



THE ROYAL SOCIETY

# Hierarchical Algorithms on Hierarchical Architectures

Hatem Ltaief, George Turkiyyah & David Keyes  
Extreme Computing Research Center (ECRC)  
King Abdullah University of Science and Technology

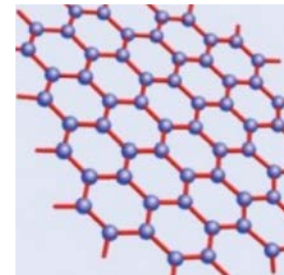
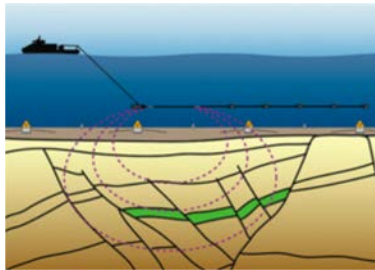
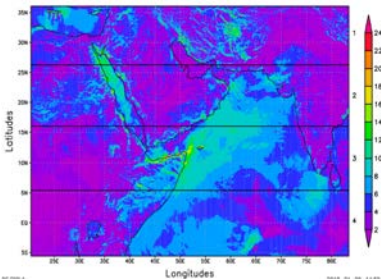


## To take away (1)

- **To better exploit emerging architectures, we need new implementations of linear, least squares, eigenvalue, and singular value solvers that**
  - **offer tunable accuracy-time tradeoffs**
  - **exploit hierarchy of precisions**
  - **may require more flops but offer more concurrency (and thus complete faster)**
- **Besides exposing more concurrency, we must**
  - **remove synchrony and over-ordering**
  - **dwell as high as possible on the memory hierarchy**

## To take away (2)

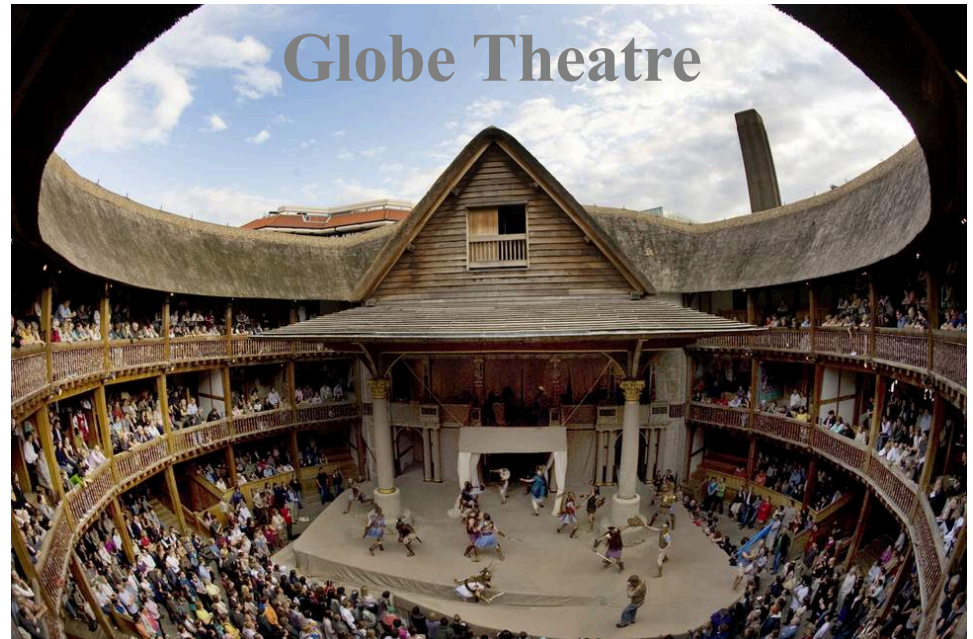
- With such new solvers, we can extend many applications that possess
  - memory capacity constraints (e.g., geospatial statistics, PDE-constrained optimization)
  - energy constraints (e.g., remote telescopes)
  - real-time constraints (e.g., wireless commun)
  - running time constraints (e.g., chem, 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 in applications
  - but not all in 30 minutes 😊
- Hope it highlights the relevance of this workshop

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



https://github.com/ecrc/

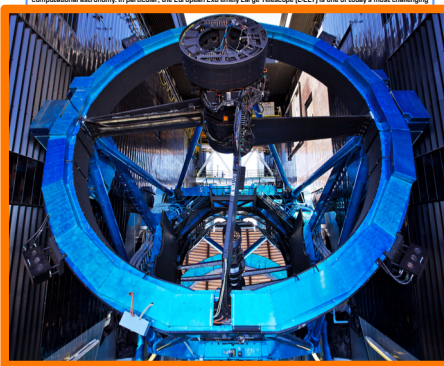
in NVIDIA cuBLAS

in Cray LibSci

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

# MOAO

The Multi-Object Adaptive Optics (MOAO) framework provides a comprehensive toolbox for high performance computational astronomy. In particular, the Extreme Geometry Large Telescope (EGLT) is one of today's most challenging



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

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

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

# ExaGeoStat

The ExaGeoStat project (ExaGeoStat) is a parallel high performance unified framework for computational geostatistics on many-core systems. The project aims at optimizing the distributed function for a given spatial data to provide an

Framework proposes a unified simulation code structure for many-core architectures. From commodity x86 to GPU accelerator-based shared and distributed memory architectures, ExaGeoStat provides practitioners to tackle computationally challenging scientific problems through state-of-the-art high-performance linear algebra.

ExaGeoStat  
Developer: [ecrc@ecrc.fr](mailto:ecrc@ecrc.fr)  
distributor: <http://github.com/ecrc/exageostat>  
Matrix: <http://github.com/ecrc/exageostat>

Current Research

- Large-scale problem for real-time processing on brown locations
- In situ processing

Performance Results (MLE)

Download the library at <http://github.com/ecrc/exageostat>

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

KAUST BASIC LINEAR ALGEBRA ROUTINES ON GPUS

# KBLAS

KAUST BLAS (KBLAS) is a high performance CUDA library implementing a subset of BLAS as well as Linear Algebra Primitives (LAPRIM) routines on NVIDIA GPUs. Using recursive and batch algorithms, KBLAS maximizes the GPU bandwidth, reuses locally cached data and increases device occupancy. KBLAS represents, therefore, a comprehensive and efficient framework versatile to various workload sizes. Located at the bottom of the usual 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

KBLAS 2.0

- Legacy Level 2 BLAS (1+0) SYMV, GEMV, HEMV
- Legacy Level 3 BLAS (1+0) TRSM, TRMM, GEMM (batch)
- Batch Level 3 BLAS (1+0) TRSM, TRMM, SYRK
- Batch Triangular (1+0) TRTR, LALLU, LPUV
- Batch Symmetric (1+0) POTRF, POTRI, POSV, PCTRI, PCTI
- Batch General (1+0) SVD, QR

Batch Precision Legacy and Batch BLAS

- Batch precision: s, e, x, x2
- Batch GPU support: Multi-GPU support, Low Rank Approximation (LRA) on GPUs, Arbitrary sizes

KBLAS HIGHLIGHTS

- KBLAS Level 1 (0) SYMV & GEMV
- KBLAS Level 1 (0) TRSM & TRMM

PERFORMANCE RESULTS

Download the library at <http://github.com/ecrc/kblas>

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

A GEMM-based SVD Software Framework on Distributed Memory Manycore Systems

# KSVD

The KAUST SVD (KSVD) is a high performance software framework for computing a dense SVD on distributed-memory manycore systems. The KSVD solver relies on the polar decomposition using the QR Dynamically-Weighted Hairy algorithm (GDWH), introduced by Nakajima and Higham (SIAM Journal on Scientific Computing, 2015). The computational challenge resides in the significant amount of extra floating-point operations required by the GDWH-based SVD algorithm, compared to the traditional on-ramp BiSVD. However, the inherent high level of concurrency associated with Level 3 BLAS-computational kernels ultimately compensates the arithmetic complexity overhead and makes KSVD a competitive SVD solver on large-scale supercomputers.

The Polar Decomposition

- A = UH V<sup>T</sup> (pencil), where U, H is orthogonal Matrix, and V is symmetric positive semi-definite matrix
- GDWH Algorithm
- Based on conventional computational kernels, i.e. Cholesky QR factorizations (6 iterations for double precision) and GEMM
- The total flop count for GDWH depends on the condition number κ of the matrix

Advantages

- Performs extra flops but less flops
- Flashes on compact memory kernels
- Supports High Precision
- May be well to GPU architectures
- Minimizes data movement
- Weakens memory synchronization

Application to SVD

- A = UH V<sup>T</sup> → UH V<sup>T</sup> → UH V<sup>T</sup>
- "LUQV" → "LUVQ" → "LSV"

Performance Results

KSVD 1.0

- GDWH-based Polar Decomposition
- Singular Value Decomposition
- QR Dynamically-Weighted Hairy Algorithm
- Support to ELPA Symmetric and HSP
- Symmetric Eigenvalue Solver
- Batch QR with Column Reordering
- Batch LAPACK-Compliant Error Handling
- Batch LAPACK-Compliant Tuning Sides
- Batch LAPACK-Compliant Error Handling

Current Research

- Asynchronous Task-Based GDWH
- Dynamic Batched GDWH
- Hardware Acceleration
- Distributed Memory Matrices
- Asynchronous Task-Based GDWH
- GDWH-based Eigensolver
- Integration into PLASMA/MAEVA

Download the software at <http://github.com/ecrc/ksvd>

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

Intel s/w for Aramco

A HIGH PERFORMANCE STENCIL FRAMEWORK USING WAFERFRONT DIAMOND TILING

# GIRIH

The GIRIH framework implements a generalized multidimensional stencil parallelization scheme for shared-cache multiprocessors that results in a significant reduction of cache size requirements for temporally blocked stencil codes. It ensures data access patterns that allow efficient hardware prefetching and TLB utilization across a wide range of architectures. GIRIH is built on a milestone waferfront diamond tiling approach to reduce horizontal data traffic in favor of locally cached data reuse. The GIRIH library reduces cache and memory bandwidth pressure, which makes it amenable to current and future cache and bandwidth-starved architectures, while enhancing performance for many applications.

STENCIL COMPUTATIONS

- Hot spot in many scientific codes
- Appear in finite difference, element, and volume discretizations of PDEs
- E.g. 3D wave acoustic wave equation:  $\nabla^2 \phi = \rho^{-1} \nabla \cdot \mathbf{f}$

MULTI-DIMENSIONAL, INTRINSIC TILING PARALLELIZATION

7-point stencil, 25-point stencil

Thread assignment in space-time dimensions

SOFTWARE INFRASTRUCTURE

- GIRIH 1.0.0
- MPI + OpenMP
- Single and double precision
- Autotuning
- Short and long stencil ranges in space and time
- Constant, variable coefficients
- LAPACK support for profiling

CURRENT RESEARCH

- Matrix power kernels
- Single and double precision
- GPU hardware accelerators
- OpenACC / CUDA
- Quasi-Newton algorithms
- Dynamic runtime systems
- Extension to CFD applications

PERFORMANCE RESULTS

- Damen size: 512 x 512 x 512
- # of time steps: 500
- 25-point star stencil
- 20-point boundary conditions
- Two-nested systems (Meyn, LS)
- 8-core Intel Xeon (E5-2680)
- 16-core Intel Xeon (E5-2680)
- 28-core Intel Xeon (E5-2680)
- Intel compiler suite v17 with AVX512 flag enabled
- Memory affinity with mwaitx1 command
- Thread binding to cores with sched\_setaffinity

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

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

Software for Testing Accuracy, Reliability and Scalability of Hierarchical computations

# STARS-H

STARS-H is a high performance parallel open-source package of Software for Testing Accuracy, Reliability and Scalability of Hierarchical computations. It provides a hierarchical matrix format in order to benchmark performance of various libraries for hierarchical matrix compression and computations (including fast). Why hierarchical matrices? Because such matrices arise in many PDEs and use much less memory while requiring less flops for computations. There are several hierarchical data formats, each one with its own performance and memory footprint. STARS-H intends to provide a standard for assessing accuracy and performance of hierarchical matrix libraries on a given hardware architecture environment. STARS-H currently supports the two low-Rank (TLR) 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 <http://www.ecrc.fr/stars-h>

Matrix Kernel

- Electrostatic (one over distance):  $A_{ij} = \frac{1}{|r_i - r_j|}$
- Electrodynamics (one over distance):  $A_{ij} = \frac{1}{|r_i - r_j|}$
- Spatial statistics (Matern kernel):  $A_{ij} = \frac{1}{|r_i - r_j|^\beta} K_\beta(\sqrt{\lambda} |r_i - r_j|)$
- And many other kernels...

STARS-H Q1.0

- Data format: The Low-Rank (TLR) format
- Operator approximation: matrix-vector multiplication, Krnlv, CG solve
- Synthetic applications in a matrix-free form: random, TLR, matrix, Cauchy matrix
- Real applications in a matrix-free form: electrostatics, electrodynamics, spatial statistics
- Programming models: OpenMP, MPI and task-based programming
- Approximation techniques: SVD, PIVOR, Randomized SVD

Reading of STARS-H

- Extend to other problems in a matrix-free form
- Support: HCLM, HSB, XE and XE data formats
- Implement other approximation schemes (e.g. ADA)
- Port to GPU accelerators
- Apply other dynamic runtime systems and programming models (e.g. PARSEC)

Performance Results

Download the software at <http://github.com/ecrc/stars-h>

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

Abstraction Layer For Standardizing APIs of Task-Based Engines

# AL4SAN

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 description of tasks and their data dependencies from existing dynamic engines. AL4SAN supports various dynamic runtime systems relying on compiler infrastructure technology or on library-defined APIs. It features an abstraction of task-based engines and runtime, enables a single-code application to access various runtimes and their respective scheduling components. The goal of AL4SAN is not to create yet another runtime system, but to further leverage the use-observability of the underlying complex hardware architectures at the dawn of the Exascale age.

AL4SAN v1.0 Features

- Standardizing Task-based runtime systems
- Using a lightweight abstraction layer
- Improving user productivity
- Supporting offshore hardware architectures
- Performing self-related linked-outside (up to 10%)

AL4SAN Roadmap

- Extending to more engines
- Leveraging data abstraction
- Composing runtime dynamic runtime systems
- Adding support to more algorithms and applications

Runtime Support

- AL4SAN Abstraction Layer for Standardizing APIs of Task-based Engines & Memory Layout M.
- Abstraction and APIs
- Standardized API

Main References

- AL4SAN Abstraction Layer for Standardizing APIs of Task-based Engines & Memory Layout M.
- Abstraction and APIs
- Standardized API

Performance Results

Download the software at <http://github.com/ecrc/al4san>

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

HiCMA

The Hierarchical Computations on Manycore Architectures (HiCMA) library aims at redesigning existing dense linear algebra libraries to exploit the data sparsity of the matrix operator. Data sparse matrices arise in many scientific problems (e.g. in geospatial-based weather forecasting, seismic imaging, and materials science applications) and are characterized by low-rank off-diagonal data structure. Numerical low-rank approximations have demonstrated attractive theoretical bounds, both in memory footprint and arithmetic complexity. The core idea of HiCMA is to design fast linear algebra computations operating on the underlying the low-rank data format, while satisfying a specified numerical accuracy and leveraging performance from massively parallel hardware architectures.

SOFTWARE STACK

- HiCMA
- OpenBLAS
- Intel MKL
- MAEVA
- PLASMA
- BLAS
- External Dependencies
- HiCMA Dataflow
- HiCMA OpenDependencies

HiCMA Q1.0

- Matrix-Matrix Multiplication
- Solver Factorization Sides
- Preconditioners
- Hardware Accelerators
- Support for Multiple Precision
- Asynchronous Task-Based Error Handling and Feed Backs
- Support for OpenMP, PIVOR and Kokkos
- Support for HCLM, HSB and XE

Performance Results

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

A collaboration of INRIA, ICL, AUB, INMEX, NVIDIA, CRAY, IOSR

# Two universes of NLA exist side-by-side



## Flat

**\* Global indices \***

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

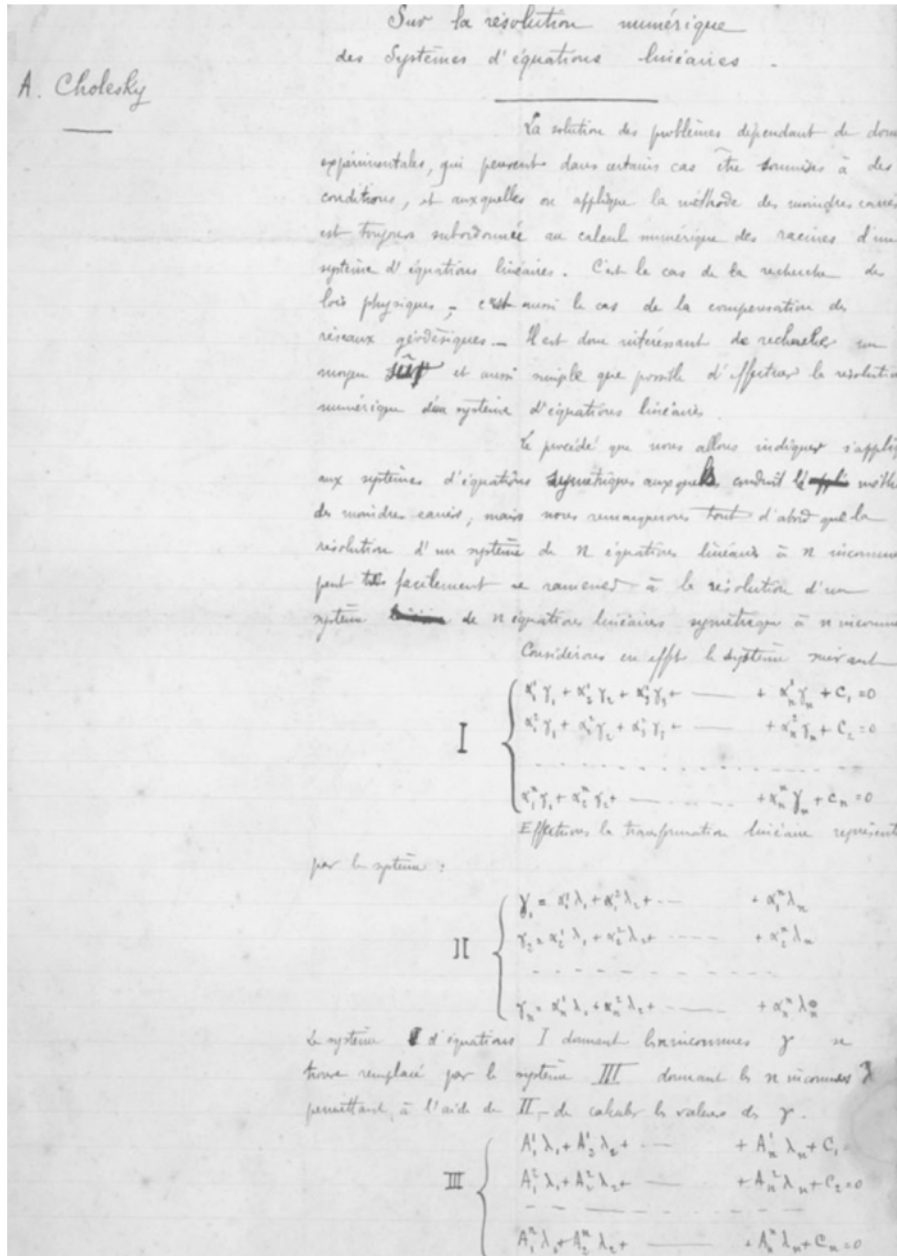
## Hierarchical

**\* 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)



```

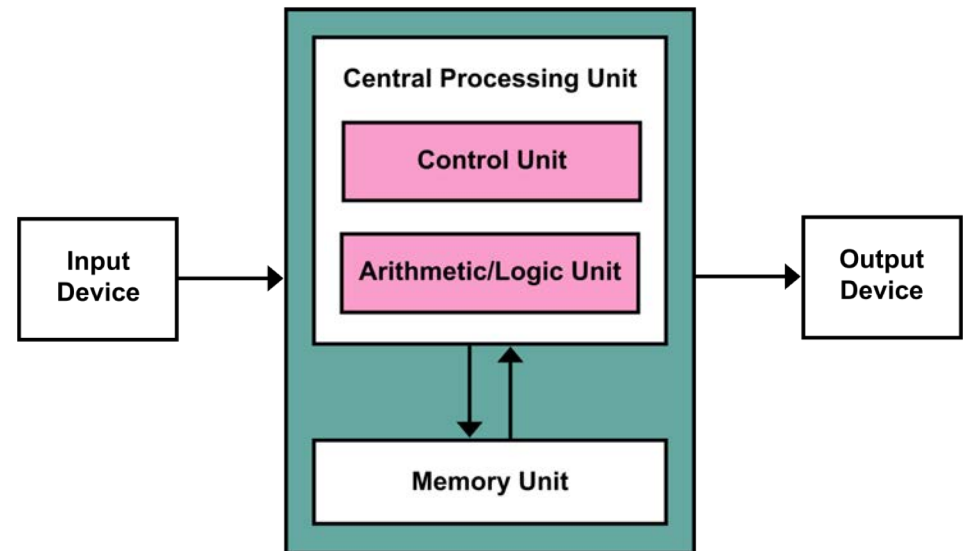
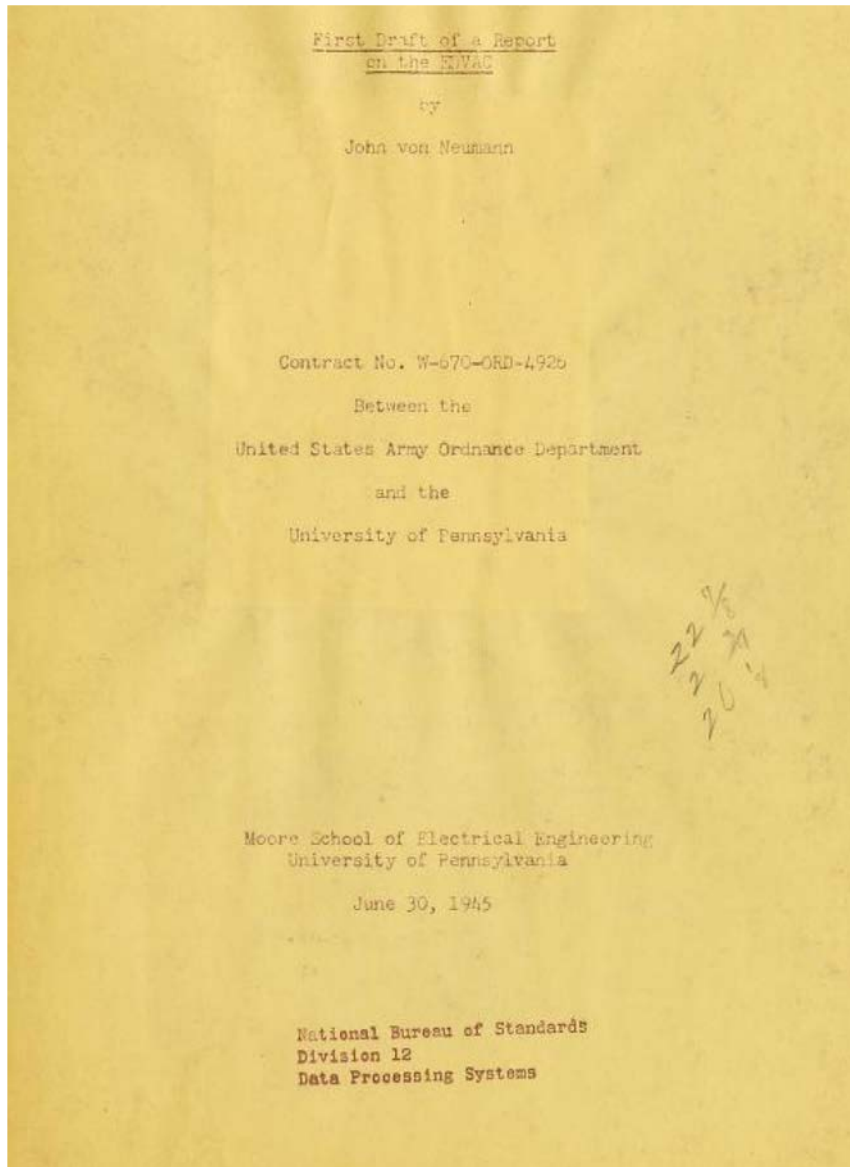
l11 = √a11;
for j = 2, ..., n do
  | lj1 = aj1/l11;
end
for i = 2, ..., n - 1 do
  | lii = (aii - ∑k=1i-1 lik2)1/2;
  for j = i + 1, ..., n do
    | lji = (aji - ∑k=1i-1 ljklik) / lii;
  end
end
lnn = (ann - ∑k=1n-1 lnk2)1/2;
end

```

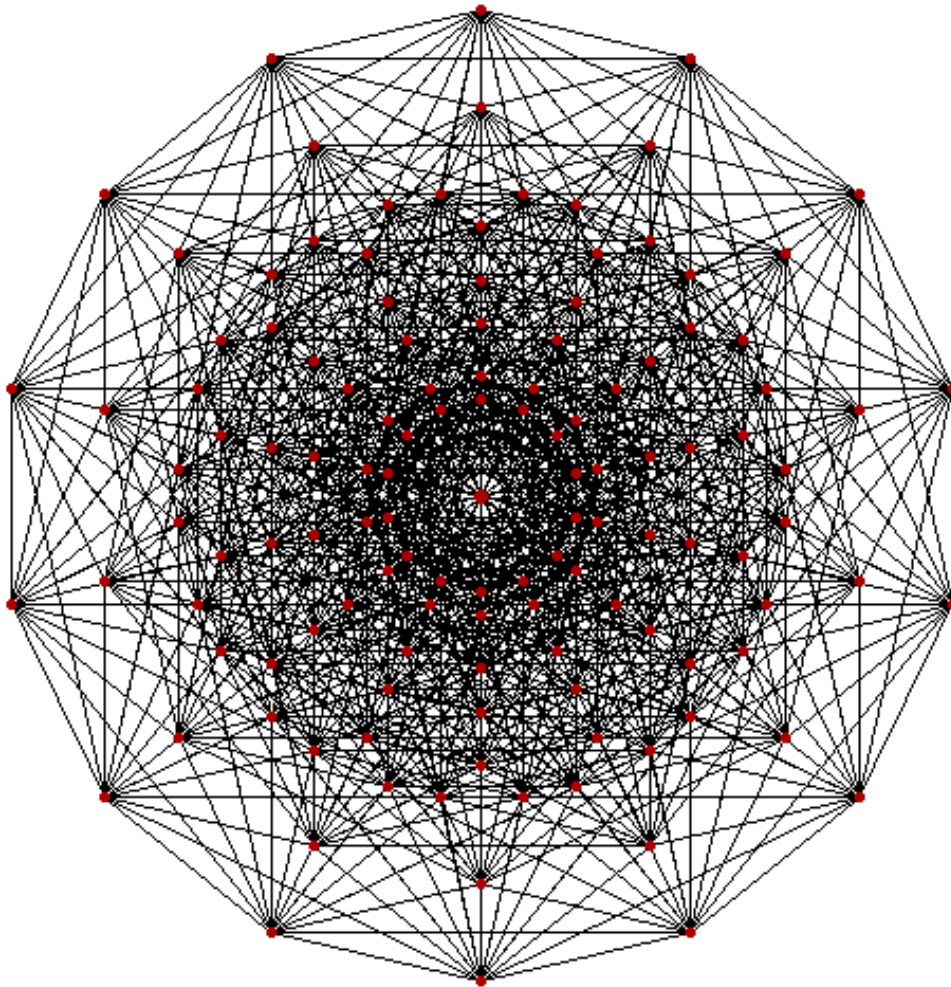
triangular recurrence



# Architectures were flat, as well (vN, 1945)



Since 1985: “horizontal” structure...

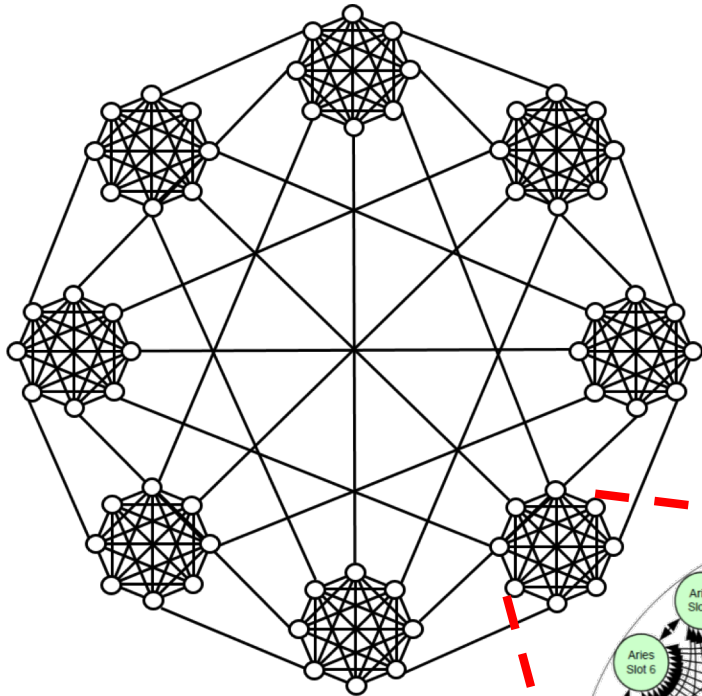


**128-node hypercube**

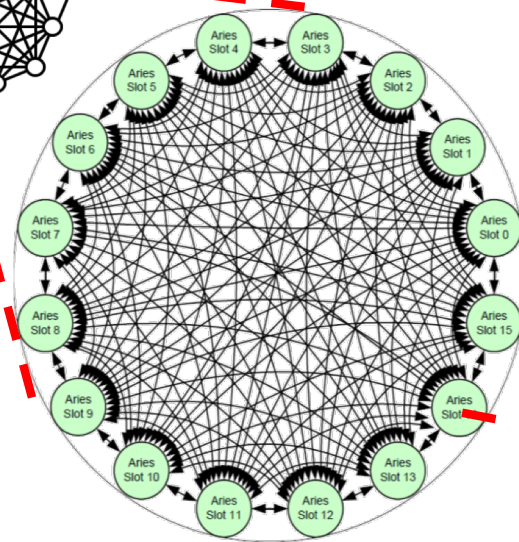


c/o Computer History Museum, Mountain View, CA

# Today's "horizontal" structure...

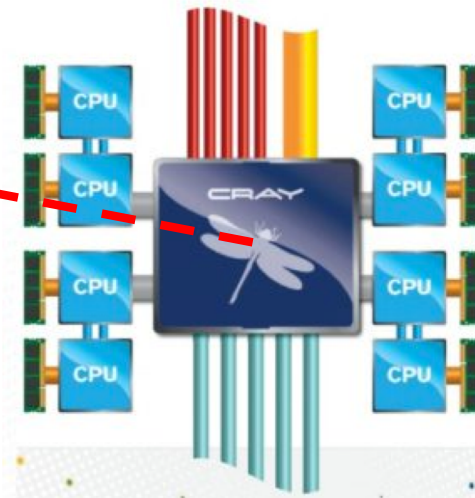


**dragonfly  
network**



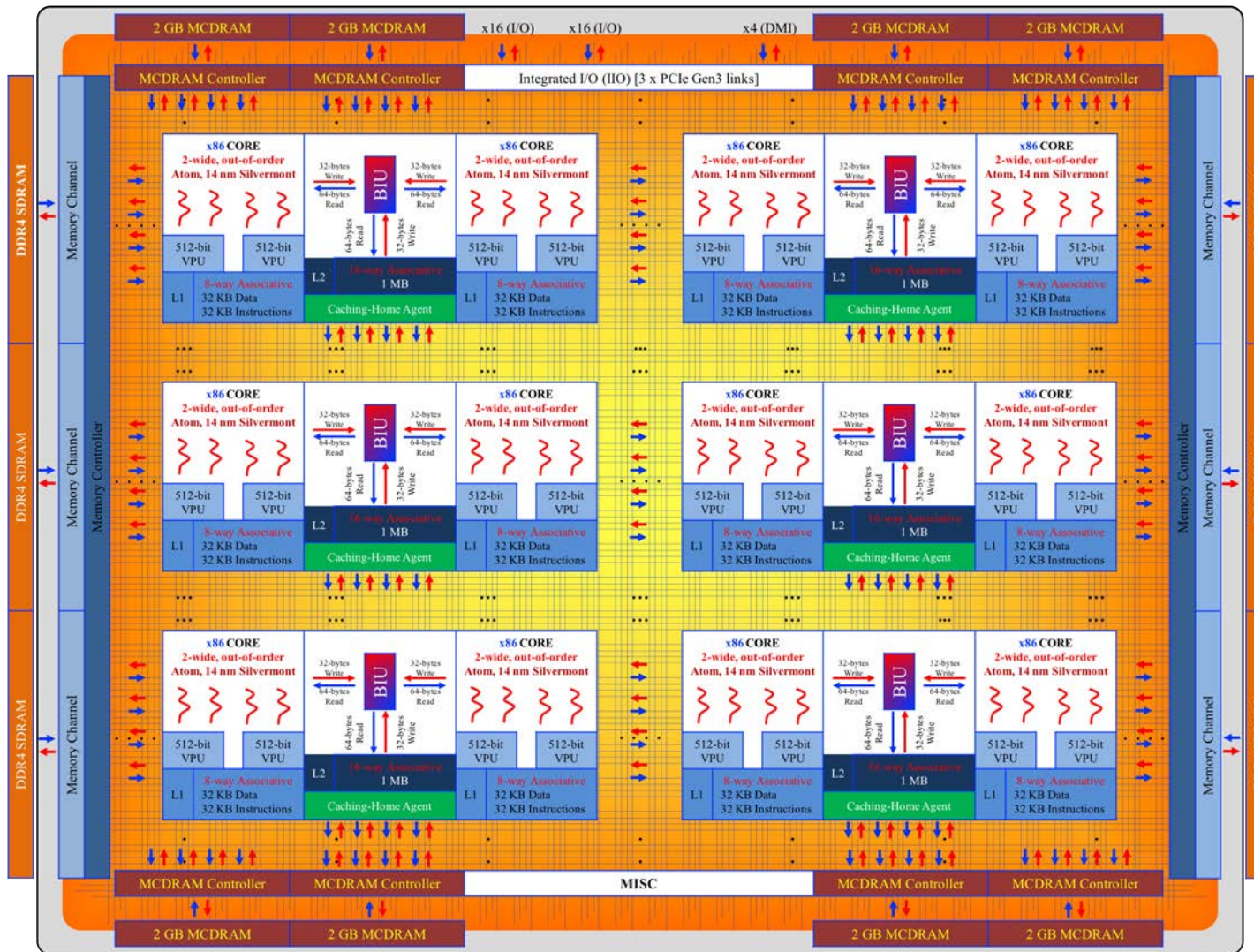
## Cray's "Aries" network

- 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

~ 1.1 TF/s, ~ 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\*

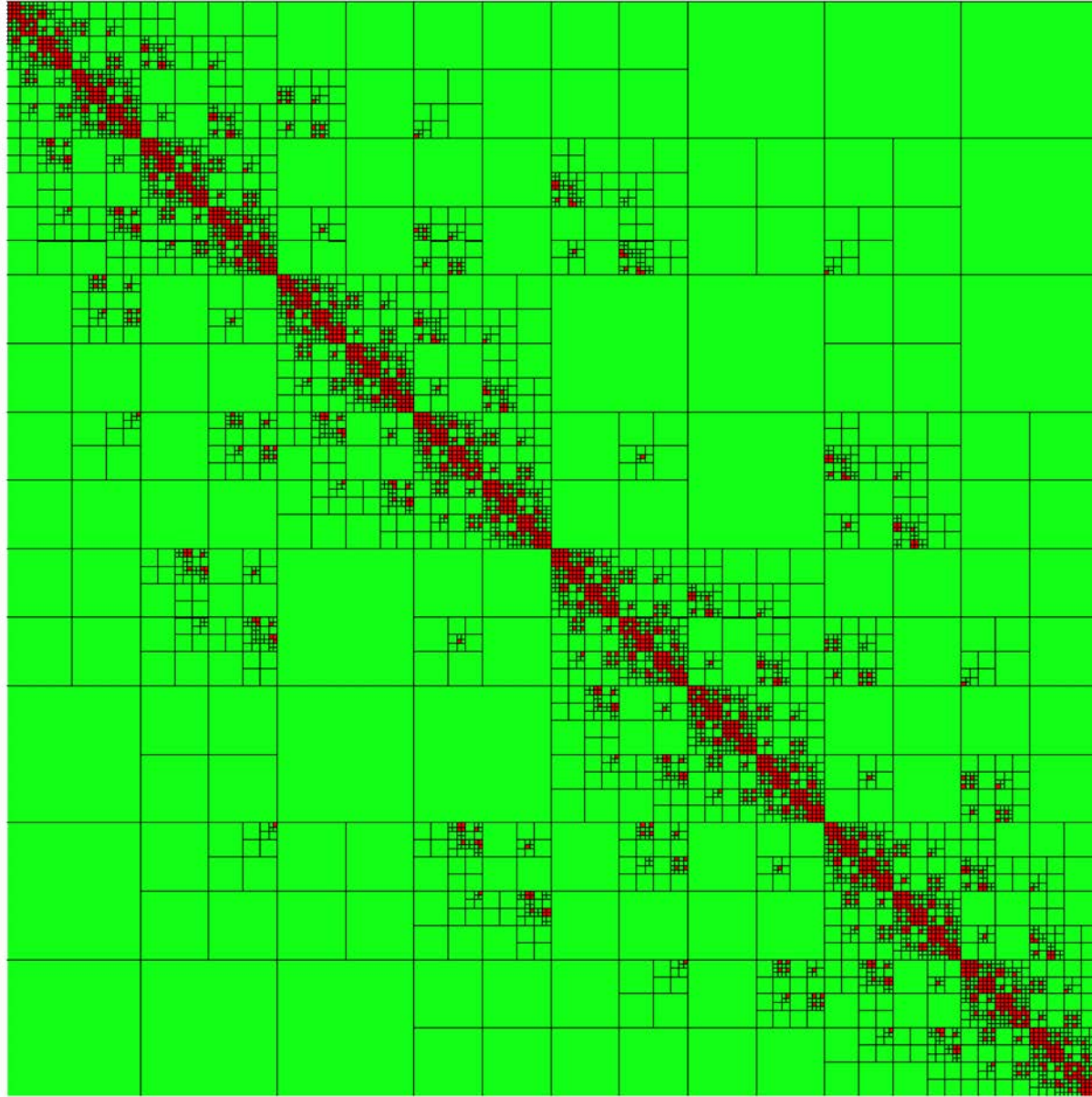
ERIN CARSON<sup>†</sup> AND NICHOLAS J. HIGHAM<sup>‡</sup>

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**
  - **stronger-than-ever 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) ~ 10 fJ/bit
- Access DRAM on chip ~ 1 pJ/bit
- Access HBM/MCDRAM (few mm) ~ 10 pJ/bit
- Access DDR3 (few cm) ~ 100 pJ/bit

**~  $10^4$  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**
  - 3) ... SIMT-style batching ... algorithm-based fault tolerance ... etc.**
-



# 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**

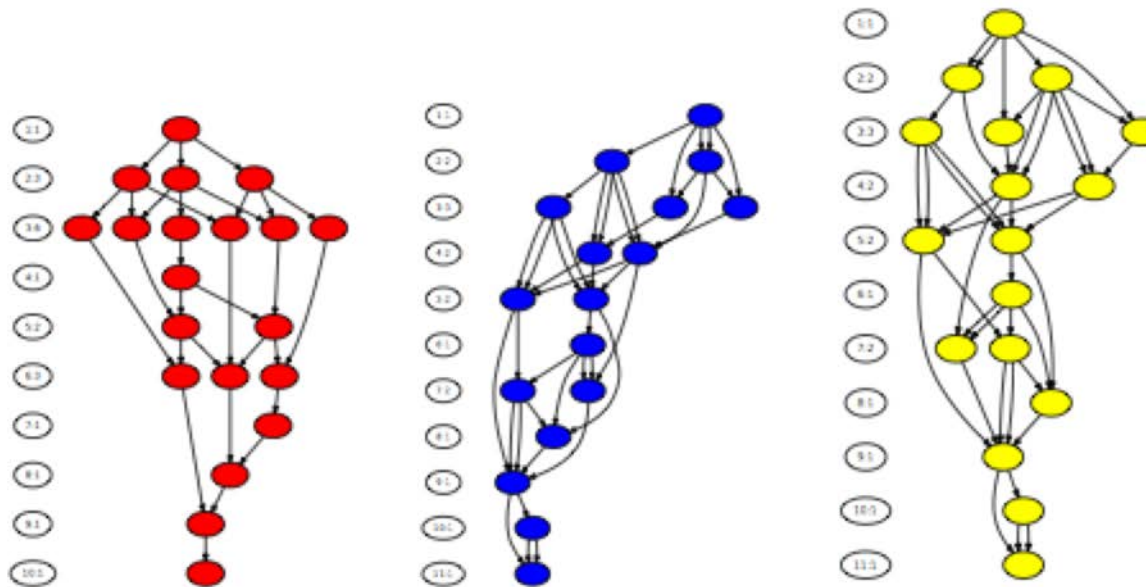
- **Disadvantages**

- ◆ **code is more complex**
  - ◆ **code is architecture-specific at the bottom**
-

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

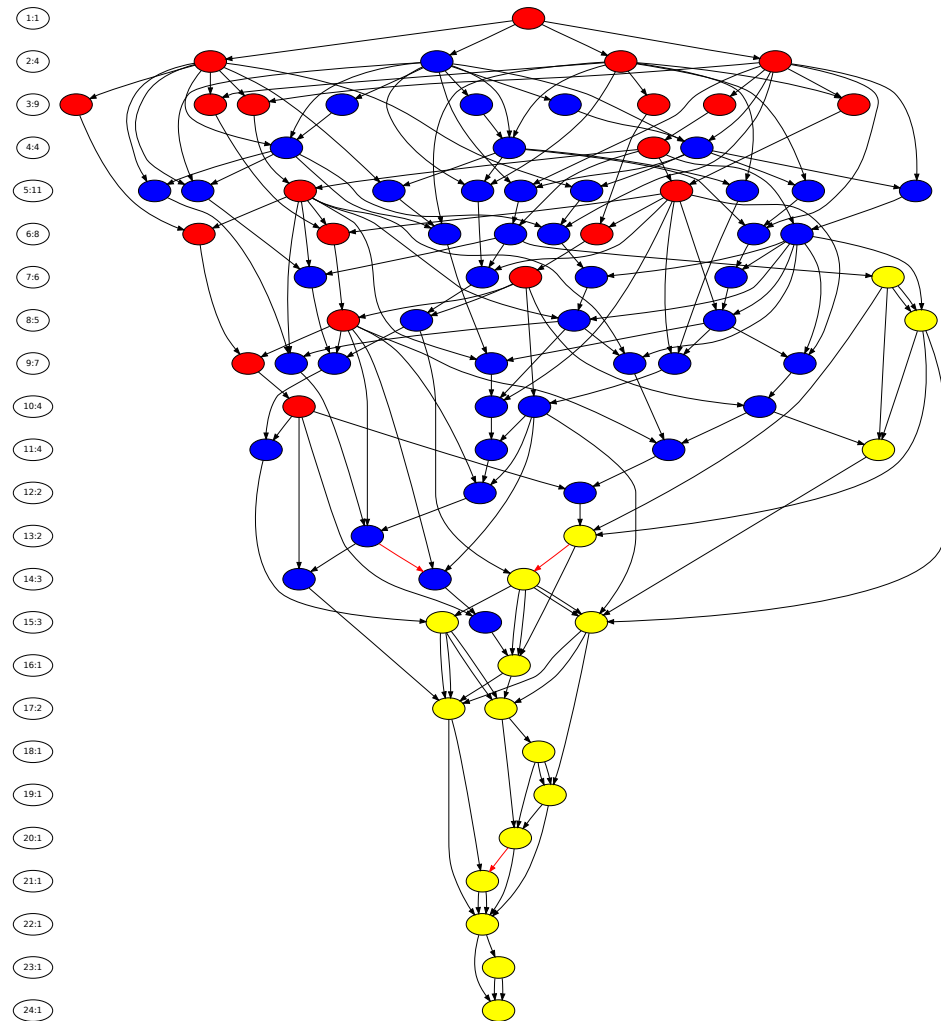
$$Ax = \lambda Bx$$

Operation	Explanation	LAPACK routine name
① $B = L \times L^T$	Cholesky factorization	POTRF
② $C = L^{-1} \times A \times L^{-T}$	application of triangular factors	SYGST or HEGST
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  $4 \times 4$  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**
-

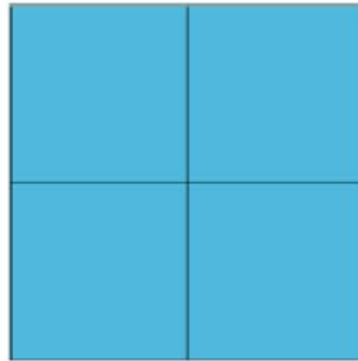
# Key tool: hierarchical matrices

- [Hackbusch, 1999] : off-diagonal blocks of typical differential and integral operators have low effective rank
- Similarly: Schur complements of the above, covariance matrices from statistics, Hessians from optimization, etc.
- 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”)

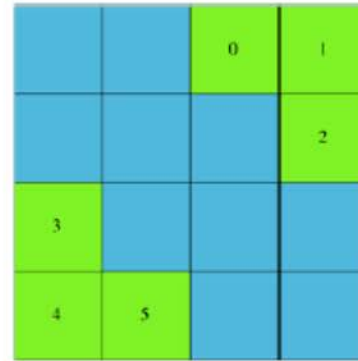
# Recursive construction of an $\mathcal{H}$ -matrix



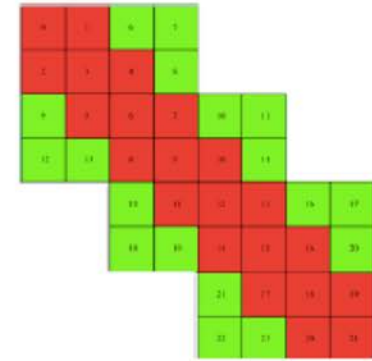
Step 0



Step 1



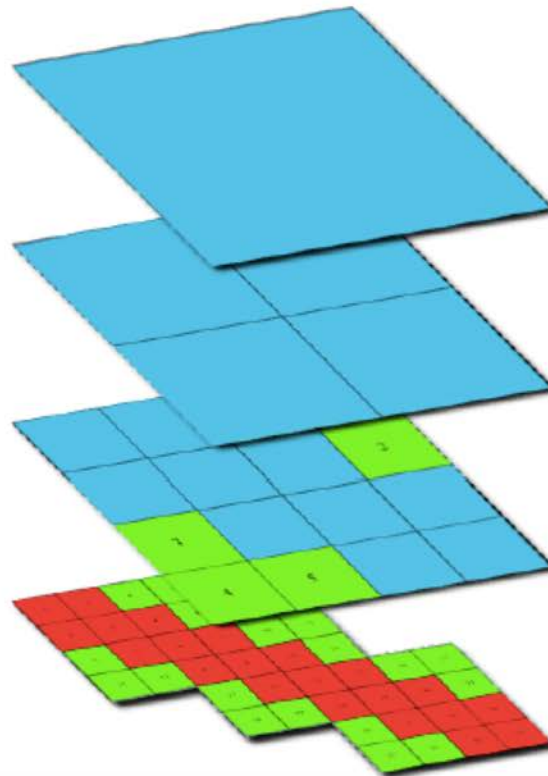
Step 2



Step 3

Specify two parameters:

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

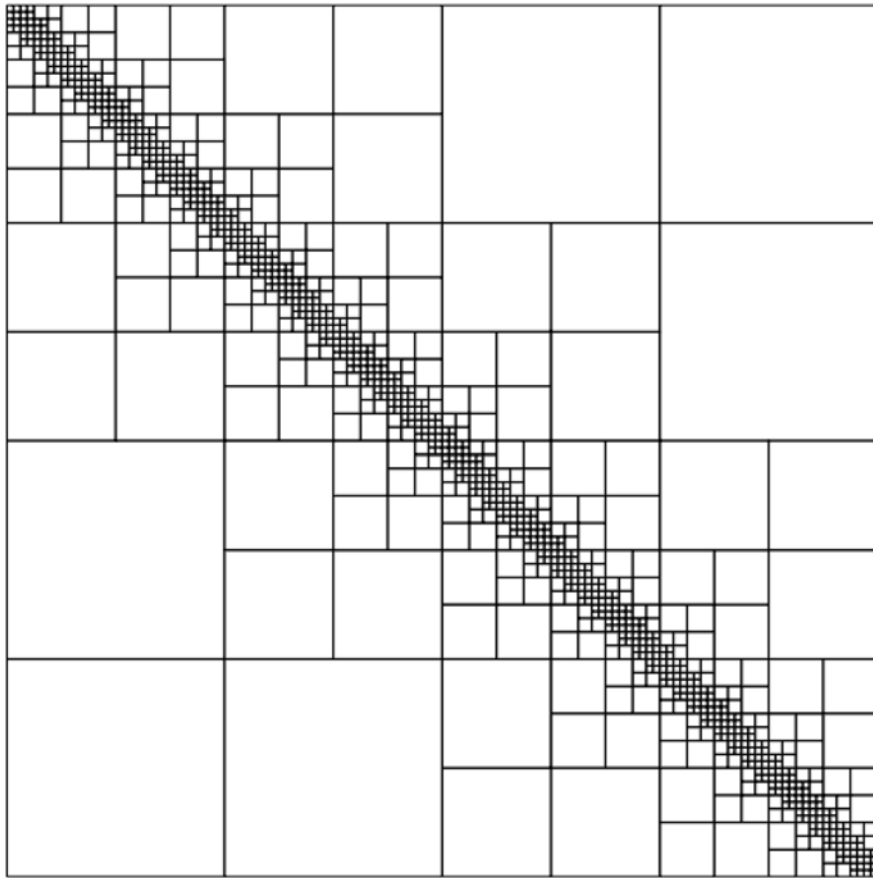


Until each block is acceptably small:

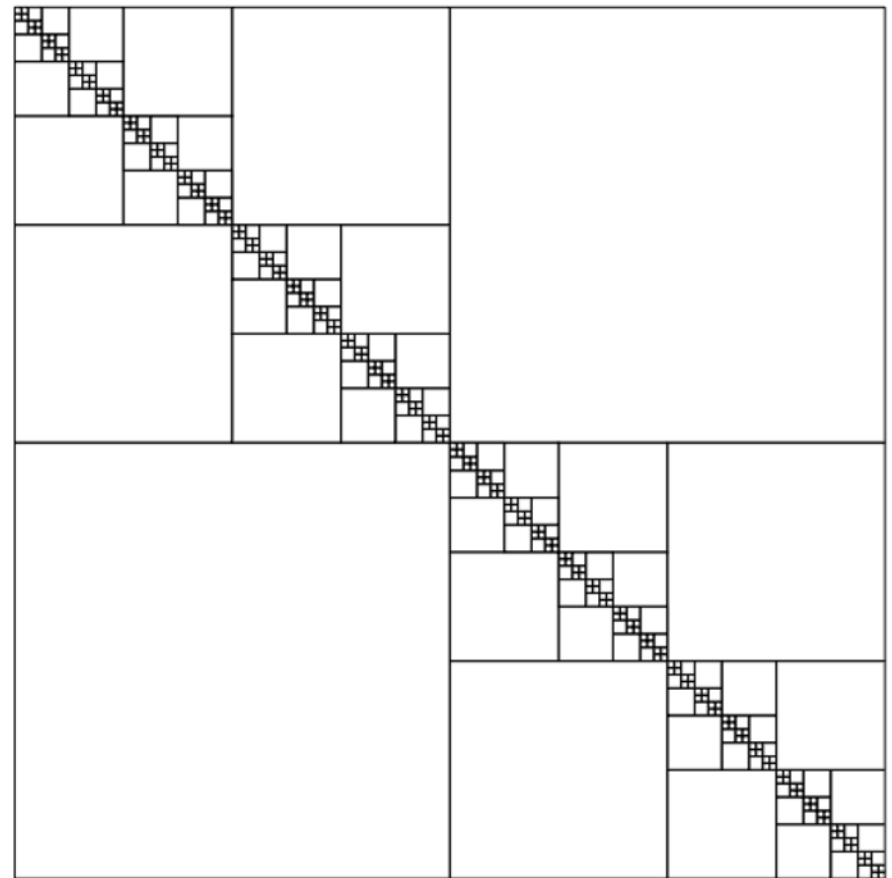
- Is rank acceptably small?
- If not, subdivide block

Take union of leaf blocks

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



**strong admissibility**



**weak admissibility**

**after Hackbusch, *et al.*, 2003**

---



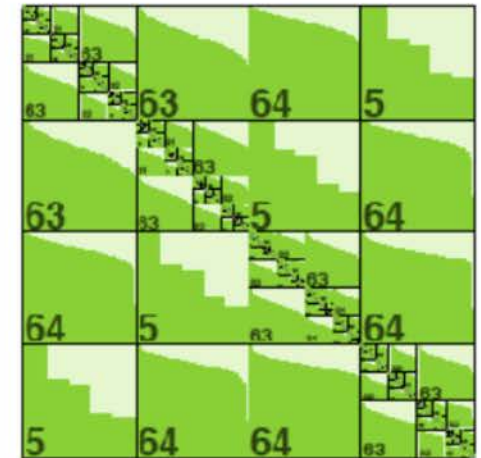
# Hierarchically low-rank “renaissance”

## Replace dense linear algebra

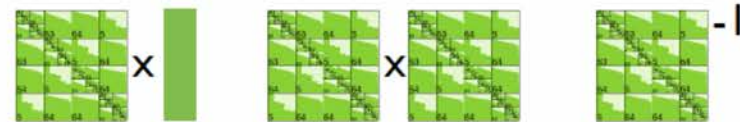
Compute :  $\mathcal{O}(N^3) \longrightarrow \mathcal{O}(k^a N \log^b N)$

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

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



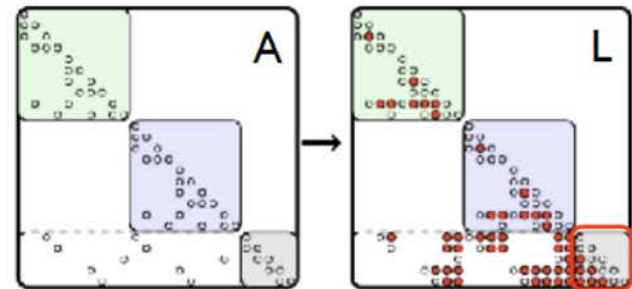
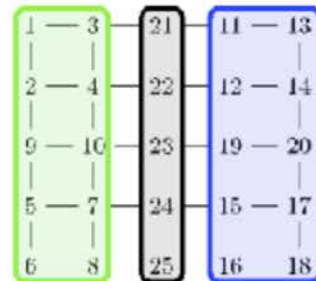
## Augment sparse linear algebra



### Sparse direct solvers

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

Nested dissection



Schur complement

### Iterative solvers

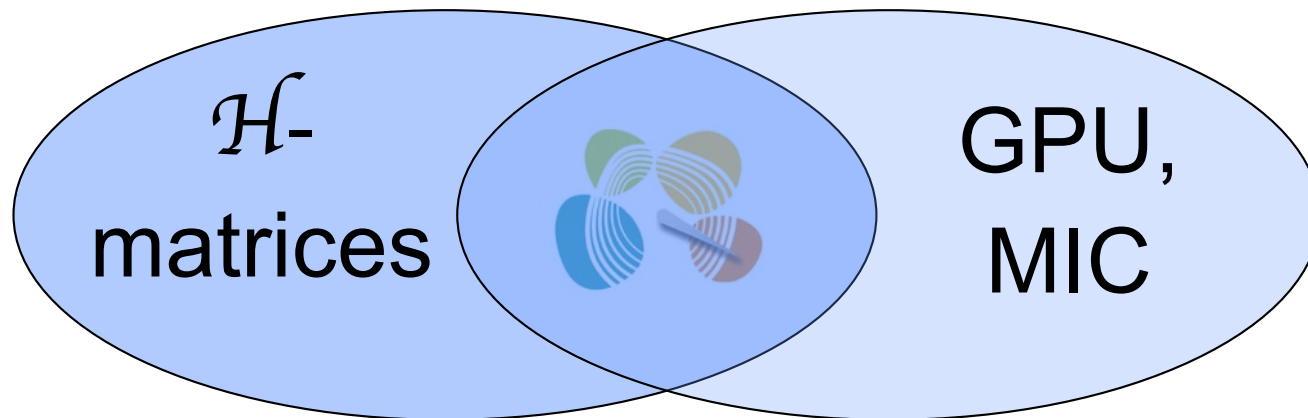
Use small  $k$  to precondition

Less sensitive to matrix condition than multigrid

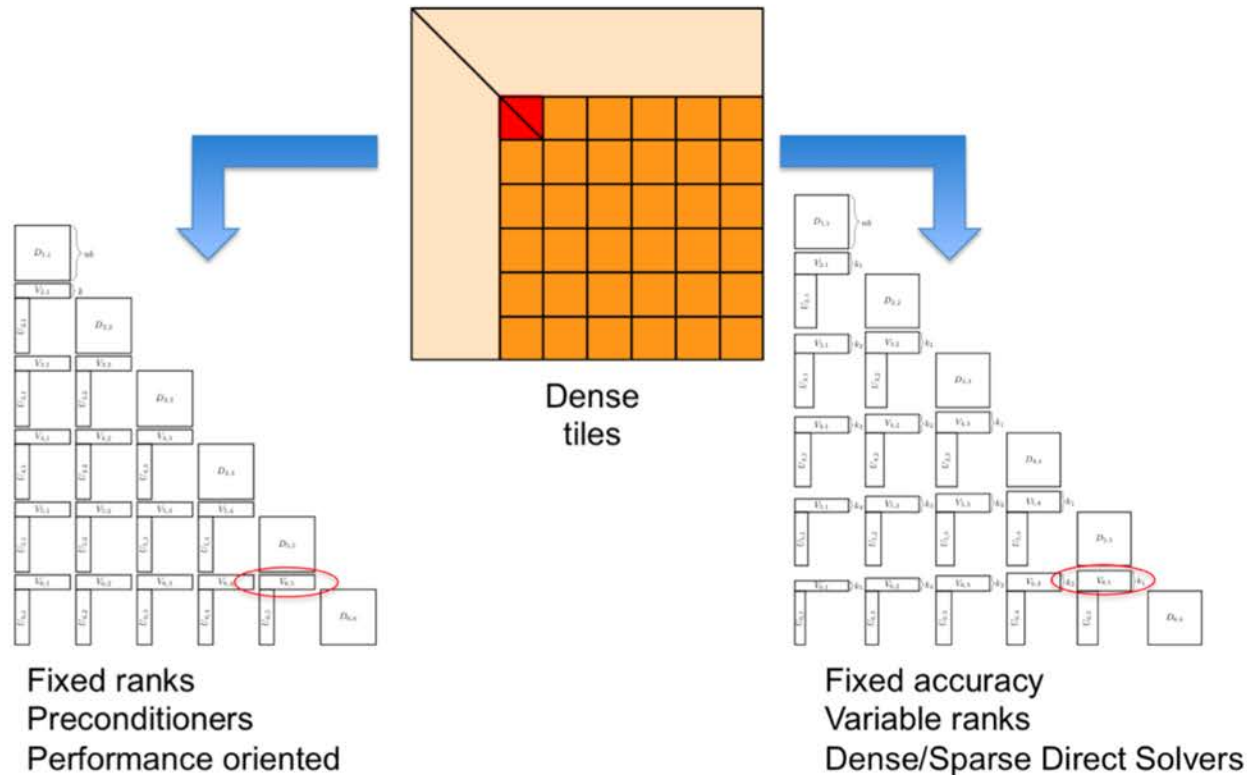
# Hierarchical algorithms and extreme scale

Must address the *tension* between

- highly uniform vector, matrix, and general SIMT 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

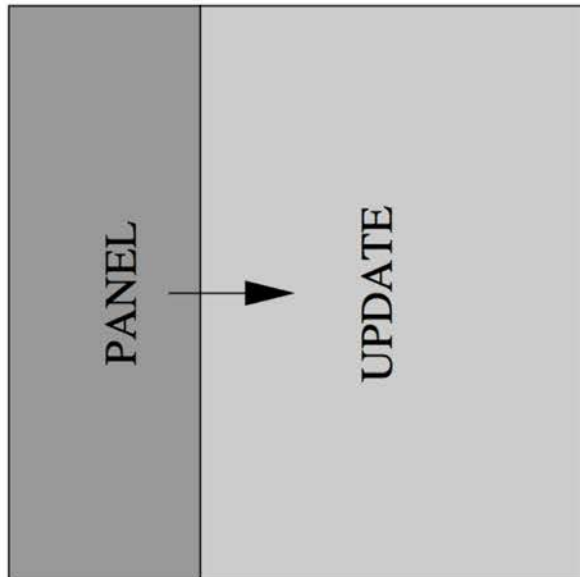
$$A = LL^T,$$

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

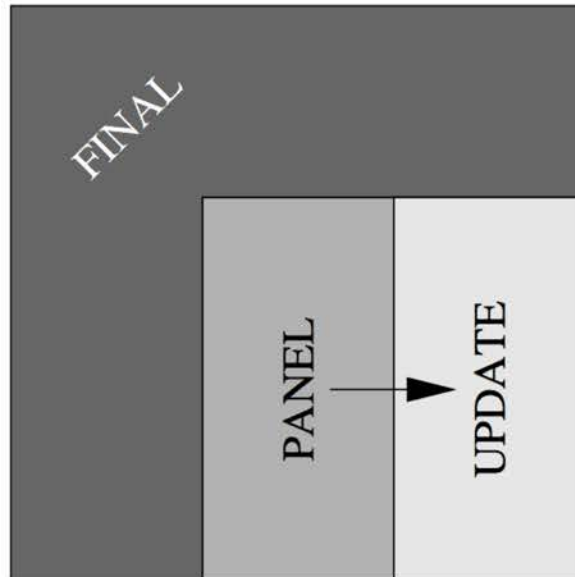
---



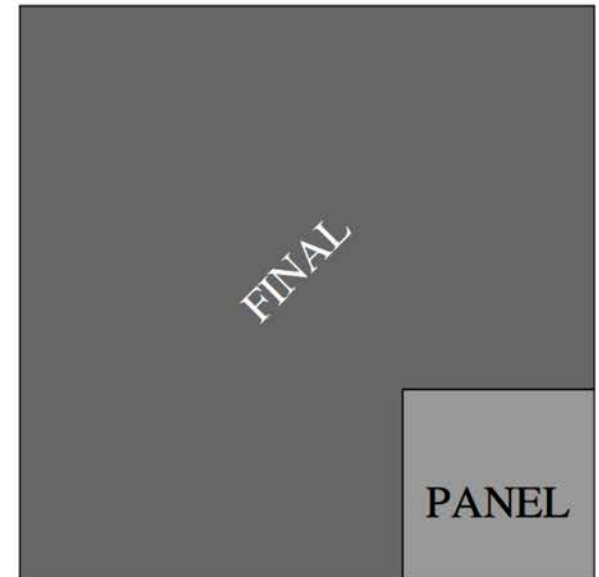
# Panel algorithm (LAPACK's POTRF)



(a) First step.



(b) Second step.



(c) Third step.

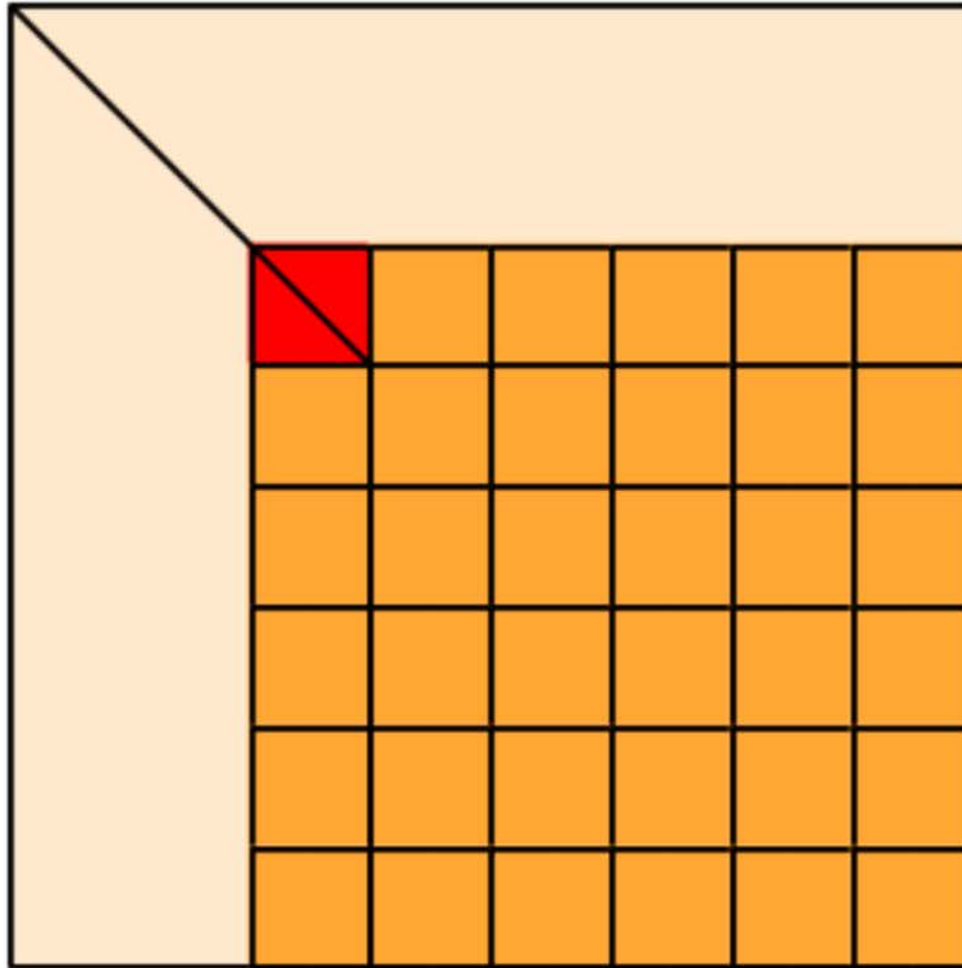
**Left- and right-looking variants, column-blocked for BLAS3**

**Decreasing concurrency**

**Artifactual over-ordering**

---

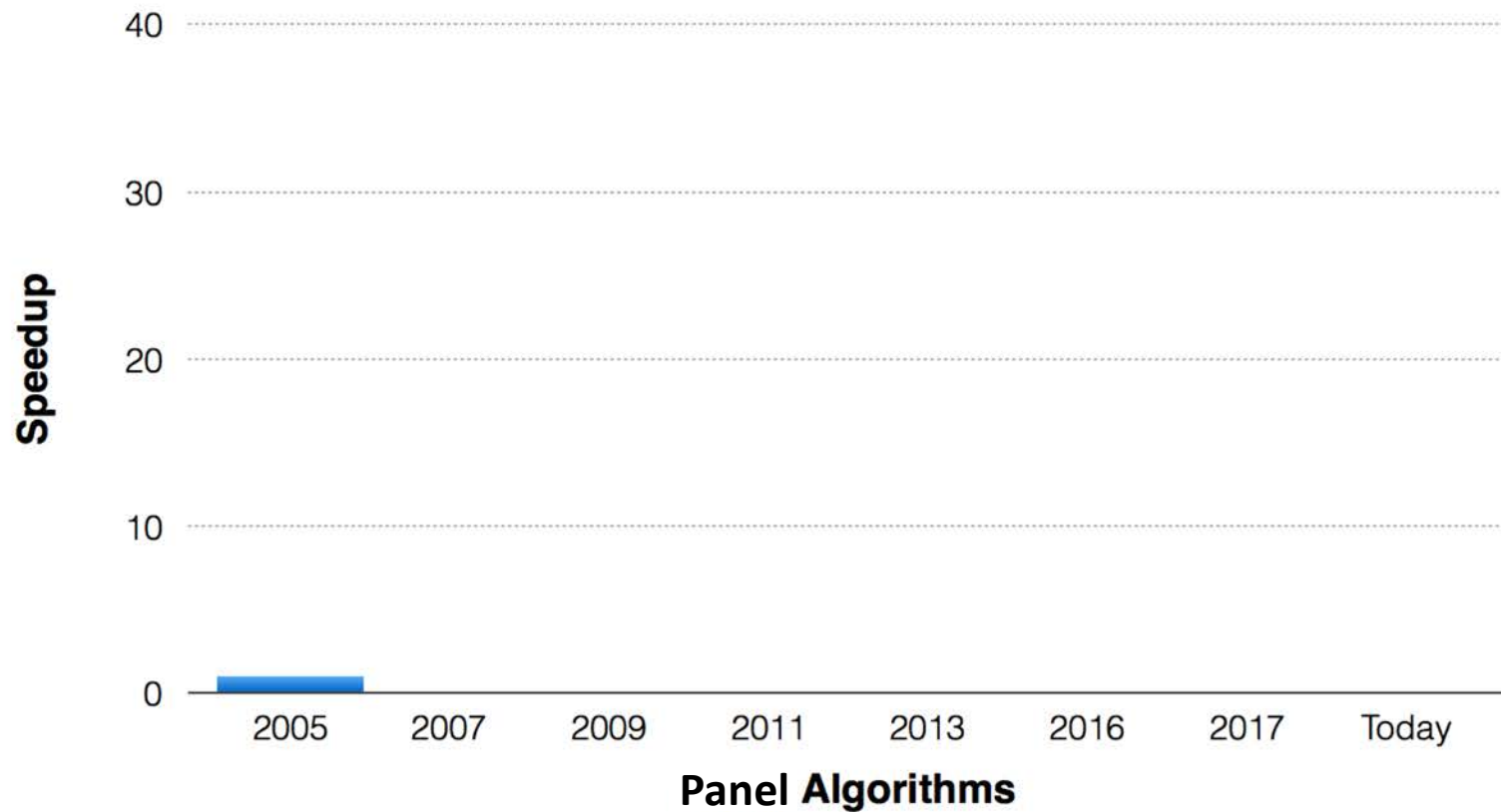
# Tile algorithm (PLASMA, MAGMA, Chameleon)



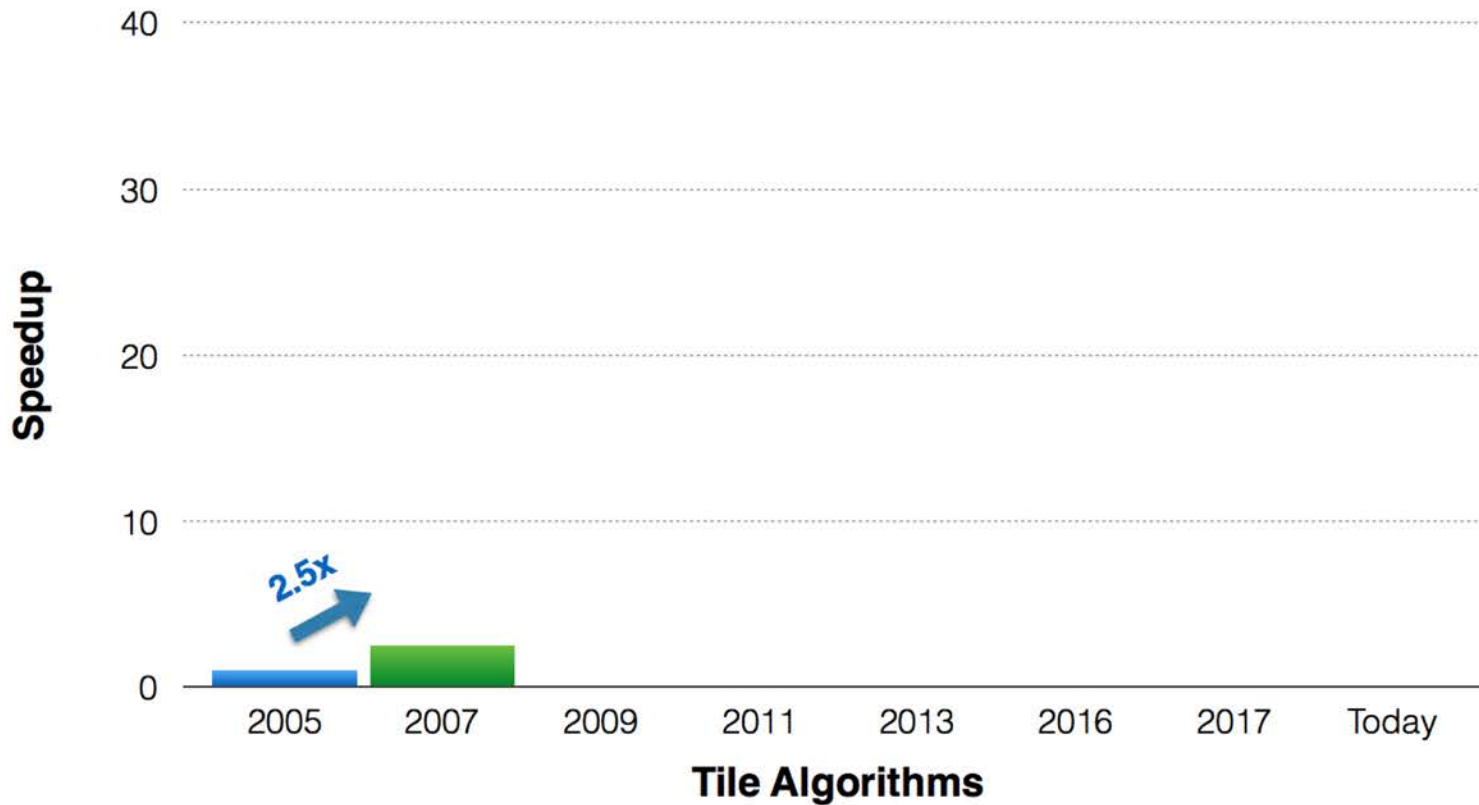
**Implemented with DAG-based scheduling**

---

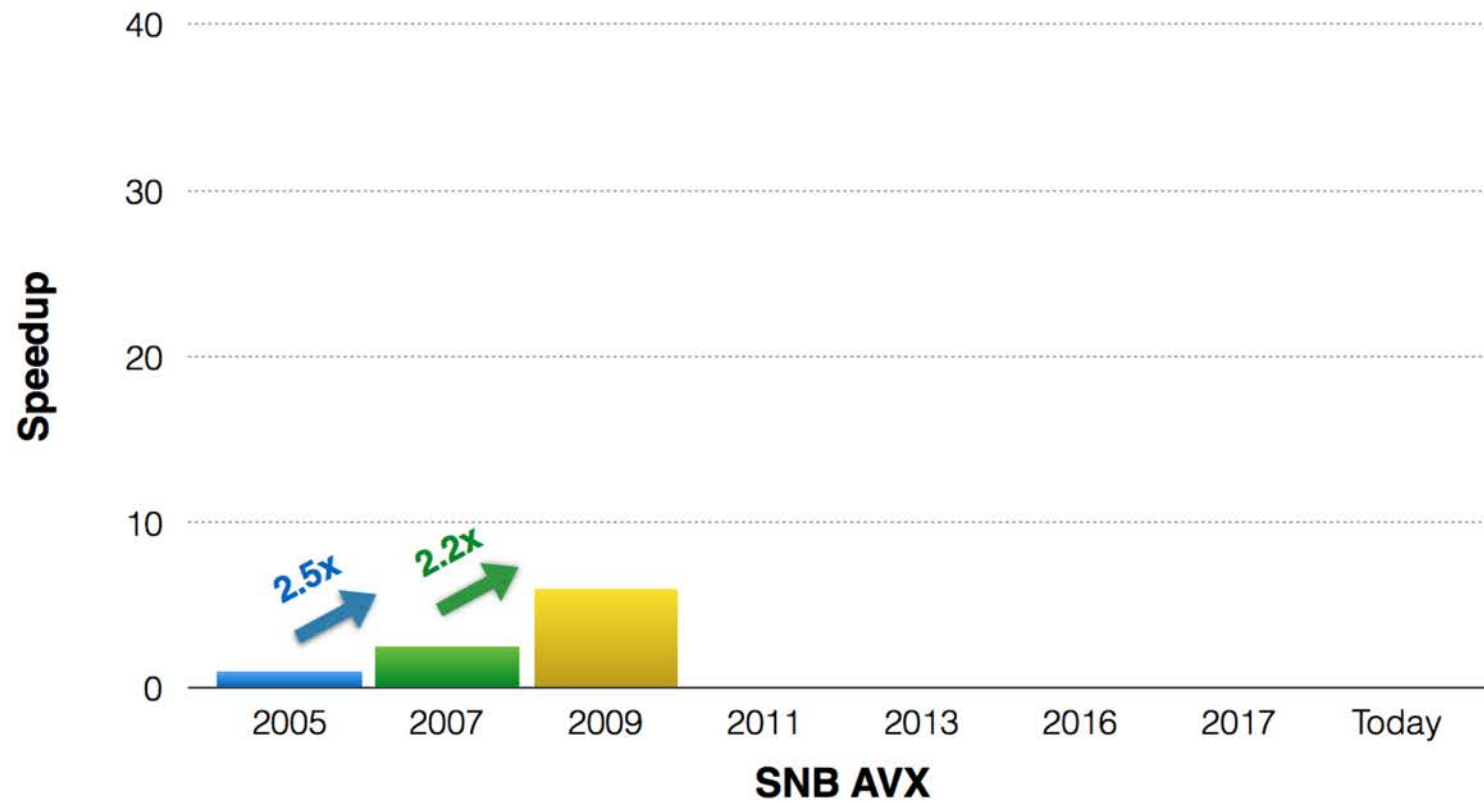
# Performance evolution of dense Cholesky



# Performance evolution of dense Cholesky

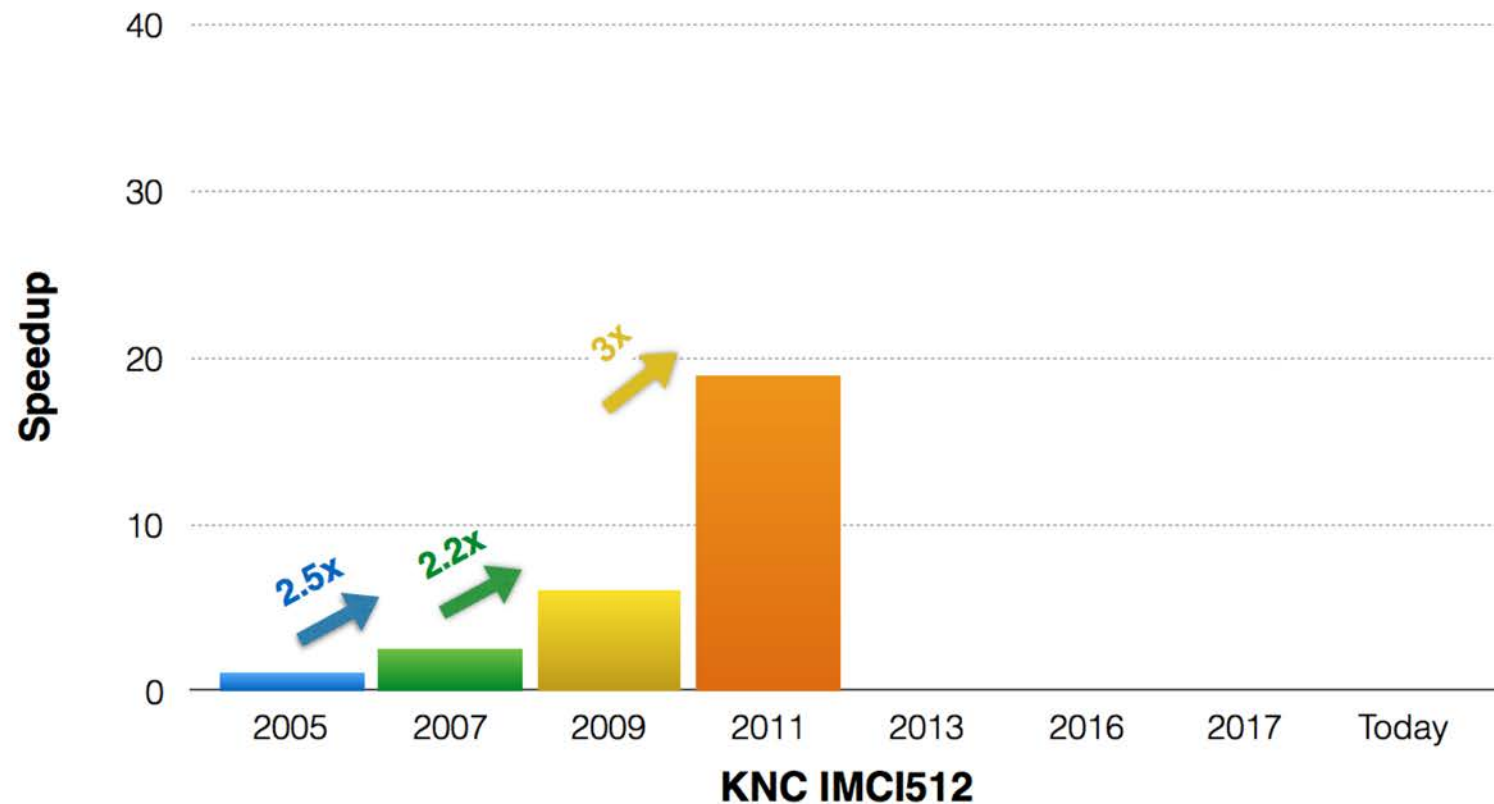


# Performance evolution of dense Cholesky

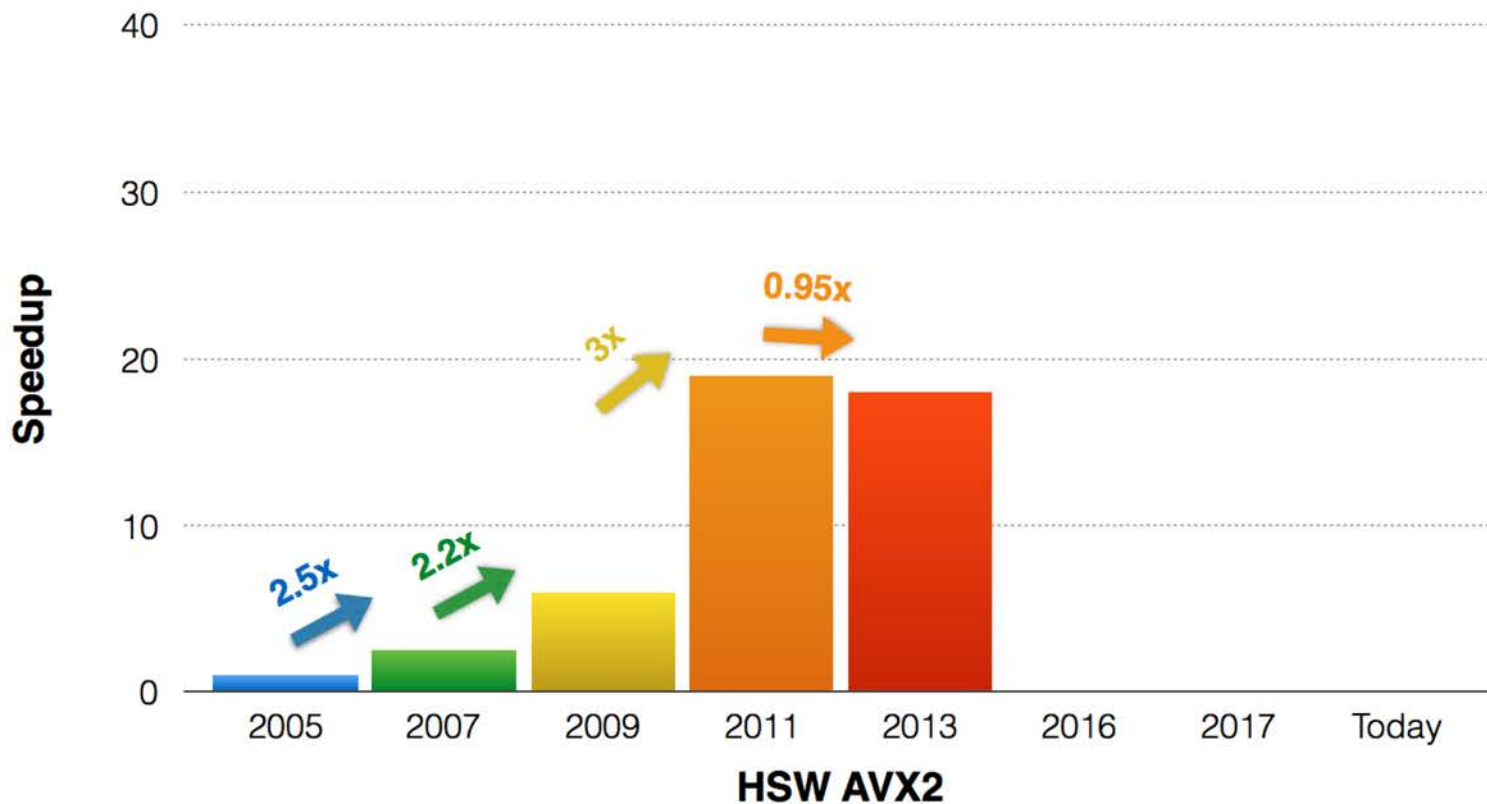




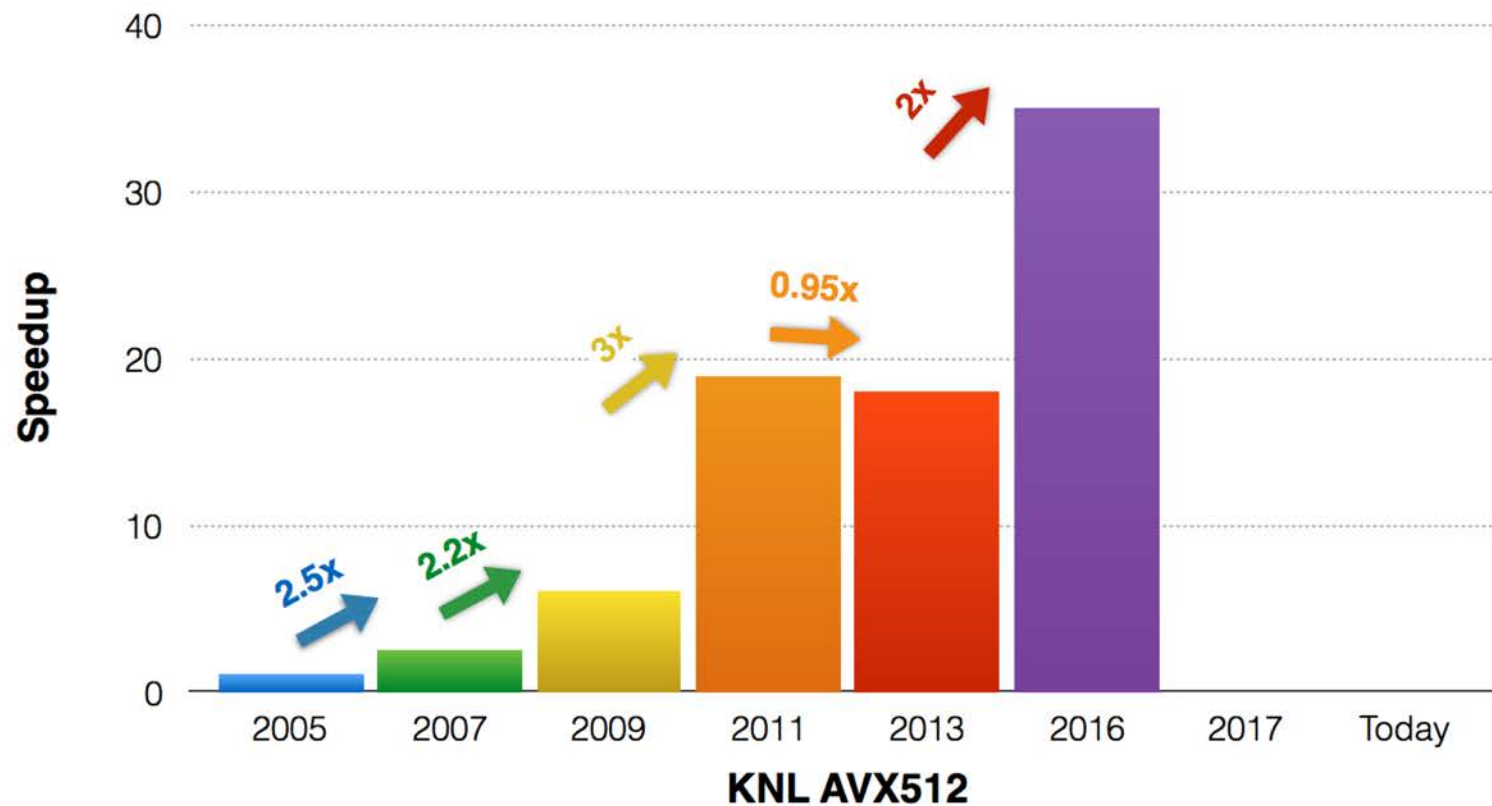
# Performance evolution of dense Cholesky



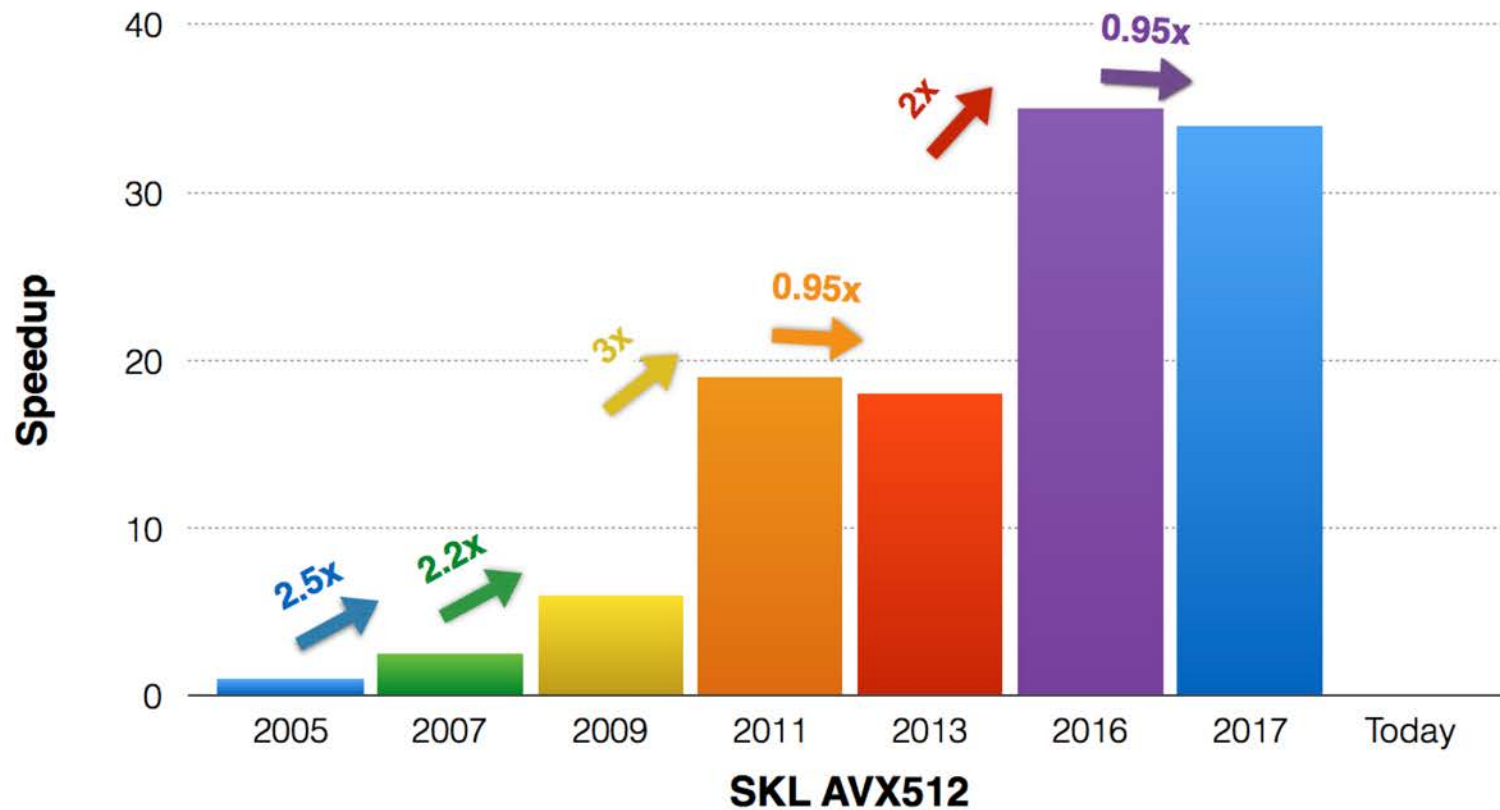
# Performance evolution of dense Cholesky



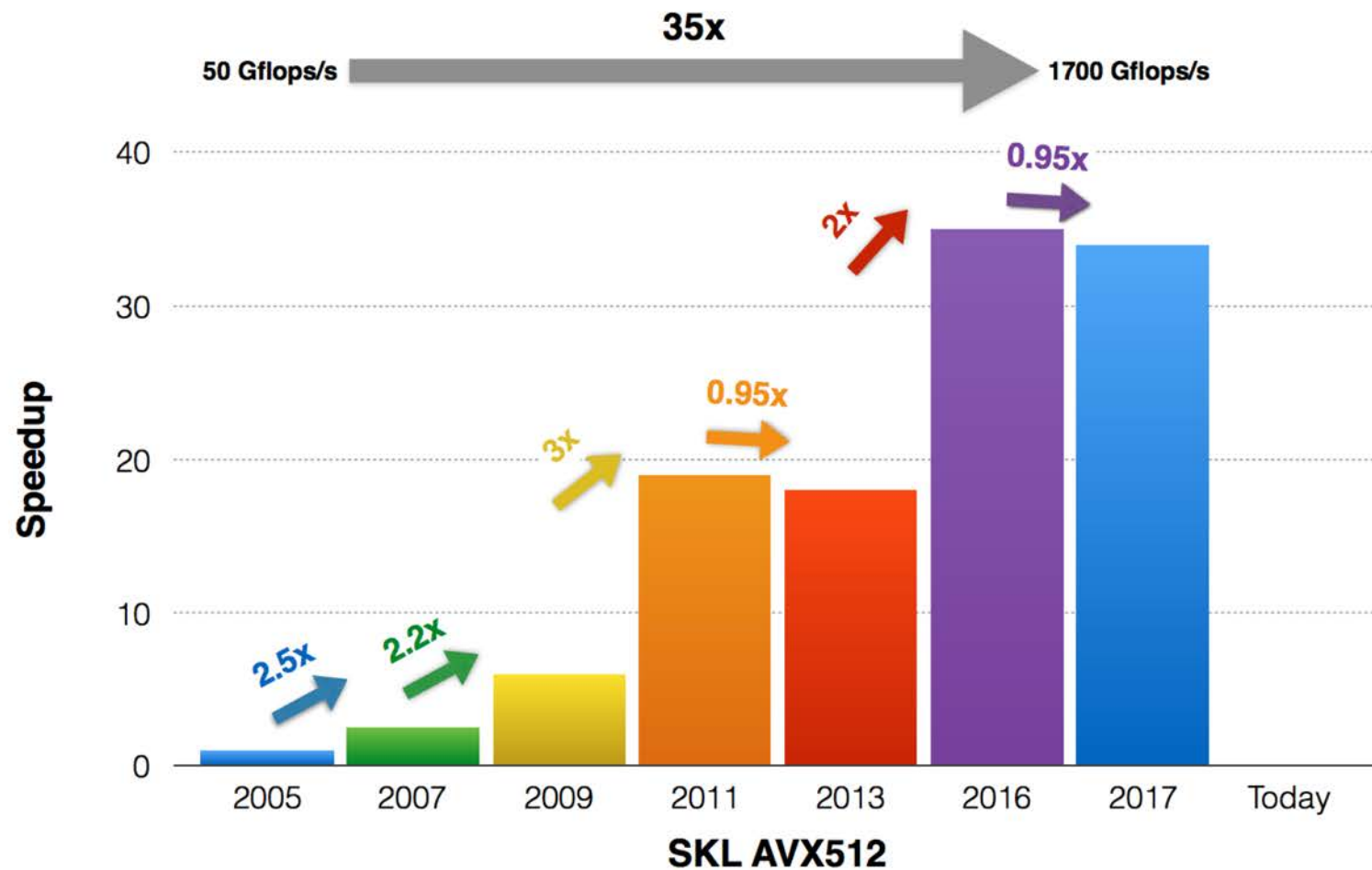
# Performance evolution of dense Cholesky



# Performance evolution of dense Cholesky

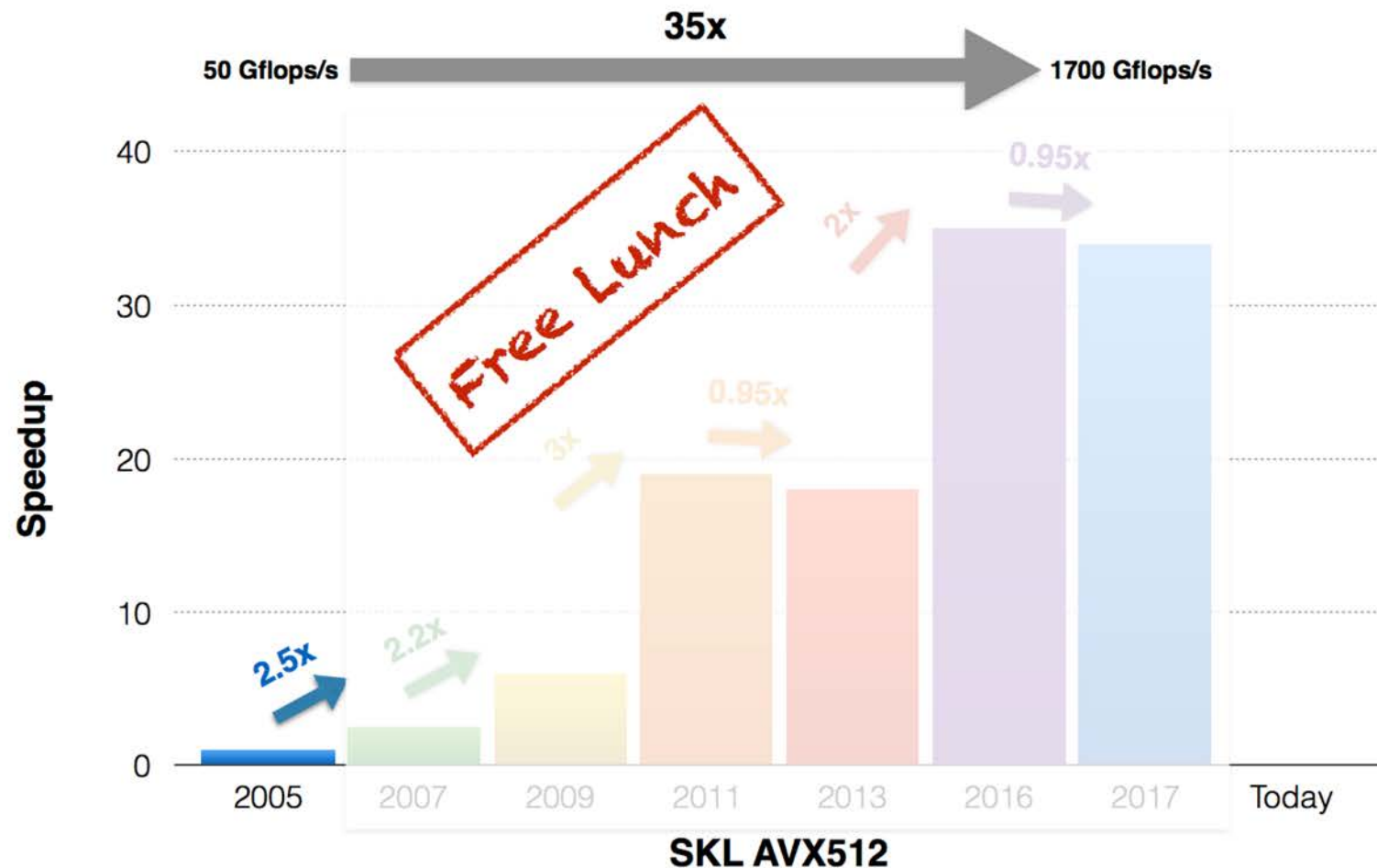


# Performance evolution of dense Cholesky



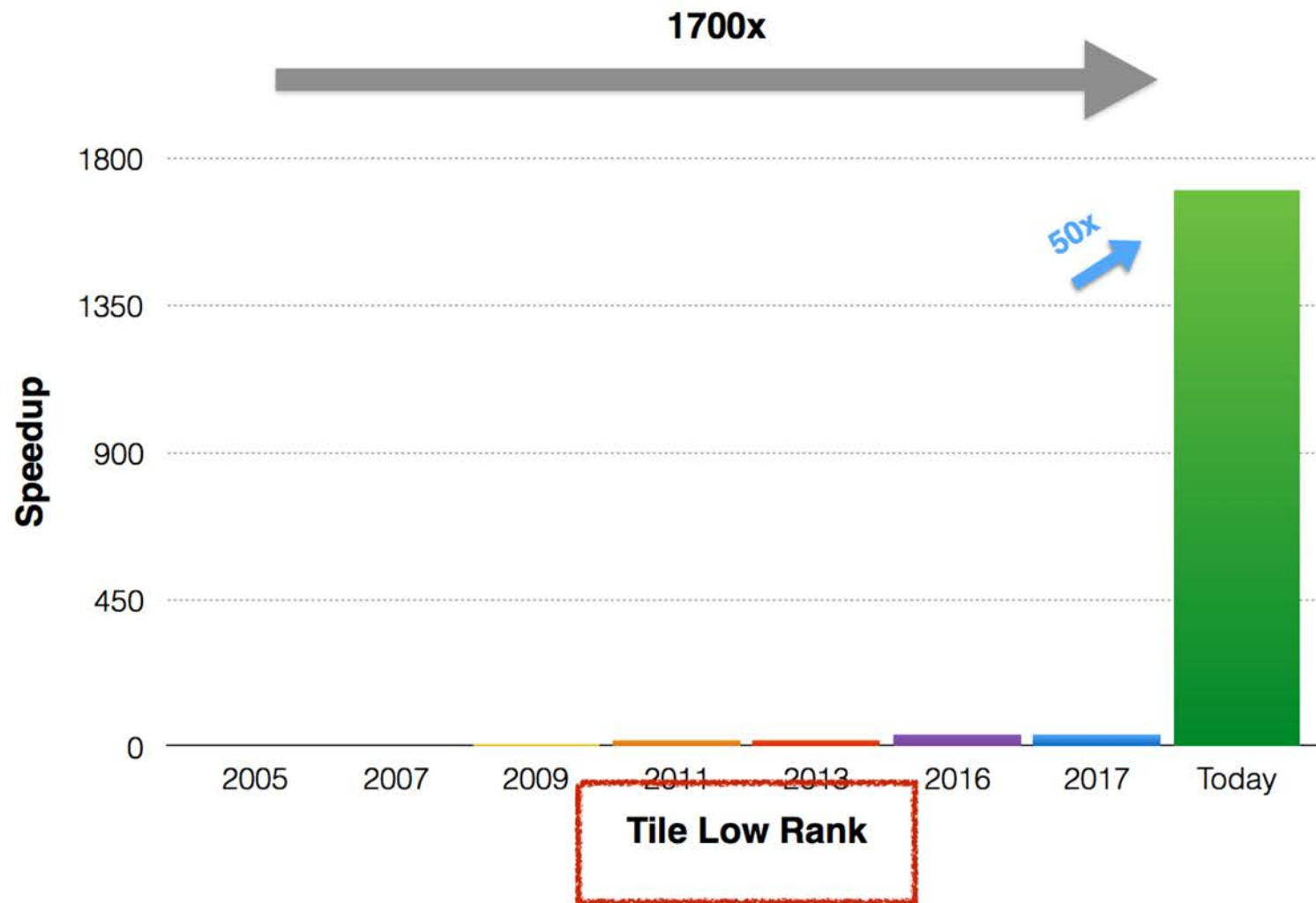


# Performance evolution of dense Cholesky



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

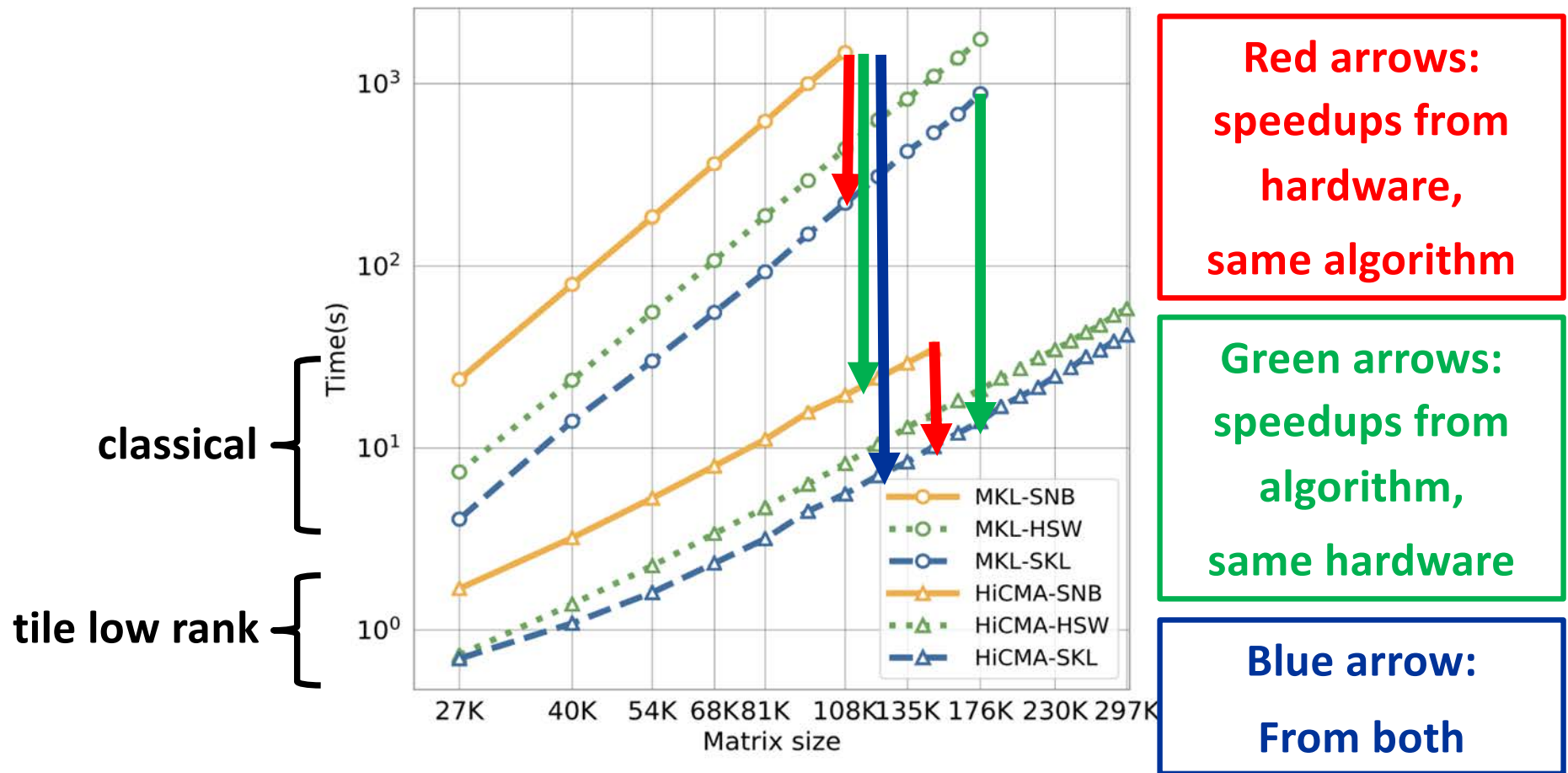
# Performance evolution of dense Cholesky



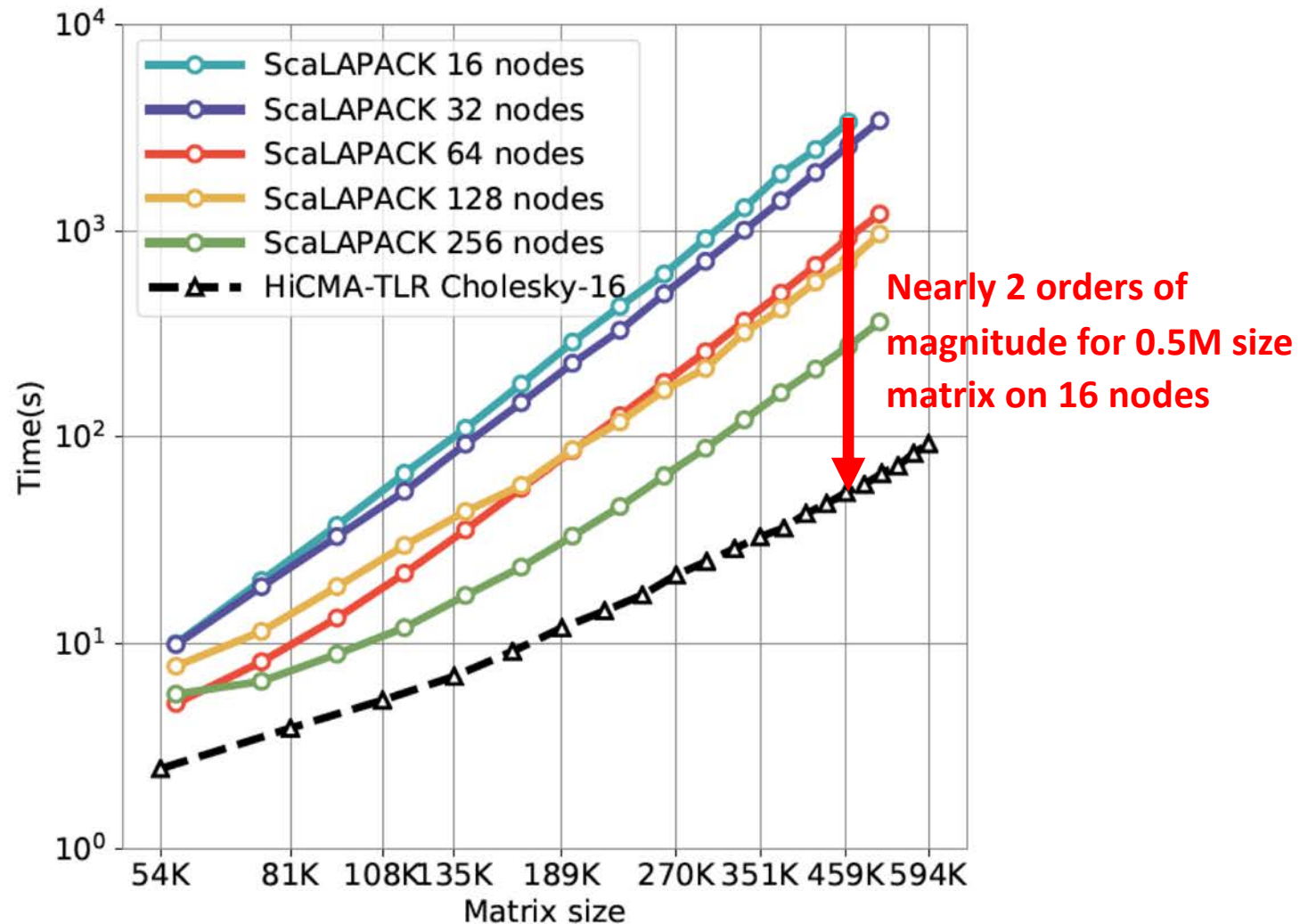
# HiCMA vs. Intel MKL on shared memory

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)

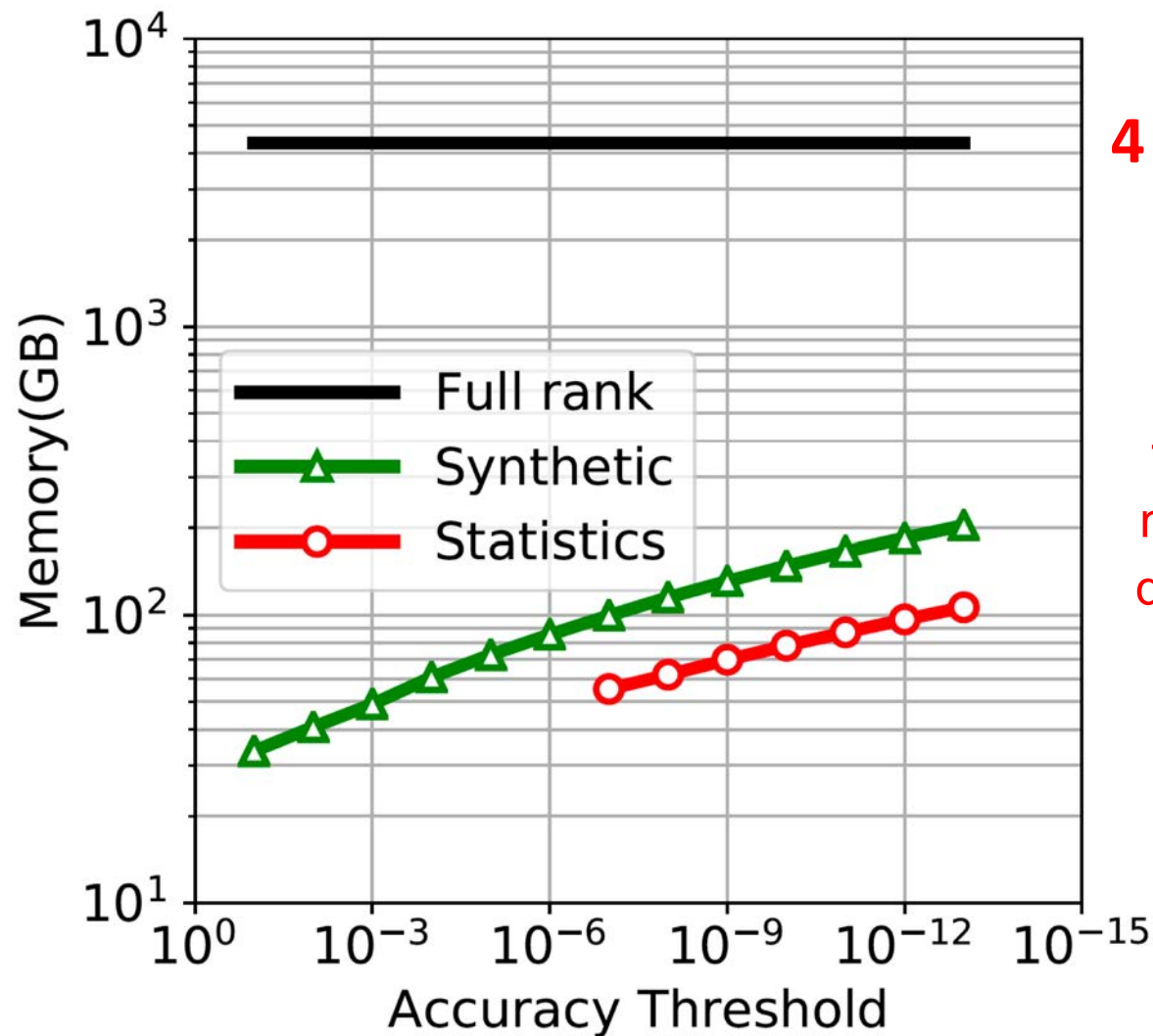


# HiCMA vs. ScaLAPACK on distributed memory



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

# Memory footprint for DP matrix of size 1M



4 TB 

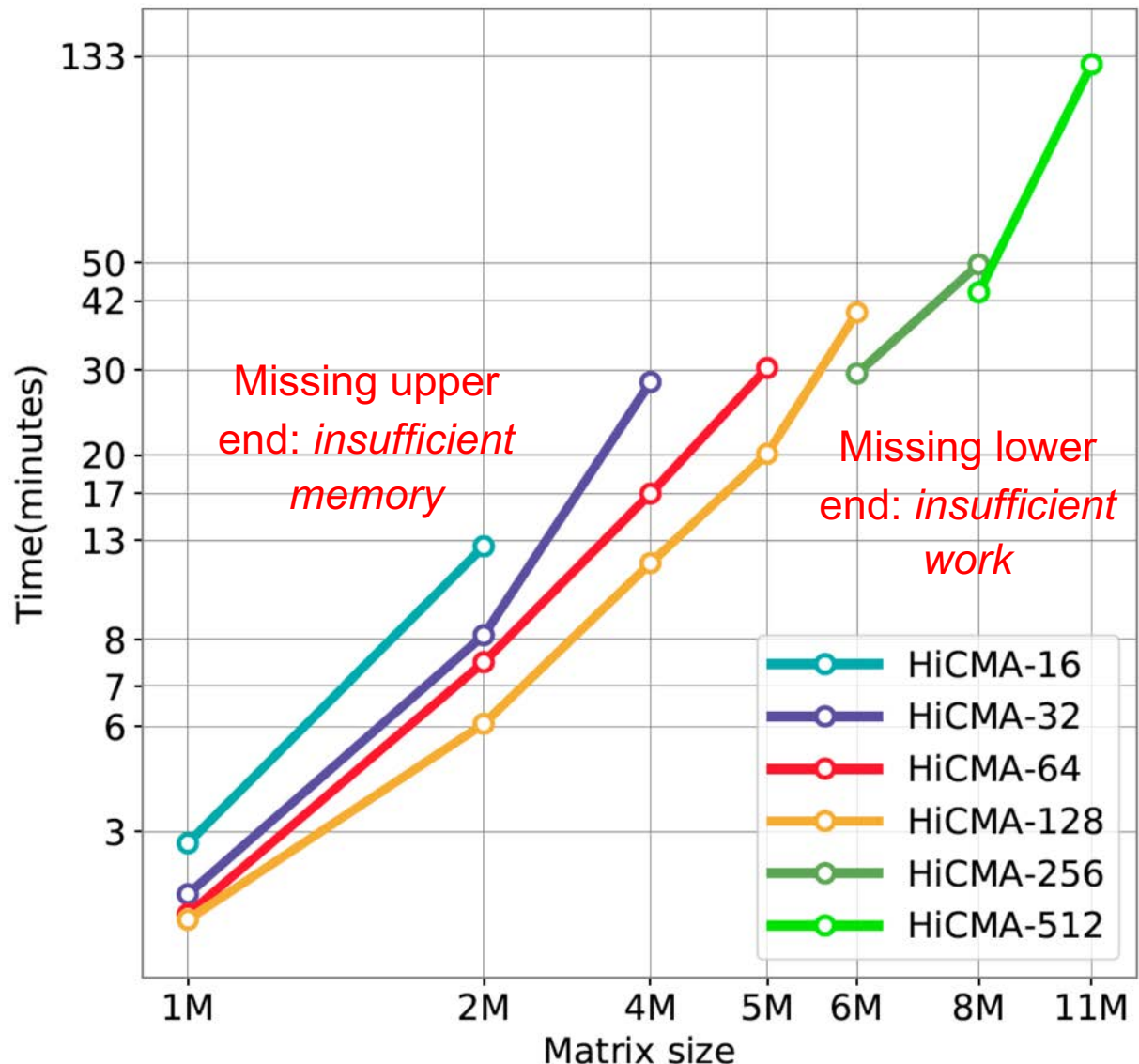
1 to 2 orders of magnitude less, depending upon accuracy



# HiCMA on distributed memory

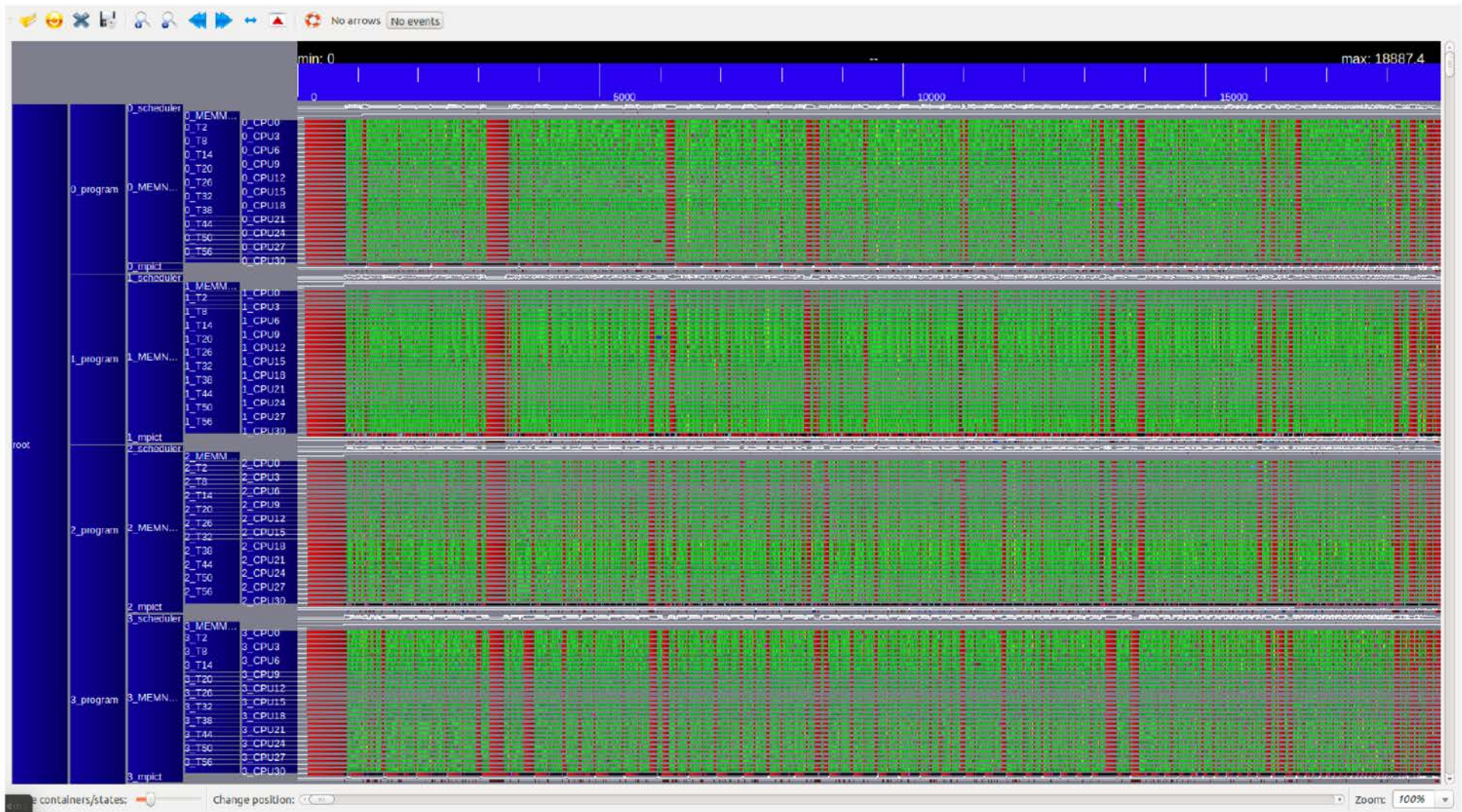
Geospatial statistics  
(Gaussian kernel) to  
accuracy  $1.0e-8$

- Cray XC40  
(“Shaheen”, 32  
Haswell cores per  
node)
- Range of problem  
sizes and core  
count



# Execution traces

Chameleon: Dense DPOTRF time **18.1s**  
4 nodes of Shaheen with a matrix size of 54K

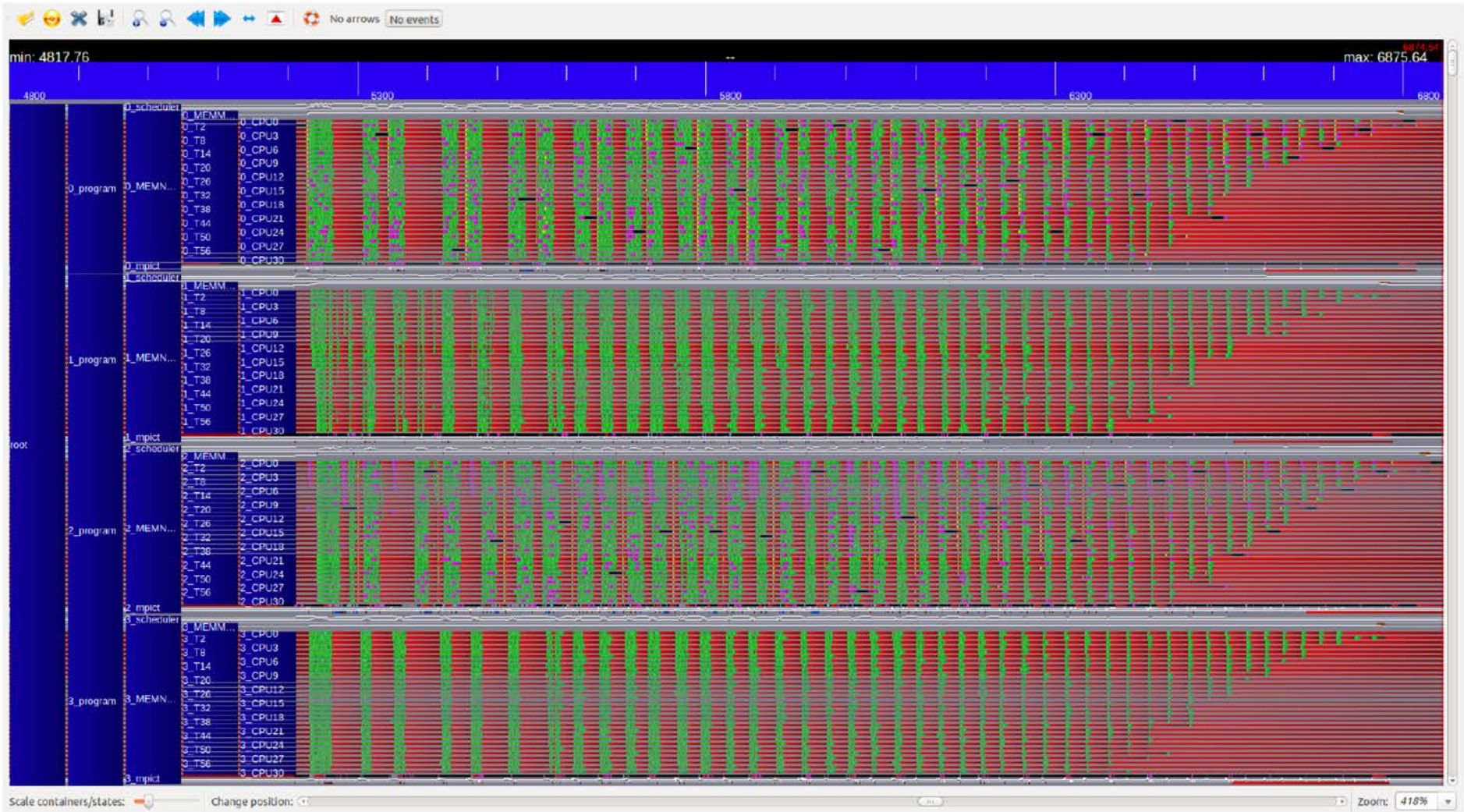


Akbadak, Ltaief, Mikhalev, Charara, & Keyes, Exploiting Data Sparsity for Large-scale Matrix Computations, Europar 2018.



# Execution traces

HiCMA: TLR DPOTRF time **1.8s** (10X faster)  
4 nodes of Shaheen with a matrix size of 54K



Akbudak, Ltaief, Mikhalev, Charara, & Keyes, Exploiting Data Sparsity for Large-scale Matrix Computations, Europar 2018.

**So far, just Tile Low Rank...**



**IT GETS  
BETTER®**

The logo consists of the words "IT GETS" on the top line and "BETTER" on the bottom line. The letters are bold and sans-serif. The colors are: 'I' is purple, 'T' is purple, 'G' is yellow, 'E' is blue, 'T' is purple, 'S' is purple, 'B' is purple, 'E' is yellow, 'T' is blue, 'T' is purple, 'E' is purple, and 'R' is purple. The letters are slightly overlapping and have a 3D effect.

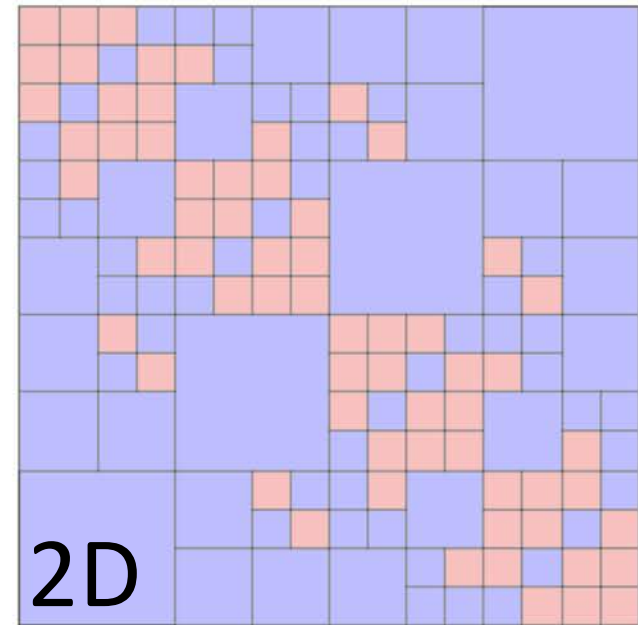
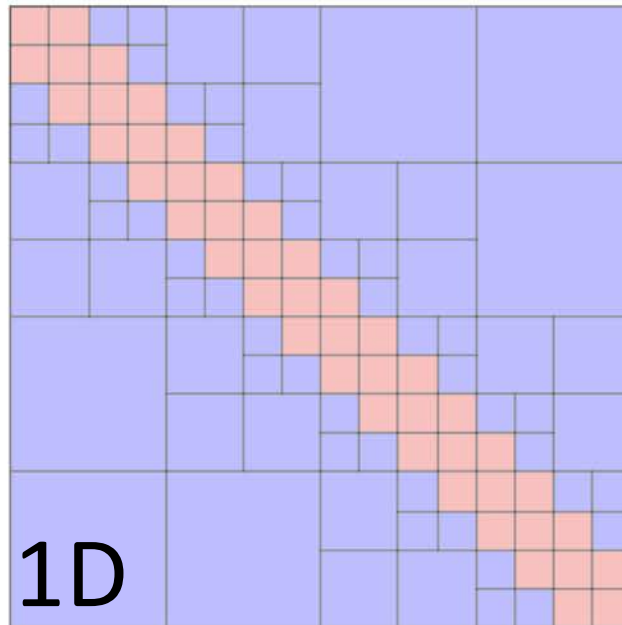
## Now: general $\mathcal{H}$ -matrices

- **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



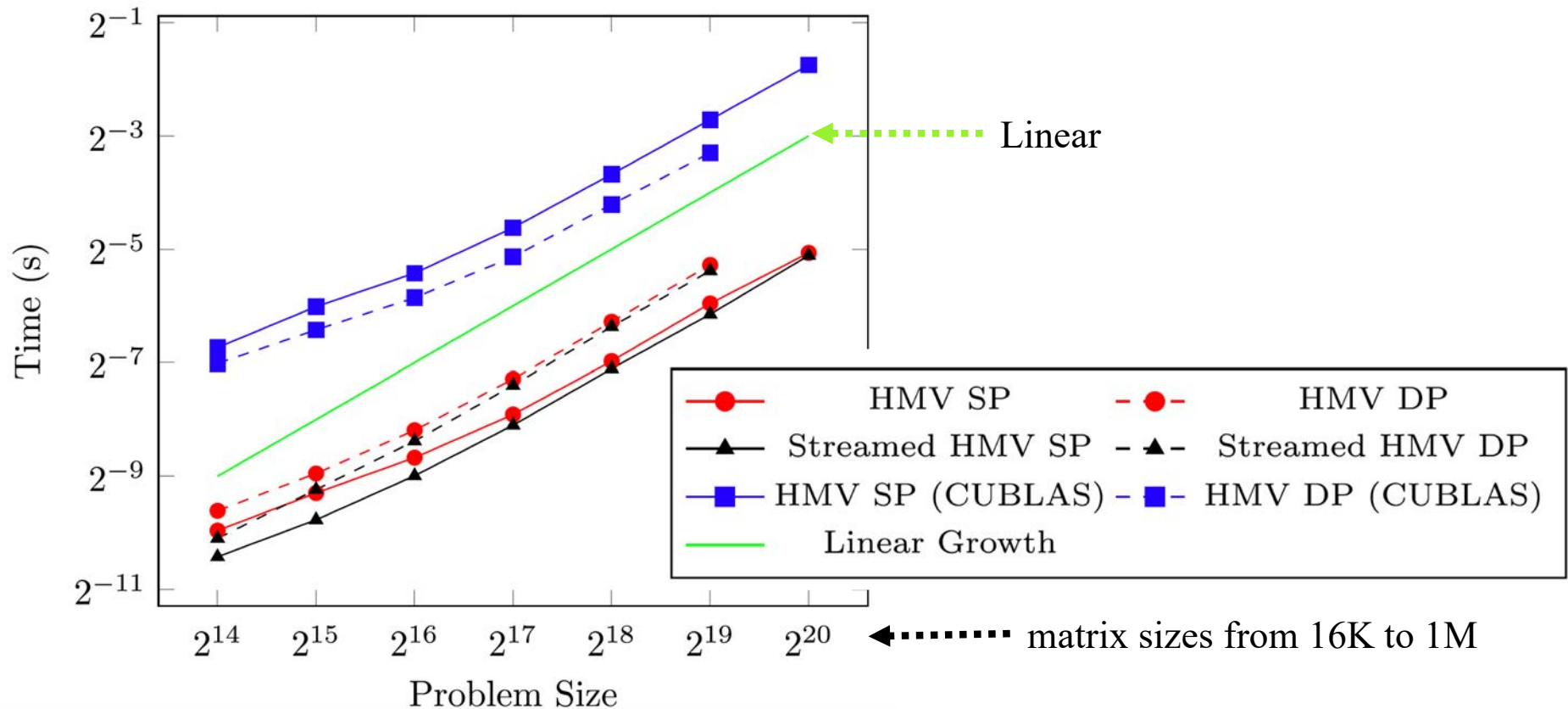
- ▶ nested bases

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

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

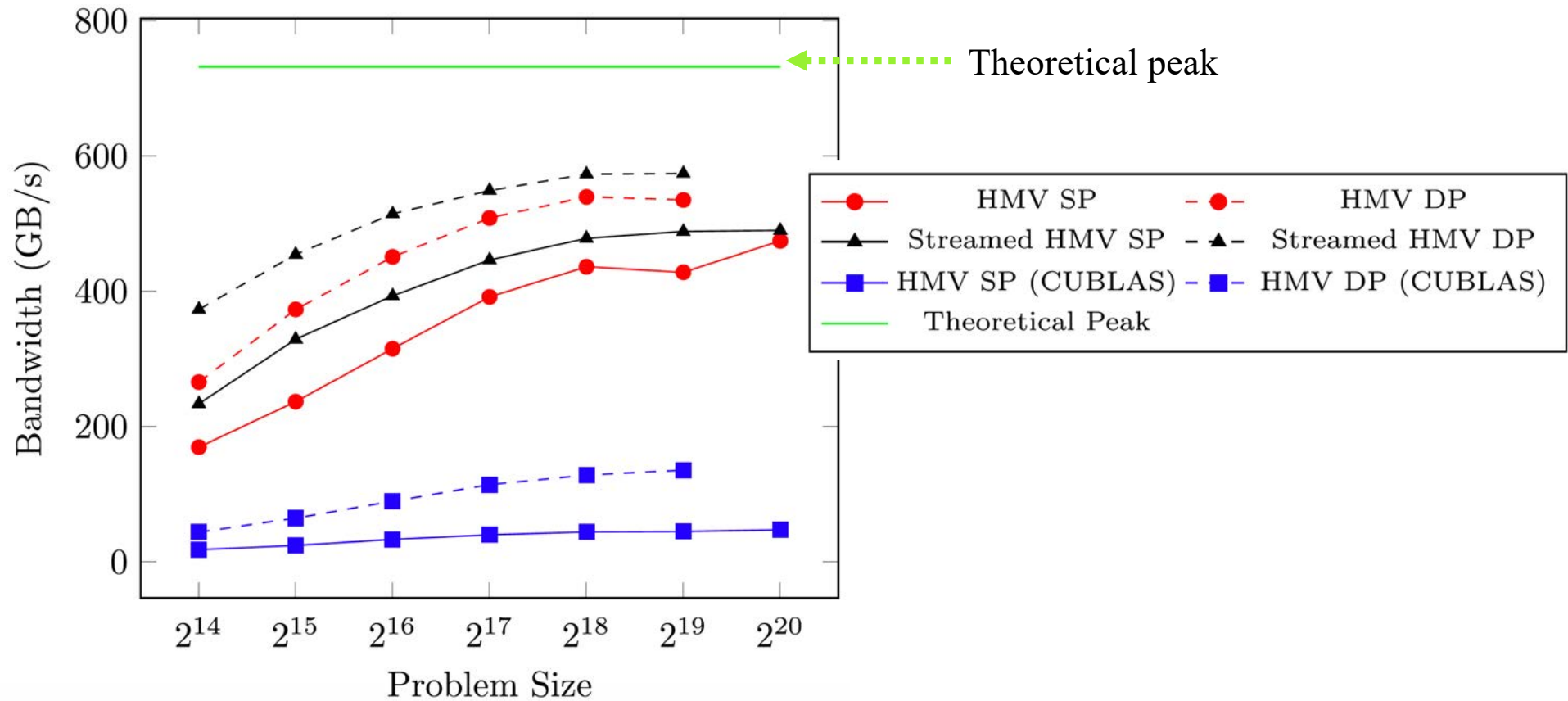
- reduces memory footprint to  $O(n)$  from  $O(n \log n)$

# Hierarchical MatVec execution time



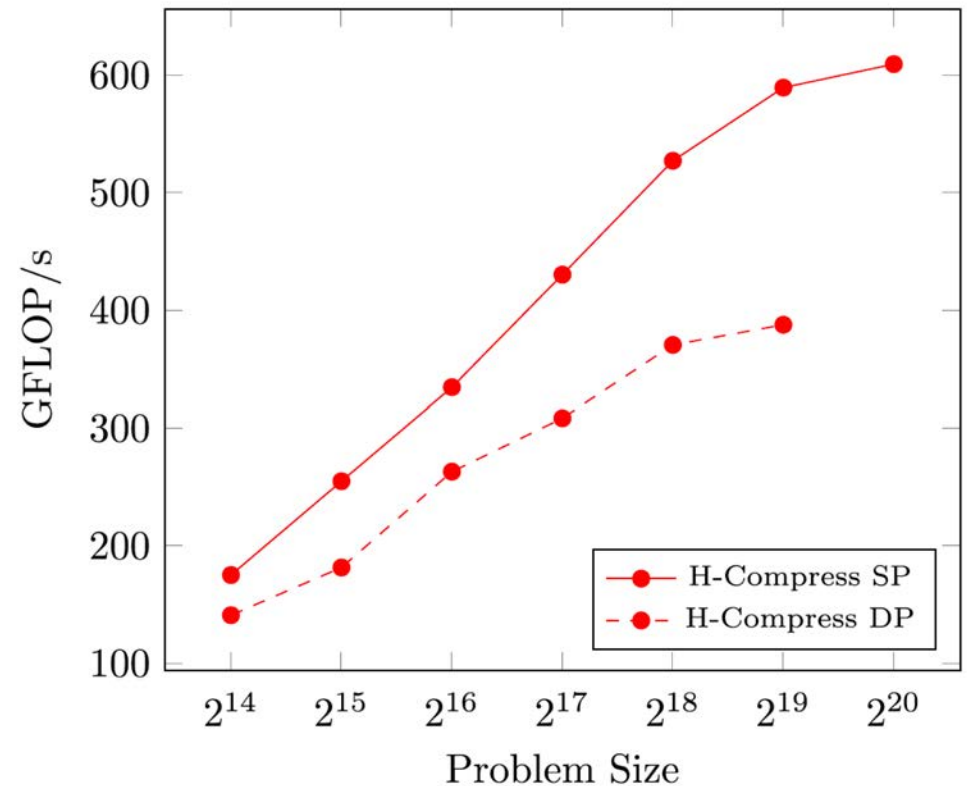
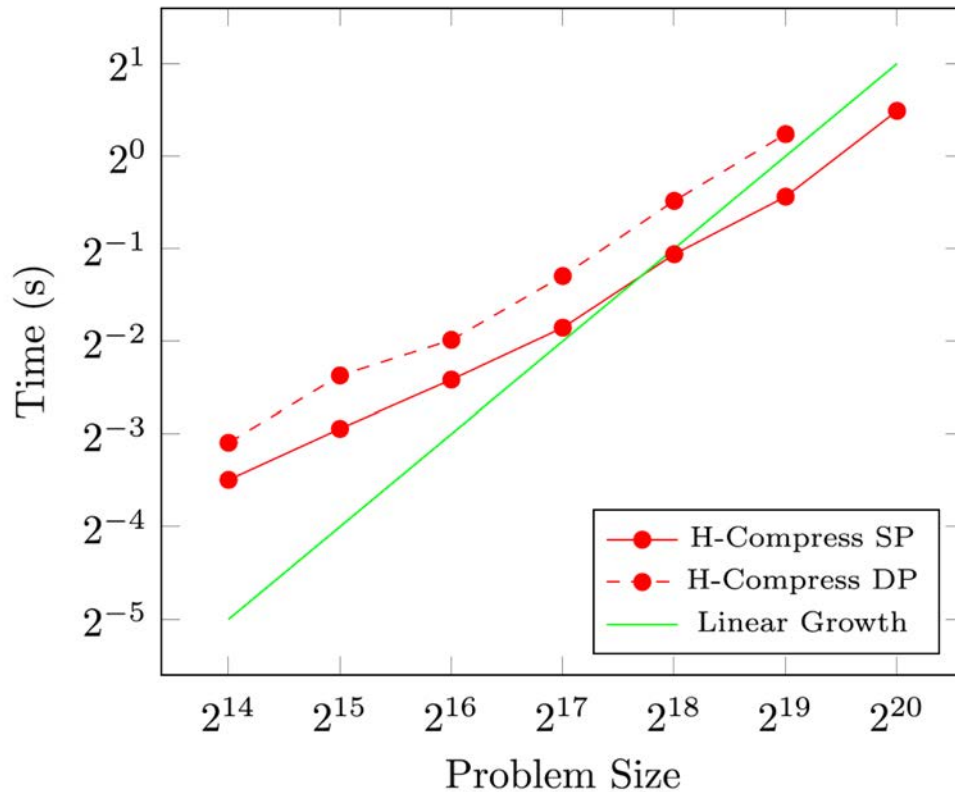
- ▶ 3D covariance matrices from spatial statistics
- ▶ running on P100 GPU
- ▶ accuracy  $10^{-3}$  computed as  $\|Ax - A^{\mathcal{H}}x\| / \|Ax\|$
- ▶ leaf size  $m = 64$

# Hierarchical MatVec bandwidth



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

# Hierarchical compression



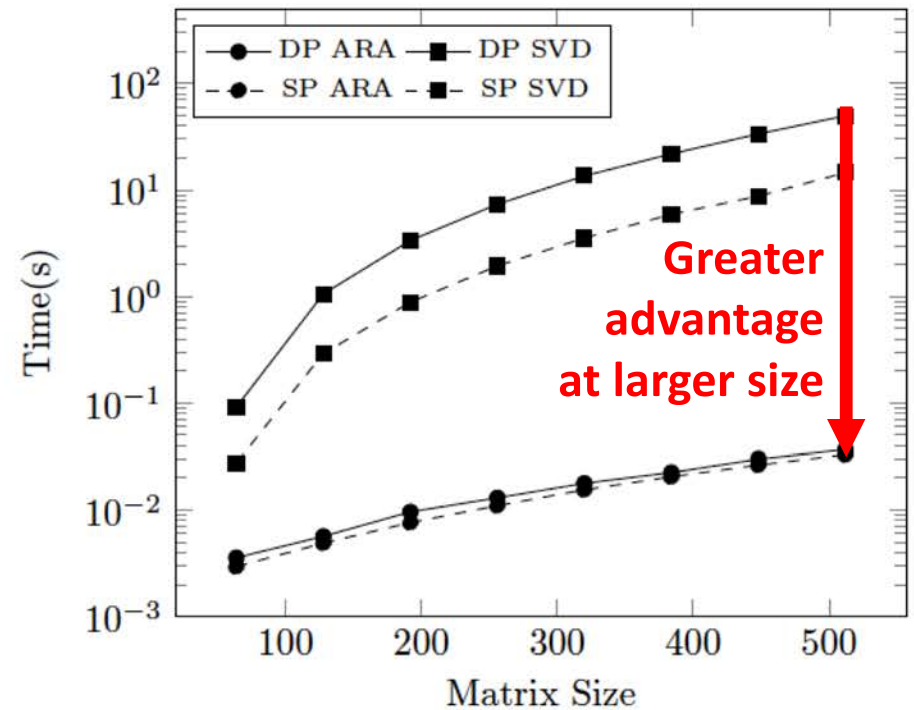
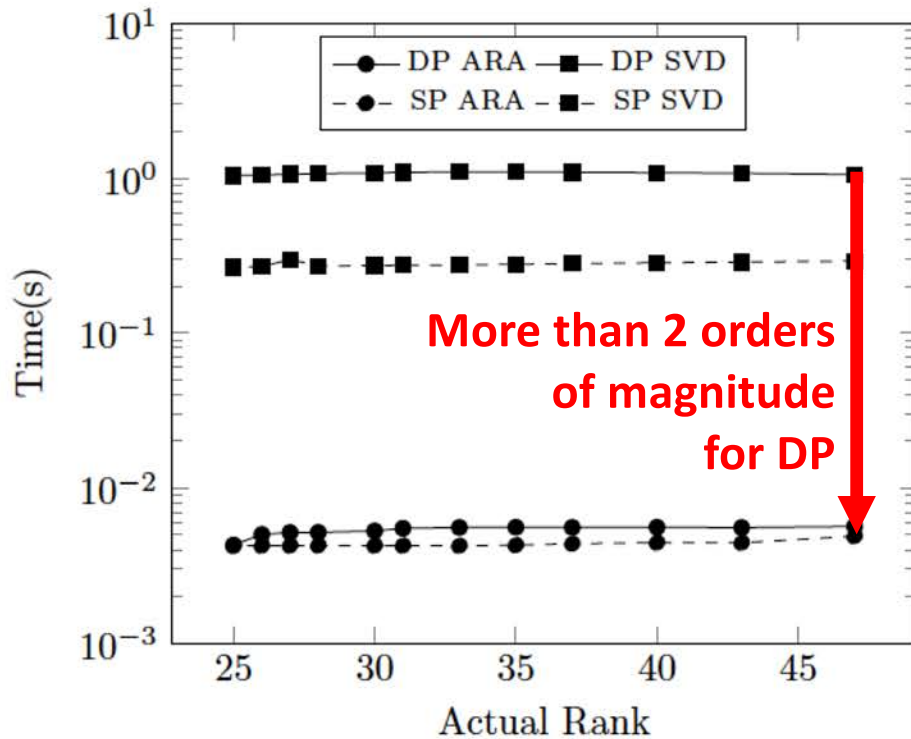
- ▶ 3D covariance matrices from spatial statistics
- ▶ running on P100 GPU
- ▶ accuracy  $10^{-3}$  computed as  $\|Ax - A^{\mathcal{H}}x\| / \|Ax\|$
- ▶ leaf size  $m = 64$

# 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
-



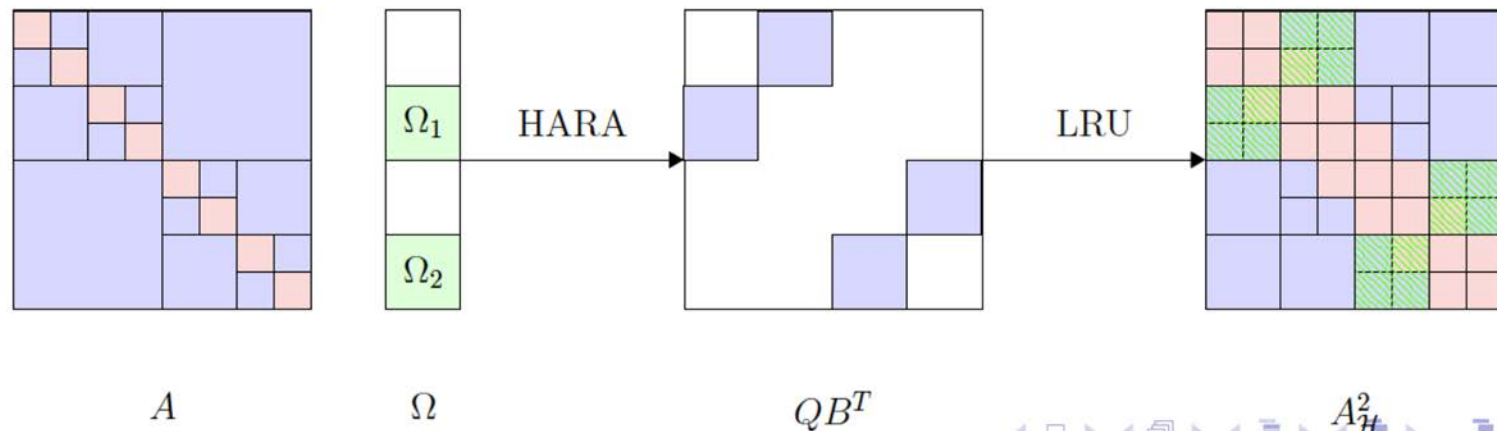
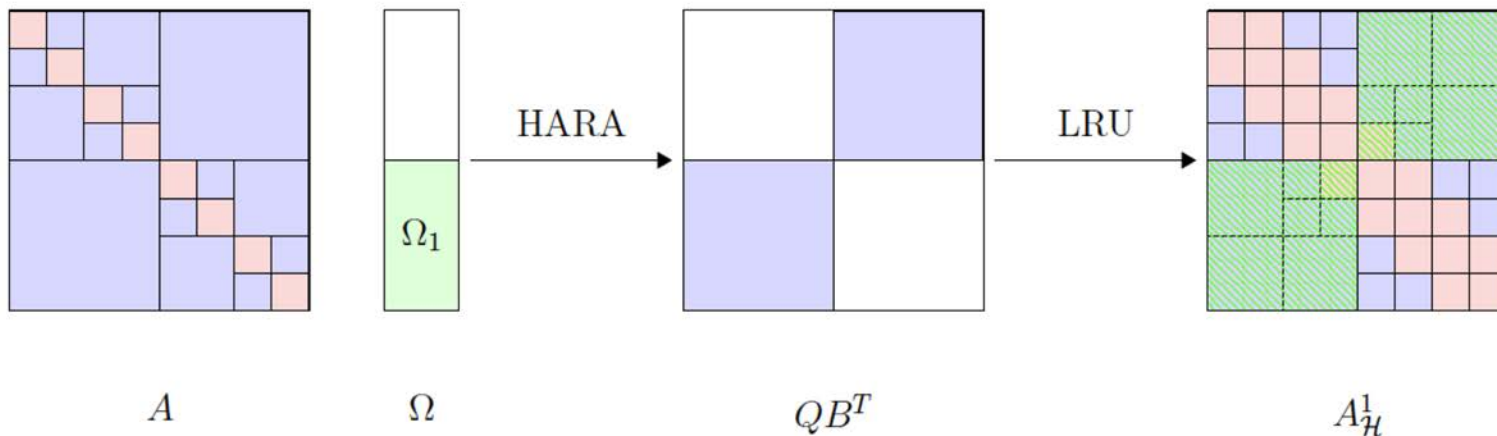
# Comparison with Jacobi (Givens) SVD



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

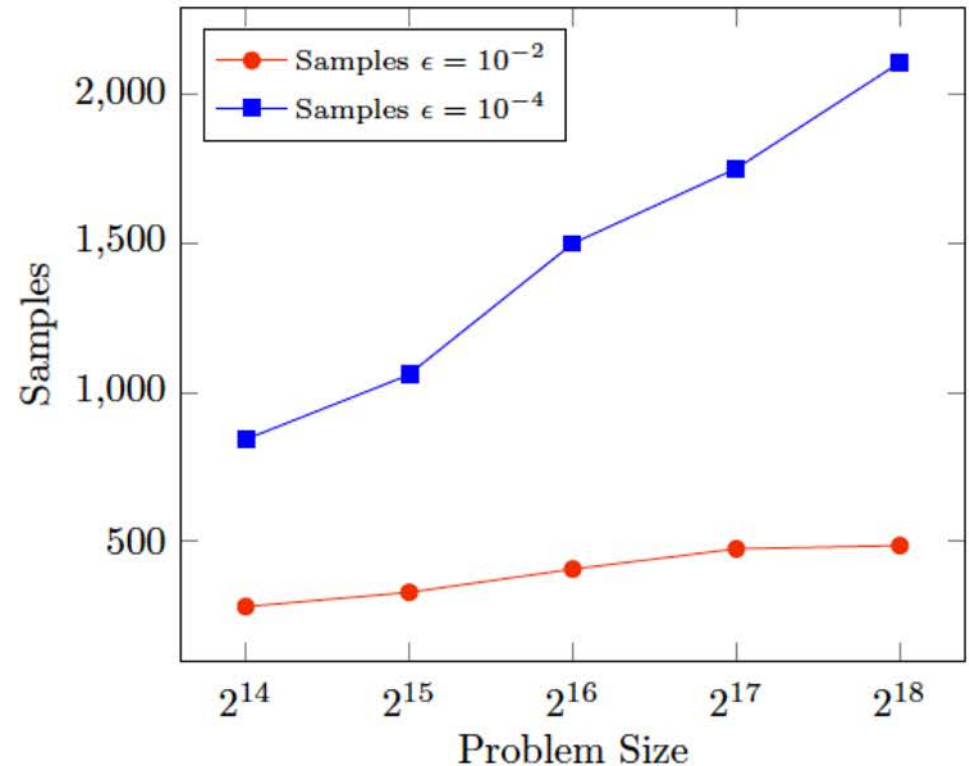
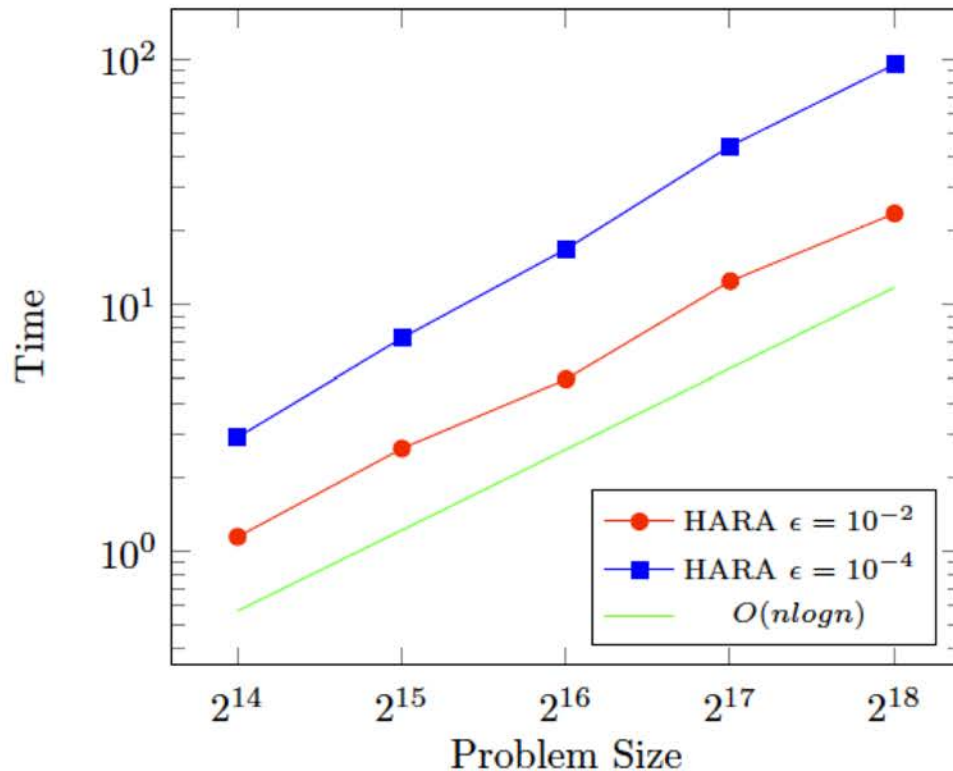
# Hierarchical Adaptive Randomized Approximation (HARA)

- ▶ 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

Matrix sizes from 16K to 256K; accuracies  $10^{-2}$  to  $10^{-4}$   
Time and no. samples to compress

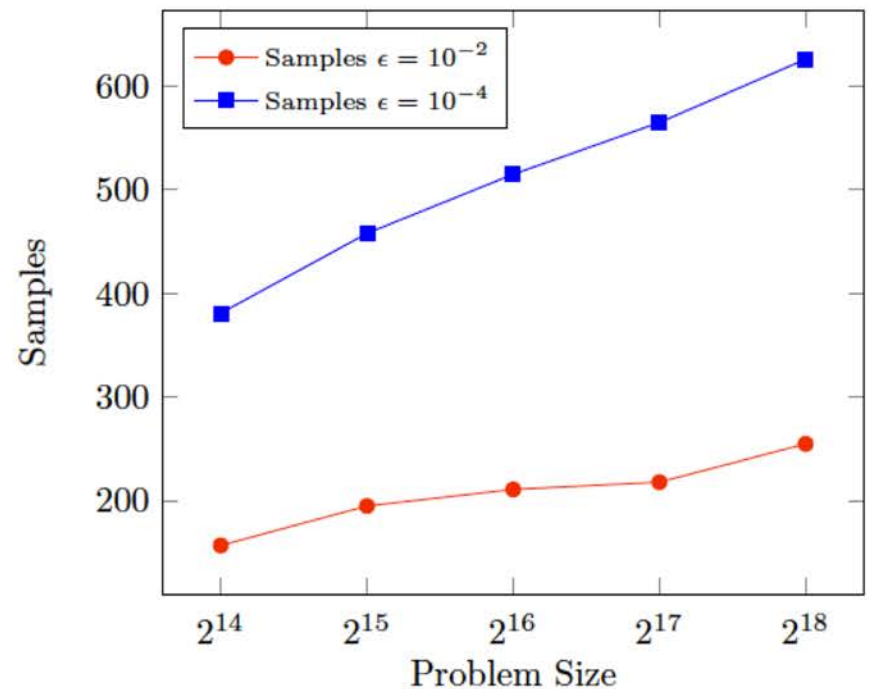
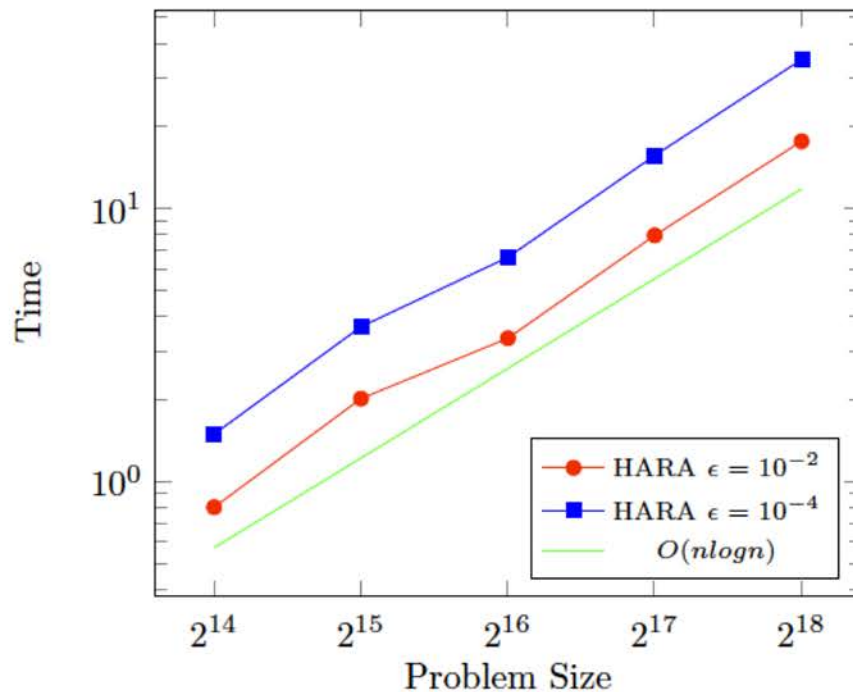


- ▶ spatial covariance matrix reconstructed from HGEMV products

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

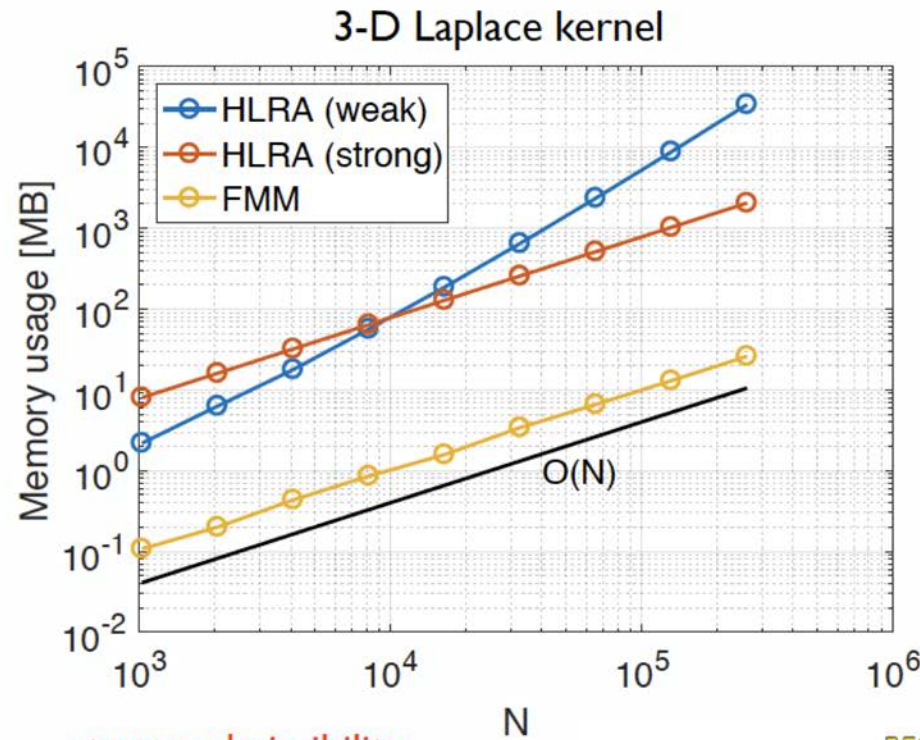
- ▶ can be cast as the problem of constructing an  $\mathcal{H}$ -matrix from matvec operations
- ▶ we can do HGEMV operations efficiently on GPUs
  - HGEMV on multiple vectors is even more efficient
- ▶ HARA construction of product also performed efficiently on the GPU

Fast matvecs  $\Rightarrow$  fast approx inversions with Newton-Schulz

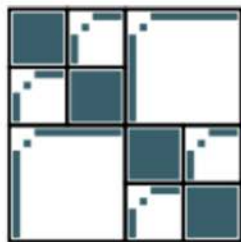




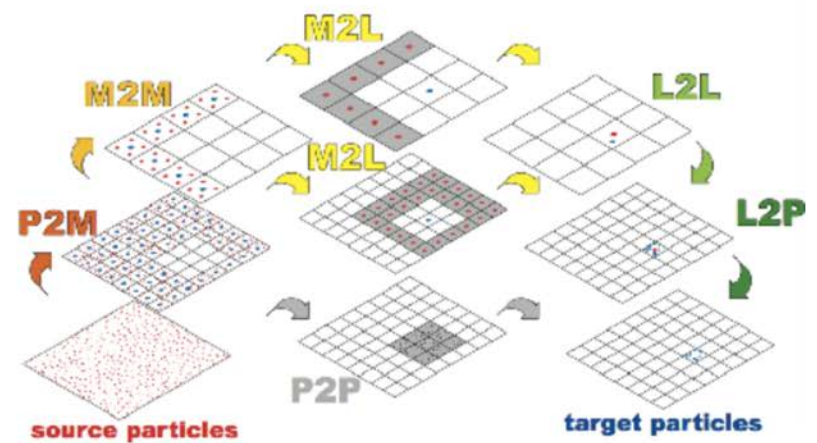
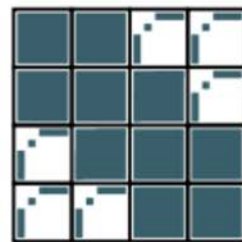
# Memory complexity of FMM vs $\mathcal{H}$ (HLRA)



weak admissibility



strong admissibility

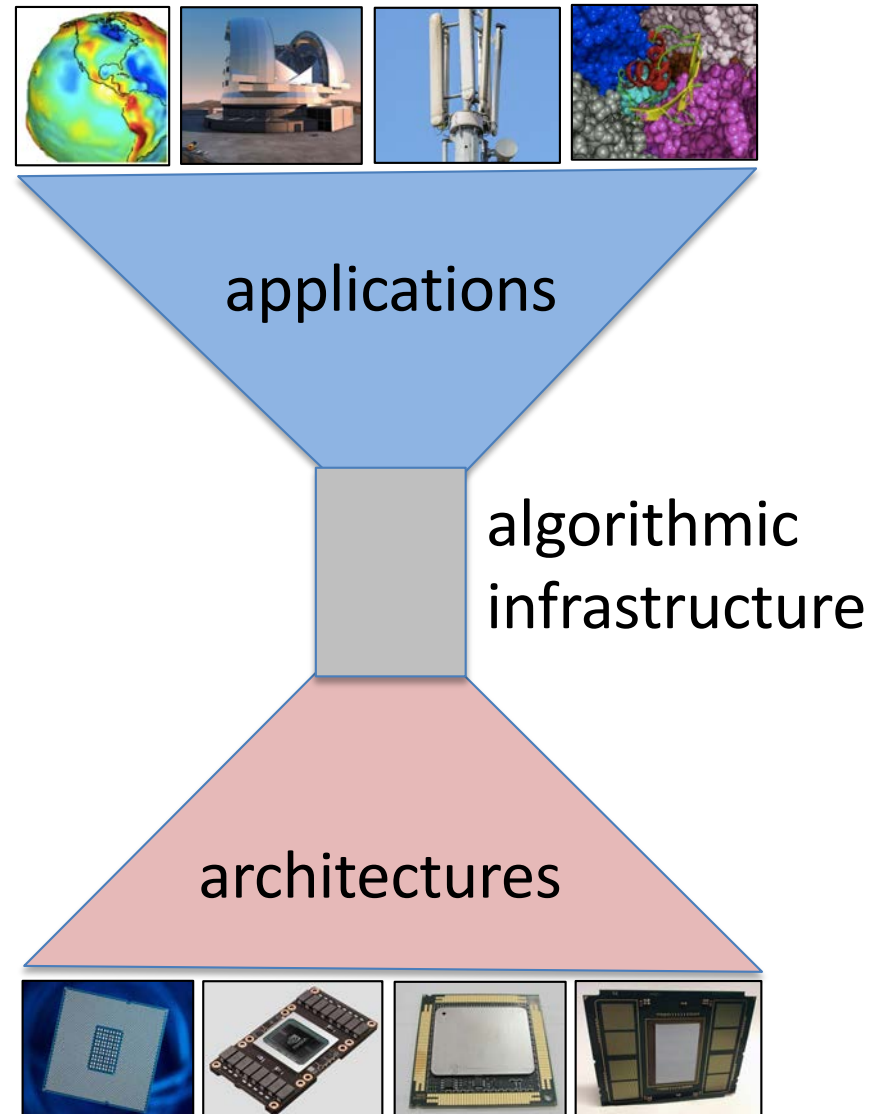


# Conclusions

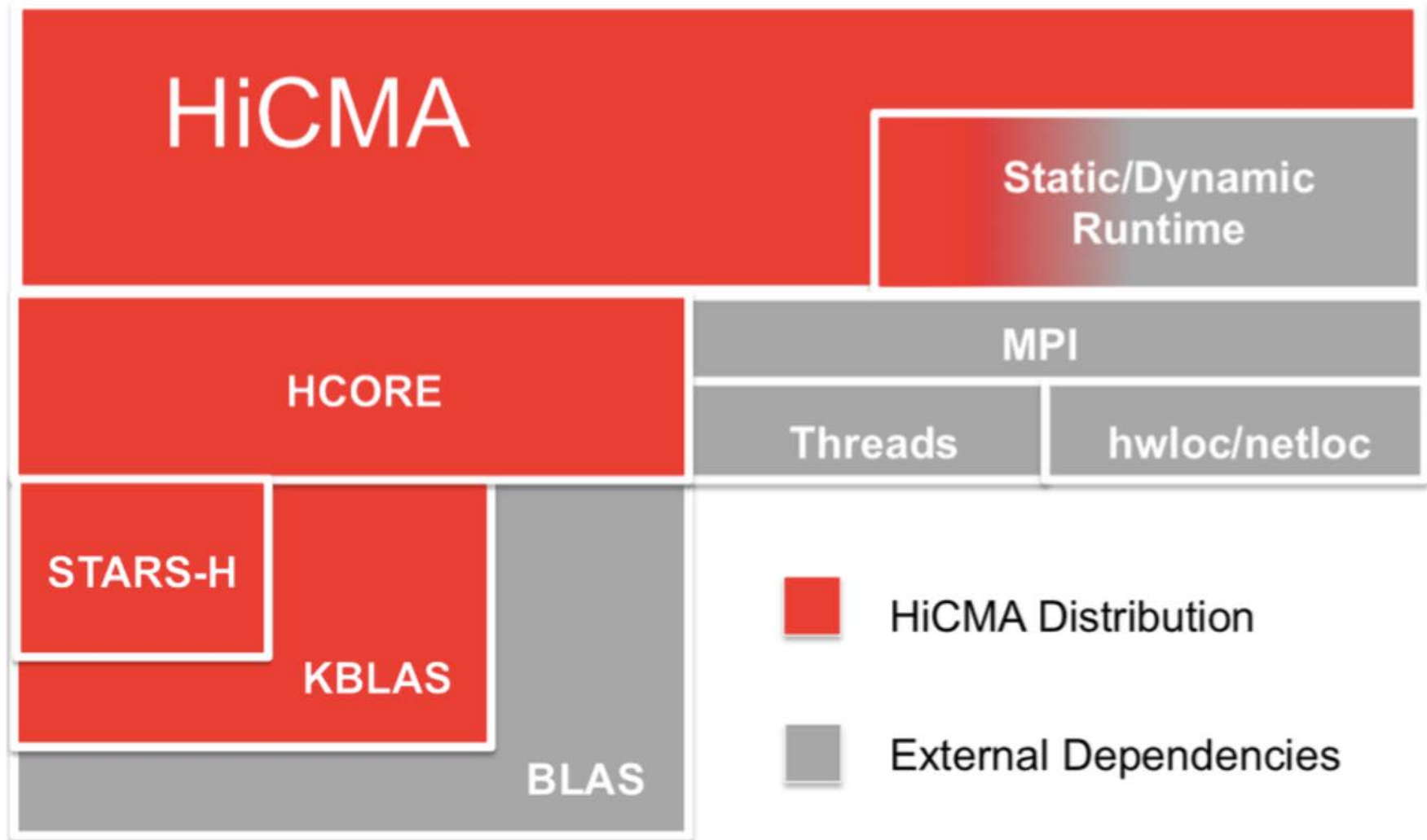
- **Plenty of ideas exist to adapt or substitute for favorite solvers with methods that have:**
    - ◆ **reduced synchrony (in frequency and/or span)**
    - ◆ **higher residence on the memory hierarchy**
    - ◆ **greater SIMT/SIMD-style shared-memory concurrency**
  - **Programming models and runtimes 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](mailto:david.keyes@kaust.edu.sa)