of the hierarchical matrix representation and the (hypothetical) dense representation of the discretized operator
- Substantial reduction in 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 ratio of 2,339
for 2M DOFs

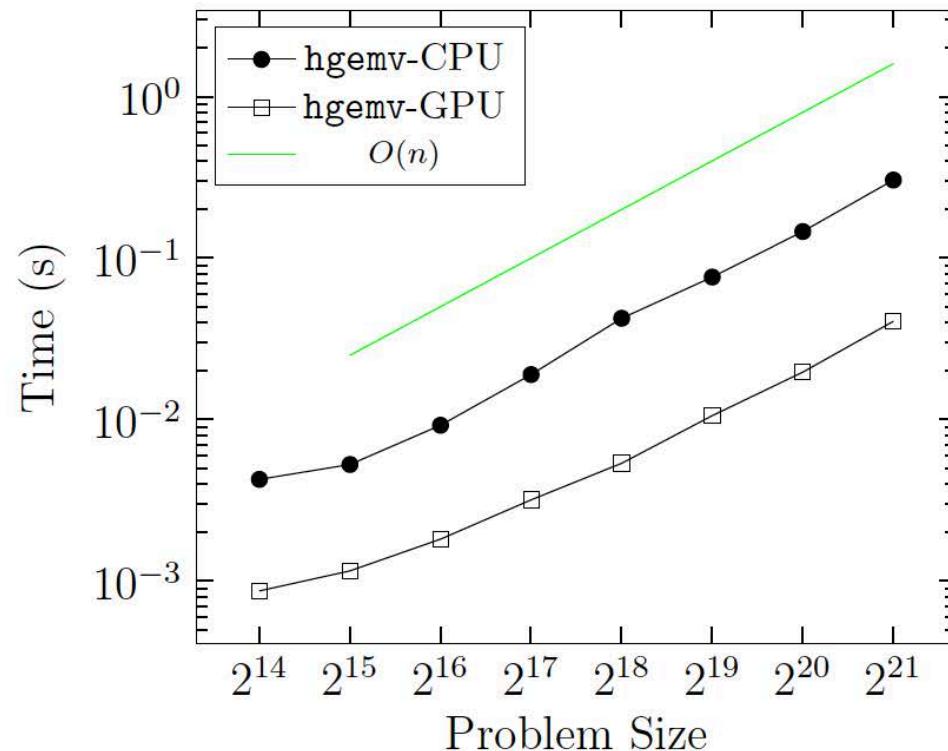
Rank variation with accuracy

- 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



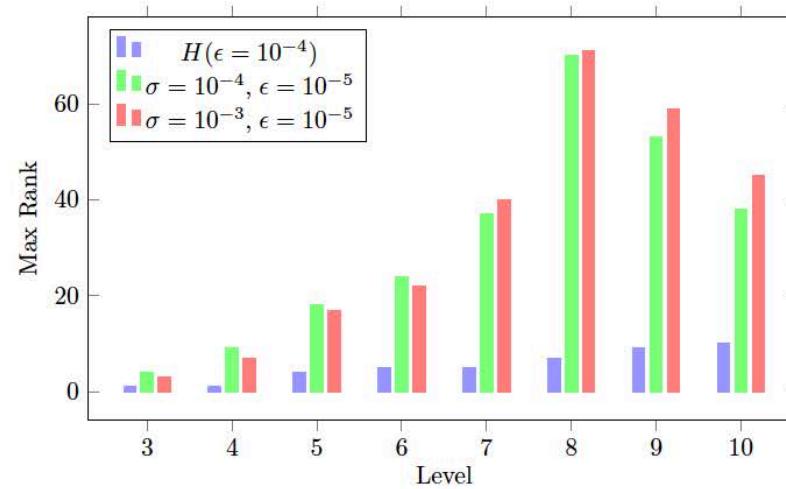
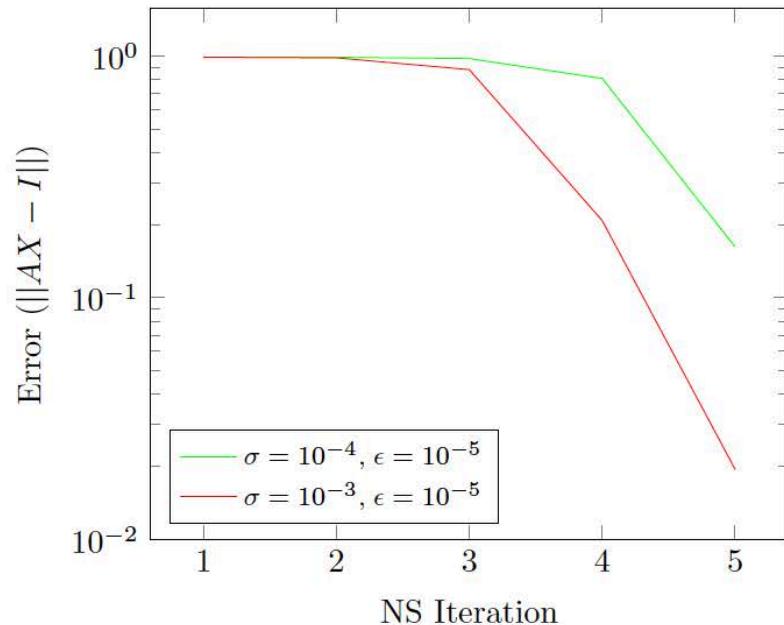
Scaling of mat-vec

- 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



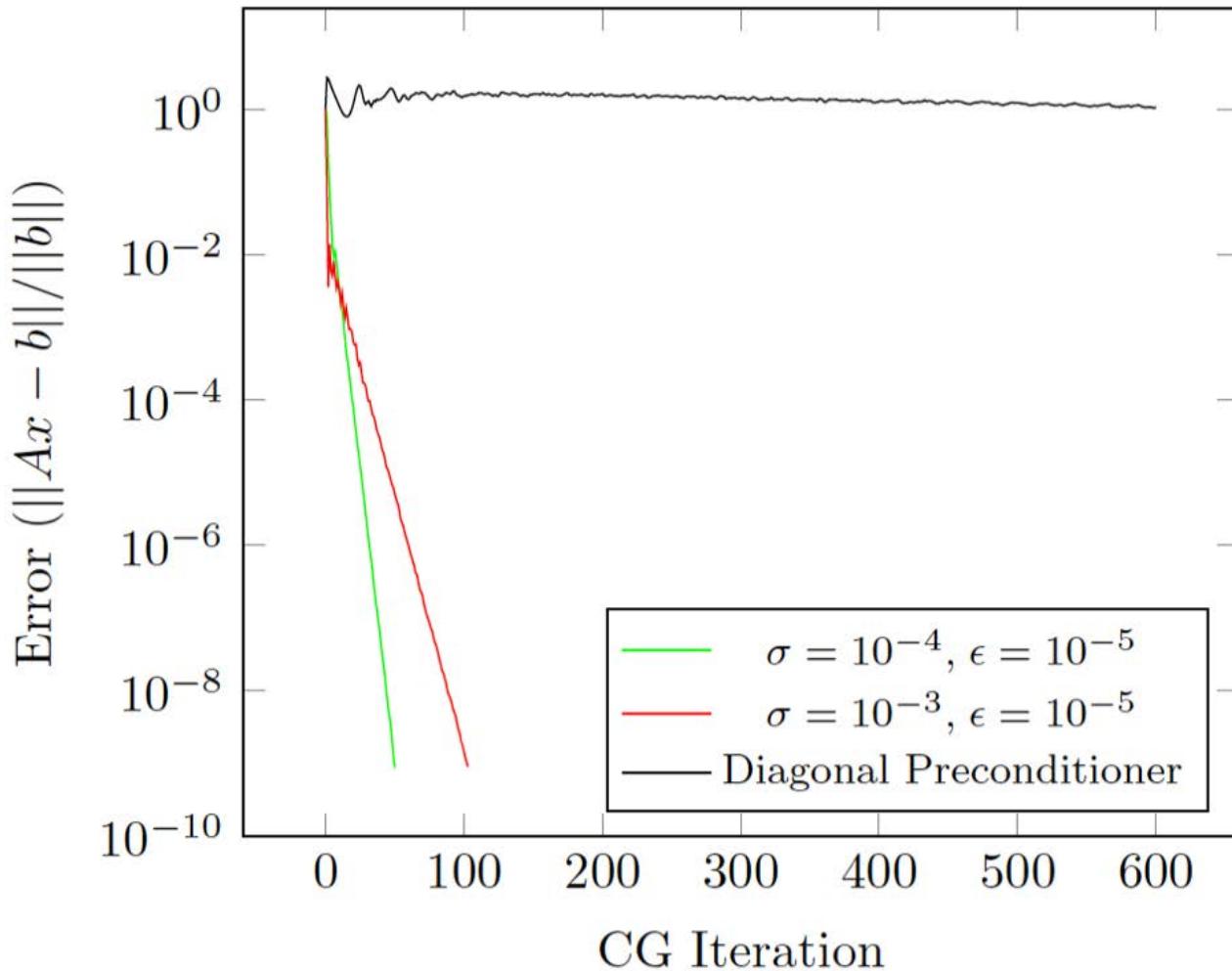
Newton-Schulz for inverse

- 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:
$$X_{p+1} = X_p (I + R_p + \cdots + R_p^{l-1}) = X_p \sum_{i=0}^{v-1} R_p^i$$
- 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 ($\alpha = 1.5, N = 2^{16}$)



Reference

Boukaram, Lucchesi, Turkiyyah, Le Maitre, Knio & K.

Computer Methods in Applied Mechanics and Engineering

369:113191 (2020)

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.

© 2020 Elsevier B.V. All rights reserved.

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

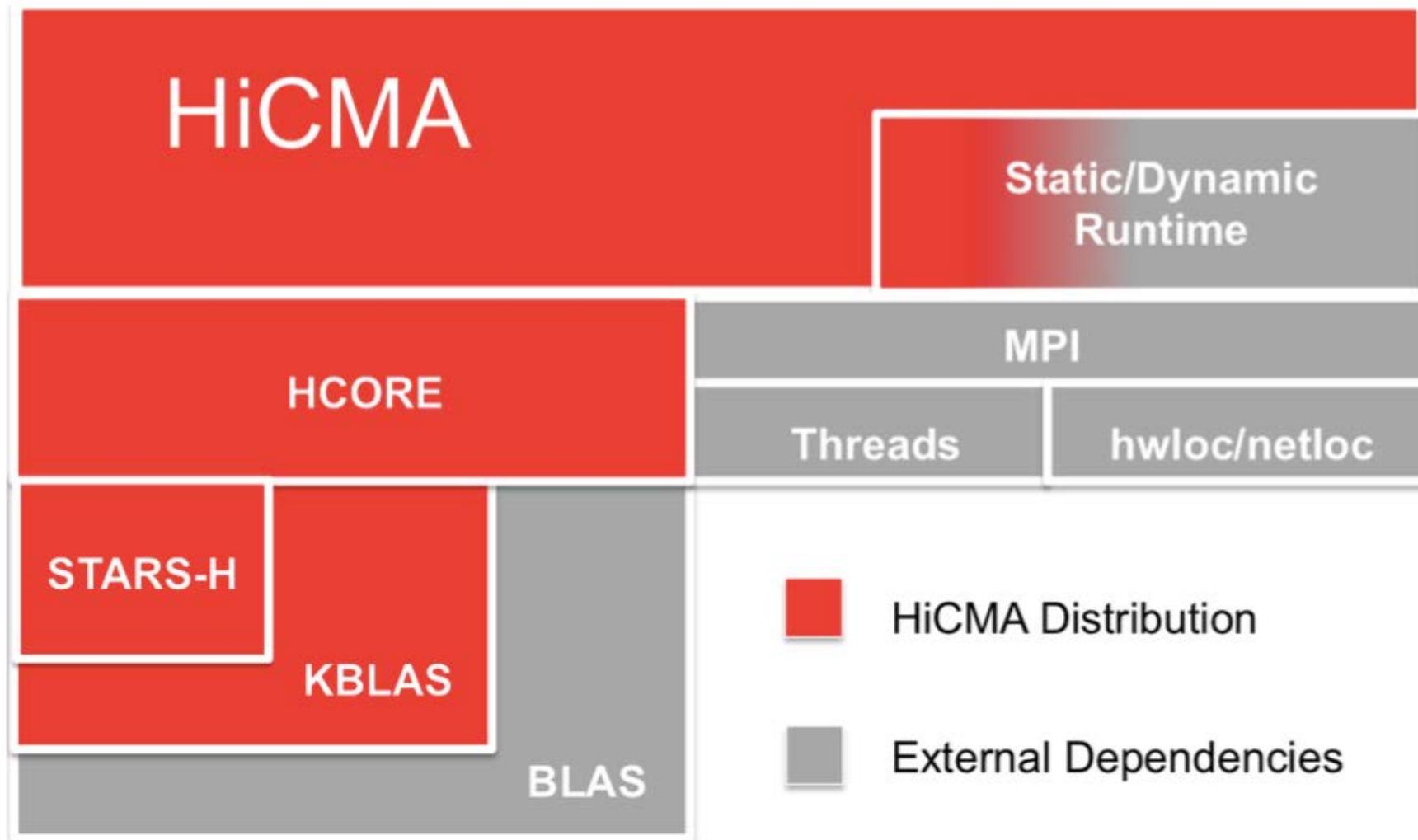
1. Introduction

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.

Hierarchical Computations on Manycore Architectures: HiCMA*



* appearing one thesis at a time at <https://github.com/ecrc>

Some ripe directions

- **Research**
 - ◆ Heuristics or theory for tuning precisions of block replacements with ultimate 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**
 - ◆ Generalization to variable rank size at leaves of the HLR tree (currently allocated for a max rank *per level*)
 - ◆ Optimizing parallelization of distributed implementation above “C-level” (in analogy to multigrid)

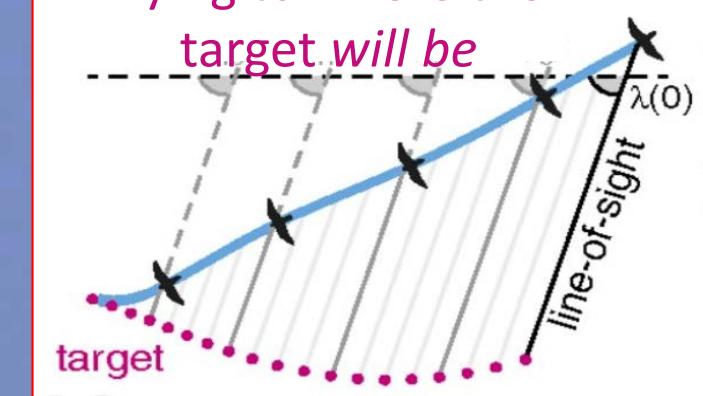
A falcon flies to where the prey *will be* ...



flying towards the target



flying to where the
target *will be*



... rather than where it is

C. H. Brighton,
et al., PNAS
(2017)

Those who did the work



Ahmad Abdelfattah



Rabab AlOmairy



Wajih Boukaram



Ali Charara



Kadir Akbudak



Hatem Ltaief



George Turkiyyah



Stefano Zampini

Very special thanks to...



Hatem Ltaief



George Turkiyyah

Reference

K., Ltaief & Turkiyyah

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

PHILOSOPHICAL
TRANSACTIONS A

rsta.royalsocietypublishing.org



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. Keyes¹, H. Ltaief¹ and G. Turkiyyah²

¹Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia

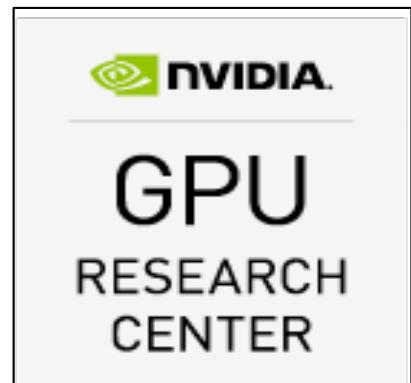
²Department of Computer Science, American University of Beirut 1107 2020, Lebanon

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 Authors. Published by the Royal Society under the terms of the Creative Commons Attribution License <http://creativecommons.org/licenses/by/4.0/>, which permits unrestricted use, provided the original author and source are credited.

THE ROYAL SOCIETY
PUBLISHING

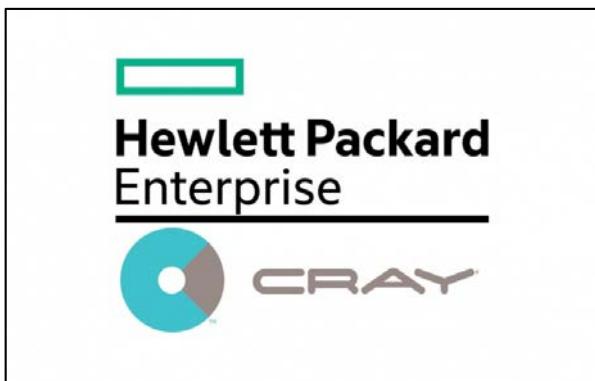
Sponsor acknowledgement



**NVIDIA GPU
Research Center**
2012



**Intel Parallel
Computing Center**
2015



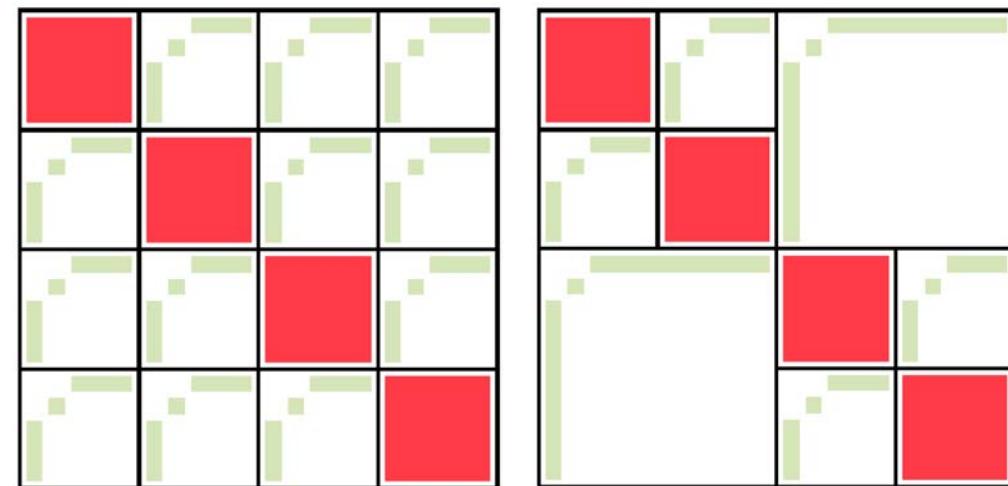
**Cray Center
of Excellence**
2015

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 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

Iconographic conclusion

I 



Poetic conclusion

“Curse of dimension,”
Can you be mitigated
By low rank’s blessing?

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



جامعة الملك عبدالله
للبعلوم والتكنولوجيا
King Abdullah University of
Science and Technology

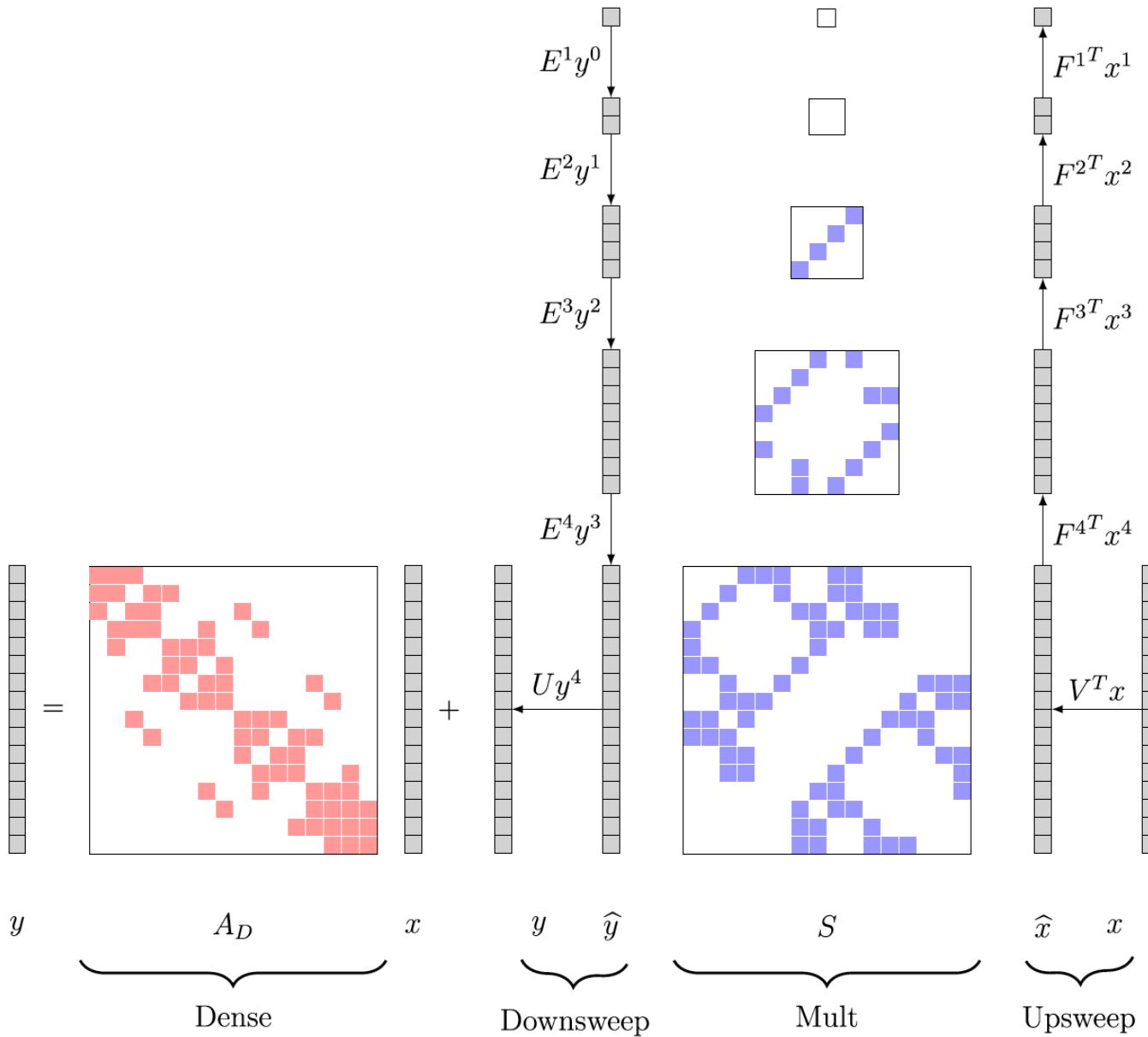
Thank you!

شكراً

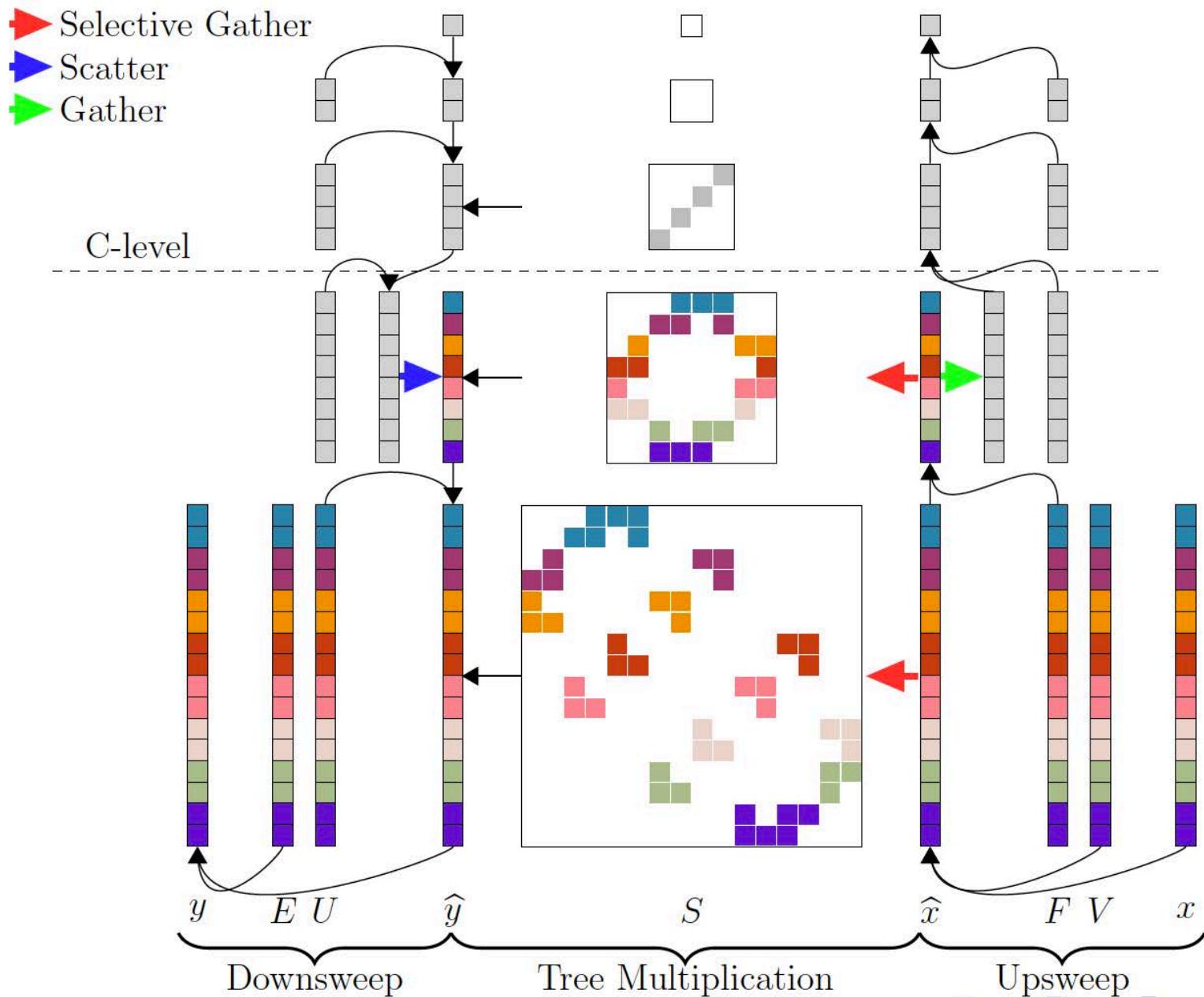
david.keyes@kaust.edu.sa

Extra slides

Tree-based mat-vec



Distributed mat-vec



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

