



Hierarchical Algorithms on Hierarchical Architectures



**David Keyes, Applied Mathematics & Computational Science
Director, Extreme Computing Research Center (ECRC)
King Abdullah University of Science and Technology**

**Kadir Akbudak, Rabab AlOmairy, Amani Alonazi, Wajih Boukaram, Ali Charara*,
Hatem Ltaief, Aleksandr Mikhalev, Dalal Sukkari, George Turkiyyah**, Rio Yokota*****

*** & ICL, UTennessee**

**** & American U of Beirut**

***** & Tokyo Tech**



Greetings from our new “boss”



Tony Chan, *formerly*:

- President, HKUST
- Director, Div Math & Phys Sci, NSF
- Dean, Phys Sci, UCLA
- Chair, Math, UCLA
- Co-founder, IPAM
- Member, NAE
- Fellow, SIAM, IEEE, AAAS
- ISI highly cited, imaging sciences, numerical analysis

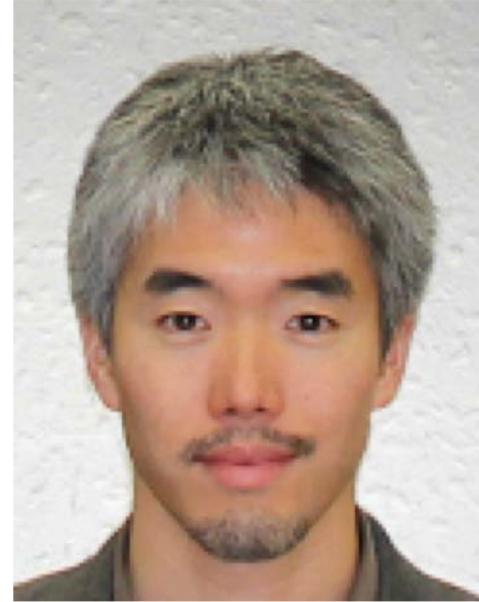
The brains behind our linear algebra and task-based runtime presentations at CSE19



Hatem
Ltaief



George
Turkiyyah



Rio
Yokota

The brawn behind our linear algebra and task-based runtime presentations at CSE19



Kadir
Akbudak



Rabab
AlOmairy



Amani
AlOnazi



Wajih
Boukaram



Ali
Charara



Aleksandr
Mikhalev



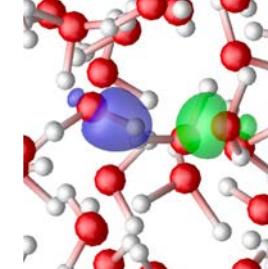
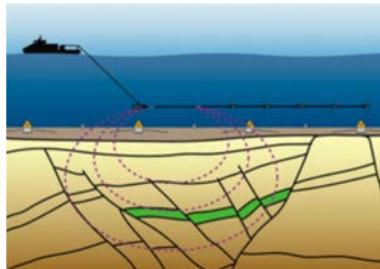
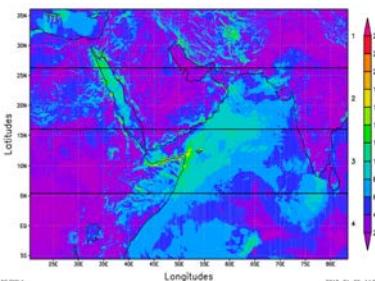
Dalal
Sukkari

To take away (1)

- To better exploit emerging architectures, we are developing new versions of linear, least squares, EVD, and SVD solvers
 - may offer tunable accuracy-time tradeoffs
 - may exploit hierarchy of precisions
 - may require more flops but offer more concurrency (and thus complete faster)
- Besides exposing more concurrency, we need to
 - remove synchrony and over-ordering
 - dwell as high as possible on the memory hierarchy

To take away (2)

- With such new solvers, we can extend many 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 associations)



To take away (3)

- If you can speed up linear algebra kernels, “*the world’s your oyster, which you with sword will open*” *
- This can all be illustrated
 - but not in 20 minutes ☺
- Hope it highlights the relevance of this double minisymposium (MS 197 Wed am, MS 231 Wed pm)
 - highlighting is all I’m good for any longer ☺

* Shakespeare (1600), *The Merry Wives of Windsor*, Act II, Scene II

HIERARCHICAL COMPUTATIONS ON MANYCORE ARCHITECTURES

Hicma

Extreme Computing Research Center

The Hierarchical Computations on Manycore Architectures (Hicma) library aims to redesign existing dense linear algebra libraries to exploit the data sparsity of the matrix operator. Data sparse matrices arise in many scientific problems (e.g., linear systems, signal processing, seismic imaging, and machine learning applications) and are characterized by low-rank structures, diagonal tile structures, and low-rank approximations and are well characterized by both in memory footprint and arithmetic complexity. The core idea of Hicma is to develop fast linear algebra computations on manycore hardware while satisfying a specified numerical accuracy and leveraging performance from massively parallel hardware architectures.

LOW-RANK ALGORITHMS & CHOLSKY FACTORIZATION

SOFTWARE STACK

GEOSPATIAL STATISTICS

HICMA 0.1.0

- LU Factorization
- Cholesky Factorization, Solve
- Double Precision
- Support for Multiple Precision
- Shared and Distributed Memory
- Support for StarPU Dynamic Dispatching
- Support for OpenMP
- Support for MPI
- Support for HPC and Kernels
- Testing Suite and Examples

PERFORMANCE RESULTS

Download the software at <https://github.com/ecrc/hicma>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

Abstraction Layer For Standardizing APIs of Task-Based Engines

AL4SAN

Extreme Computing Research Center

The abstraction layer for standardizing APIs of task-based engines (AL4SAN) is designed as a lightweight software library, which provides a collection of APIs to unify the expression of tasks and their data dependencies. It is built on top of the widely used and well-known task-based engines, such as StarPU, and leverages compiler infrastructure technology or on library-defined APIs. It features an abstraction of task-based engines and, therefore, enables a single-code application to assess various runtimes and their respective scheduling strategies. This allows users to quickly create yet efficient execution system, but also to leverage the user-benefices of the underlying parallel hardware architectures.

AL4SAN v1.0 Features

- Standardized Task-based Engine API
- Using a lightweight abstraction
- Improving application scalability
- Supporting different hardware architectures
- Performance overhead relatively limited overhead (See Fig. 10)
- Supports:
 - OpenBLAS
 - StarPU
 - StarLIB
 - StarPU-K
 - QUARK
- Performance Assessment

AL4SAN Roadmap

- Extending to more original tasks
- Integrating C++ constructs
- Supporting more scientific computing runtimes
- Adding support to more applications and benchmarks

AL4SAN Abstraction Layer For Task-Based Engines

AL4SAN Main Reference

KBLAS HIGHLIGHTS

- KBLAS Level-2 (3) GMV & HEMV
- KBLAS Level-2 (3) TRMM & TRSM
- Batch Cholesky (4x4)
- LAUHM
- POTRS
- POTRI
- Batch General (4x4)
- Single GPU support
- Fast precision & k=1
- Multi GPU support
- Uniform batch tests
- Arbitrary sizes

CURRENT RESEARCH

- Half Precision Legacy and Batch BLAS
- TUF GPU (TULLAPL) on GPUs
- TUF GPU (TULLAPL) Matrix Combinations
- Adaptive Cross Approximation (ACA) on GPUs

PERFORMANCE RESULTS

Download KBLAS at <https://github.com/cecm/kblas>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

in NVIDIA cuBLAS

KBLAS

Extreme Computing Research Center

KAUST BASIC LINEAR ALGEBRA ROUTINES ON GPUs

KBLAS is a high performance CUDA library implementing a subset of BLAS as well as Linear Algebra PACKAGE (LAPACK) routines on NVIDIA GPUs. Using recursive and batch algorithms, KBLAS maximizes the GPU bandwidth, reduces locally cached data and increases data locality. KBLAS targets a broad range of applications and is highly efficient. In many cases, KBLAS outperforms its CPU counterparts. Located at the heart of the software stack, KBLAS enables higher-level numerical libraries and scientific applications to extract the expected performance from GPU hardware accelerators.

RECURSIVE ALGORITHMS: TRMM and TRSM

- Legacy Level-2 BLAS (1x1) GMV, HEMV, TRMM, TRSM
- Legacy Level-3 BLAS (1x-n) TRMM, TRSM, SYRK
- Batch Level-2 BLAS (4x4) TRMM, TRSM, SYRK
- Batch Cholesky (4x4)
- LAUHM
- POTRS
- POTRI
- Batch General (4x4)
- Single GPU support
- Fast precision & k=1
- Multi GPU support
- Uniform batch tests

BATCH ALGORITHMS: Recursive Cholesky POTRS

- 1 Rec-POTRS
- 2 Rec-TRMM
- 3. Rec-TRSM
- 4. Rec-POTRF

AL4SAN

AL4SAN Main Reference

AL4SAN Abstraction Layer For Task-Based Engines

AL4SAN Current Research

AL4SAN Performance Assessment

AL4SAN Roadmap

AL4SAN Performance Results

KBLAS HIGHLIGHTS

- Half Precision Legacy and Batch BLAS
- TUF GPU (TULLAPL) on GPUs
- TUF GPU (TULLAPL) Matrix Combinations
- Adaptive Cross Approximation (ACA) on GPUs

CURRENT RESEARCH

- Half Precision Legacy and Batch BLAS
- TUF GPU (TULLAPL) on GPUs
- TUF GPU (TULLAPL) Matrix Combinations
- Adaptive Cross Approximation (ACA) on GPUs

PERFORMANCE RESULTS

Download KBLAS at <https://github.com/cecm/kblas>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

in Cray LibSci

KSVD

Extreme Computing Research Center

A QDWH-Based SVD Software Framework on Distributed-Memory Manycore Systems

The KAUST SVD (KSVD) is a high performance software framework for computing a dense SVD on distributed-memory manycore systems. The KSVD software relies on the parallelism using the QR Dynamically-Weighted Halley algorithm (QDWHA) for the computation of the singular value decomposition (SVD). The computational challenge resides in the significant amount of extra floating-point operations required by the QDWHA-based SVD algorithm, compared to the traditional one-stage bidiagonal SVD. However, the inherent high level of concurrency associated with the iterative nature of the algorithm compensates the arithmetic complexity overhead and makes KSVD a competitive SVD solver on large-scale supercomputers.

QDWH Algorithm

- Parallel QR factorization using the QDWH-based SVD
- Efficient communication-computation balance
- Support for Singular Value Decomposition
- Support for Singular Value Computation
- Support for LU Factorization and MP
- Support for Cholesky Factorization
- Minimizes data movement
- Minimizes resource synchronizations

Advantages

- Fast convergence rate
- Relies on compute intensive kernels
- Supports orthogonal Matrices and symmetric positive semi-definite matrix
- Supports Cholesky Factorizations for R-B band matrices
- Minimizes data movement
- Minimizes resource synchronizations

Application to SVD

- $\mathbf{U} \mathbf{\Sigma} \mathbf{V}^T = \mathbf{U} \mathbf{\Sigma}' \mathbf{V}^T$
- $\mathbf{U} \mathbf{\Sigma}' \mathbf{V}^T = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$

Performance Results

Download KSVD at <https://github.com/cecm/ksvd>

Current Research

- Anisotropic Task-Based QDWH
- Dynamically Weighted Halley
- Distributed Memory Machines
- Adaptive, Task-Based QDWH
- QDWH-based Eigensolver (LU-QDWH)
- Integration into PLASMA/METIS

ACKNOWLEDGMENTS

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

Software for Testing Accuracy, Reliability and Scalability of Hierarchical computations

STARS-H

Extreme Computing Research Center

STARS-H is a high performance parallel open-source package of Software for Testing Accuracy, Reliability and Scalability of Hierarchical computations. It provides a benchmark in order to benchmark performances of various libraries for hierarchical computations. The main idea is to benchmark the performance of various libraries for hierarchical computations, each one with its own performance and memory footprint. STARS-H intends to provide a standard for assessing the performance of hierarchical computations. STARS-H supports the standard TUF data format for approximation on shared and distributed-memory systems, using MPI, OpenMP and task-based programming models. STARS-H package is available online at <https://github.com/ecrc/stars-h>.

Matrix Kernels

- Electrodynamics (far and near distances)
- $A_{ij} = \frac{1}{r_{ij}}$
- Electrostatics (far and near distances)
- $A_{ij} = \frac{\cos(kr_{ij})}{r_{ij}}$
- Spatial autocorrelation (Matern kernel)
- $A_{ij} = \frac{2(1-\beta)}{1+\beta} \left(\frac{2\pi r_{ij}}{l} \right)^2 K_0 \left(\frac{2\pi r_{ij}}{l} \right)$
- And many others...

Stochastic Problem Setting

Stochastic estimation problem for a quasi uniform distribution in a unit square [0,1] x [0,1] with exponential kernel:

$$A_{ij} = \frac{\beta}{r_{ij}}$$

where $\beta = 0.1$ is a correlation length parameter and r_{ij} is a distance between i-th and j-th spatial points.

ROADMAP OF STARS-H

- Extend to other problems in a matrix-free form
- Support: HODLR, HSL, HX, X², X³ data formats
- Integrate other approximation schemes (e.g., ACA)
- Port to GPU accelerators
- Adaptation to different memory systems and programming models (e.g., PARSEC)

3D problem on different two-level shared memory Intel dBB architectures

3D problem on a different amount of nodes (from 4 up to 1024) of a distributed-memory CUBLAS system for a different error threshold ϵ

Performance Results (MLE)

Download the library at <https://github.com/vzn/stars-h>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

PARALLEL HIGH PERFORMANCE UNIFIED FRAMEWORK FOR GEOSTATISTICS ON MANY-CORE SYSTEMS

ExaGeoStat

Extreme Computing Research Center

This poster presents ExaGeoStat, a high performance unified framework for geostatistics on many-core systems. The framework provides a general solution for generating a covariance function for a given spatial data to provide an efficient way to predict missing values. ExaGeoStat proposes a multi-level approach to accelerate the computation of the covariance function, which is computationally challenging since it involves the inversion of a large number of n×n matrices.

ExaGeoStat 1.0

- Data formats: TUF, GMV, HEMV
- Operations: approximation, matrix-vector multiplication, Krylov CG solve.
- Supports: shared and distributed memory, random TUF, GMV, HEMV, Cauchy matrix.
- Real applications in a matrix-free form: electrostatics, electromagnetism, optics, atmospheric sciences.
- Programming models: OpenMP, MPI and task-based (StarPU).
- Approximation methods: SVD, RIRG, Randomized SVD.

ROADMAP OF EXAGEOSTAT

- Extend to other problems in a matrix-free form
- Support: HODLR, HSL, HX, X², X³ data formats
- Integrate other approximation schemes (e.g., ACA)
- Port to GPU accelerators
- Adaptation to different memory systems and programming models (e.g., PARSEC)

ExaGeoStat 1.0 Maximum Likelihood Estimation (MLE)

ExaGeoStat 1.0 Covariance Function

ExaGeoStat 1.0 Performance Results (MLE)

Download the library at <https://github.com/vzn/exageostat>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

A HIGH PERFORMANCE MULTI-OBJECT ADAPTIVE OPTICS FRAMEWORK FOR GROUND-BASED ASTRONOMY

MOAO

Extreme Computing Research Center

MOAO is a high performance multi-object adaptive optics framework for ground-based astronomy. It is designed to handle multiple objects simultaneously, providing high-resolution images of distant galaxies and stars. The framework uses a combination of hardware and software components to achieve this. The hardware includes a large telescope and a deformable mirror, while the software includes a complex control system and a powerful computer to process the data. The MOAO framework has been successfully used to capture images of multiple objects in a single exposure, greatly increasing the efficiency of astronomical observations.

Performance Results (MLE)

Download the software at <http://github.com/cecm/moao>

Acknowledgment of PARIS-DIDIER, INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from PARIS-DIDIER, INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by PARIS-DIDIER, INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

in Aramco GeoDrive

GIRIH

Extreme Computing Research Center

The Girih framework implements a generalized multi-dimensional intra-tile parallelization scheme for shared-address multicore processors that results in a significant reduction of cache size requirements for temporally blocked stand still tiles. The Girih framework is designed to handle the challenges of modern multicore architectures, including irregular memory access patterns and varying numbers of cores per node. It uses a novel approach to handle these challenges, called "Girih", which involves a combination of intra-tile parallelization and inter-tile communication. The Girih framework has been shown to be effective in improving the performance of scientific applications running on multicore processors.

MULTIDIMENSIONAL INTRA-TILE PARALLELIZATION

STENCIL COMPUTATIONS

- Hot spot in many scientific codes
- Supports: shared memory, element, and volume decompositions of PDEs
- Eg.: 3D wave acoustic wave equation $\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$

7-point stencil

25-point stencil

SOFTWARE INFRASTRUCTURE

GIRIH system components

- Matrix power kernels
- Single and double precision
- Overlapping domain decomposition
- GRB4MPI: GRB4MPI: MPI support for C/C++
- OpenACC/CUDA
- Dynamic runtime systems
- Constant/variable coefficients
- LU-factorization support for C/C++

PERFORMANCE RESULTS (1D ORDERED INTEGRATION WITH 25 POINTS)

- Domain size: 512 x 512 x 512 x 512
- 4x4 grid
- 25-point star stencil
- Two socket system (Mem. 128GB)
- 16-core Intel Skylake (2.1GHz, 20MB L3 Cache, 32GB RAM, 20GbE, 40GbE)
- 20-core Intel Skylake (2.9GHz, 32GB RAM, 20GbE, 40GbE)
- Memory affinity enabled
- Memory affinity with mklmt.c1 command
- Thread binding to cores with sse2, #aff 0x1 as command
- Thread binding to cores with sse2, #aff 0x1 as command

GPU PRECISION

Element slicing performance across Intel #S generations

Download the software at <http://github.com/cecm/girih>

Acknowledgment of INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. With support from INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT. Sponsored by INRIA, IOLeT, NVIDIA, CRAY, OSR, and Green IT.

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



Exploring a “Parallel Universe” for Numerical Linear Algebra: **HiCMA**



**David Keyes, Applied Mathematics & Computational Science
Director, Extreme Computing Research Center (ECRC)
King Abdullah University of Science and Technology**

**Kadir Akbudak, Rabab AlOmairy, Amani Alonazi, Wajih Boukaram, Ali Charara*,
Hatem Ltaief, Aleksandr Mikhalev, Dalal Sukkari, George Turkiyyah**, Rio Yokota*****

*** & ICL, UTennessee**

**** & American U of Beirut**

***** & Tokyo Tech**



Two universes of NLA exist side-by-side



c/o Instageeked.com

* 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 unités canoniques 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 auxquels conduit le ~~appelé~~ méthode des unités canoniques, mais nous remarquons 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~~ de n équations linéaires symétriques à n inconnues.

Considérons en effet le système suivant

I
$$\begin{cases} x_1^2 \gamma_1 + x_2^2 \gamma_2 + x_3^2 \gamma_3 + \dots + x_n^2 \gamma_n + c_1 = 0 \\ x_2^2 \gamma_1 + x_3^2 \gamma_2 + x_4^2 \gamma_3 + \dots + x_n^2 \gamma_n + c_2 = 0 \\ \vdots \\ x_n^2 \gamma_1 + x_1^2 \gamma_2 + x_2^2 \gamma_3 + \dots + x_{n-1}^2 \gamma_n + c_n = 0 \end{cases}$$

Effectuons la transformation linéaire représentée par le système :

II
$$\begin{cases} \gamma_1 = x_1^2 \lambda_1 + x_2^2 \lambda_2 + \dots + x_n^2 \lambda_n \\ \gamma_2 = x_2^2 \lambda_1 + x_3^2 \lambda_2 + \dots + x_n^2 \lambda_n \\ \vdots \\ \gamma_n = x_n^2 \lambda_1 + x_1^2 \lambda_2 + \dots + x_{n-1}^2 \lambda_n \end{cases}$$

Le système II d'équations I demande n inconnues γ_j et sera remplacé par le système III demandant les n inconnues λ_k permettant, à l'aide de II, de calculer les valeurs de γ_j .

III
$$\begin{cases} A_1^2 \lambda_1 + A_2^2 \lambda_2 + \dots + A_n^2 \lambda_n + C_1 = 0 \\ A_2^2 \lambda_1 + A_3^2 \lambda_2 + \dots + A_n^2 \lambda_n + C_2 = 0 \\ \vdots \\ A_n^2 \lambda_1 + A_1^2 \lambda_2 + \dots + A_{n-1}^2 \lambda_n + C_n = 0 \end{cases}$$



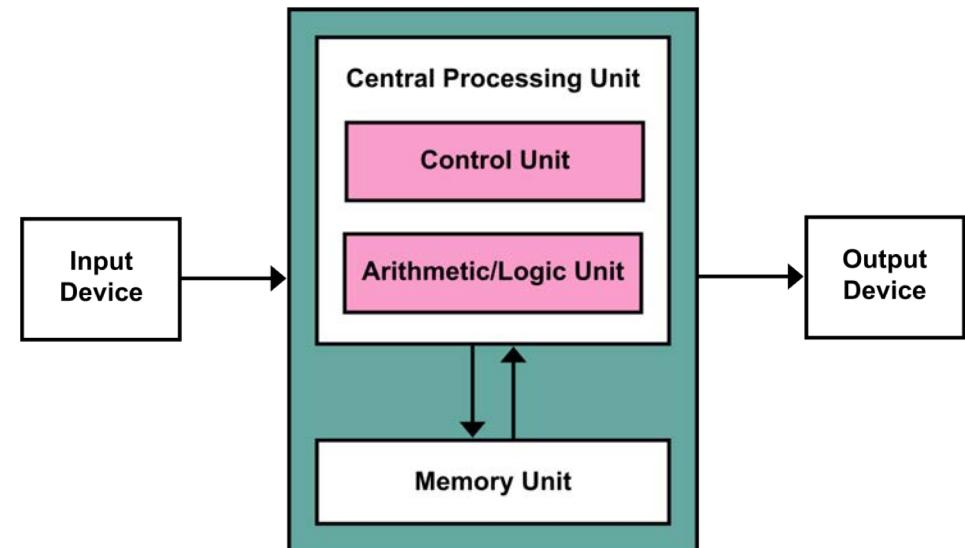
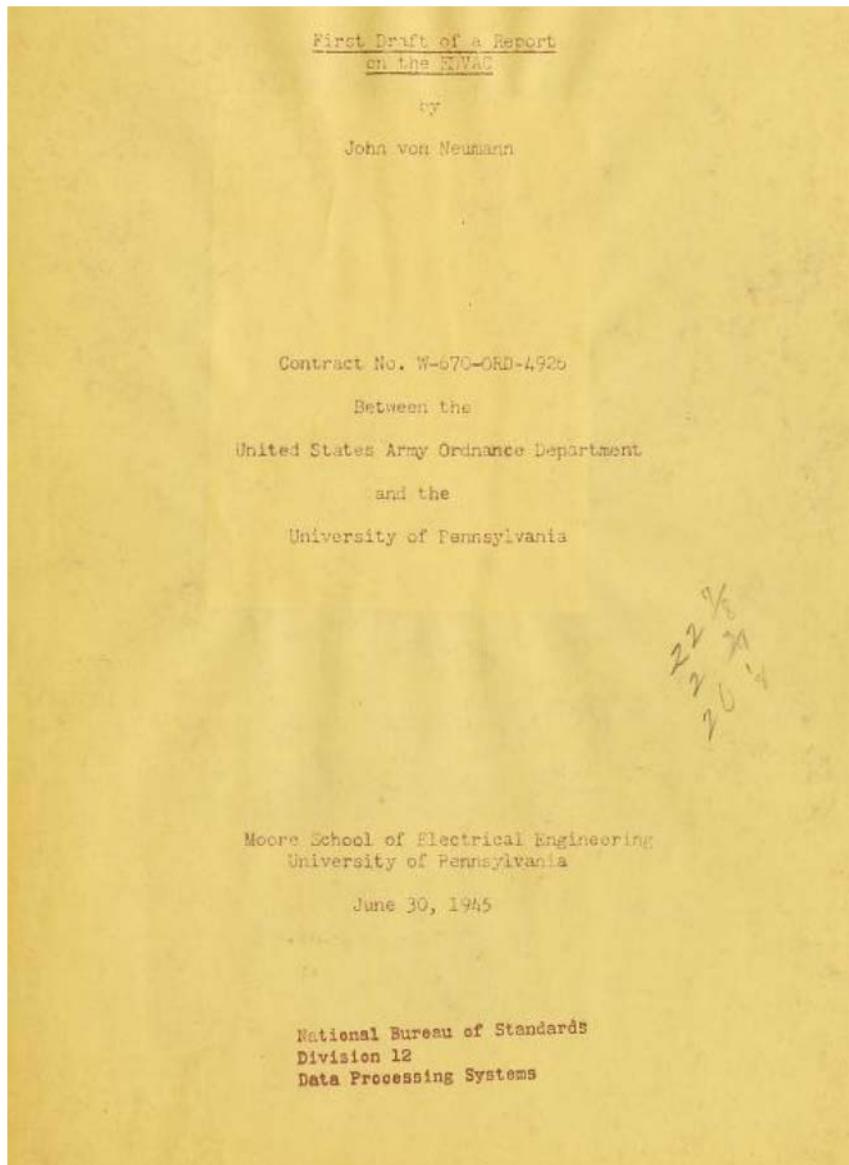
```

 $\ell_{11} = \sqrt{a_{11}};$ 
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

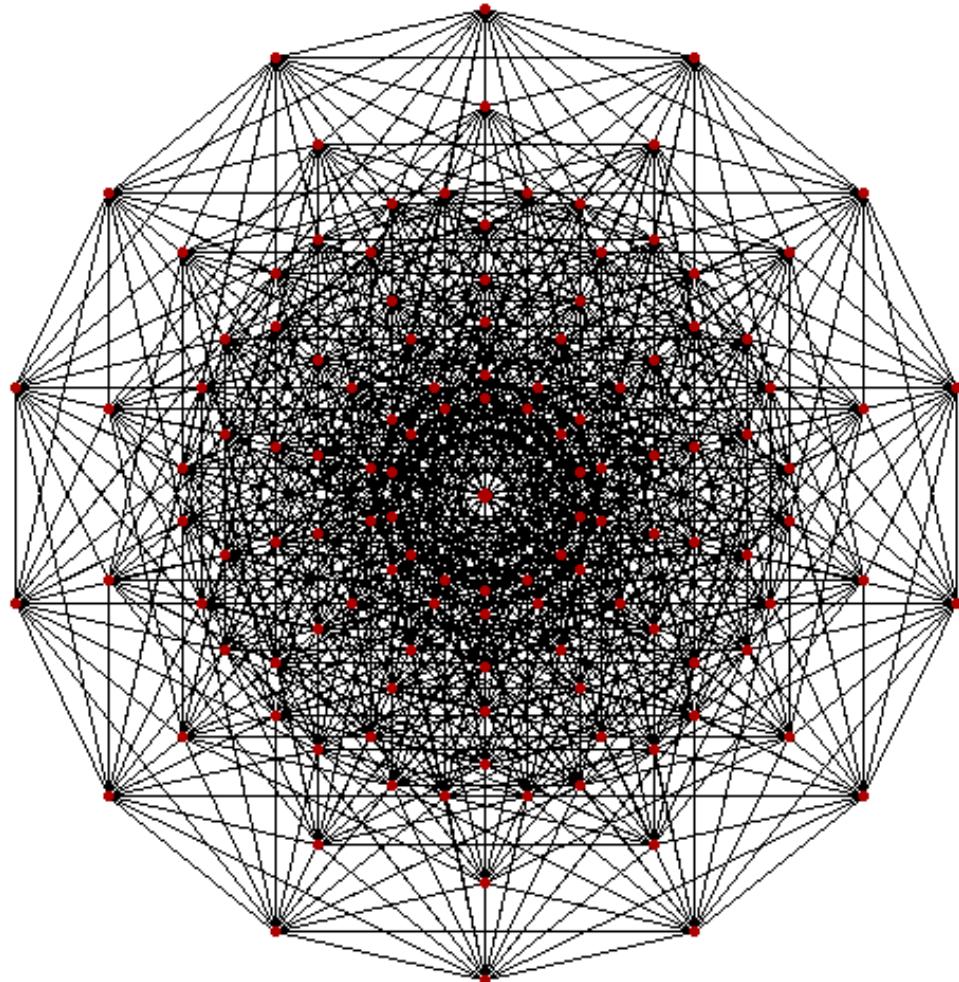
```

triangular recurrence

Architectures were flat, as well (vN, 1945)



Since 1985: “horizontal” structure...

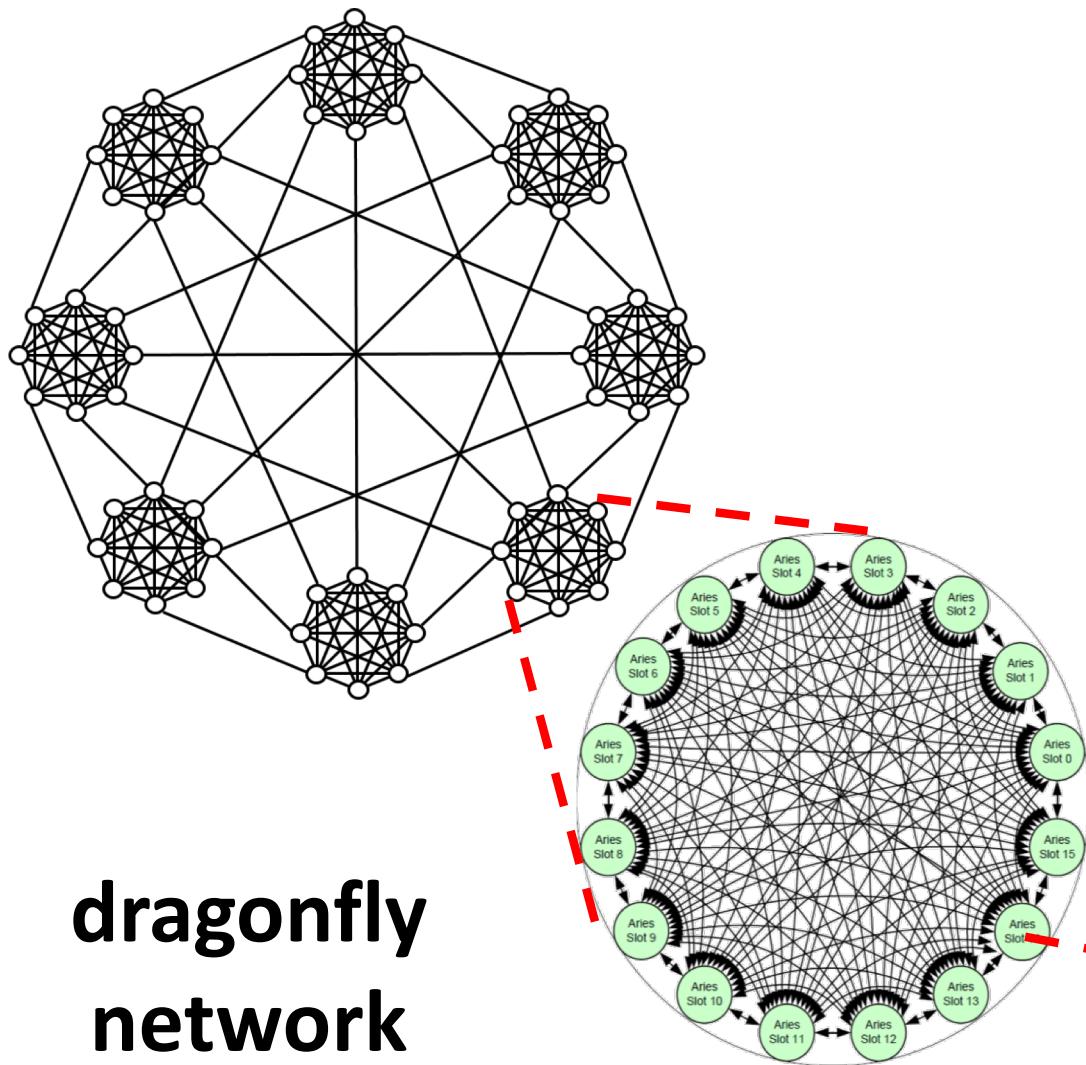


128-node hypercube



c/o Computer History Museum, Mountain View, CA

Today's “horizontal” structure...

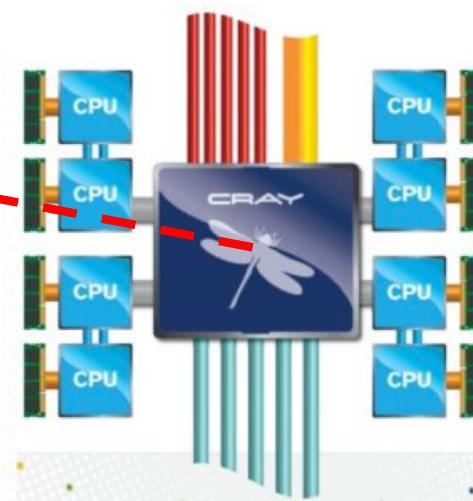


dragonfly
network

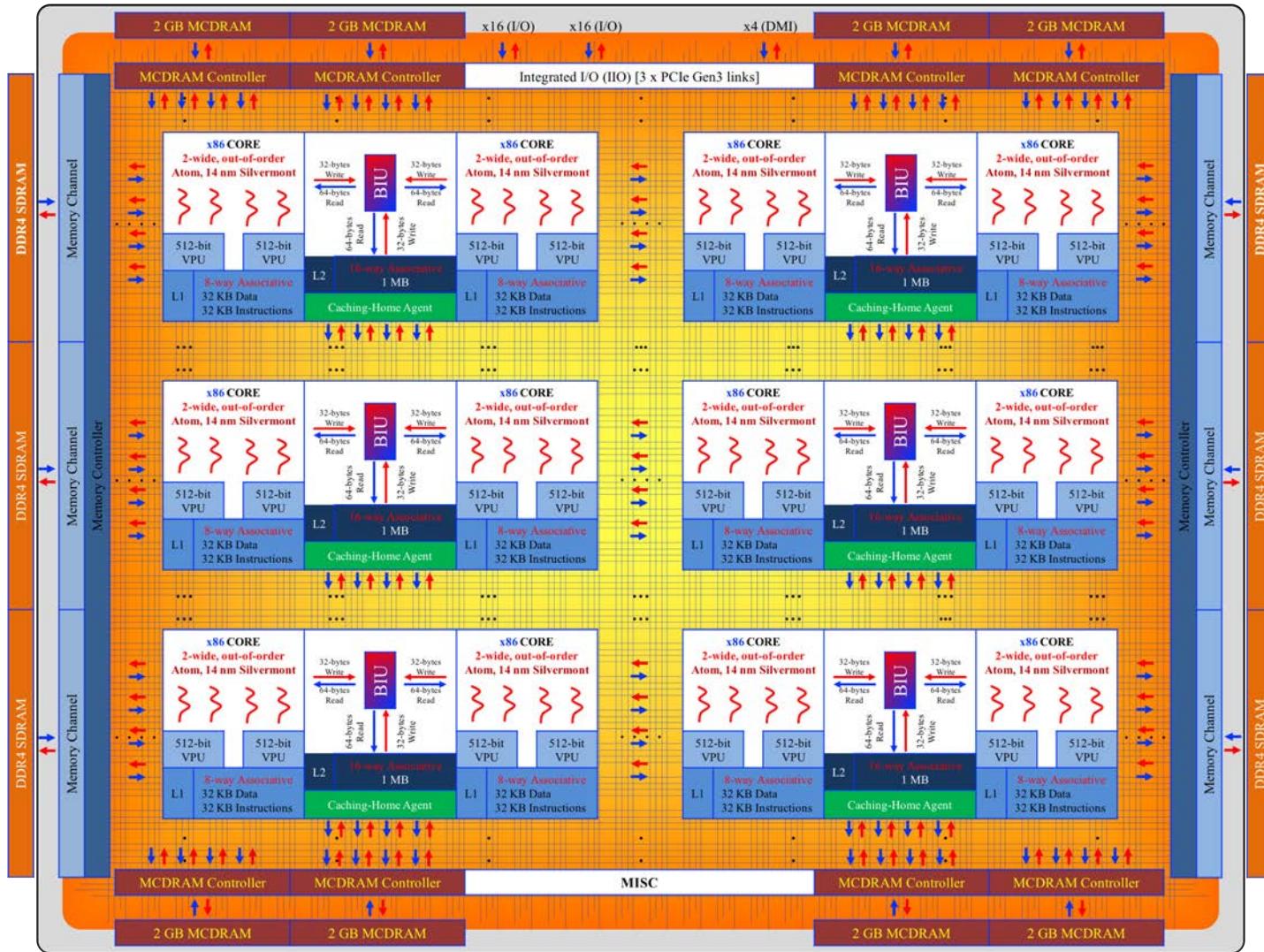
c/o Cray

Cray's “Aries” network

- copper *within* a cabinet
- optical *between* cabinets
- scalability of a fat tree
- cost of a torus
- maximum of three hops between any pair of the 200,000 Xeon cores in KAUST's Cray XC40



And now add: “vertical” structure



Intel's 256-core Knights Landing Processor

- nested levels of cache
 - MCDRAM
 - DDR4 SDRAM
- all within a 3 Tflop/s node

Two decades of evolution

1997



2017



ASCI Red at Sandia

1.3 TF/s, 850 KW

Cavium ThunderX2

\sim 1.1 TF/s, \sim 0.2 KW

3.5 orders of
magnitude

Hierarchies do not necessarily match!

As humans managing implementation complexity,
we would all prefer:

- hierarchical algorithms on flat architectures

or even (suboptimally)

- flat algorithms on hierarchical architectures

Reality

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

- A 4,000-node subset of ORNL's Summit sustains 1.88 ExaOp/s of mixed precision on a genomics application
- Majority of these operations are half-precision (16-bit floating point) NVIDIA tensor-core matrix-matrix multiplies

* paraphrase of D. Rumsfeld, Cable News Network, 8 Dec 2004.

Now: hierarchy of precisions

SIAM J. SCI. COMPUT.
Vol. 40, No. 2, pp. A817–A847

© 2018 SIAM. Published by SIAM under the terms
of the Creative Commons 4.0 license

ACCELERATING THE SOLUTION OF LINEAR SYSTEMS BY ITERATIVE REFINEMENT IN THREE PRECISIONS*

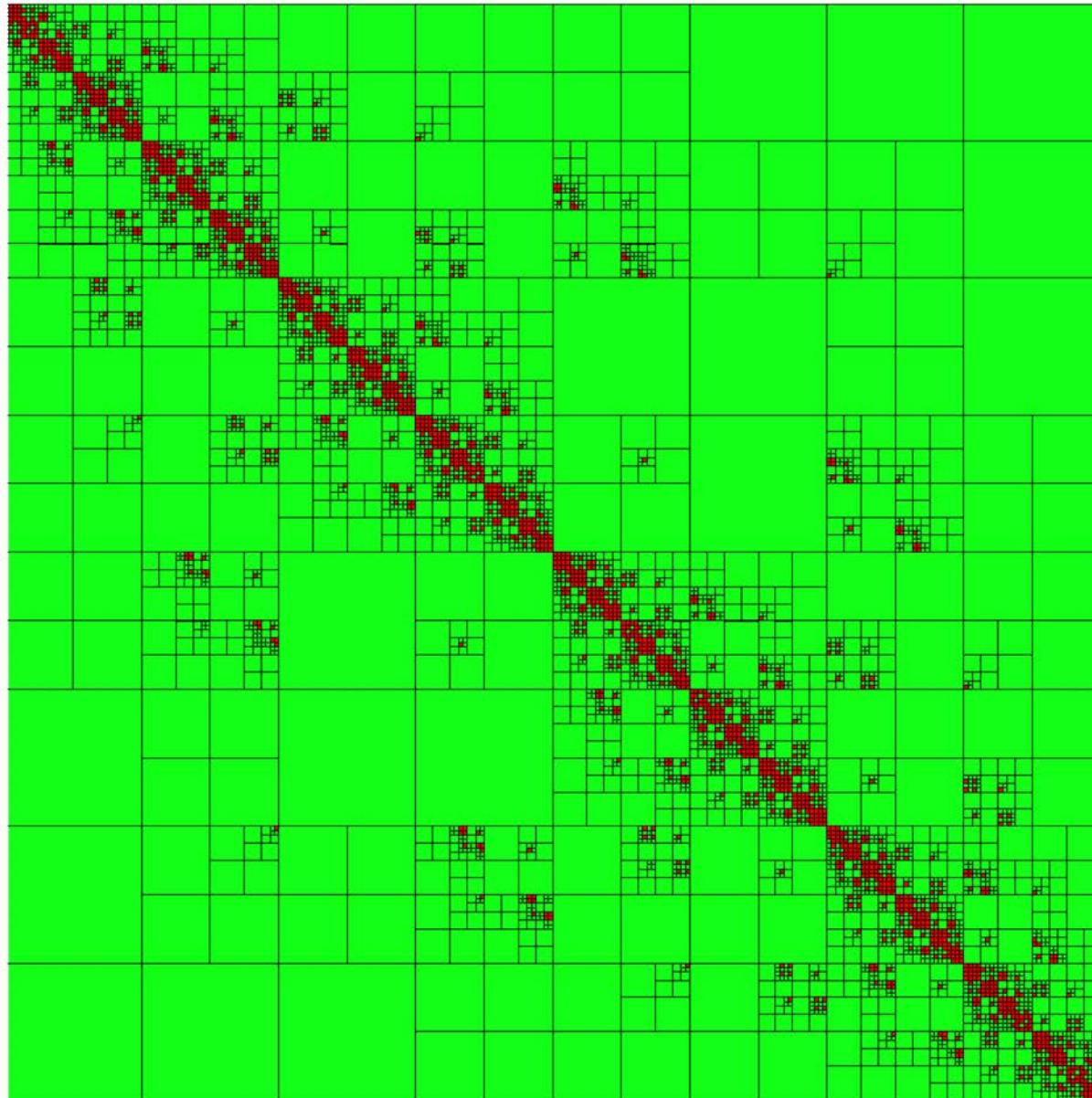
ERIN CARSON[†] AND NICHOLAS J. HIGHAM[‡]

TABLE 1.1

Summary of existing rounding error analyses for iterative refinement in floating point arithmetic indicating (a) whether the analyses apply to LU factorization only or to an arbitrary solver, (b) whether the backward or forward error analyses are componentwise (“comp”) or normwise (“norm”), and (c) the assumptions on the precisions u_f , u_s , u , u_r in Algorithm 1.1 ($u_f = u$ and $u_s = u_f$ unless otherwise stated).

	Year	Solver	Forward error	Backward error	Precisions
Moler [27]	1967	LU	norm	—	$u \geq u_r$
Stewart [36]	1973	LU	norm	—	$u \geq u_r$
Jankowski et al. [22]	1977	arb.	norm	norm	$u = u_r$
Skeel [34]	1980	LU	comp	comp	$u \geq u_r$
Higham [17]	1991	arb.	comp	comp	$u = u_r$
Higham [18], [19]	1997	arb.	comp	comp	$u \geq u_r$
Tisseur [37]	2001	arb.	norm	norm	$u \geq u_r$
Langou et al. [24]	2006	LU	norm	norm	$u_f \geq u = u_r$
Carson and Higham [9]	2017	arb.	comp	—	$u \geq u_r$
This work	2017	arb.	comp	comp, norm	$u_f \geq u_s \geq u \geq u_r$

Now: hierarchy of ranks



Architectural challenge

- **Memories are hierarchical in an increasing number of levels**
 - incentive to tune algorithms for register & cache reuse
 - additional flop/s cost little, within a given workingset of data that fits in highest level cache
 - more computation leading to less communication and/or synchronization may be a good trade-off

It's not just bandwidth; it's energy

- Access SRAM (registers, cache) $\sim 10 \text{ fJ/bit}$
- Access DRAM on chip $\sim 1 \text{ pJ/bit}$
- Access HBM/MCDRAM (few mm) $\sim 10 \text{ pJ/bit}$
- Access DDR3 (few cm) $\sim 100 \text{ pJ/bit}$

$\sim 10^4$ advantage in energy for staying in cache!

similar ratios for *latency* as for *bandwidth* and *energy*

Algorithmic imperatives

- 1) Reside “high” on the memory hierarchy**
 - ◆ as close as possible to the processing elements
- 2) Reduce communication and synchrony**
 - ◆ in frequency and/or span



Widely applicable strategies

- 1) Employ dynamic runtime systems based on directed acyclic task graphs (DAGs)
 - ◆ e.g., Charm++, Quark, StarPU, Legion, OmpSs, HPX, ADLB, Argo, ParSec
 - ◆ dynamic scheduling capabilities in OpenMP
 - 2) Exploit data sparsity of hierarchically low-rank type
 - ◆ meet the “curse of dimensionality” with the “blessing of low rank”
 - 3) Code libraries to various architecture while presenting high-level application programmer interface
-

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
-

2) Hierarchically low-rank 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 compression
 - ◆ not all operators compress well
-

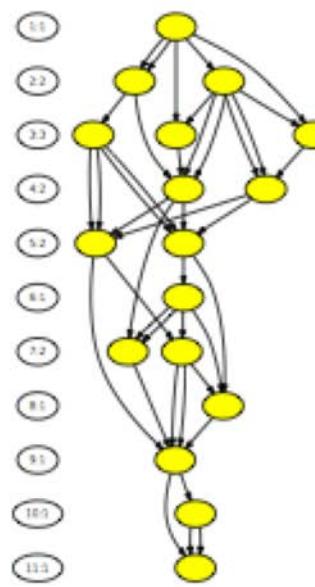
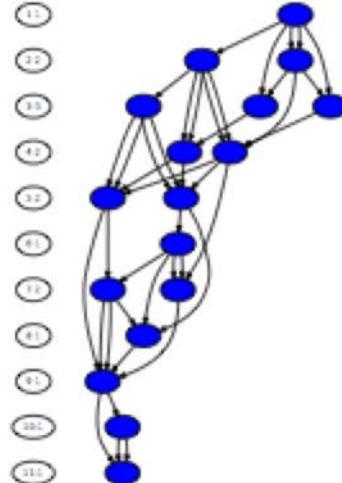
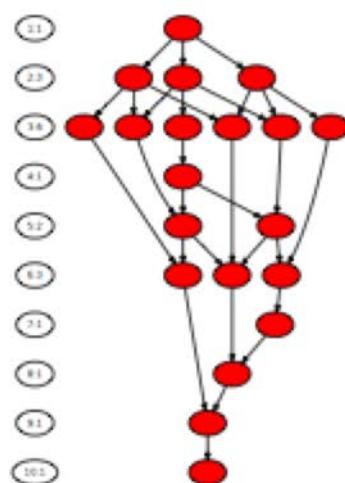
3) Code to the architecture

- **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
-

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

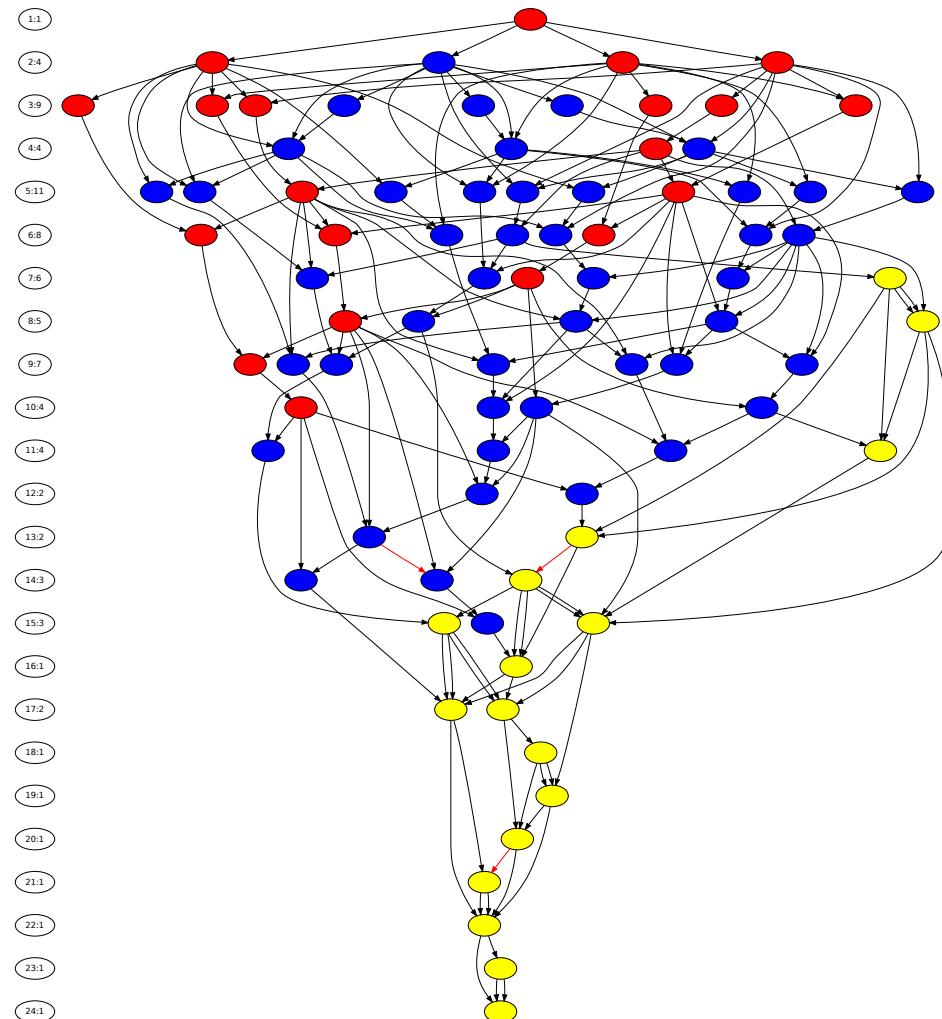
$$Ax = \lambda Bx$$

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



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) Reduce memory footprint and operation complexity with low rank

- Replace dense blocks with hierarchical representations when they arise during matrix operations
 - use high accuracy (high rank, but typically less than full) to build “exact” solvers
 - use low accuracy (low rank) to build preconditioners
 - Tune block structure and rank parameters to variety of hardware configurations
-

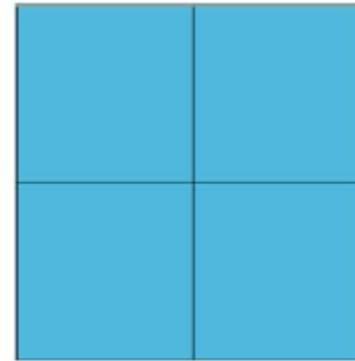
Key tool: hierarchical matrices

- [Hackbusch, 1999] : off-diagonal blocks of typical differential and integral operators have low effective rank
 - By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n :
 - polynomial in k
 - lin-log in n
 - constants carry the day
 - Such hierarchical representations navigate a compromise
 - fewer blocks of larger rank (“weak admissibility”) or
 - more blocks of smaller rank (“strong admissibility”)
-

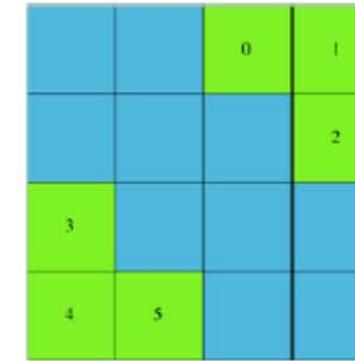
Recursive construction of an \mathcal{H} -matrix



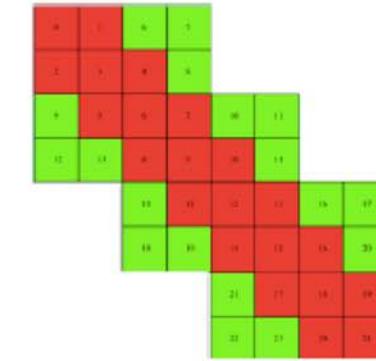
Step 0



Step 1



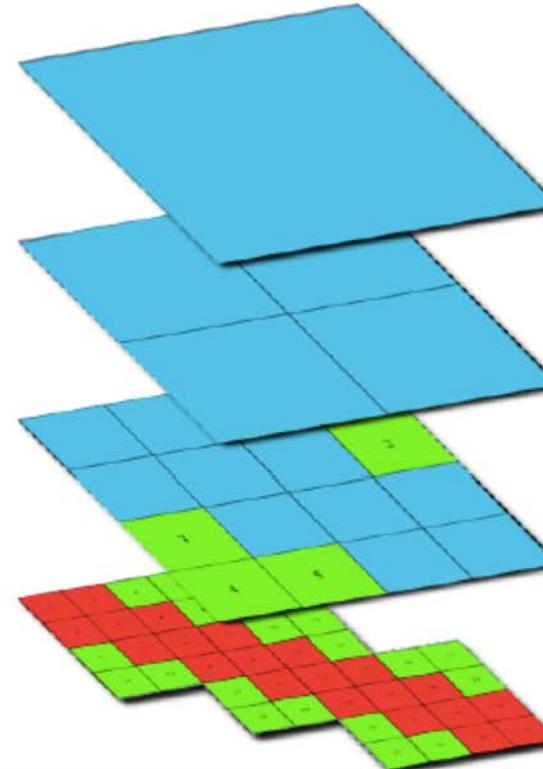
Step 2



Step 3

Specify two parameters:

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

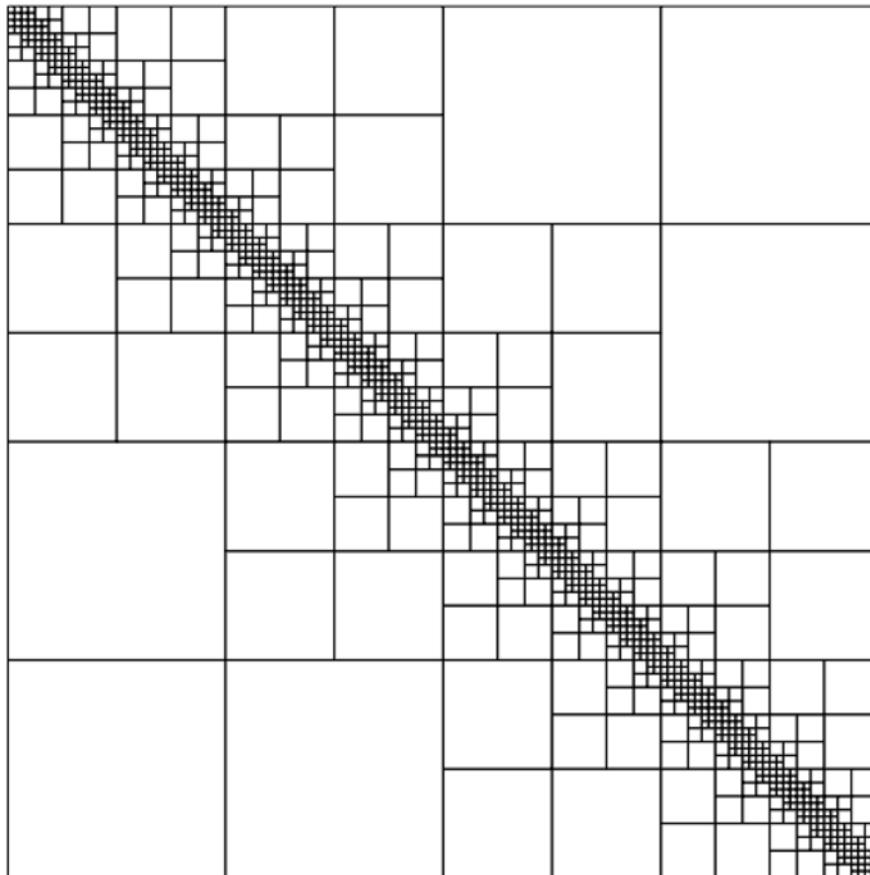


Until each block is acceptably small:

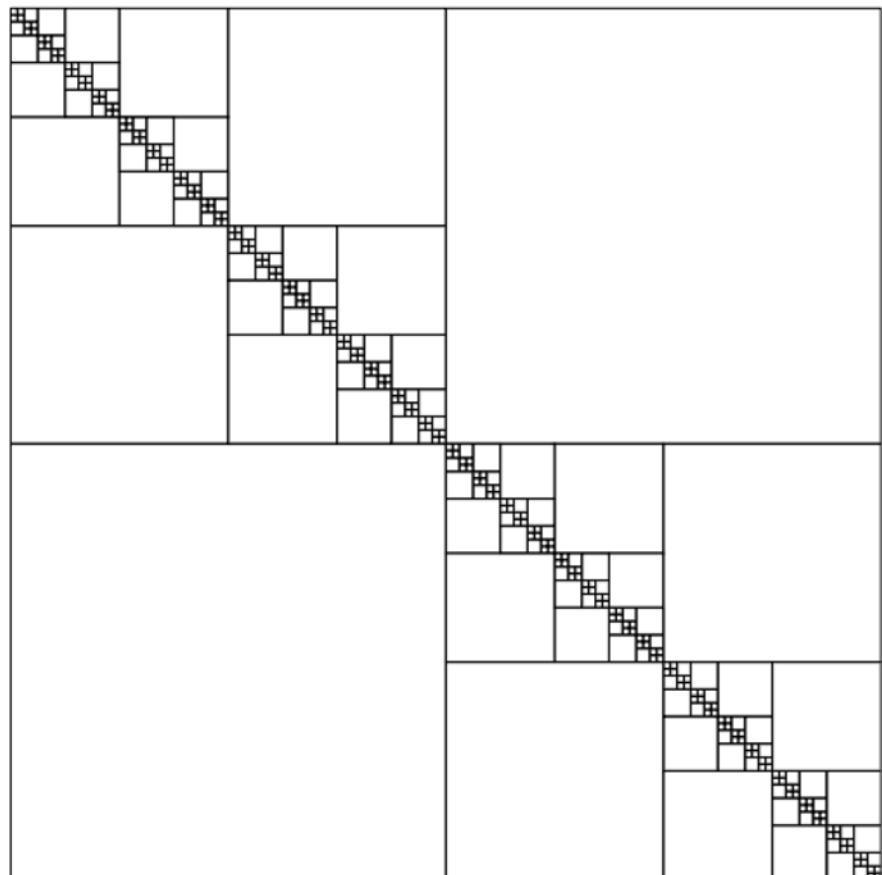
- Is rank acceptably small?
- If not, subdivide block

Take union of leaf blocks

Tree-like structures of “Standard (strong)” vs. “weak” admissibility



strong admissibility



weak admissibility

after Hackbusch, *et al.*, 2003

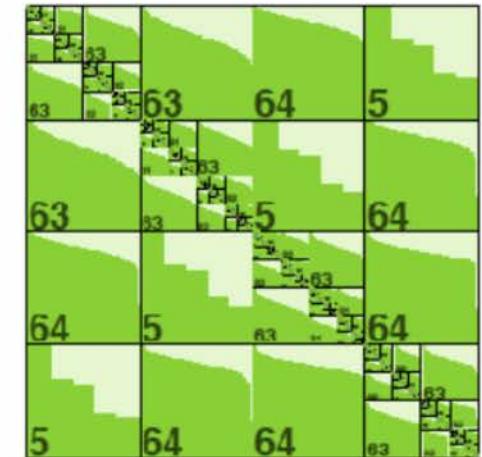
Hierarchically low-rank “renaissance”

Replace dense linear algebra

Compute : $\mathcal{O}(N^3) \longrightarrow \mathcal{O}(k^a N \log^b N)$

Memory : $\mathcal{O}(N^2) \longrightarrow \mathcal{O}(kN)$

Hierarchical off-diagonal blocks
Approximated with rank k
 a and b are small constants



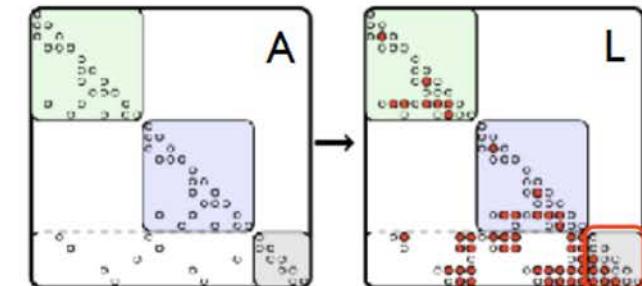
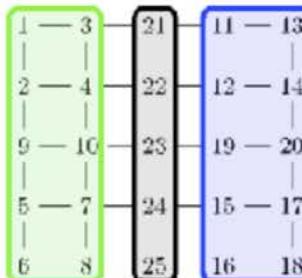
Augment sparse linear algebra

$$\begin{matrix} \text{Sparse matrix} & \times & \text{Vector} \\ \text{Sparse matrix} & \times & \text{Matrix} \\ \text{Sparse matrix} & - & \text{I} \end{matrix}$$

Sparse direct solvers

Schur complement (frontal matrix) is dense but numerically low-rank

Nested dissection



Schur complement

Iterative solvers

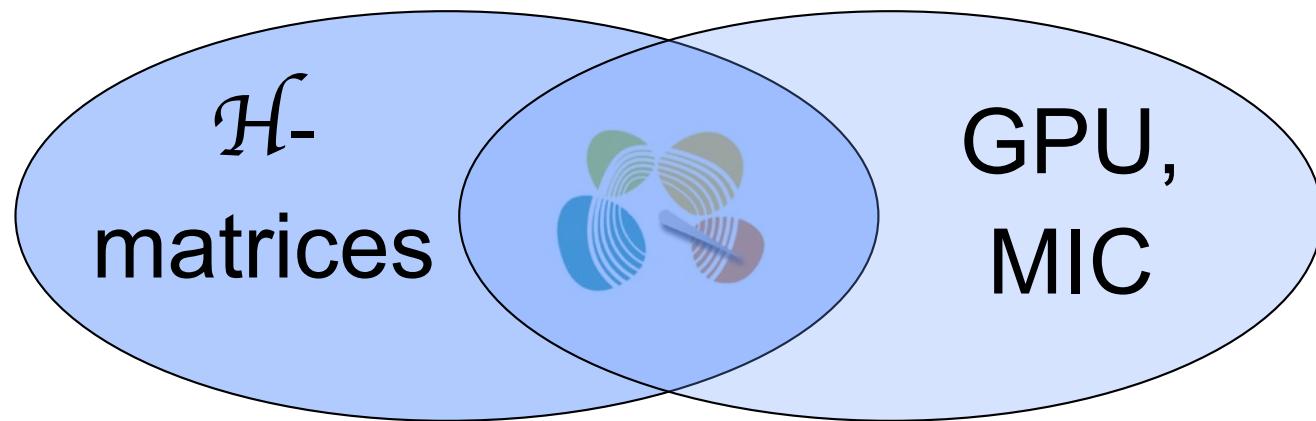
Use small k to precondition

Less sensitive to matrix condition than multigrid

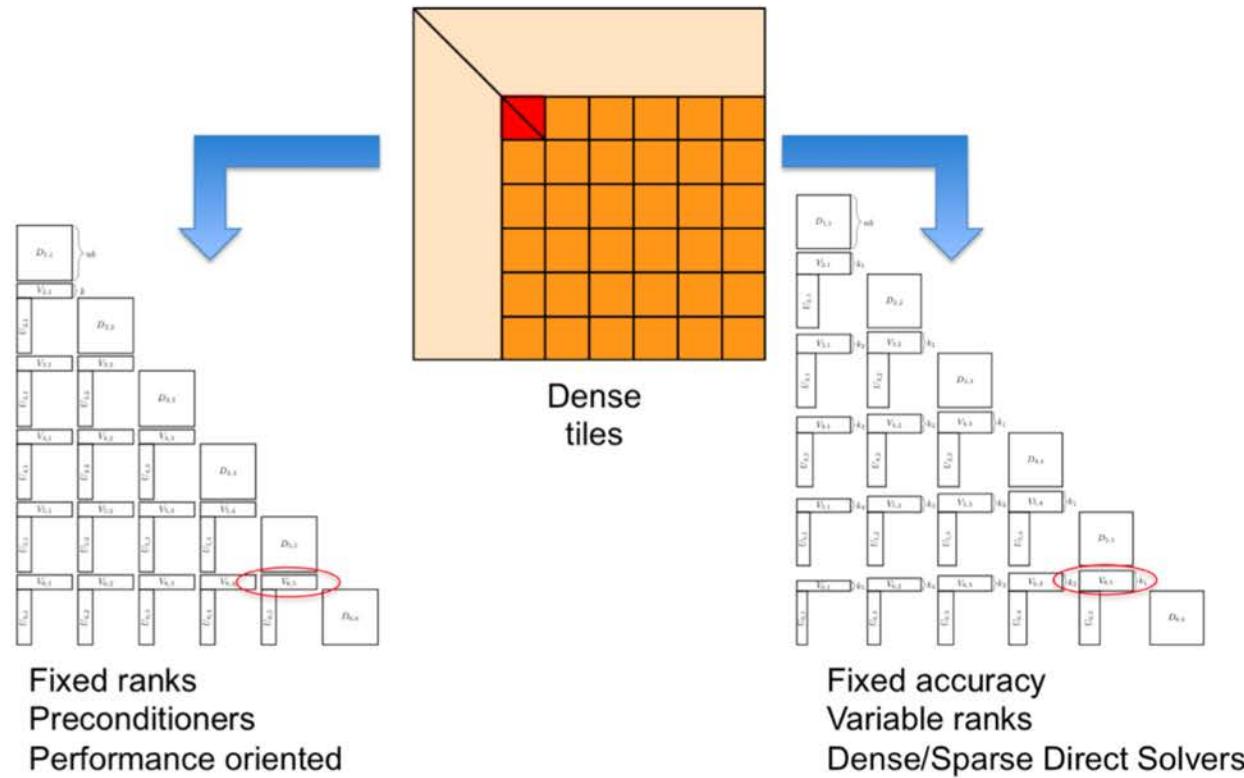
Hierarchical algorithms and extreme scale

Must address the *tension* between

- highly uniform vector, matrix, and general SIMD operations
- hierarchical algorithms with tree-like data structures and scale recurrence



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.

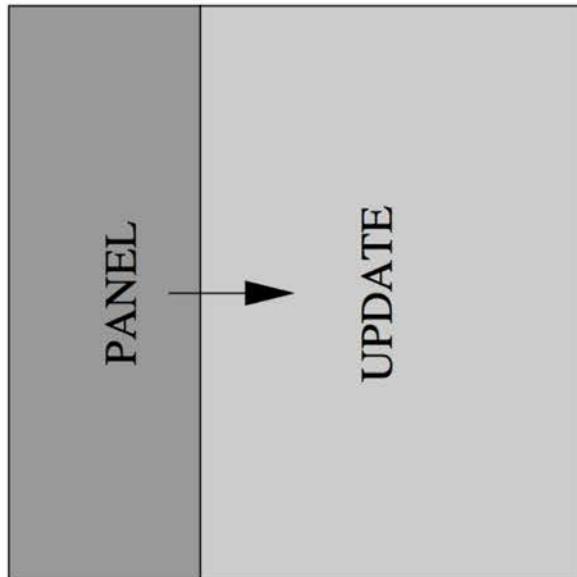
Example: Cholesky

The Cholesky factorization of an $N \times N$ real symmetric, positive-definite matrix A has the form

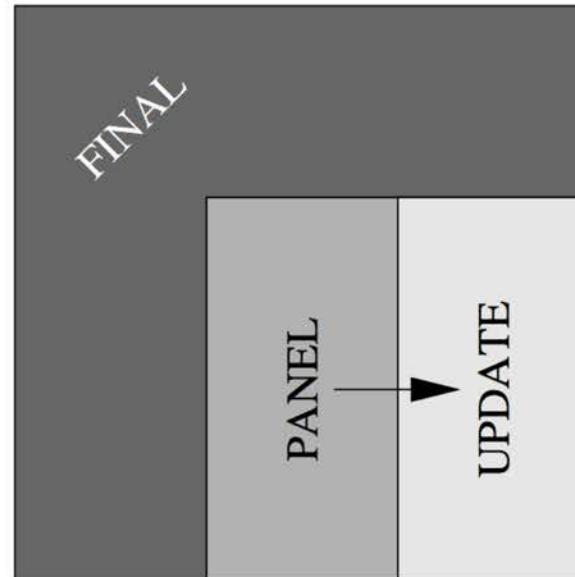
$$A = LL^T,$$

where L is an $N \times N$ real lower triangular matrix with positive diagonal elements.

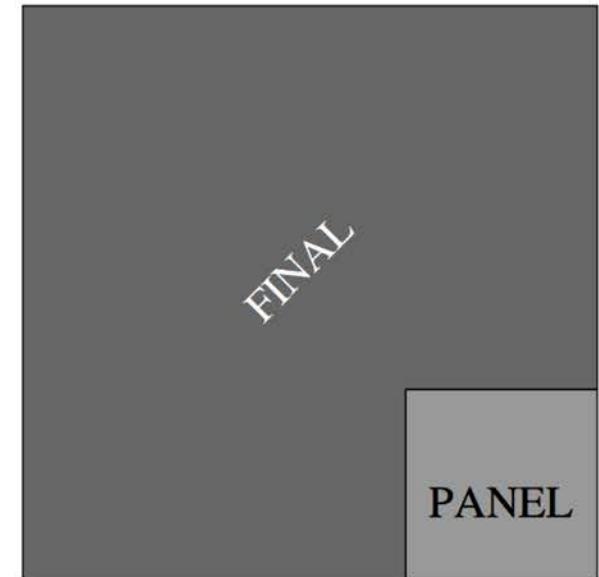
Panel algorithm (LAPACK's POTRF)



(a) First step.



(b) Second step.



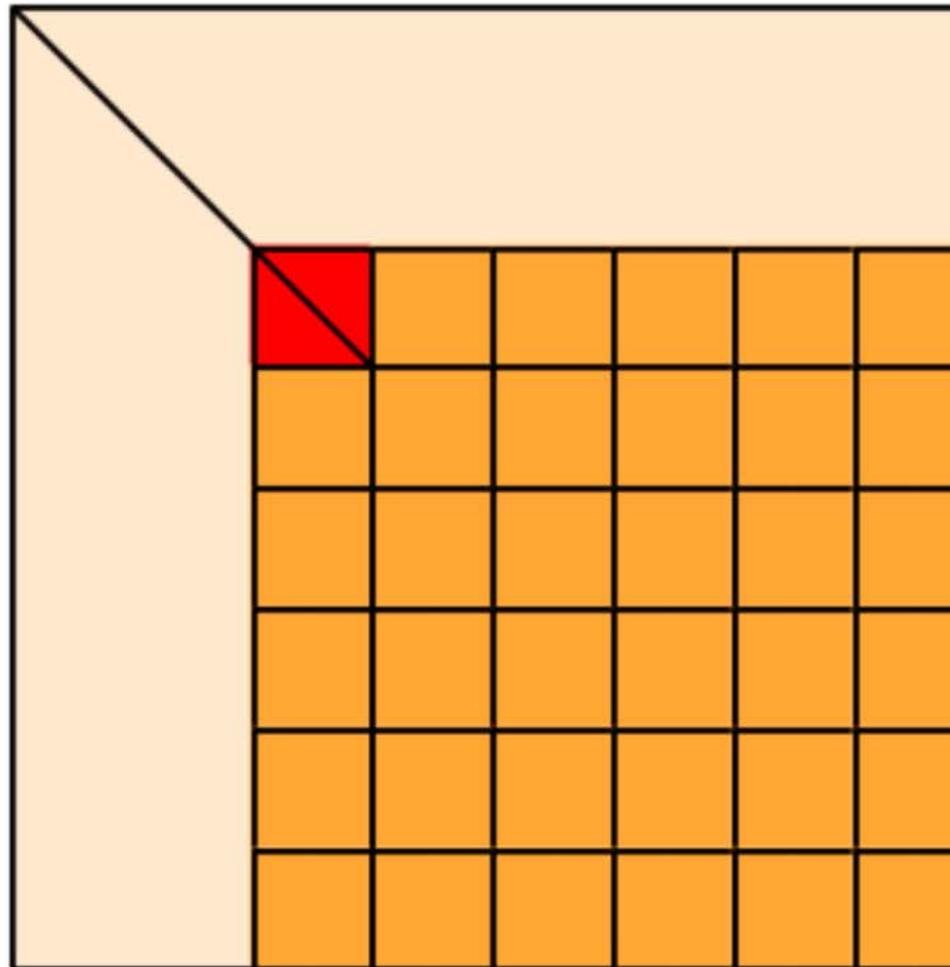
(c) Third step.

Left- and right-looking variants, column-blocked for BLAS3

Decreasing concurrency

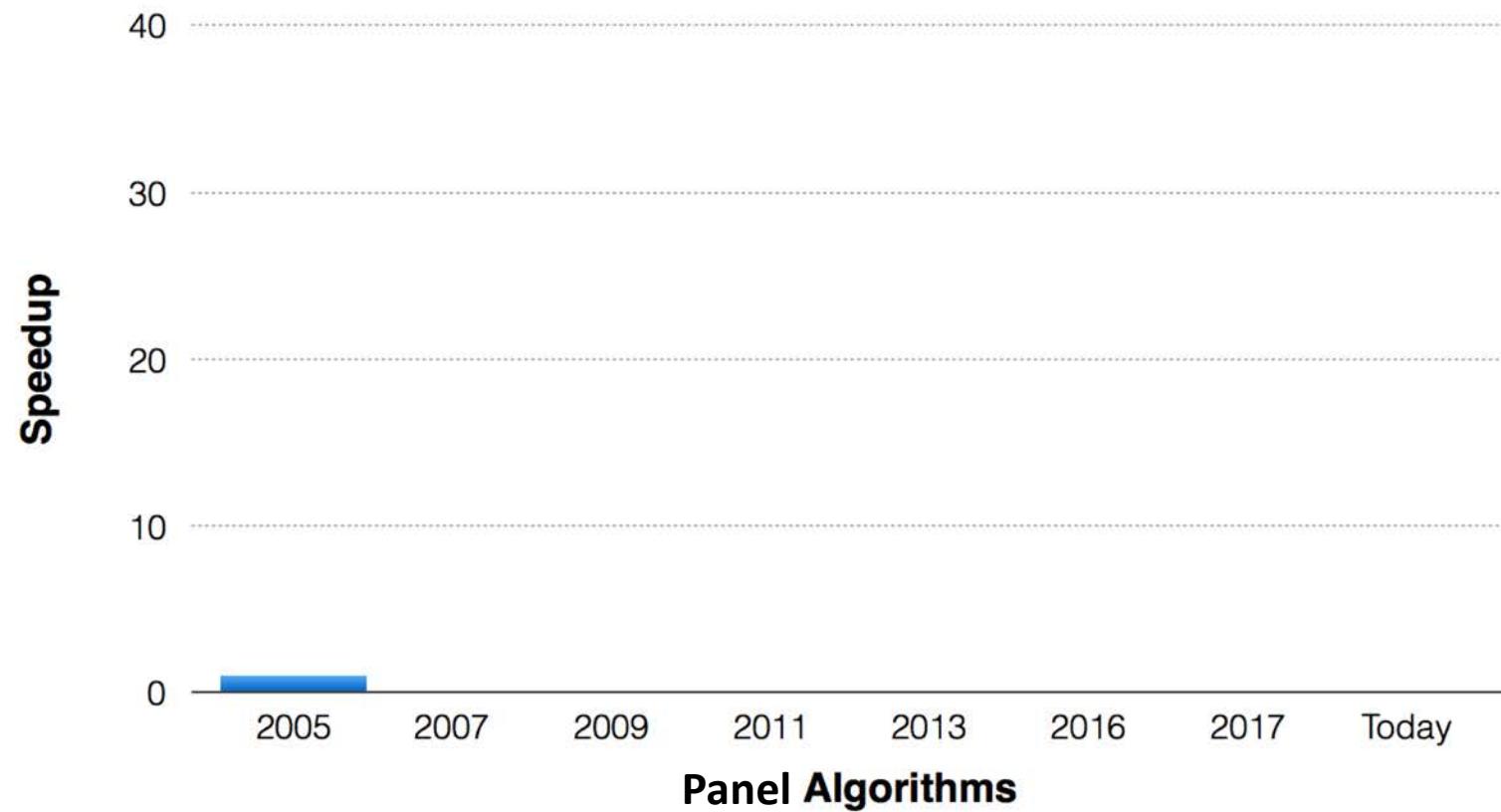
Artifactual over-ordering

Tile algorithm (PLASMA, MAGMA, Chameleon)

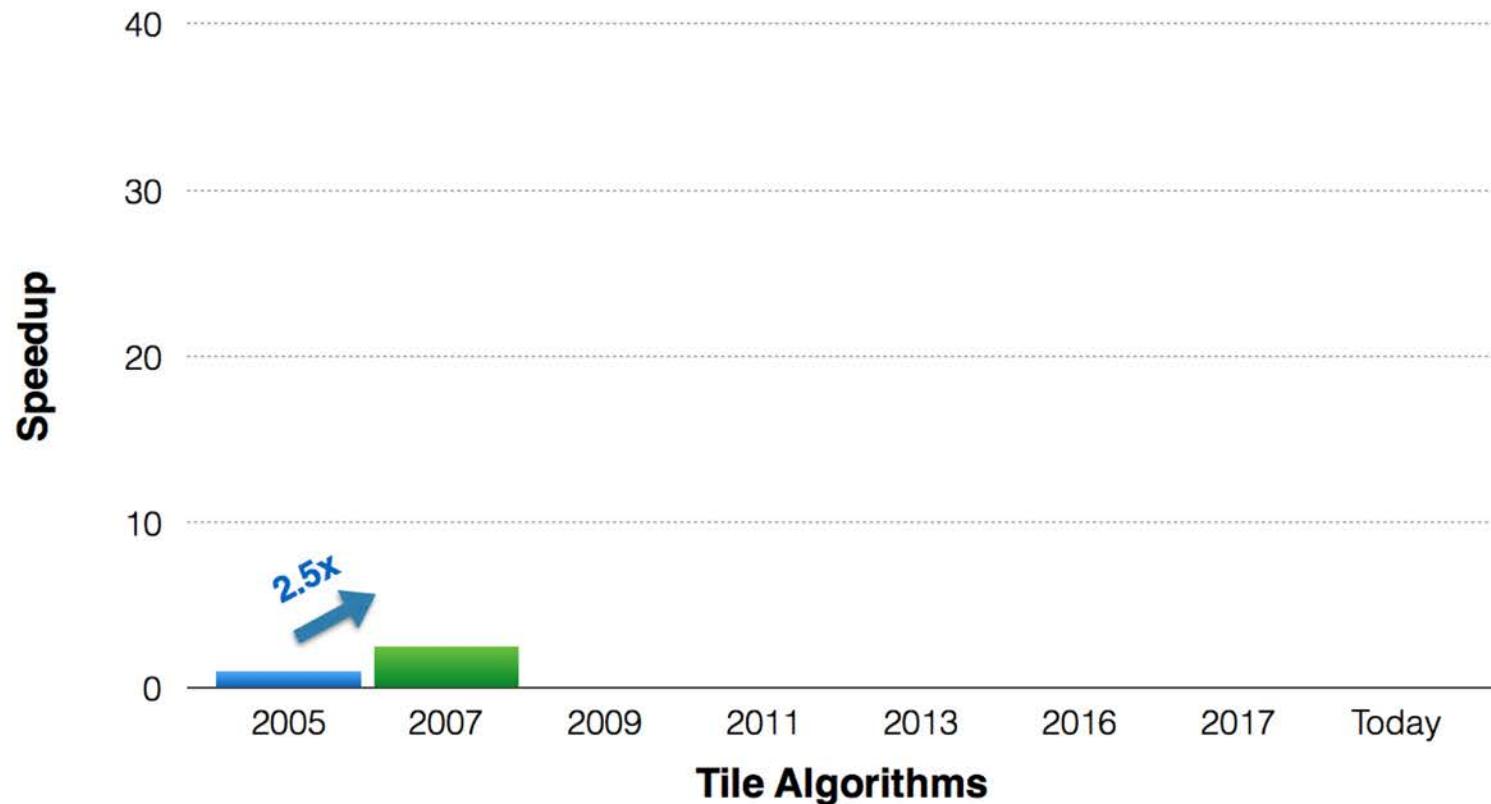


Implemented with DAG-based scheduling

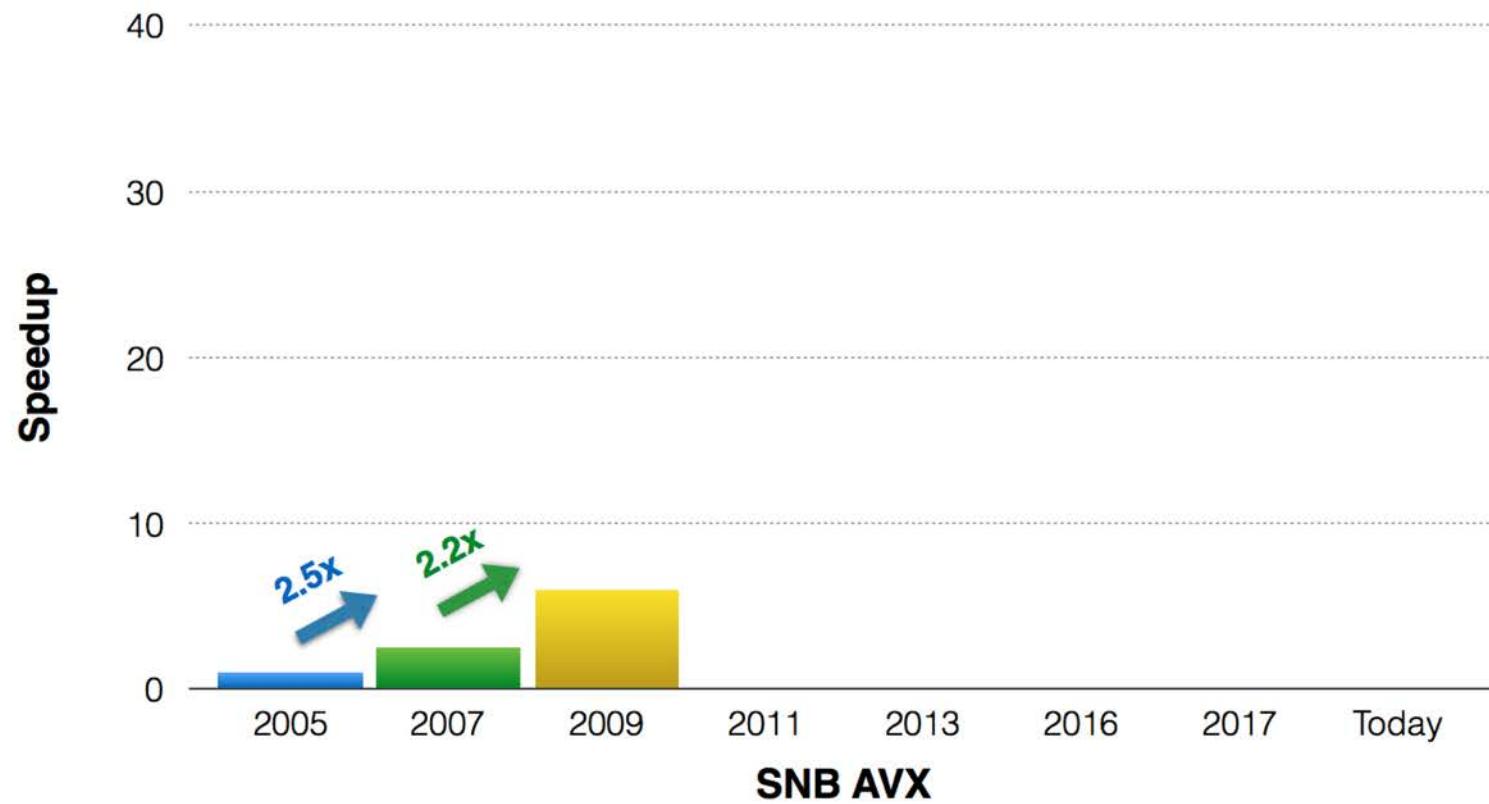
Performance evolution of dense Cholesky



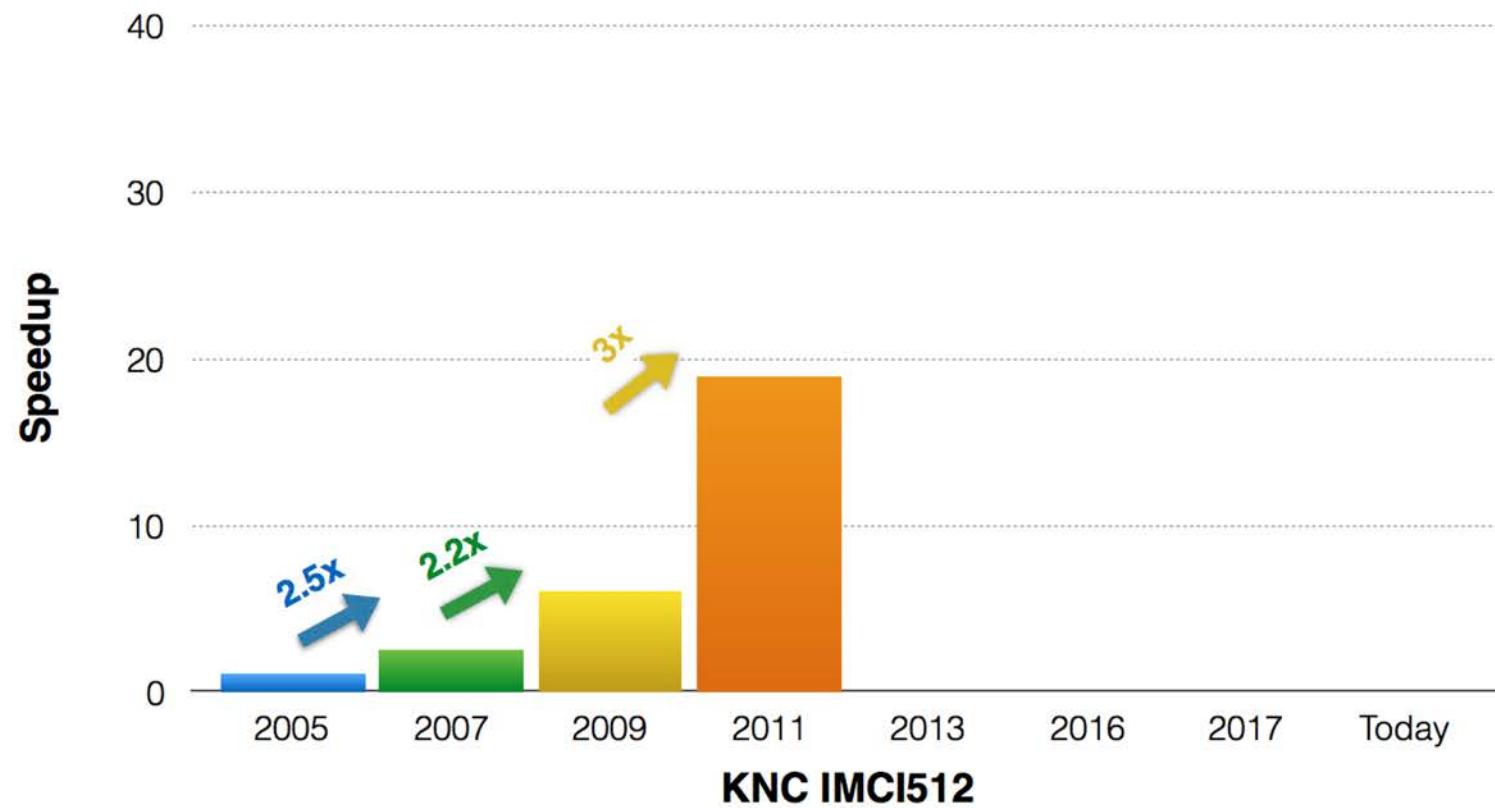
Performance evolution of dense Cholesky



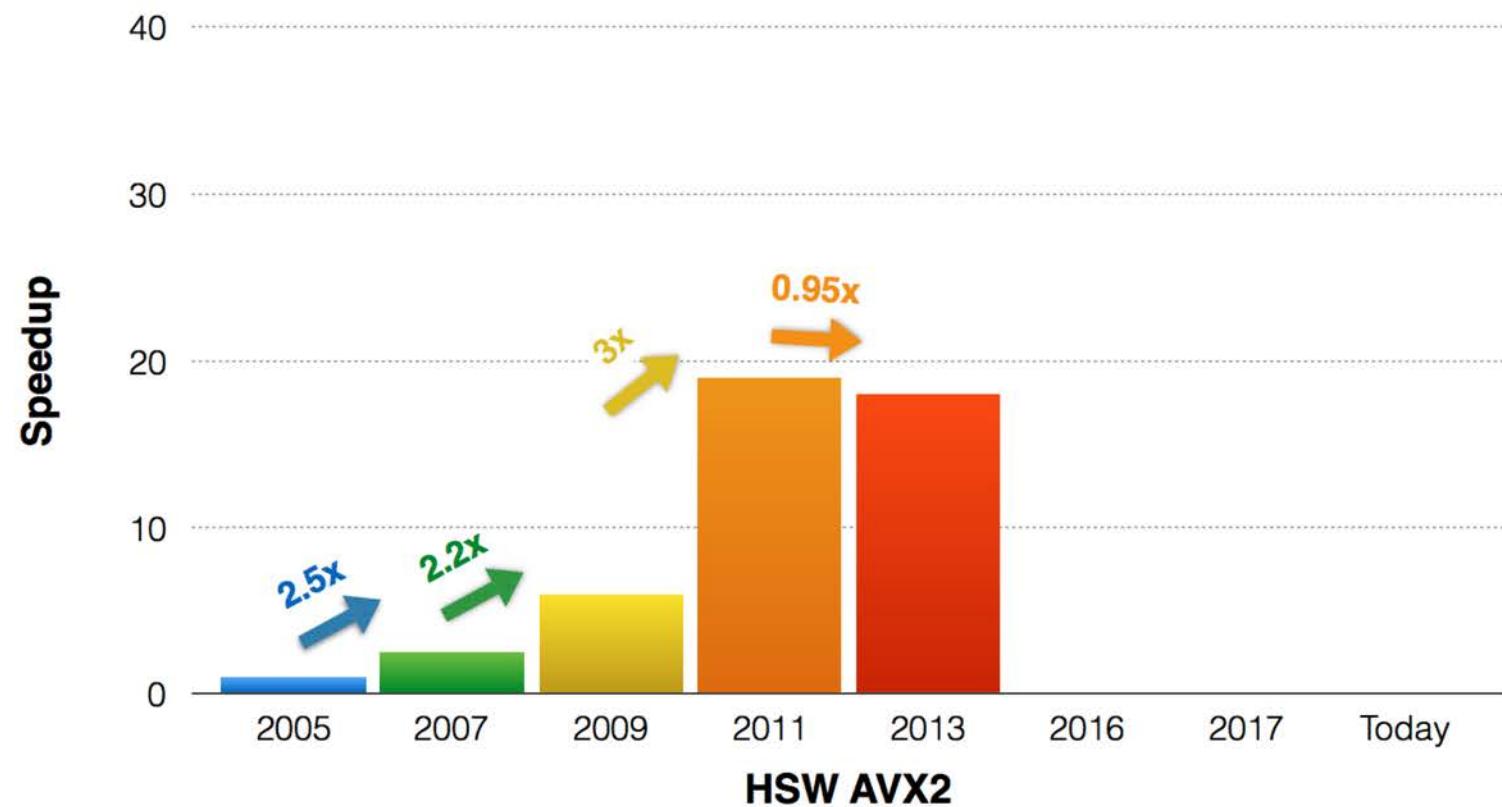
Performance evolution of dense Cholesky



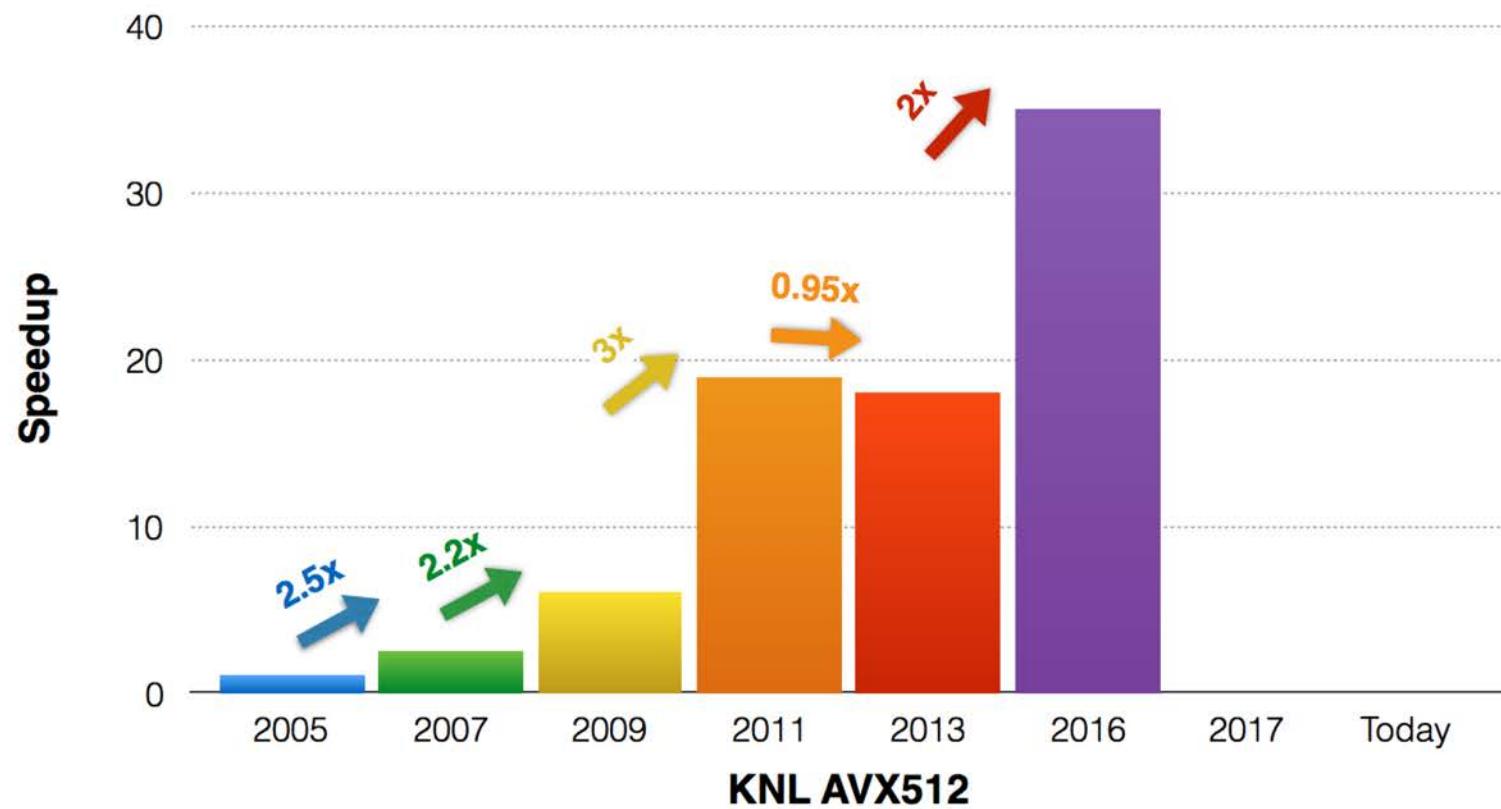
Performance evolution of dense Cholesky



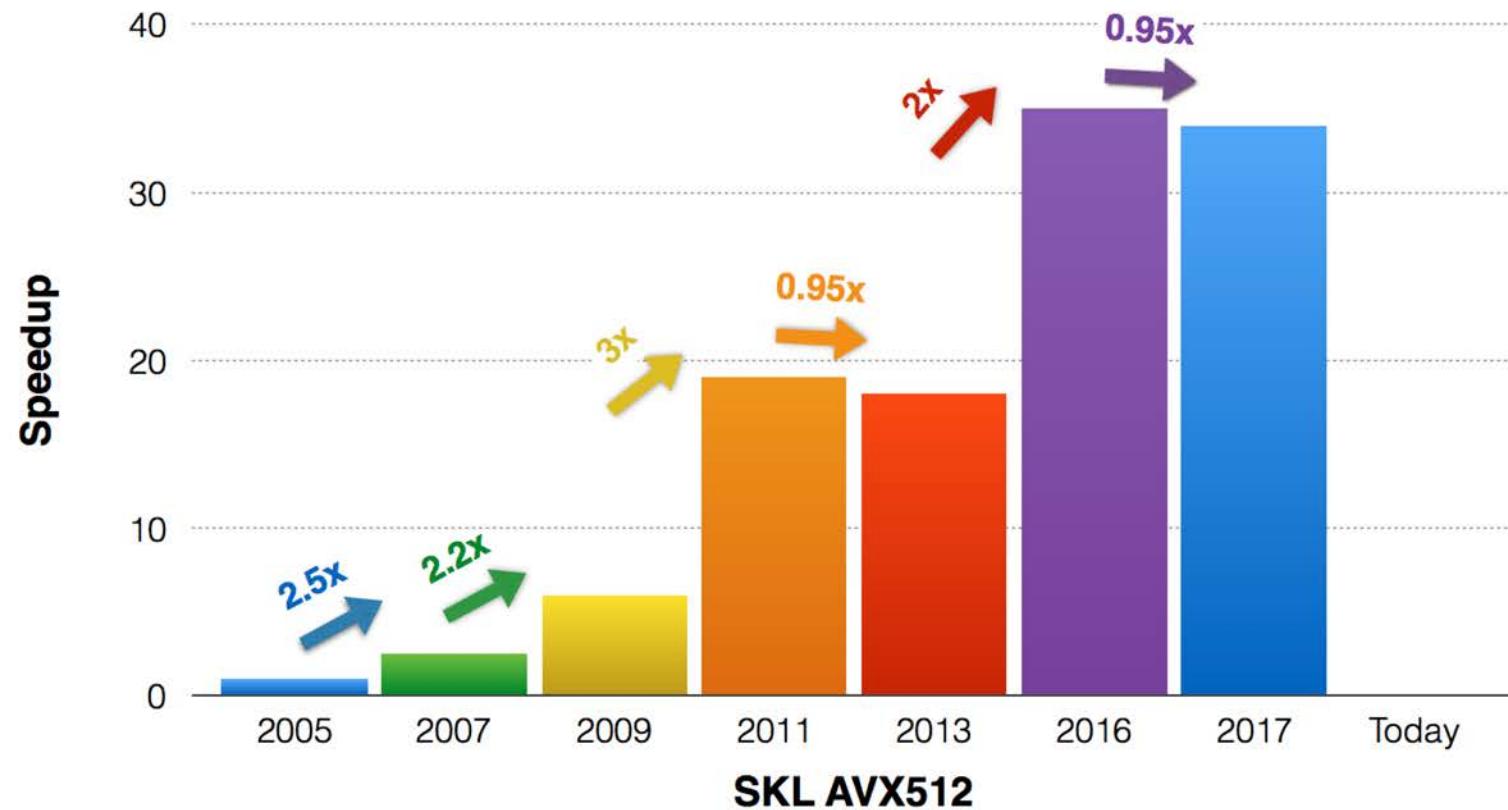
Performance evolution of dense Cholesky



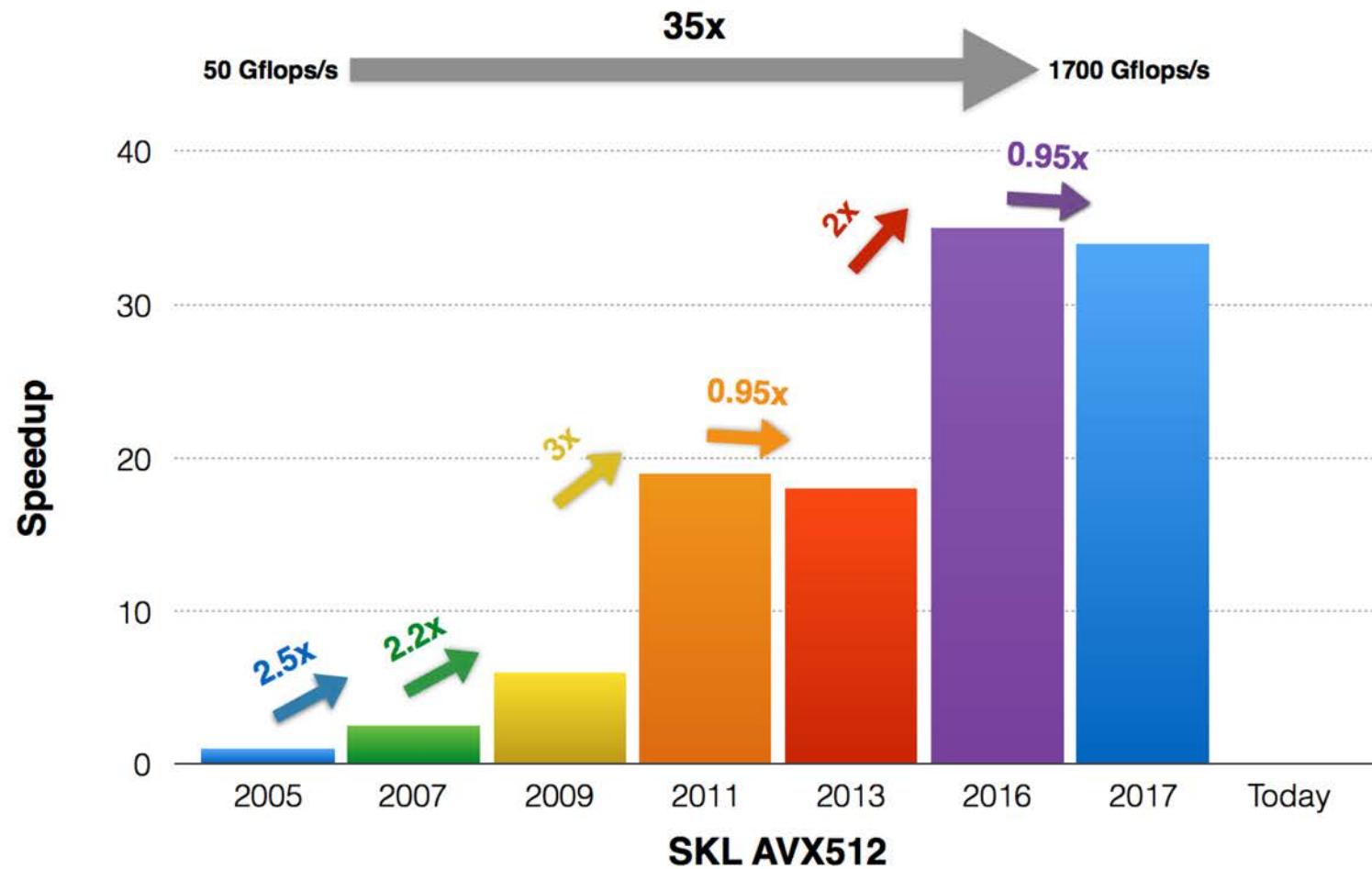
Performance evolution of dense Cholesky



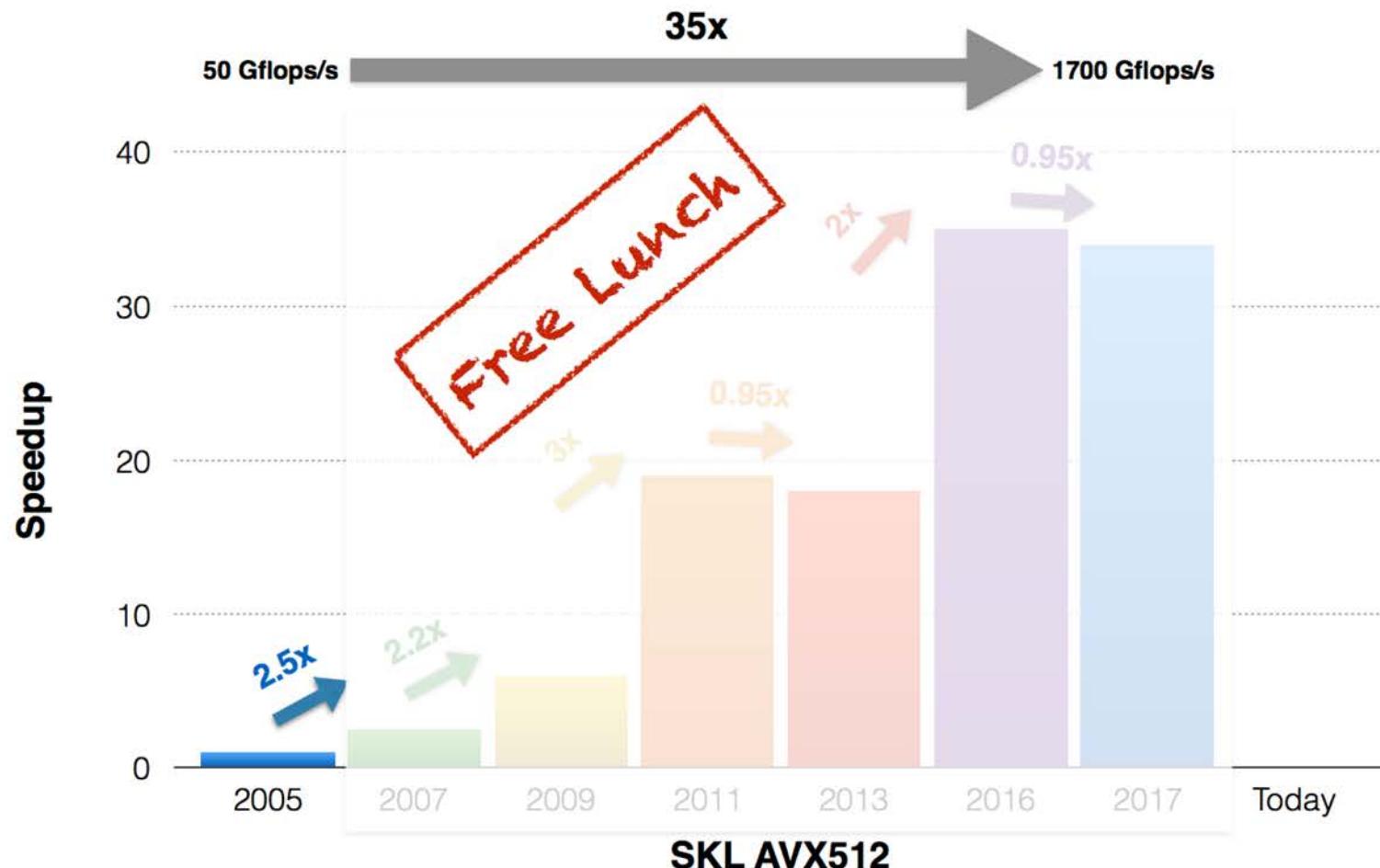
Performance evolution of dense Cholesky



Performance evolution of dense Cholesky

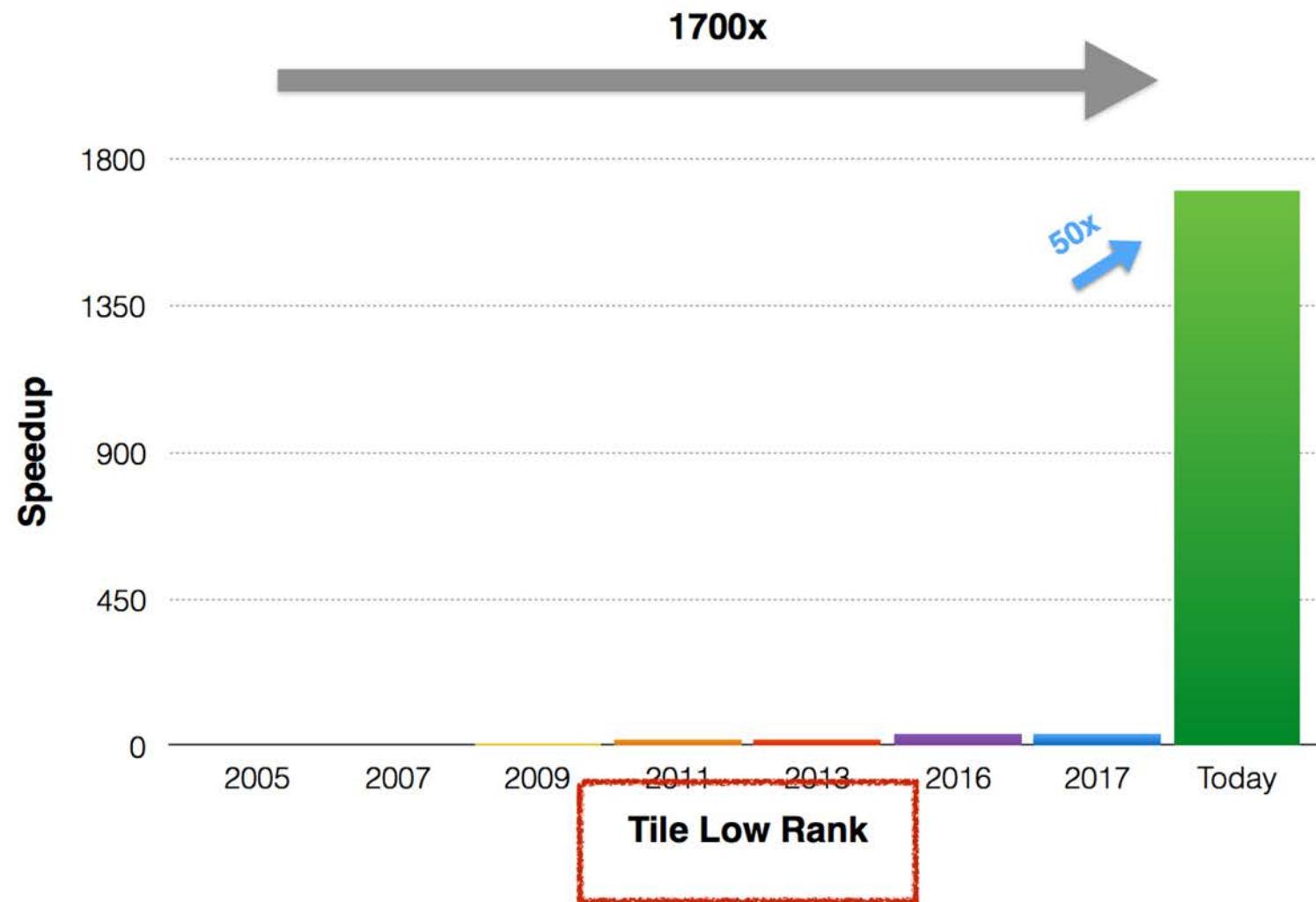


Performance evolution of dense Cholesky

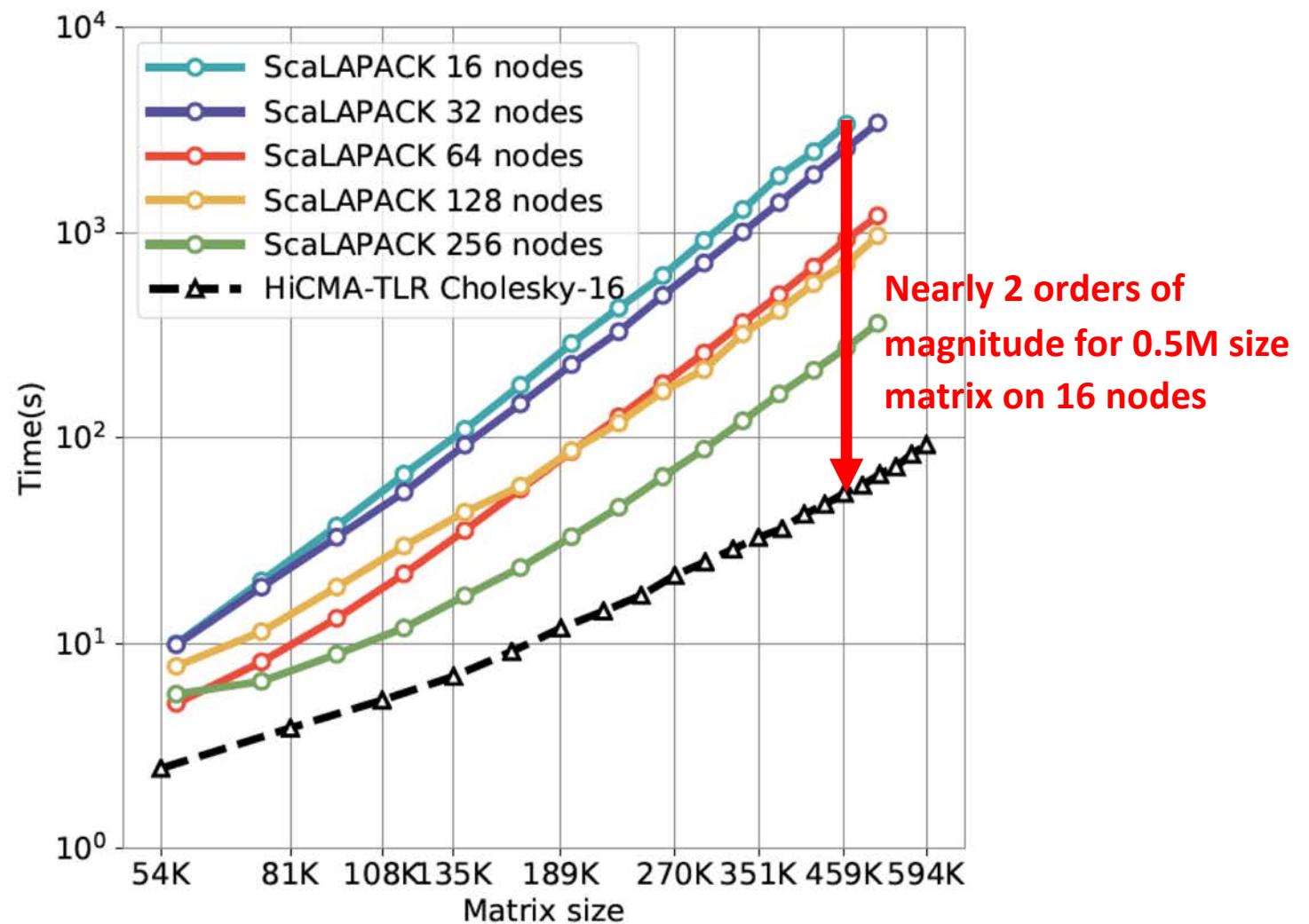


(first factor based on tiling;
successive factors, 2007-2017, based on Top500 hardware generations)

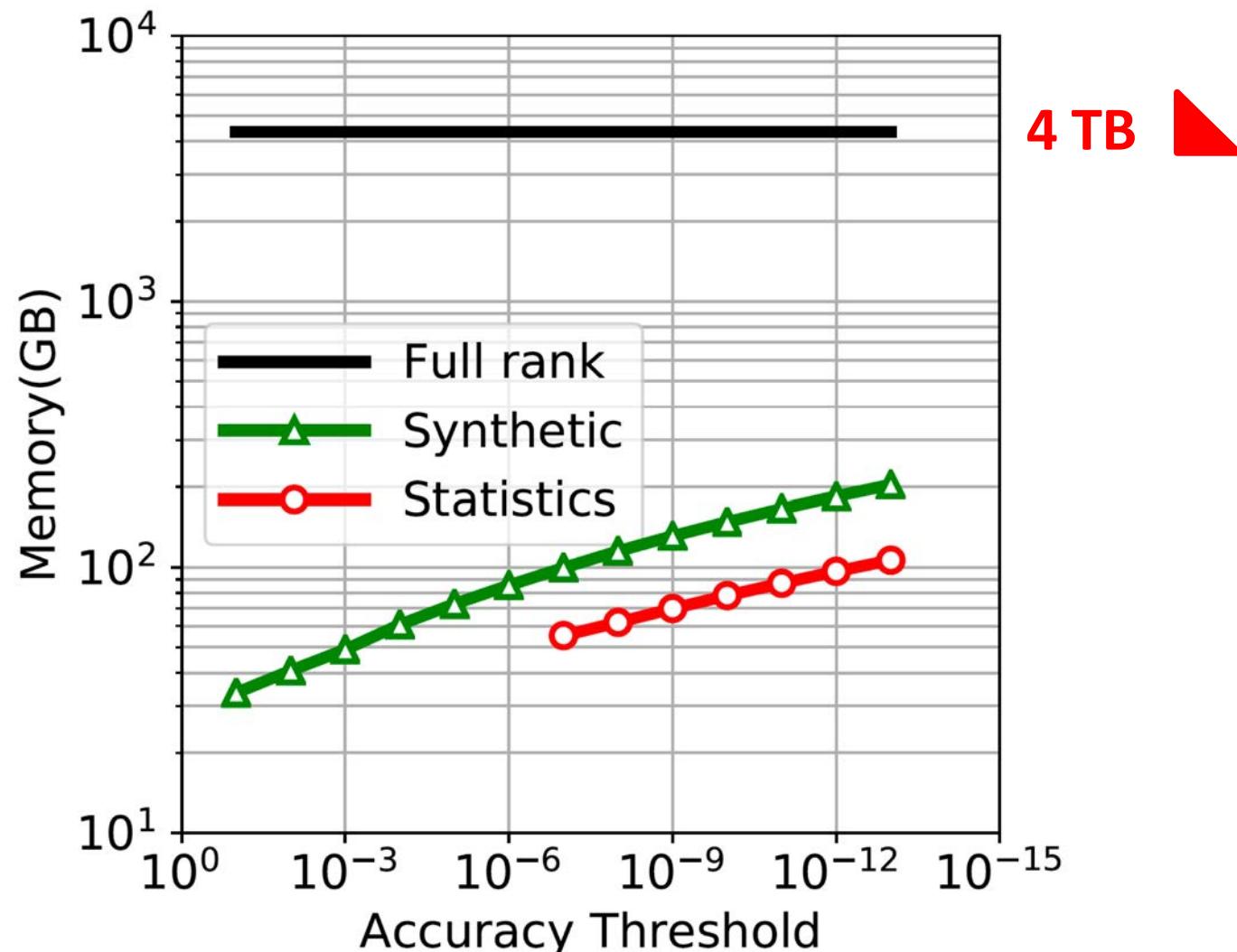
Performance evolution of dense Cholesky



HiCMA vs. ScaLAPACK on distributed memory



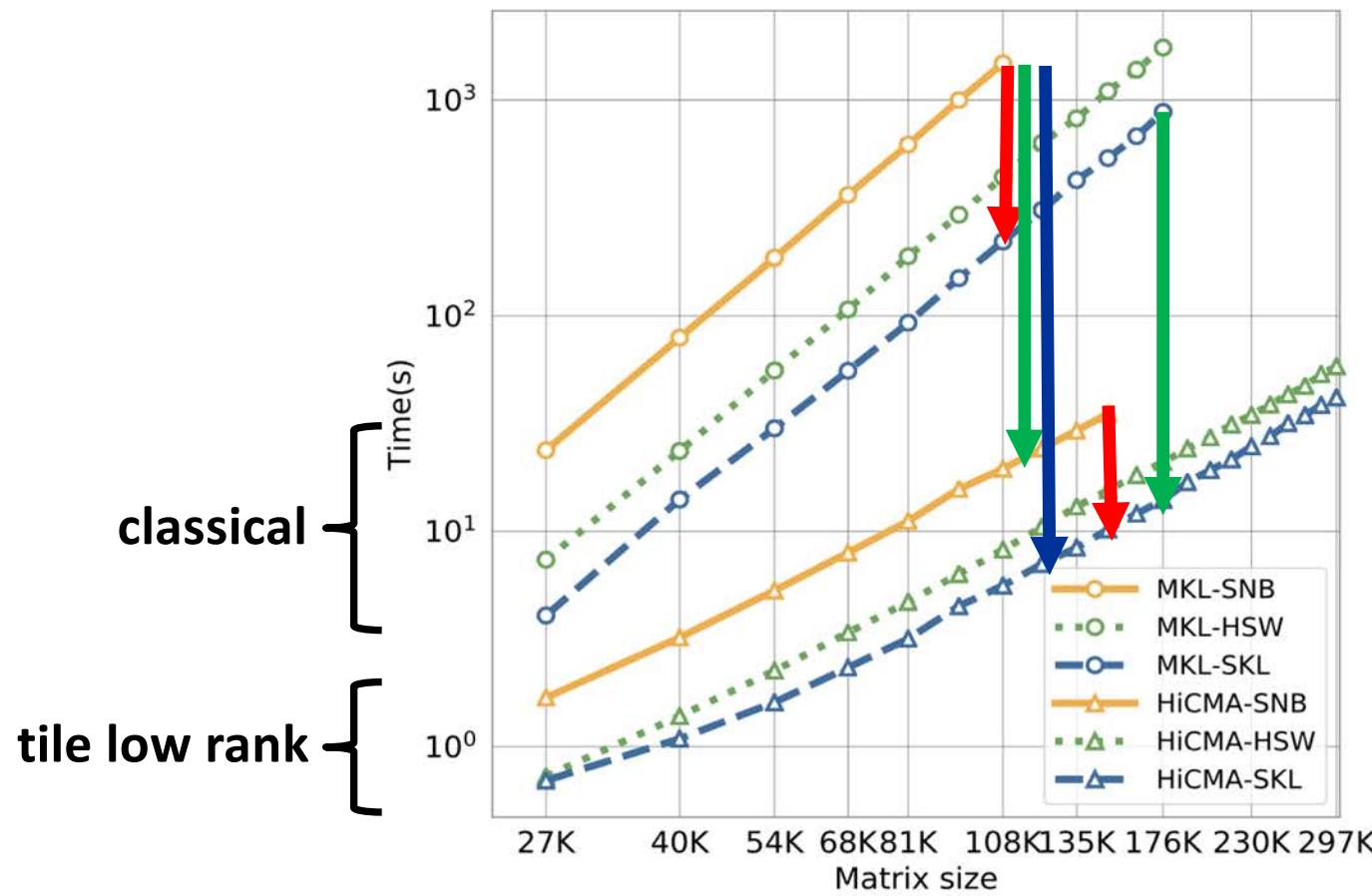
Memory footprint for DP matrix of size 1M



K. Akbudak, H. Ltaief, A. Mikhalev, A. Charara, and D. E. Keyes, Exploiting Data Sparsity for Large-Scale Matrix Computations, Accepted at EuroPar, 2018.

HiCMA vs. Intel MKL on shared-memory

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



Red arrows:
speedups from
hardware,
same algorithm

Green arrows:
speedups from
algorithm,
same hardware

Blue arrow:
From both

So far, just Tile Low Rank...

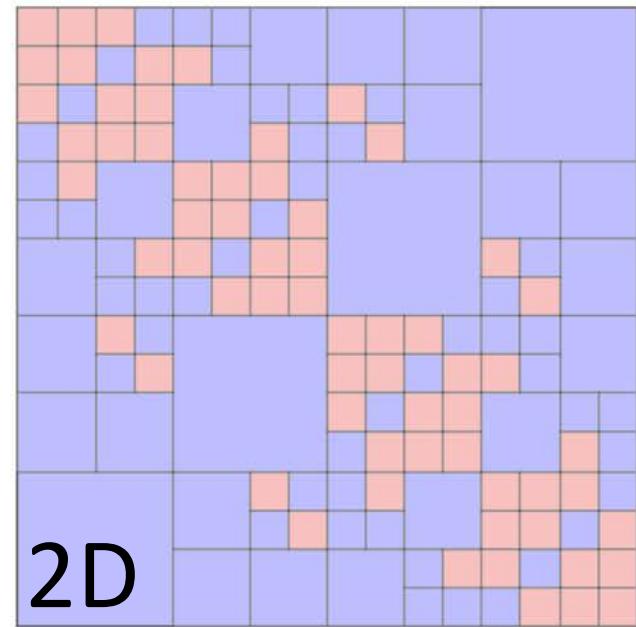
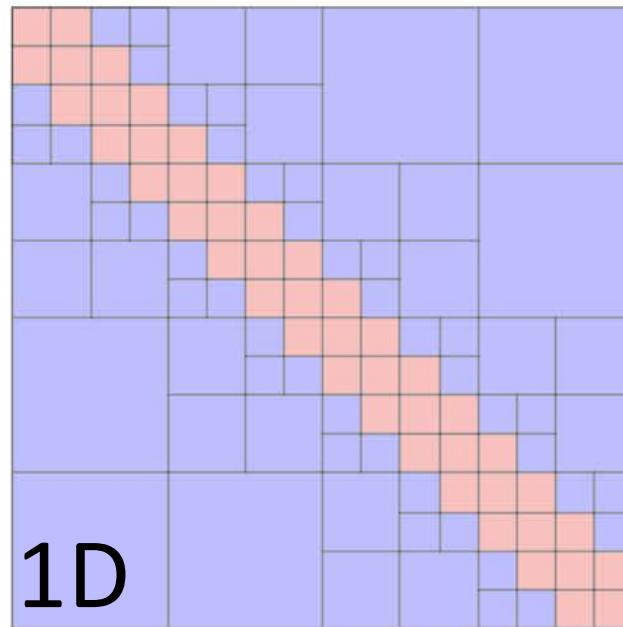


Now: general \mathcal{H} -matrices

- First advance: adopt \mathcal{H}^2 matrix structure
- Second advance: use randomized SVD (Halko, Martinsson & Tropp, 2009) to form the low-rank blocks at the leaves
 - an easy, flop-intensive GEMM-based flat algorithm
- Third advance: implement using “batches” on GPUs/multi-GPUs

\mathcal{H}^2 hierarchical matrix representation

- ▶ general blocking structure



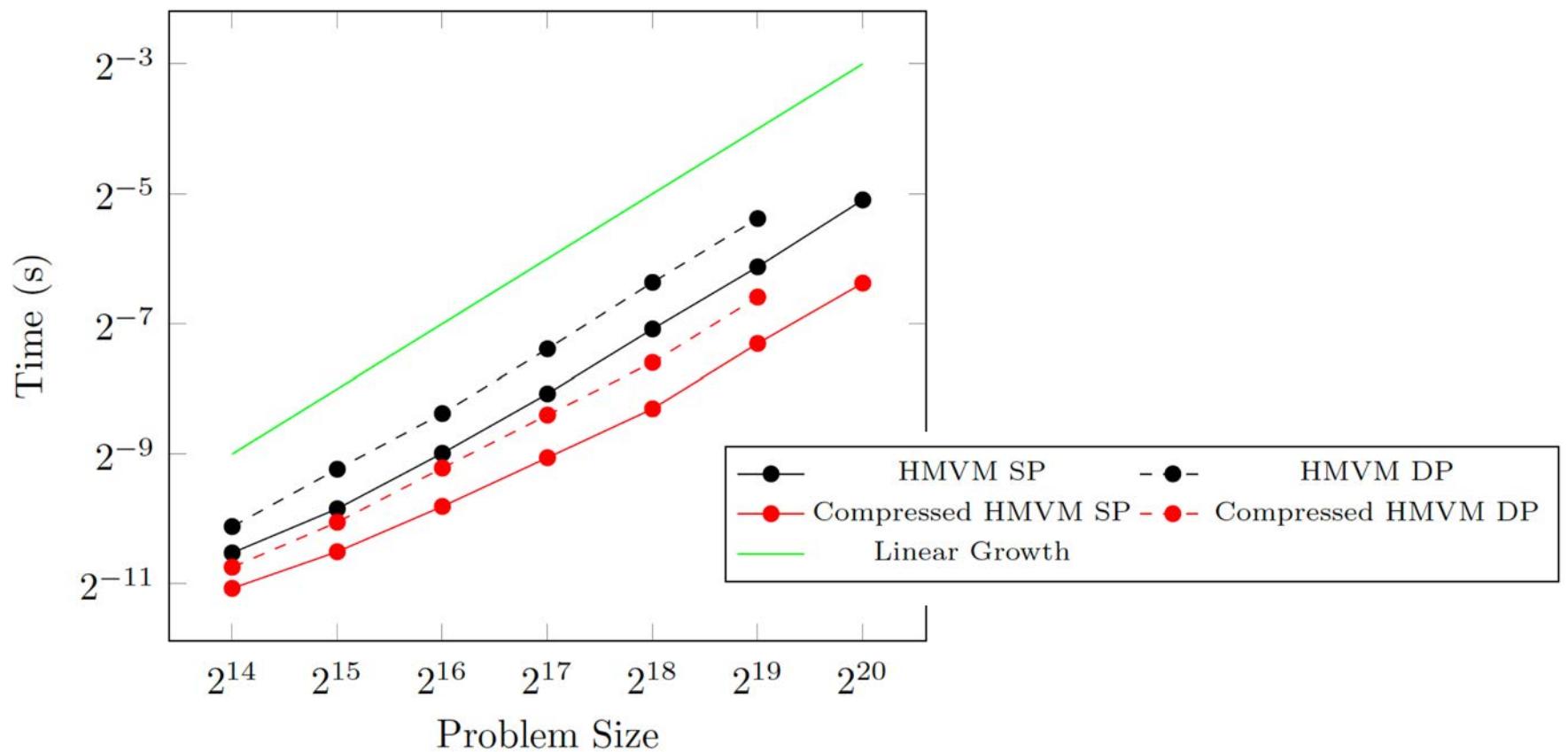
- ▶ nested bases

- $A_{ij}^l = U_i^l S_{ij}^l V_j^{lT}$

- $U_i^{l-1} = \begin{bmatrix} U_{i_1}^l & \\ & U_{i_2}^l \end{bmatrix} \begin{bmatrix} E_{i_1}^l \\ E_{i_2}^l \end{bmatrix}$

- reduces memory footprint to $O(n)$ from $O(n \log n)$

HGEMV performance

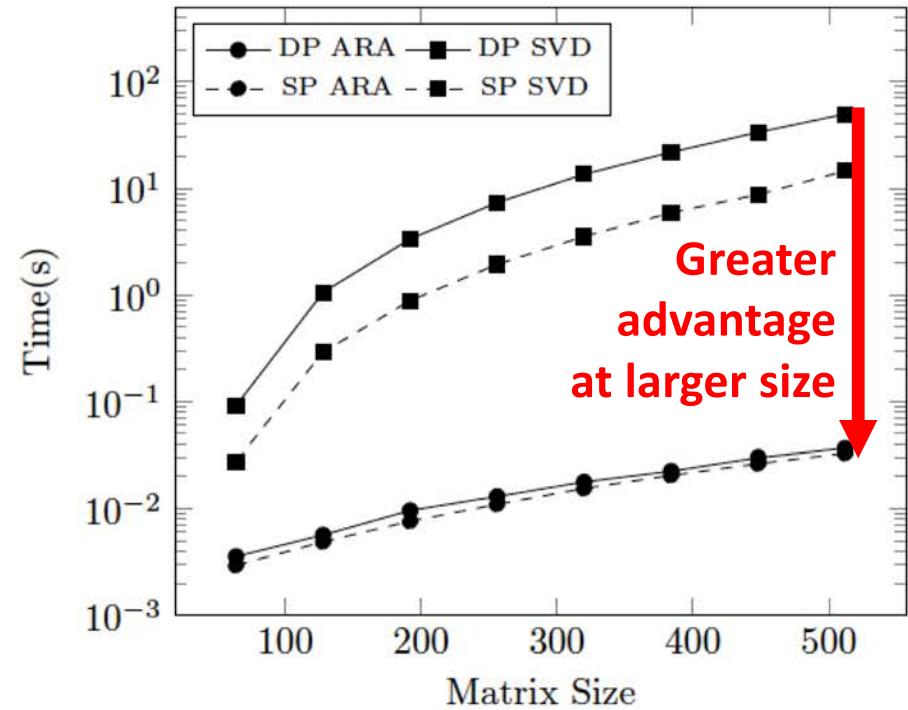
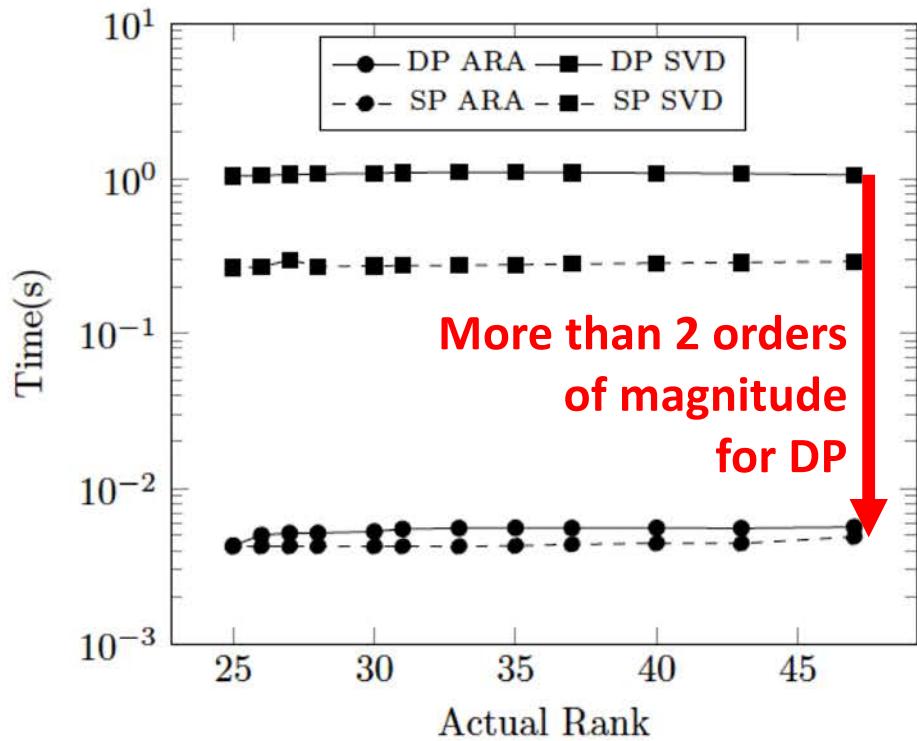


- ▶ 3D covariance matrices from spatial statistics
- ▶ running on P100 GPU
- ▶ accuracy 10^{-3} computed as $\|Ax - A^Hx\|/\|Ax\|$
- ▶ leaf size $m = 64$

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

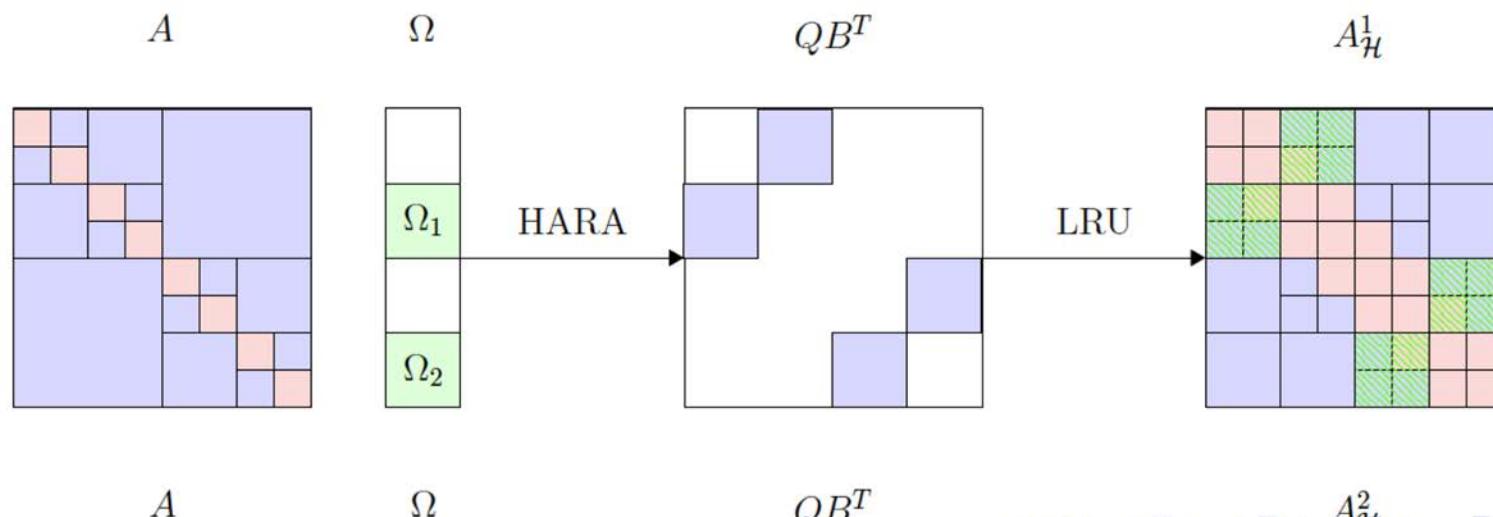
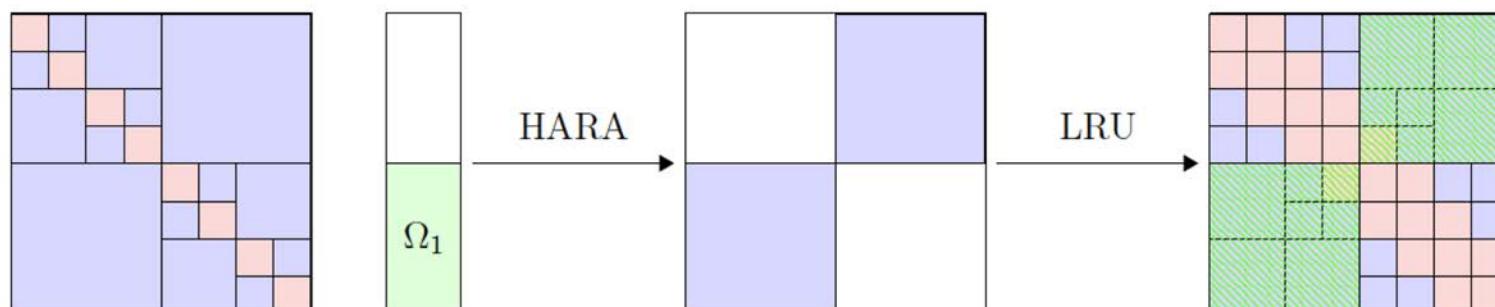
Comparison with Jacobi (Givens) SVD



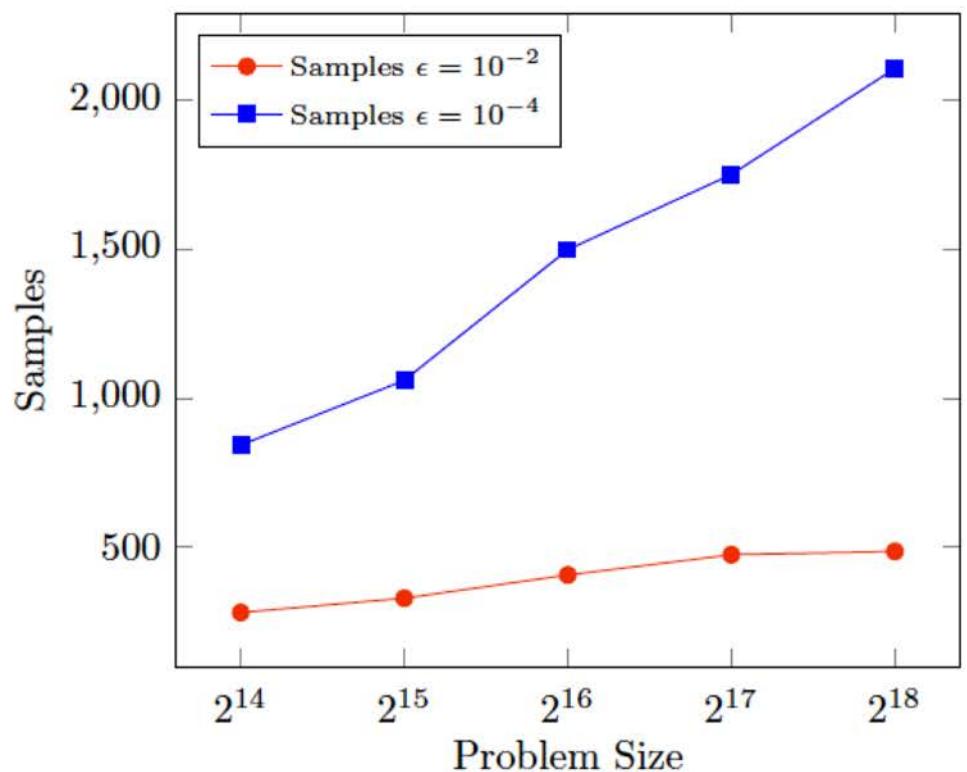
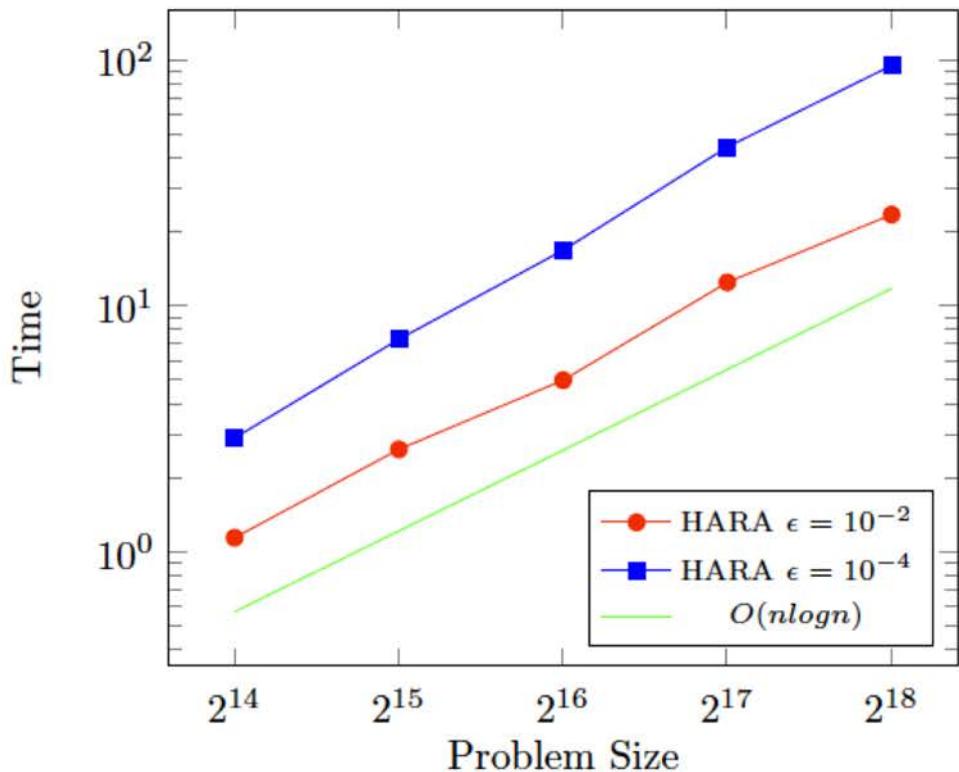
- ▶ batch of 1000 matrices in single and double precision
- ▶ varying rank for fixed size (128)
- ▶ varying size for fixed rank (47)

Hierarchical Adaptive Randomized Approximation (HARA)

- ▶ extends the basic idea to hierarchical matrices
- ▶ samples blocks of the matrix and accumulates the local low rank updates into an \mathcal{H} -matrix that is recompressed



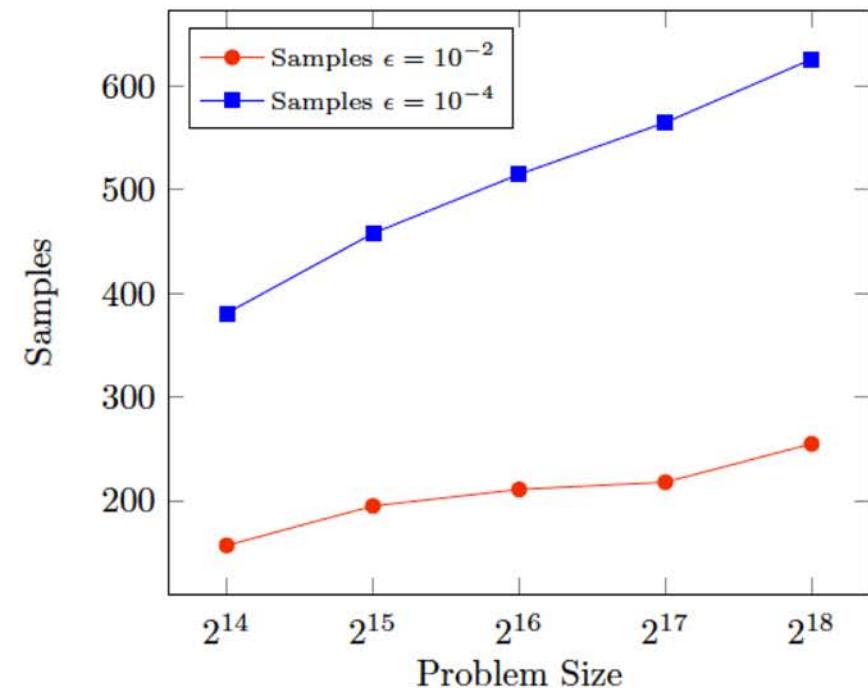
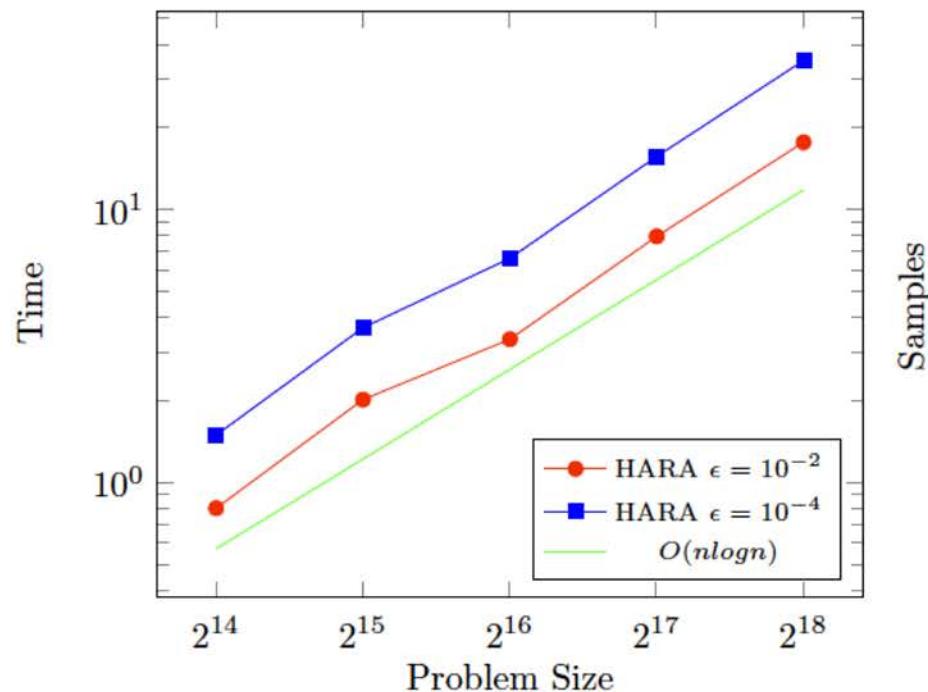
Performance of HARA on GPU



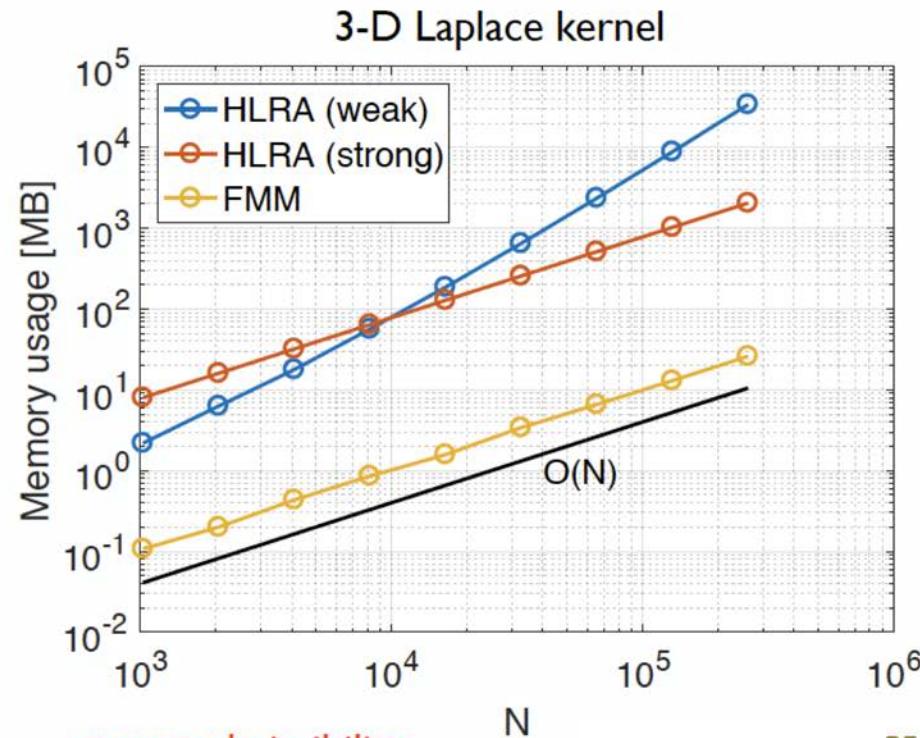
- spatial covariance matrix reconstructed from HGEMV products

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

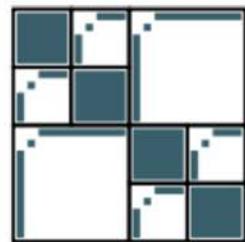
- ▶ can be cast as the problem of constructing an \mathcal{H} -matrix from matvec operations
- ▶ we can do HGEMV operations efficiently on GPUs
 - HGEMV on multiple vectors is even more efficient
- ▶ HARA construction of product also performed efficiently on the GPU



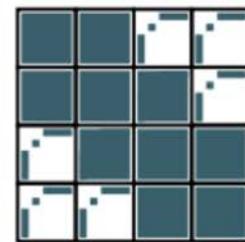
Memory complexity of FMM vs \mathcal{H} (HLRA)



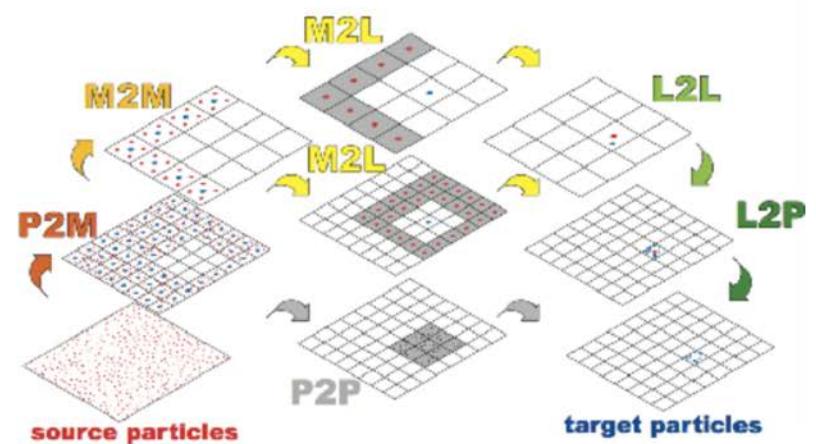
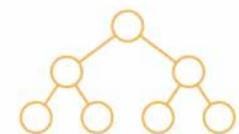
weak admissibility



strong admissibility



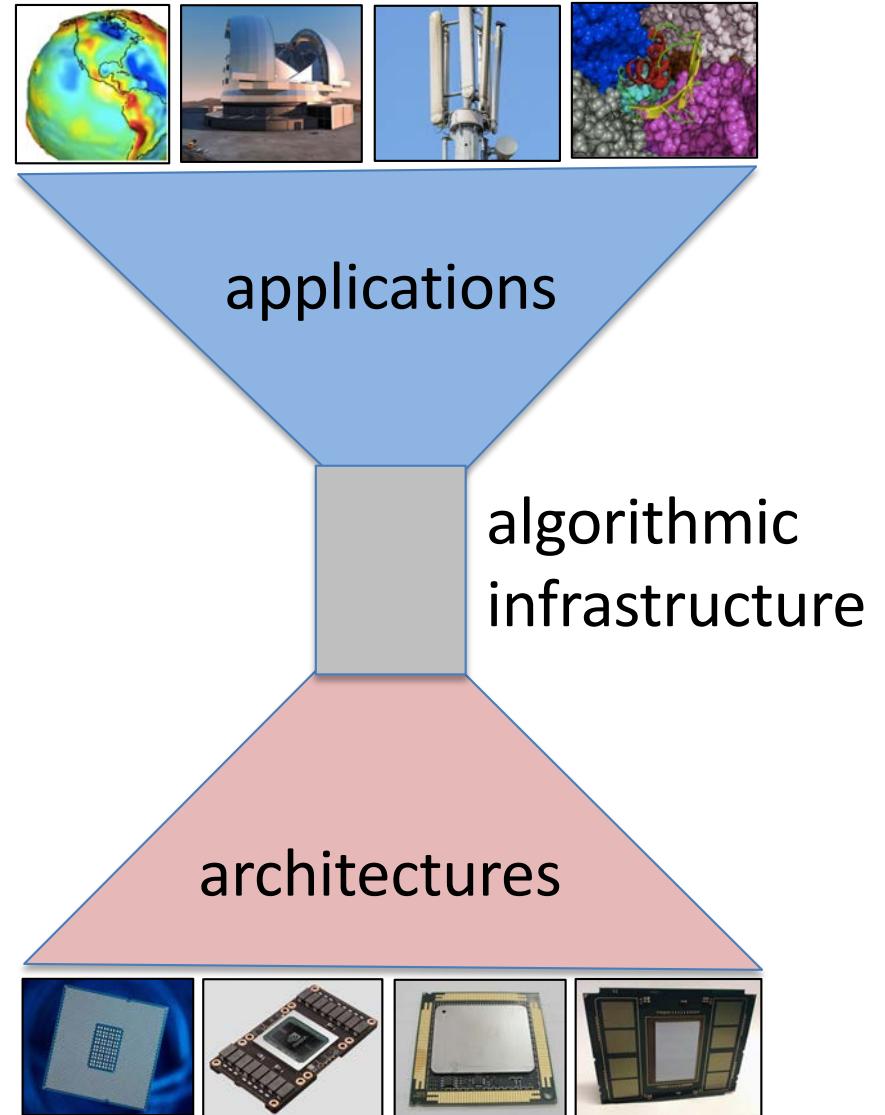
N



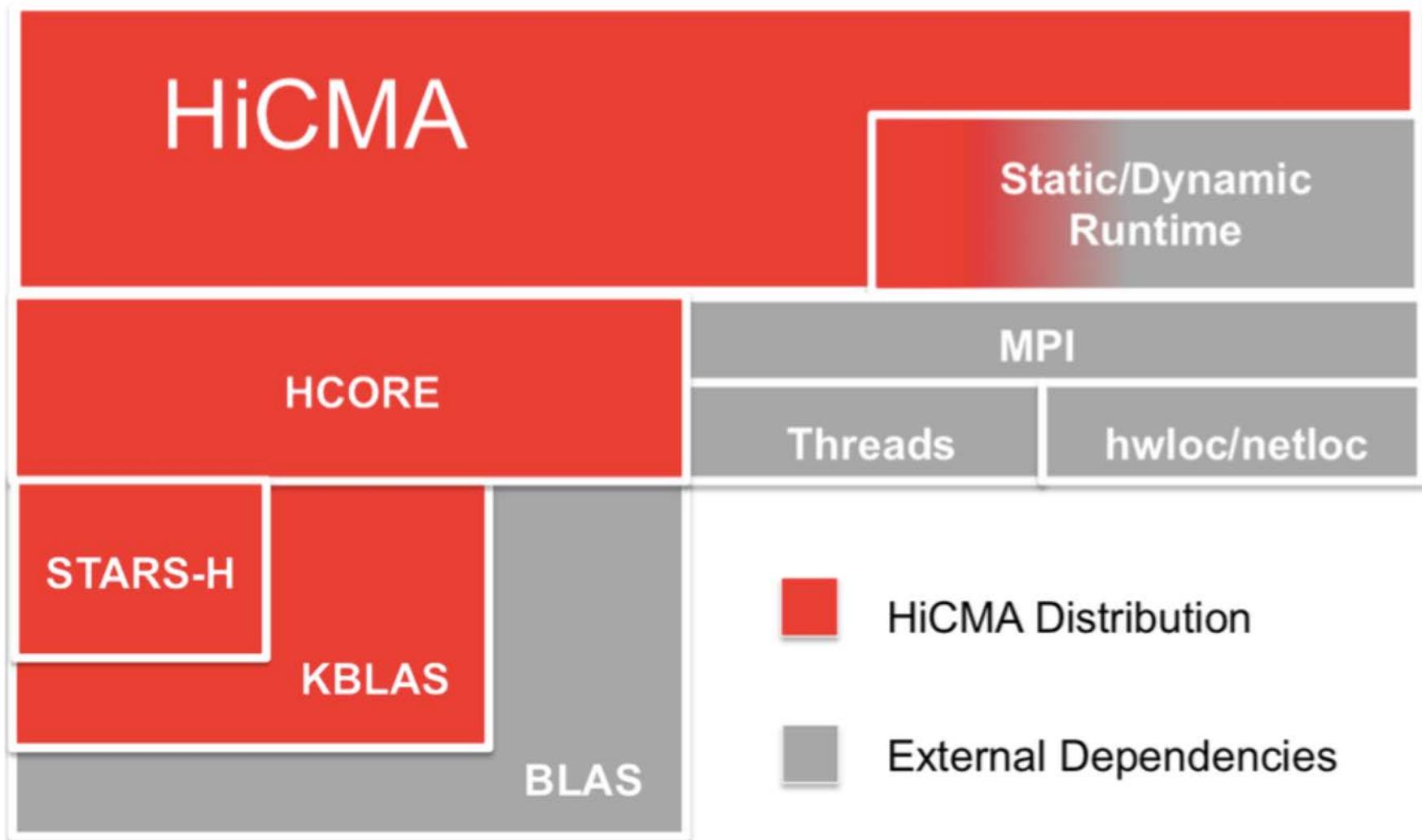
Conclusions

- Plenty of ideas exist to adapt or substitute for favorite solvers with methods that have:
 - reduced synchrony (in frequency and/or span)
 - higher residence on the memory hierarchy
 - greater SIMD-style shared-memory concurrency
 - Programming models and runtimes may have to be stretched to accommodate
 - Everything should be on the table for trades, beyond disciplinary thresholds → “co-design”
-

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



Hierarchical Computations on Manycore Architectures: HiCMA*



* appearing incrementally at <https://github.com/ecrc>

Acknowledgments

This talk was made possible through baseline KAUST support for ECRC research scientists and our vendor-sponsored efforts on hosting FMM and \mathcal{H} -matrix methods on MIC and GPU architectures



Thank you!

شكرا

david.keyes@kaust.edu.sa