University of Arkansas, Fayetteville

ScholarWorks@UARK

Mathematical Sciences Spring Lecture Series | Mathematical Sciences

4-6-2021

Lecture 02: Tile Low-rank Methods and Applications (w/review)

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

Follow this and additional works at: https://scholarworks.uark.edu/mascsls

 Part of the Algebra Commons, Analysis Commons, Non-linear Dynamics Commons, Numerical Analysis and Computation Commons, and the Ordinary Differential Equations and Applied Dynamics Commons

Citation

Keyes, D. (2021). Lecture 02: Tile Low-rank Methods and Applications (w/review). *Mathematical Sciences Spring Lecture Series.* Retrieved from https://scholarworks.uark.edu/mascsls/4

This Video is brought to you for free and open access by the Mathematical Sciences at ScholarWorks@UARK. It has been accepted for inclusion in Mathematical Sciences Spring Lecture Series by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.

University of Arkansas Department of Mathematical Sciences

46th Spring Lecture Series

David Keyes

Extreme Computing Research Center

King Abdullah University of Science and Technology

5-9 April 2021

Lecture 2

Tile Low-rank Methods and Applications (w/review)

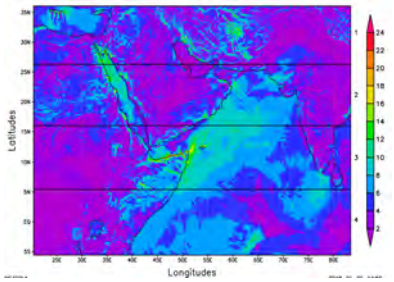
NLA “renaissance” theme, recapped

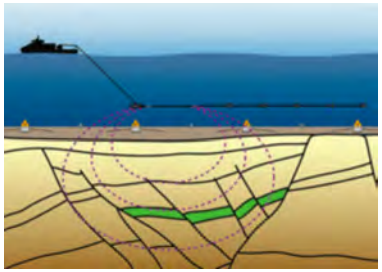
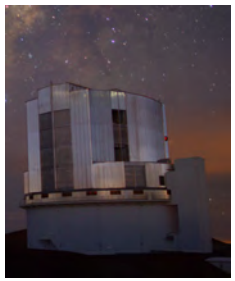
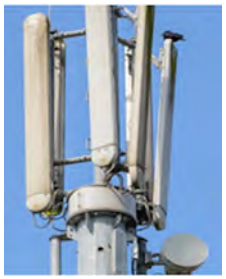
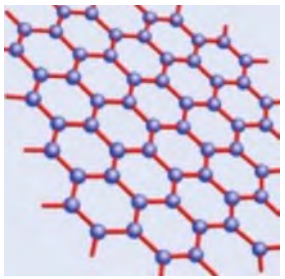
- With controllable trade-offs, many numerical linear algebra operations adapt well for high performance on emerging architectures through
  - higher residency on the memory hierarchy
  - greater SIMT/SIMD-style concurrency
  - reduced synchronization and communication
- Rank-structured matrices, based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

Algorithmic opportunity

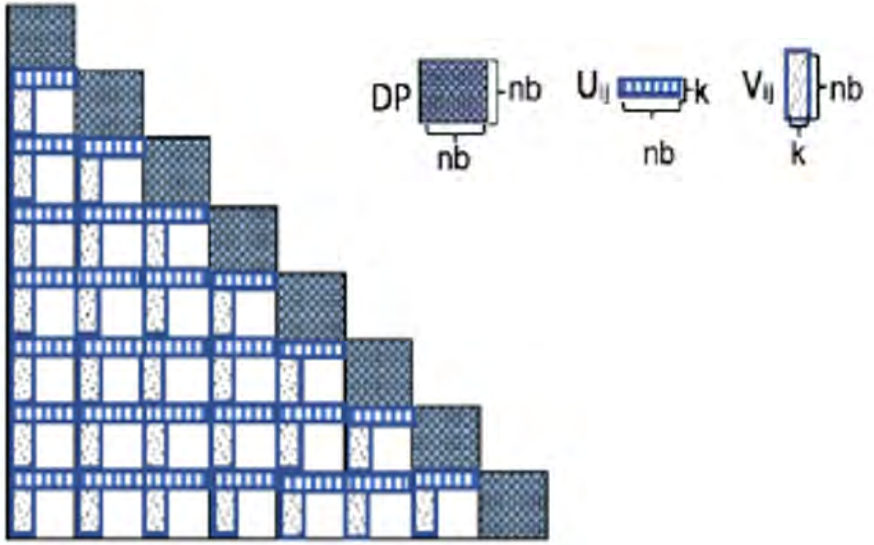
With such new algorithms, we can extend many applications that possess

- 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)

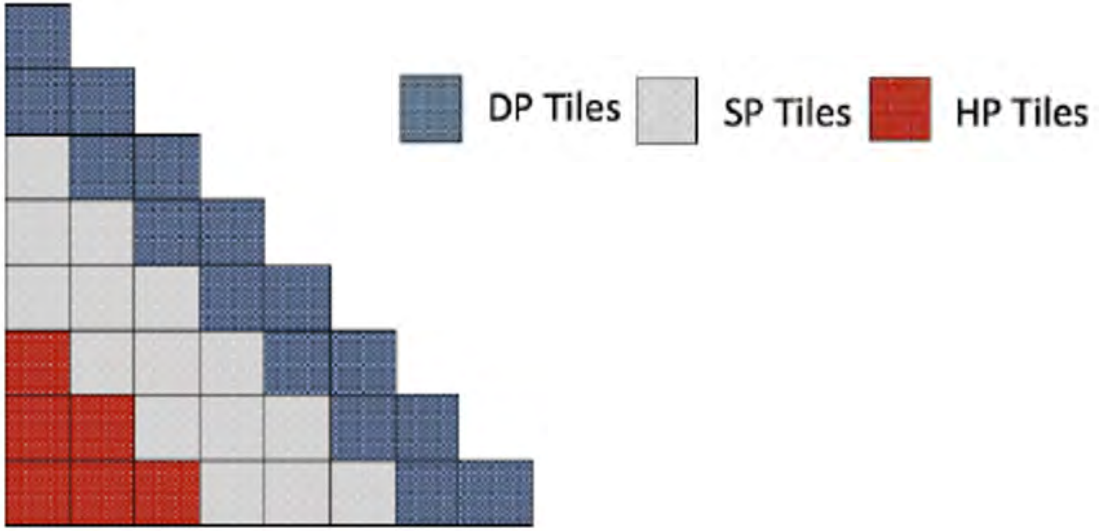



We'll introduce a hierarchy of ranks …



… and a hierarchy of precisions



Today's renaissance in numerical linear algebra

A "renaissance" is a rebirth, revival, or renewed interest

Numerical linear algebra has been vital for over 70 years

- has never endured any "dark ages" or "winter"
- it simply became a bottleneck as problem sizes grew
  - dense problems: cubic complexity, quadratic storage
  - sparse problems: memory bandwidth-constrained

Renewal, in the form of reinvention, comes from these:

- many practical accuracy requirements are less stringent than traditional
- many important applications are "data sparse"
- many high-complexity tasks can be performed in lower precision than traditional
- randomness (typically by random selection) beats systematic coverage

A great time to be looking for research topics

- re-examine important existing algorithms in this light

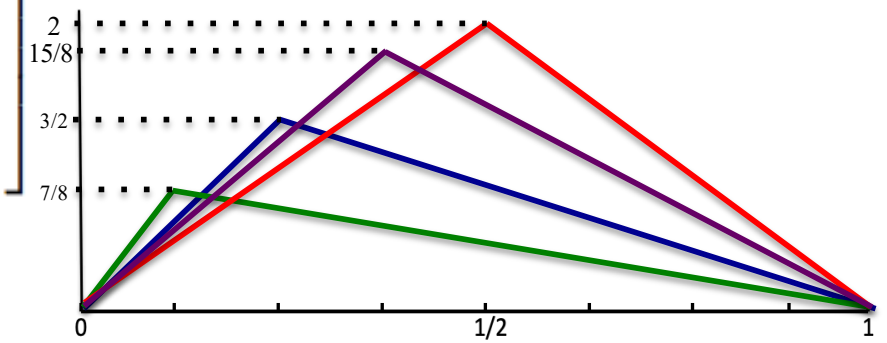
Simple example of data sparsity with the 1D Laplacian

$$A = \begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \leftrightarrow = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}$$

Conformal off-diagonal blocks of A and A^{-1} admit low-rank representation with the same low rank

(Fiedler & Markham, 1986)

$$A^{-1} = \frac{1}{8} \times \begin{bmatrix} 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 12 & 10 & 8 & 6 & 4 & 2 \\ 5 & 10 & 15 & 12 & 9 & 6 & 3 \\ 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 15 & 10 & 5 \\ 2 & 4 & 6 & 8 & 10 & 12 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix} \leftrightarrow = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$$

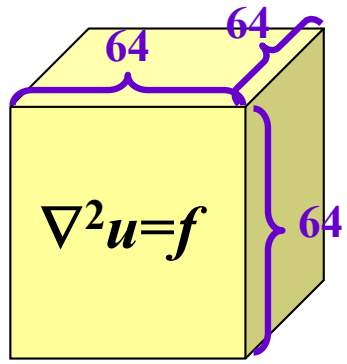


A is sparse but A^{-1} is dense

Solvers have always evolved beneath "$Ax=b$"

- **Advances in algorithmic efficiency rival advances in hardware architecture**
- **Consider Poisson's equation on a cube of size $N=n^3$**

Year	*Method*	*Reference*	*Storage*	*Flops*
1947	GE (banded)	Von Neumann & Goldstine	n^5	n^7
1950	Optimal SOR	Young	n^3	$n^4 \log n$
1971	CG	Reid	n^3	$n^{3.5} \log n$
1984	Full MG	Brandt	n^3	n^3

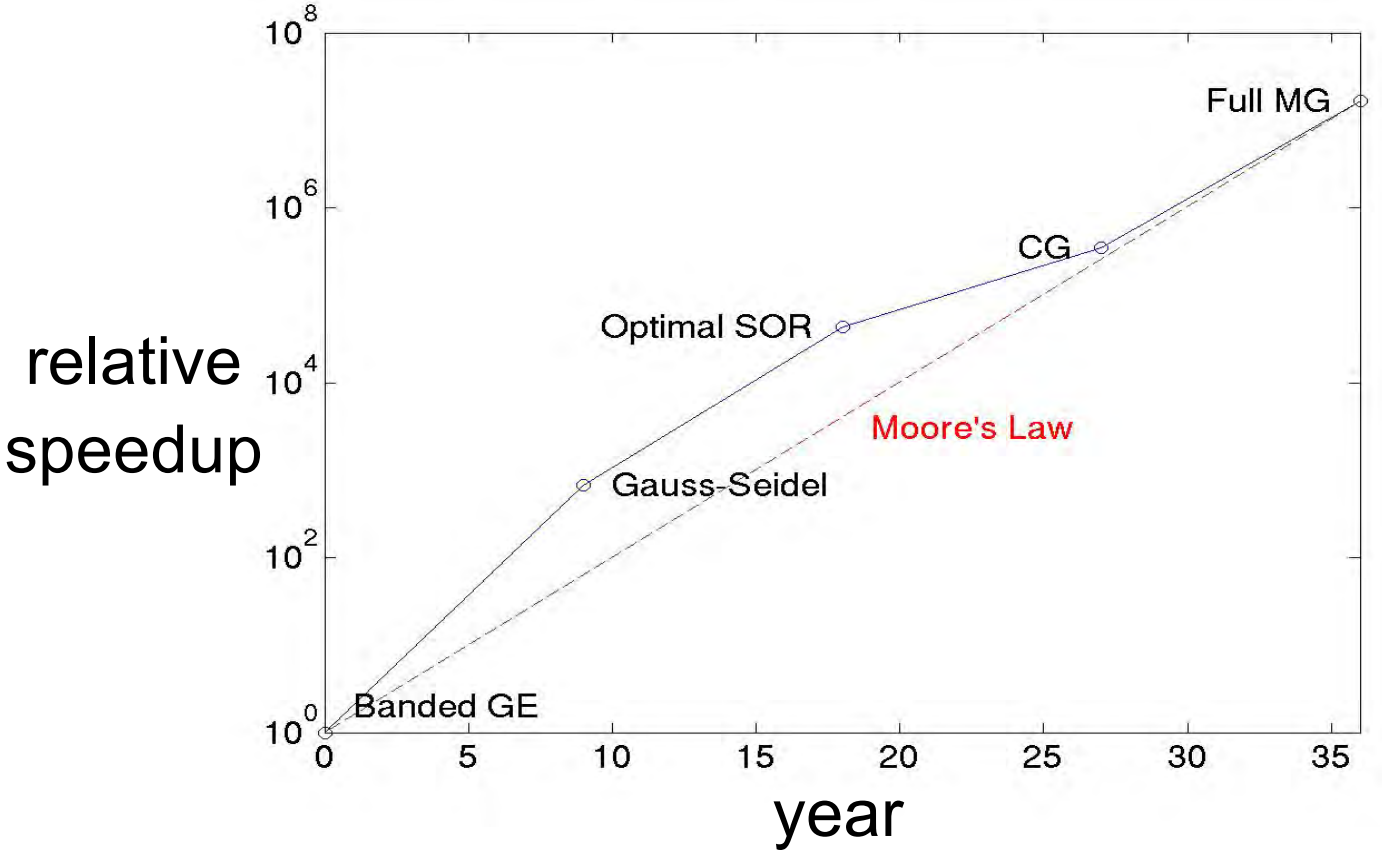


- **If $n=64$, this implies an overall reduction in flops of ~16 million***

*Six months is reduced to 1 second

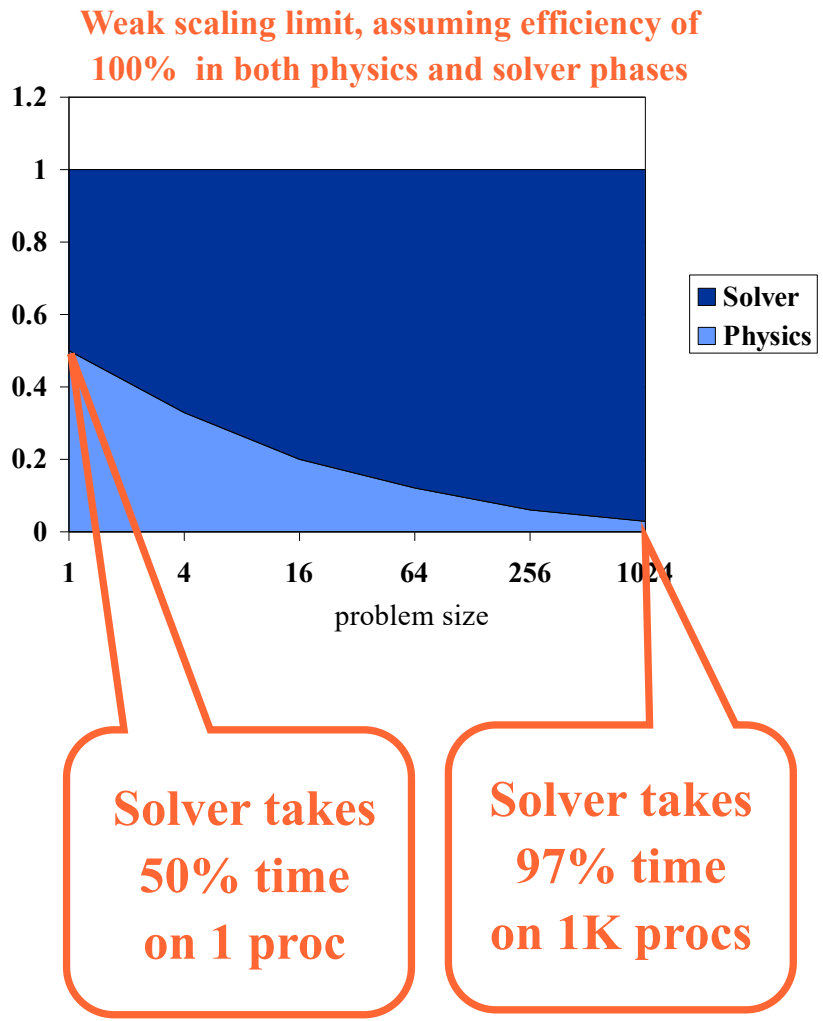
Solver algorithms and Moore's Law

- **This advance took place over a span of about 36 years, or 24 doubling times for Moore's Law**
- **$2^{24} \approx 16$ million $\Rightarrow$ the same as the factor from algorithms alone!**



Large-scale brings focus on complexity

- **Given, for example:**
  - **a "physics" phase that scales as *O(N)***
  - **a "solver" phase that scales as $O(N^{3/2})$**
  - **computation is almost all solver after several doublings**
- **Increasingly, this is felt in the gut or the utility bills of users**



A brief review of beneficial background

In a short lecture series that gets to the *reinvention*, we need to *assume* the invention, namely knowledge of

- **where the problems come from**
- **how they are traditionally solved**

so we undertake next a 15-slide review on

- **types of applications that lead to "bottleneck" linear algebra operations**
- **what results we need to extract**
- **algebraic building blocks**

It could be "homework" tonight to look up what you might be missing

We make convenient assumptions, without the caveats for every possible pathology related to conditioning, etc.

\begin{review}

Primary problem class for my lectures

Symmetric positive definite matrices arising from

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

Schur complements

Suppose p, q are nonnegative integers, and suppose A, B, C, D are respectively $p \times p$, $p \times q$, $q \times p$, and $q \times q$ matrices of complex numbers. Let

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

so that M is a $(p + q) \times (p + q)$ matrix.

If D is invertible, then the **Schur complement** of the block D of the matrix M is the $p \times p$ matrix defined by

$$M/D := A - BD^{-1}C.$$

This eliminates the bottom q unknowns in terms of the remaining p unknowns and is known as “static condensation” in the finite element method.

The Schur complement arising from an SPD matrix is still SPD, but is generally dense even for a sparse original, because D^{-1} is dense.

Schur complements arise when block Gauss elimination is performed to factorize a sparse matrix into its Cholesky (or more generally LU) factors, due to such fill in.

⇒ Condensing sparse finite element, finite difference, and finite volume matrices leads to dense blocks that are often data sparse

c/o wikipedia

Integral equations

The most basic type of integral equation is called a *Fredholm equation of the first type*,

$$f(x) = \int_a^b K(x,t)\,\varphi(t)\,dt.$$

One method to solve numerically requires discretizing variables and replacing integral by a quadrature rule

$$\sum_{j=1}^{n} w_j K\left(s_i, t_j\right) u(t_j) = f(s_i), \qquad i = 0, 1, \cdots, n.$$

This is $A\,u = f$, where A is n x n. If the kernel $K(s,t)$ depends only on the distance between s and t and not on either separately, it is a "displacement kernel" and invariant upon interchange of argument, so A is a symmetric matrix and generally dense.

For a *second type Fredholm* integral equation, the unknown u also appears outside, giving diagonal weight to $A = I + K$.

⇒ Integral equations of interest are often strongly diagonally dominant and data-sparse (leading, e.g., to fast multipole methods)

c/o wikipedia

Hessian matrices

Suppose $f\colon \mathbb{R}^n \to \mathbb{R}$ is a function taking as input a vector $\mathbf{x} \in \mathbb{R}^n$ and outputting a scalar $f(\mathbf{x}) \in \mathbb{R}$. If all second partial derivatives of f exist and are continuous over the domain of the function, then the Hessian matrix $\mathbf{H}$ of f is a square $n \times n$ matrix, usually defined and arranged as follows:

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

, symmetric by interchange of partial derivatives.

Hessian matrices are used in large-scale optimization problems within Newton-type methods because they are the coefficient of the quadratic term of a local Taylor expansion of a function. That is,

$$y = f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\,\Delta\mathbf{x}^{\mathrm{T}}\mathbf{H}(\mathbf{x})\,\Delta\mathbf{x}$$

where ∇f is the gradient $(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$. Computing and storing the full Hessian matrix takes $\Theta(n^2)$ memory, which is infeasible for high-dimensional functions such as the loss functions of neural nets, conditional random fields, and other statistical models with large numbers of parameters. For such situations, truncated-Newton and quasi-Newton algorithms have been developed.

⇒ Hessian matrices require approximation for optimization

Often sum of compact and low-rank and often hierarchically low rank

c/o wikipedia

Fractional derivatives

A fairly natural question to ask is whether there exists a linear operator H, or half-derivative, such that

$$H^2 f(x) = Df(x) = \frac{d}{dx} f(x) = f'(x)\,.$$

It turns out that there is such an operator, and indeed for any $a > 0$, there exists an operator P such that

$$\left(P^a f\right)(x) = f'(x),$$

or to put it another way, the definition of $\frac{d^n y}{dx^n}$ can be extended to all real values of n.

There is the Caputo fractional derivative defined as:

$$D^\nu f(t) = \frac{1}{\Gamma(n-\nu)} \int_0^t (t-u)^{(n-\nu-1)} f^{(n)}(u) du \qquad (n-1) < \nu < n$$

where n is the smallest integer larger than the fractional power ν . When discretized, the operator D^ν is symmetric and dense except when ν is an integer. For instance, if $n = 2$, we get the discrete 1D Laplacian matrix, which is sparse, with trivially low-rank off-diagonal blocks. For fractional values of ν we lose sparsity, *but we still have data sparsity*. The off-diagonal blocks have low rank.

⇒ Discretized fractional derivative operators – which are currently extremely hot – are natural candidates for hierarchically low-rank approximation, which is crucial in dimensions higher than one

c/o wikipedia

Covariance matrices

Remember that for a scalar-valued random variable X

$$\sigma_X^2 = \operatorname{var}(X) = \operatorname{E}[(X - \operatorname{E}[X])^2] = \operatorname{E}[(X - \operatorname{E}[X]) \cdot (X - \operatorname{E}[X])].$$

If the entries in the column vector

$$\mathbf{X} = (X_1, X_2, \ldots, X_n)^{\mathrm{T}}$$

are random variables, each with finite variance and expected value, then the covariance matrix $\mathrm{K}_{\mathbf{XX}}$ is the matrix whose (i, j) entry is the covariance[1]:p. 177

$$\mathrm{K}_{X_i X_j} = \operatorname{cov}[X_i, X_j] = \operatorname{E}[(X_i - \operatorname{E}[X_i])(X_j - \operatorname{E}[X_j])]$$

where the operator E denotes the expected value (mean) of its argument.

In other words,

$$\mathrm{K}_{\mathbf{XX}} = \begin{bmatrix} \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_n - \operatorname{E}[X_n])] \\ \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_n - \operatorname{E}[X_n])] \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_n - \operatorname{E}[X_n])] \end{bmatrix}$$

This matrix is the generalization of the scalar variance to n dimensions.

The entries on the diagonal are the variances of each individual element of $\mathbf{X}$.

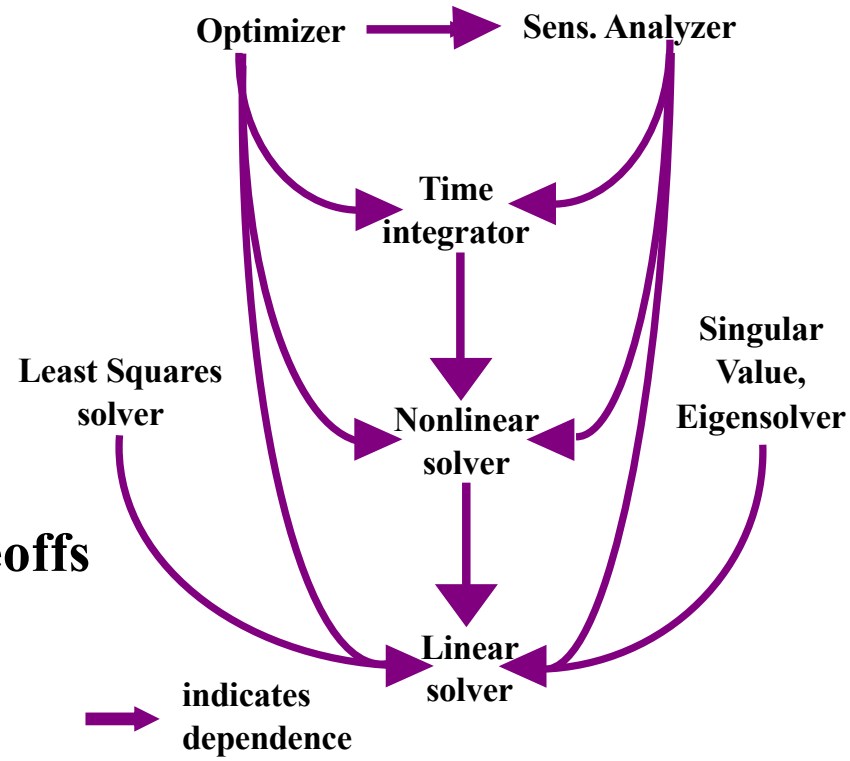
⇒ Covariance matrices are symmetric and positive semi-definite with elements that generally decay with distance by causality, and data sparse when properly ordered (e.g., space-filling curves)

c/o wikipedia

Solvers and their nestedness

To "go big" and achieve the potential of emerging architectures for scientific applications, we need implementations of fast

- **linear and least squares solvers**
- **eigensolvers & singular value solvers**
- **nonlinear and optimization solvers**
- **time integrators & sensitivity solvers**
- **stencil solvers**



that

- **offer tunable accuracy-time-space tradeoffs**
- **exploit data sparsity**
- **exploit hierarchy of precisions**
- ***may* require more flops but complete earlier, thanks to more concurrency or less communication or synchronization**
- **are energy efficient**

A week of solvers ☺

	Mon, April 5th	Tue, April 6th	Wed, April 7th	Thu, April 8th	Fri, April 9th
9:00am	**David Keyes** Universals of Extreme Scale Computing	**David Keyes** Tile Low Rank Methods & Applications	**David Keyes** Hierarchically Low Rank Methods & Applications	**David Keyes** Spatial Statistics with HRL, TLR & Mixed Precision	**David Keyes** Convergence of Big Data and Extreme Computing
10:15am	**Ulrike Yang** Algebraic Multigrid Methods	**Laura Grigori** Multilevel and Randomized Methods	**Xiao-Chuan Cai** Nonlinear Preconditioning Methods	**Zhaojun Bai** Eigen Decomposition Methods	**Rio Yokota** Low-rank & Kroenecker Methods
11:30am	**Edmond Chow** Preconditioned Iterative Methods	**Jack Dongarra** Dense Linear Methods	**Carol Woodward** Time Integration Methods	**Sherry Li** Low-rank Factorization Methods	**Ilse Ipsen** Randomized Least Squares Methods
6:00pm	**Ann Almgren** Mathematics for High Performance		**AWM Panel** Women in STEM		

Linear, least squares, eigen & singular value solvers of "direct" type

Since Householder (1954), direct matrix computations have been built on a foundation of *decompositions* or *factorizations:*

- $A = LL^T = \text{Ł}D\text{Ł}^T$ **(Cholesky, L, Ł lower tri)**
- $A = LU$ **(LU (pivots not shown), L lower tri, U upper tri)**
- $A = QR$ **(QR, Q orthogonal, R right tri)**
- $A = UH$ **(Polar, U unitary, H HPD if A has full rank)**
- $A = UTU^H$ **(Schur, U unitary, T upper tri)**
- $A = V\Lambda V^T$ **(Eigen, V orthogonal if A symmetric, Λ diagonal,)**
- $A = U\boldsymbol{\Sigma}V^T$ **(Singular, U, V unitary, $\boldsymbol{\Sigma}$ diagonal)**

Mathematically, these decompositions were all known by 1910.

If A is $n \times n$, dense, and full rank, their complexities are all $O(n^3)$.

In factored form, determinants, inverses for repeat solves, and low-rank updates are comparatively inexpensive.

Linear solvers of iterative type

Linear iterative methods are primarily useful for sparse problems, because their inner loops typically contain matrix-vector multiplications and (approximate) preconditioning solves.

These typically exploit special spectral, mathematical, or geometric structure, e.g., coming from a PDE:

- **stationary iterative methods**
- **Chebyshev iterative methods**
- **Krylov subspace iterative methods**
- **multilevel iterative methods**

and may thus achieve a complexity as low as $O(n)$.

(Sparse problems are also solved by direct methods, especially those coming from PDE discretizations, in less than $O(n^3)$ by exploiting geometric structure, e.g., nested dissection.)

Nonlinear solvers

Nonlinear solvers for $f(x) = 0$ are invariably iterative.

For high-dimensional problems, they are of fixed-point type

$$x_{k+1} = g(x_k)$$

or accelerated (e.g., Anderson), or are variants of Newton's method

$$x_{k+1} = x_k - J(x_k)^{-1} f(x_k)$$

where the inverse of the Jacobian is effected by incomplete application of an iterative linear method, often on an approximate Jacobian.

Jacobian-free Newton-Krylov methods that access $J(x_k)$ only by Fréchet derivatives

$$J(u)v \approx \frac{1}{\varepsilon}[F(u+\varepsilon v) - F(u)]$$

are popular.

Newton's method may be nonlinearly preconditioned and is usually "globalized" by line-search, trust-region, or physics-informed "continuation" methods.

Continuation can be in the form of problem resolution (mesh), problem parameters (Reynolds), or artificial means.

Time integration solvers

Integrators for $f(x_t,x) = 0$ with important special case $x_t + F(x) = 0$ are usually semi-discretized in space, and sometimes fed to an *explicit* ODE integrator.

When arising in parabolic, differential algebraic, or mixed contexts, they are typically *implicitly* discretized in time because the temporal stability bound is typically more severe than the accuracy bound.

Hyperbolic systems admitting wave behavior tend to have accuracy and stability limits of the same order and may be solved explicitly.

However, in multi-wavespeed systems, fast wave phenomena may be unimportant to the dynamics of interest, so implicit methods are used.

As such, time integrators tend to reduce on each timestep to a nonlinear solve, which, in turn through Newton reduces to a linear solve.

Time-integration methods may carry extra sensitivity variables along, which provide a form of uncertainty quantification.

Optimization solvers

Large-scale optimization problems typically come with PDE constraints and piggyback on implicit nonlinear solvers.

Consider the rootfinding problem derived from the necessary gradient conditions for minimization, with design variables u

- **Objective** $\min_u \Phi(x,u)\,;\Phi \in \Re$
- **Constraints** $f(x,u)=0\,;x\in\Re^N;u\in\Re^M;f\in\Re^N$
- **Lagrangian** $\Phi(x,u)+\lambda^T f(x,u)\,;\lambda\in\Re^N$
- **Form the gradient of the Lagrangian with respect to each of x, u, and λ to get a large coupled root-finding problem:**

$$\Phi_x(x,u)+\lambda^T f_x(x,u)=0$$
$$\Phi_u(x,u)+\lambda^T f_u(x,u)=0$$
$$f(x,u)=0$$

Newton Reduced Sequential Quadratic Programming (RSQP)

- **Applying Newton's method leads to the KKT system for states x , designs u , and multipliers λ**

$$\begin{bmatrix} W_{xx} & W_{ux}^T & J_x^T \\ W_{ux} & W_{uu} & J_u^T \\ J_x & J_u & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta u \\ \delta \lambda \end{bmatrix} = - \begin{bmatrix} g_x \\ g_u \\ f \end{bmatrix}$$

- **Newton Reduced SQP solves the Schur complement system $H\,\delta u = g$, where H is the reduced Hessian**

$$H = W_{uu} - J_u^T J_x^{-T} W_{ux}^T - (J_u^T J_x^{-T} W_{xx} - W_{ux}) J_x^{-1} J_u$$

$$g = -g_u + J_u^T J_x^{-T} g_x - (J_u^T J_x^{-T} W_{xx} - W_{ux}) J_x^{-1} f$$

- **Then**

$$J_x \delta x = -f - J_u \delta u$$

$$J_x^T \delta \lambda = -g_x - W_{xx} \delta x - W_{ux}^T \delta u$$

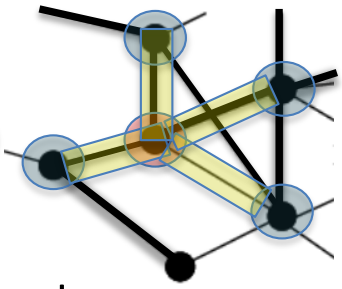
Stencil ops: 4 very different types of $y=Ax$

Explicit positions
Explicit values

store no positions

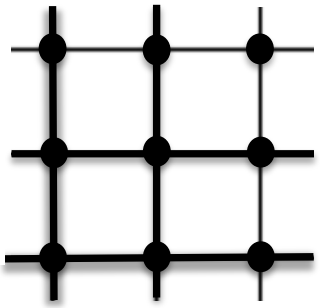
Implicit positions
Explicit values

$$y_i = d_i x_i - \sum_{j \in S_i} c_{ij} x_j$$



general sparse matvec

$$y_i = d_i x_i - \sum_{j=i+K} c_{ij} x_j$$



diffusion on Cartesian mesh, $\nabla \cdot A \nabla$

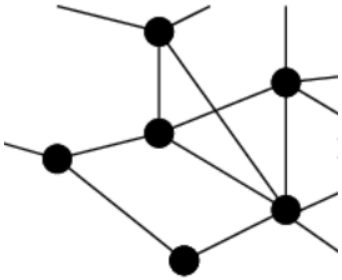
store no values

store neither

Explicit positions
Implicit values

$$y_i = d_i x_i - \sum_{j \in S_i} x_j$$

$$d_i = |S_i|$$

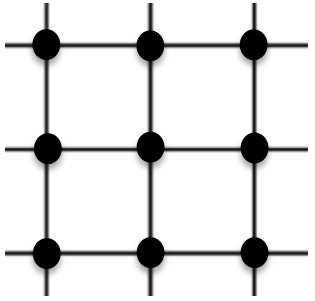


graph Laplacian

Implicit positions
Implicit values

$$y_i = d x_i - \sum_{j=i+K} x_j$$

$$d = |K|$$



Laplacian on Cartesian mesh, ∇^2

e.g., for 1D Laplacian, $K = \{-1, +1\}$ and $d = 2$

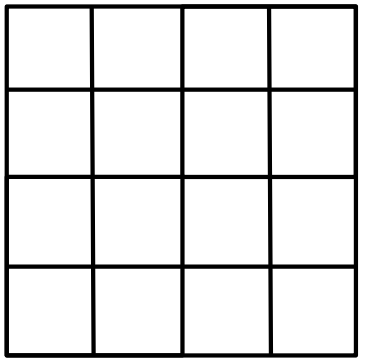
\end{review}

Rank-structured operators

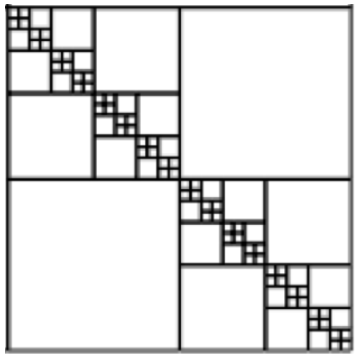
- **Advantages**
  - **shrink memory footprints to live higher on the memory hierarchy**
    - **higher means quick access (↑ arithmetic intensity)**
  - **reduce operation counts**
  - **tune work to accuracy requirements**
    - **e.g., preconditioner versus solver**
- **Disadvantages**
  - **pay cost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

Data sparsity from rank-structured matrices*

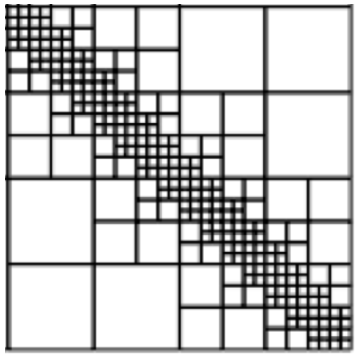
- **Tile low rank (TLR)**
  - all blocks at a single level of subdivision (could in principle vary in size)
- **Hierarchically low rank (HLR)**
  - blocks are left at various levels upon recursive subdivision
  - weak and strong "admissibility" variants
- **HLR more than two decades old**
  - Hackbusch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



TLR



HLR
weakly admissible



HLR
strongly admissible

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

Reduce memory footprint and operation complexity with low rank

- **Replace dense blocks with reduced rank representations, whether "born dense" or as arising during matrix operations**
  - **use high accuracy (high rank) to build "exact" solvers**
  - **use low accuracy (low rank) to build preconditioners**
- **Consider hardware parameters in tuning block sizes and maximum rank parameters**
  - **e.g., cache sizes, warp sizes**
- **Use randomized SVD** (Halko, Martinsson & Tropp, 2009) **to form low-rank blocks**
  - **a flop-intensive GEMM-based flat algorithm**
- **Implement in "batches" of leaf blocks**
  - **flattening trees in the case of HLR**

Complexities of rank-structured factorization

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

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

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

Rank-structure also relevant to sparse problems

Classical factorizations fill in with elimination

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

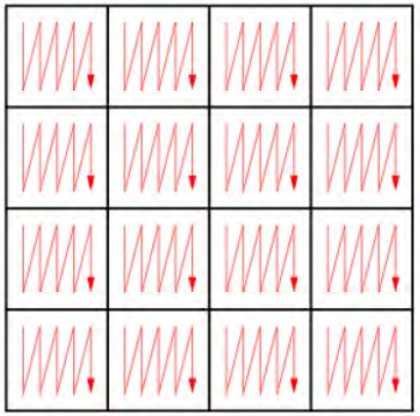
- **Classical nested dissection generally requires $O(N^2)$ operations**
- **Tile low-rank can yield $O(N^{4/3})$ operations**

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

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

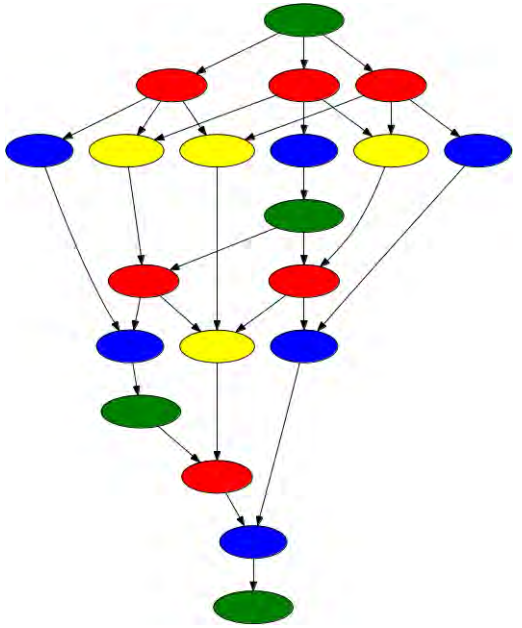
Tile Low Rank begins with tile algorithms

- **PLASMA, Chameleon, and FLAME implementations**
- **Dense matrix is divided into tiles**
- **Remove artifactual synchronization by bringing task parallelism to the fore**



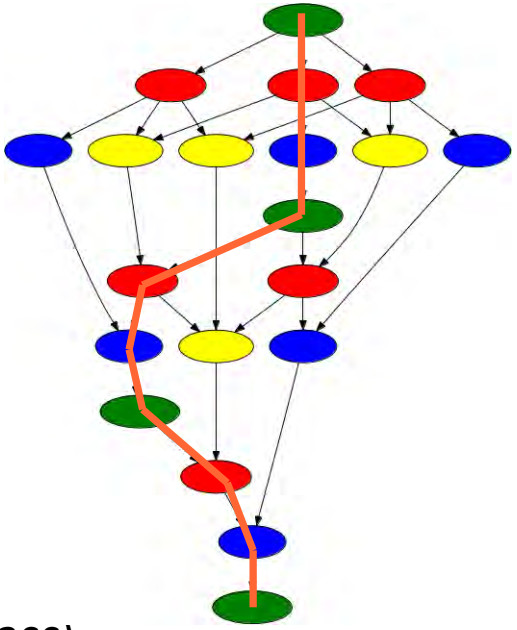
Tile Cholesky of a 4x4 matrix

```
for k = 1 to T do
    POTRF(D(k,k))
    for i = k+1 to T do
        TRSM(V(i,k), D(k,k))
    for j = k+1 to T do
        SYRK(D(j,j), U(j,k), V(j,k))
        for i = j+1 to T do
            GEMM(U(i,k), V(i,k), U(j,k), V(j,k), U(i,j), V(i,j), acc)
```



Tile Cholesky factorization (dense)

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



A serial and incompressible critical path of TLR Cholesky:

(T-1) * (POTRF + TRSM + SYRK) + POTRF

Kernel	Dense Cholesky
POTRF	1/3 * B^3
TRSM	B^3
SYRK/LR_SYRK	B^3
GEMM/LR_GEMM	2 * B^3
Total	O(N^3)

The block algorithm has T steps down the diagonal, each of which is *O*(T^2) for updates, and the block update operations are each *O*(B^3), so there is no reduction in complexity,

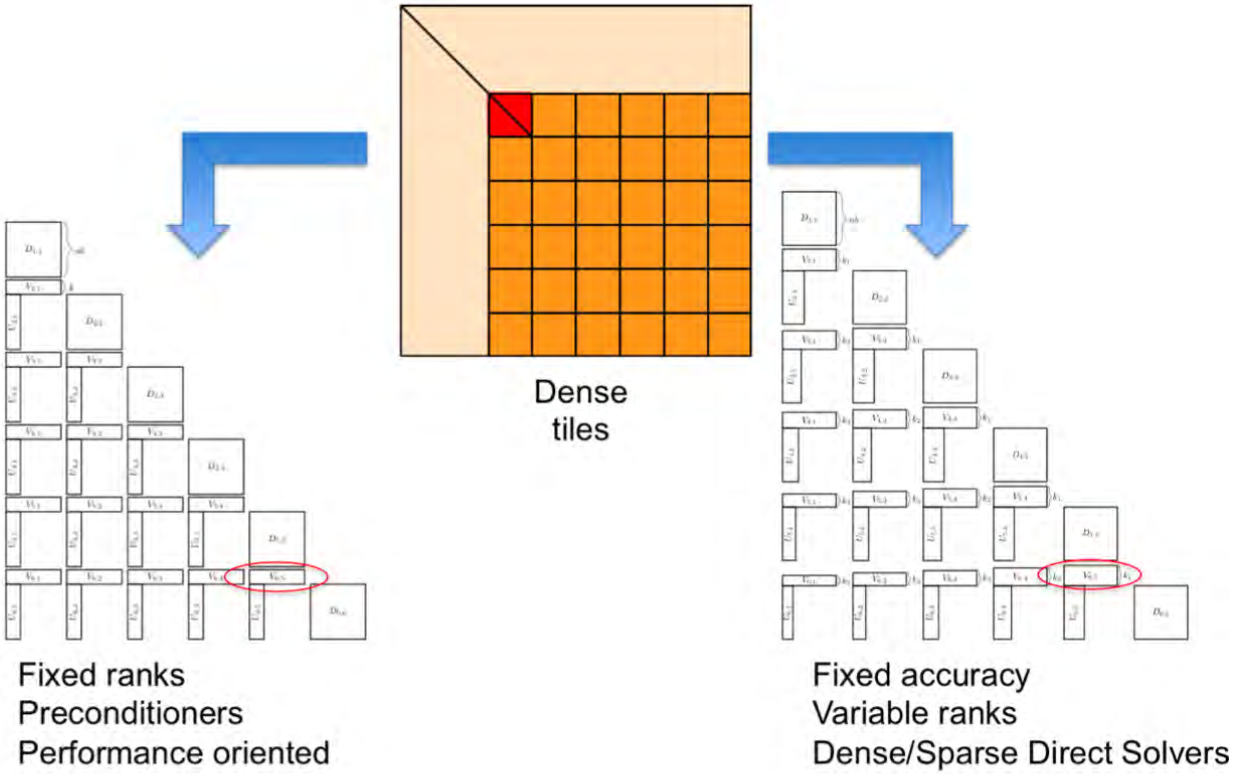
T^3 * B^3 = N^3 ,

just better scheduling.

Dynamic runtime systems

- Operate directly from the sequential code
- Ensure that data dependencies are not violated
- Schedule the tasks across appropriate available hardware resources
- Optimize memory placement for nonuniform access
- Enhance software productivity by abstracting the hardware
- Examples (all avail for shared memory, distributed memory, and GPUs):
  - OpmSs
    - BSC, Barcelona
    - Pragma-based, extending OpenMP to asynchronous execution
  - ParSEC
    - ICL, University of Tennessee
    - Parallel runtime scheduling and execution control
  - StarPU
    - INRIA, Bordeaux
    - Unified runtime system for heterogeneous multicore architecture

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

TLR Cholesky factorization

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

Swapping out a B for a rank, where rank ≪ B, provides an easy win.

However, one must pay the cost of initial compression and of recompression after updates.

Kernel	Dense Cholesky	TLR Cholesky
POTRF	1/3 * B^3	1/3 * B^3
TRSM	B^3	B^2 * rank
SYRK/LR_SYRK	B^3	2 * B^2 * rank + 4 * B * rank^2
GEMM/LR_GEMM	2 * B^3	36 * B * rank^2 + 157*rank^3
Total	O(N^3)	O(N^2 * rank)

Low rank approximation for off-diag tiles

There are several means of forming data sparse representations of the amenable off-diagonal blocks

- **standard SVD:** $O(n^3)$**, too expensive for initial and especially for repeated compressions after manipulations**
- **randomized SVD*** (Halko et al, 2011)**:** $O(n^2 \log k)$ **for rank** k**, requires only a small number of passes over the data, saving over the SVD in memory accesses as well as operations**
- **adaptive cross approximation (ACA)*** (Bebendorf, 2000)**:** $O(k^2 n \log n)$**, motivated by integral equation kernels**
- **matrix skeletonization (representing a matrix by a representative collection of row and columns), such as CUR, sketching, or interpolatory decompositions**

*** RSVD and ACA routines offered in KBLAS at** github.com/ecrc/kblas-gpu

Jacobi-SVD also offered in KBLAS for small matrices in batched mode

Geospatial statistics motivation

"Increasing amounts of data are being
by remote sensing instruments and nun
while techniques to handle millions of o
historically lagged behind…

Computational implementations that w
irregularly-spaced observations are still

- Dorit Hammerling, N

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

Traditional approaches:

- **Global low rank**
- **Zero outer diagonals**

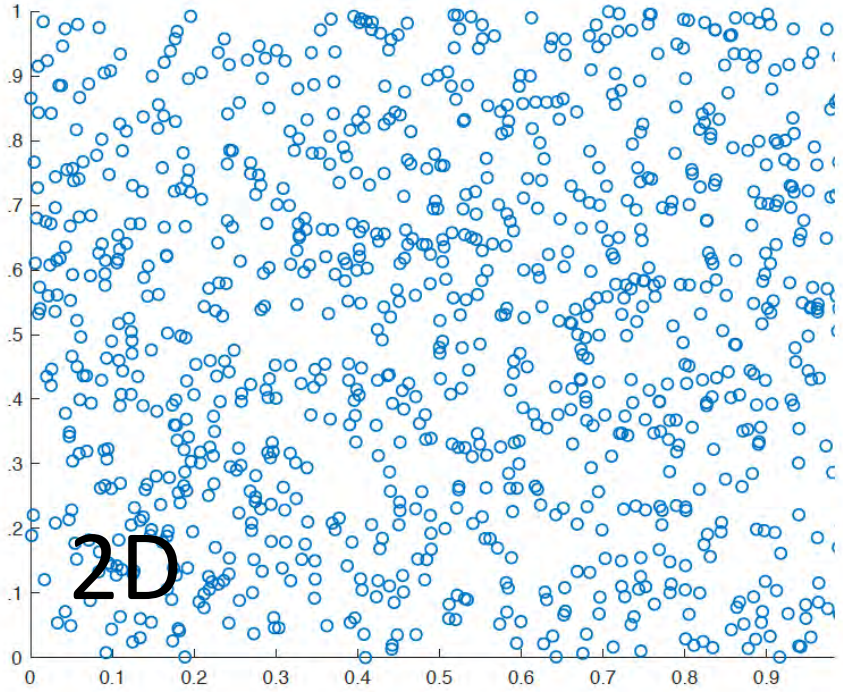
Better approaches:

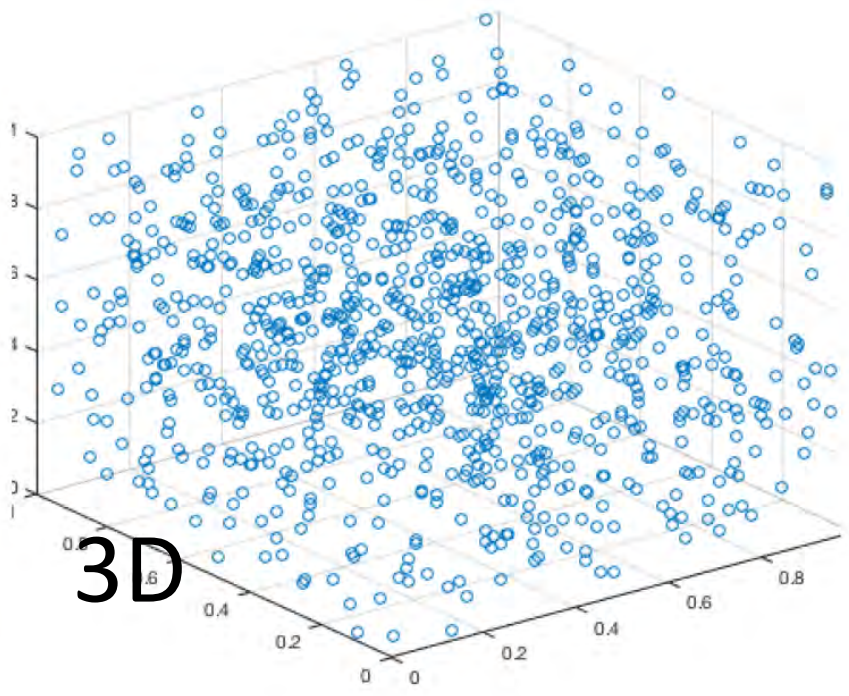
- **Hierarchical low rank**
- **Reduced precision outer diagonals**

Geospatial statistics applications

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

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





Large dense symmetric systems arise as covariance matrices in spatial statistics

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

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

inverse

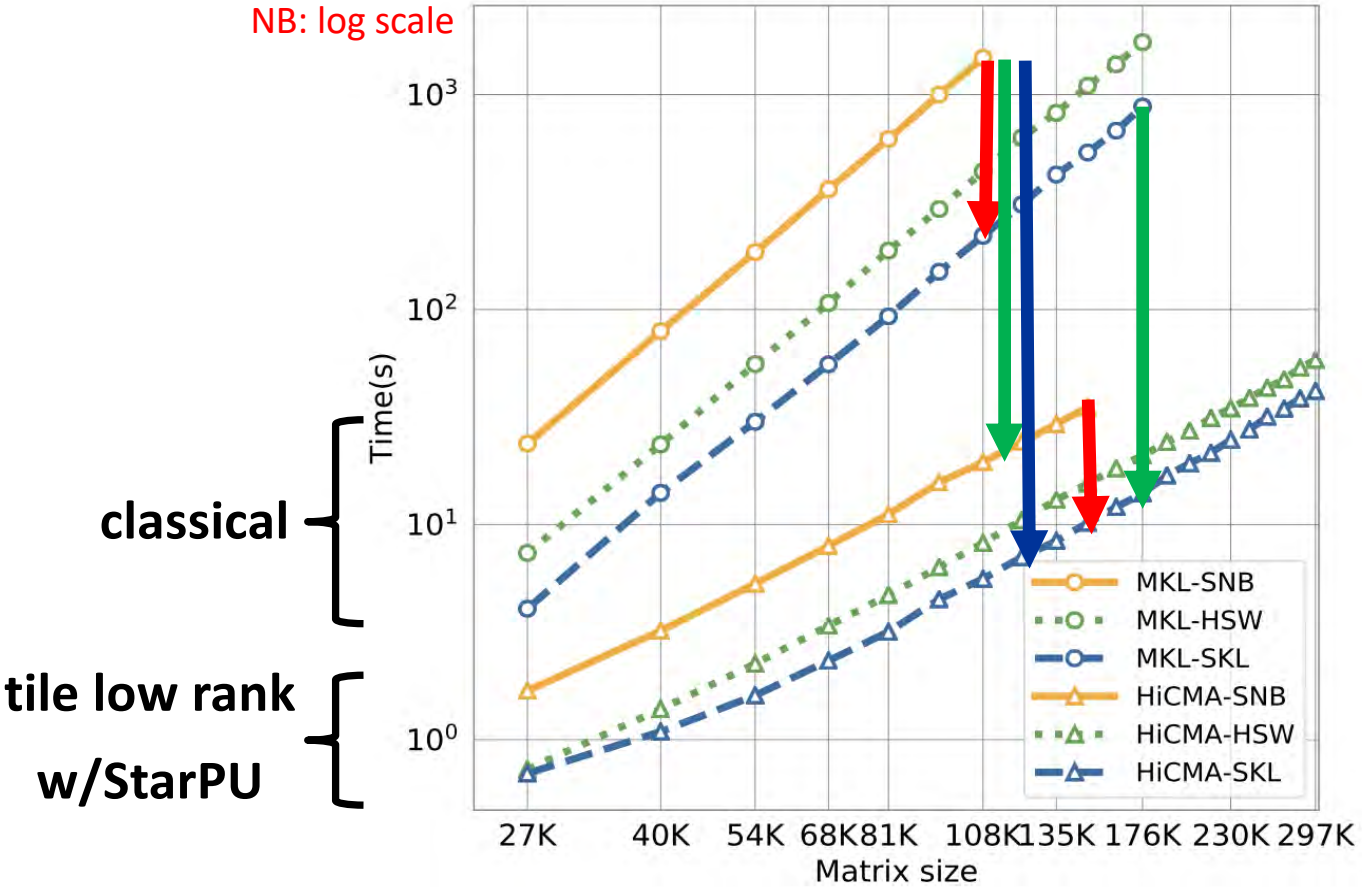
determinant

- Solve Σ^{-1} and determinant $|\Sigma|$ and depend upon Cholesky, dominated by DPOTRF factorization routine (next slides)

TLR vs. Intel MKL on shared memory

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

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



Red arrows: speedups from hardware, same algorithm

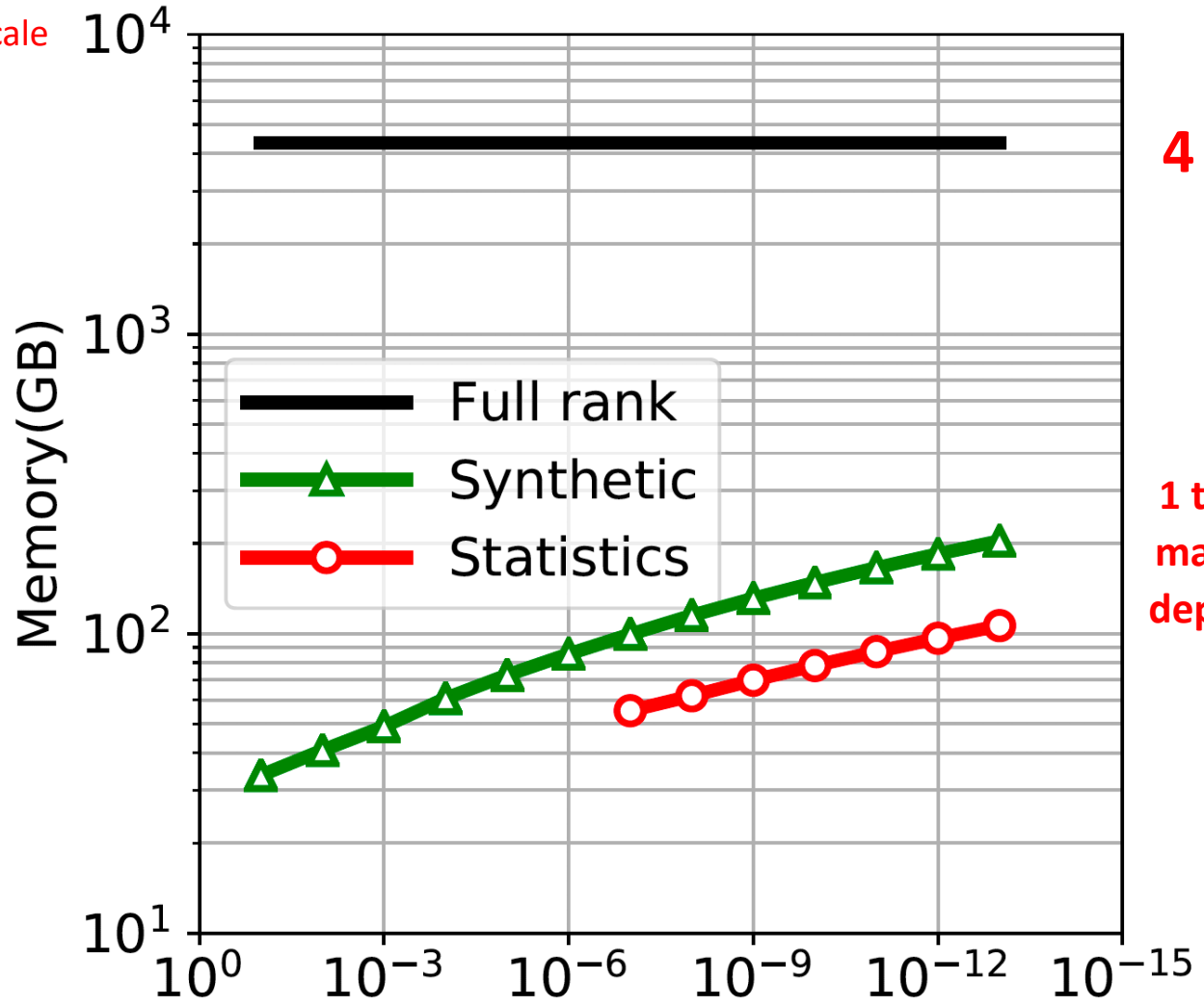
Green arrows: speedups from algorithm, same hardware

Blue arrow: from both

Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations,* Euro-Par 2018

Memory footprint for DP matrix of size 1M

NB: log scale



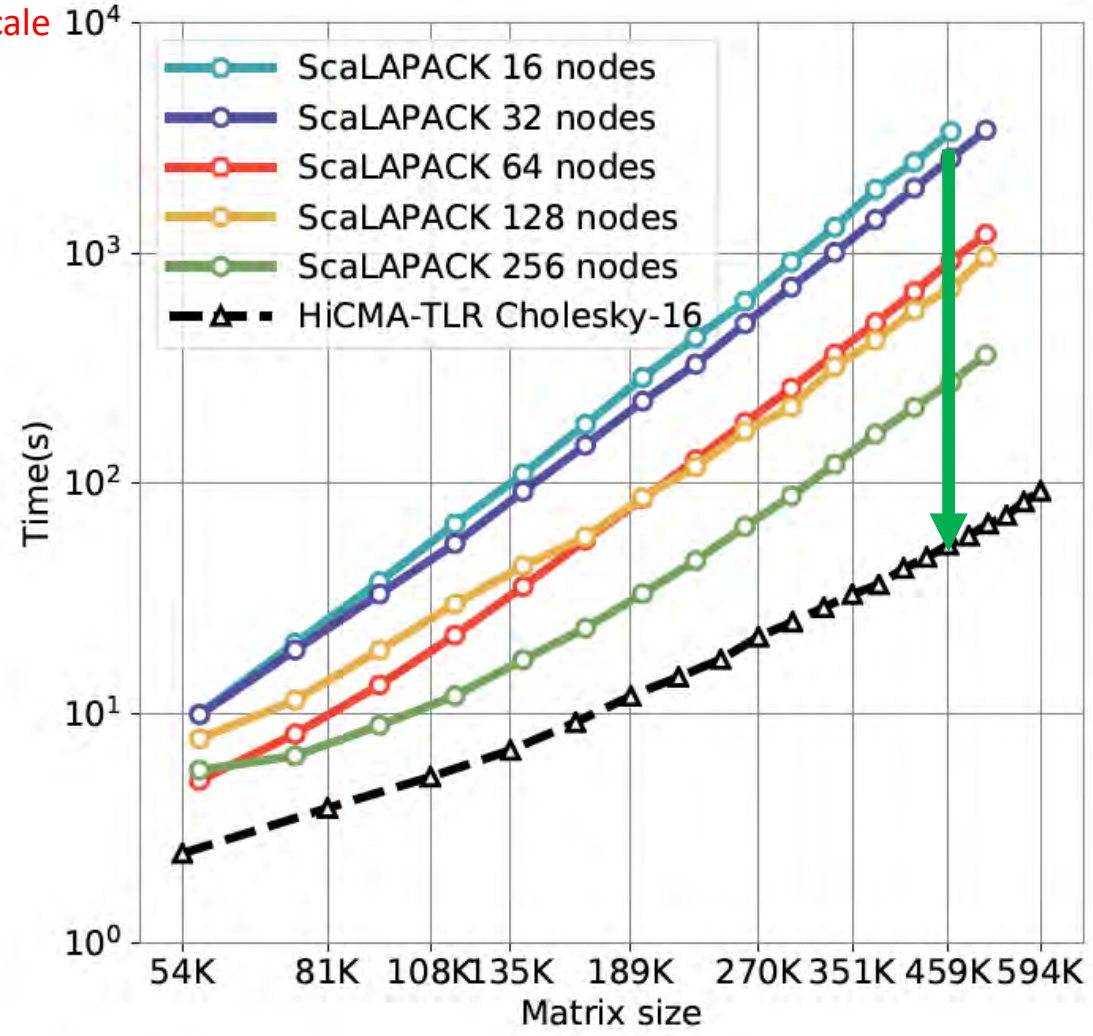
4 TB

1 to 2 orders of magnitude less, depending upon accuracy

Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations*, EuroPar 2018

TLR vs. ScaLAPACK on distributed memory

NB: log scale



Green arrow: speedup from algorithm, same 16 nodes

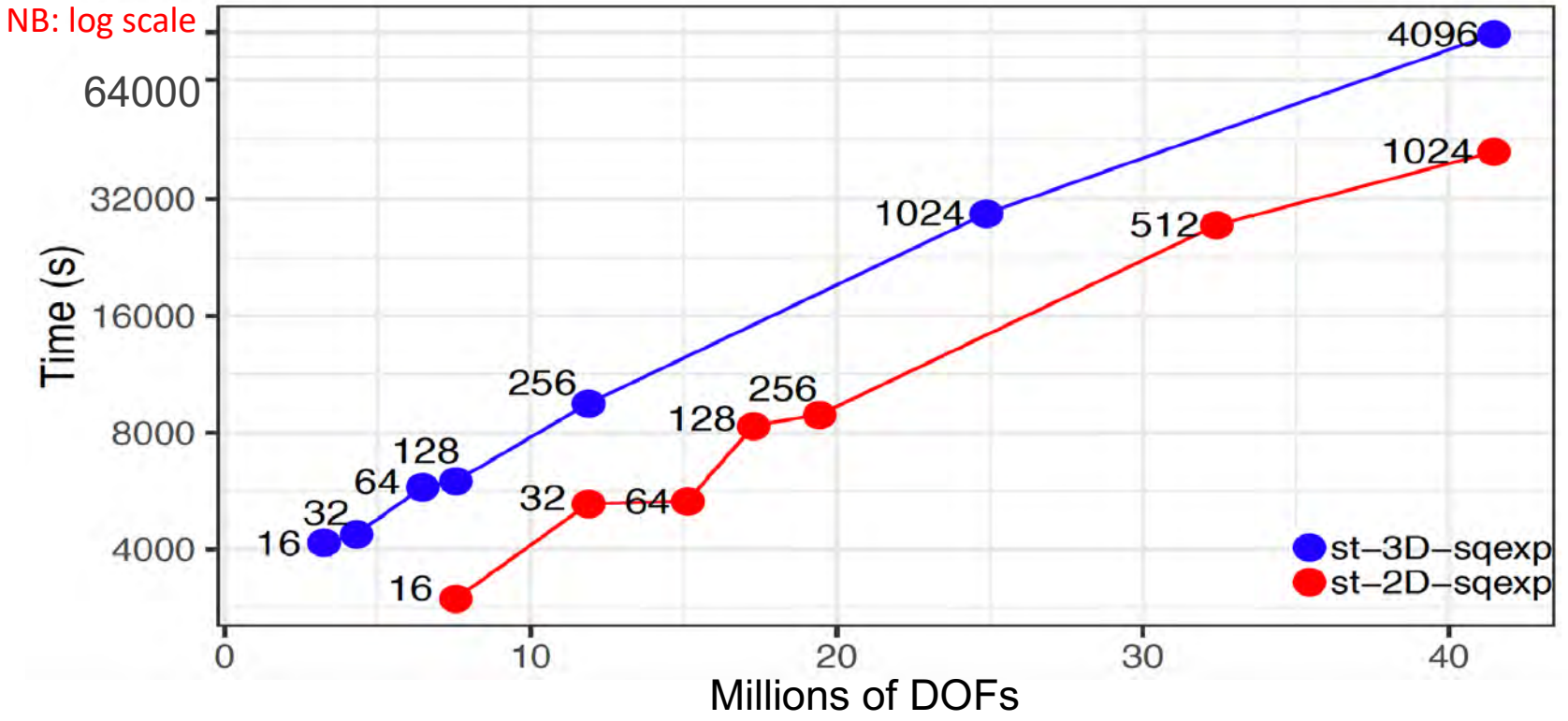
Shaheen II at KAUST: a Cray XC40 system with 6,174 compute nodes, each of which has two 16-core Intel Haswell CPUs running at 2.30 GHz and 128 GB of DDR4 main memory

Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations,* Euro-Par 2018

TLR *tour de force*

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

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

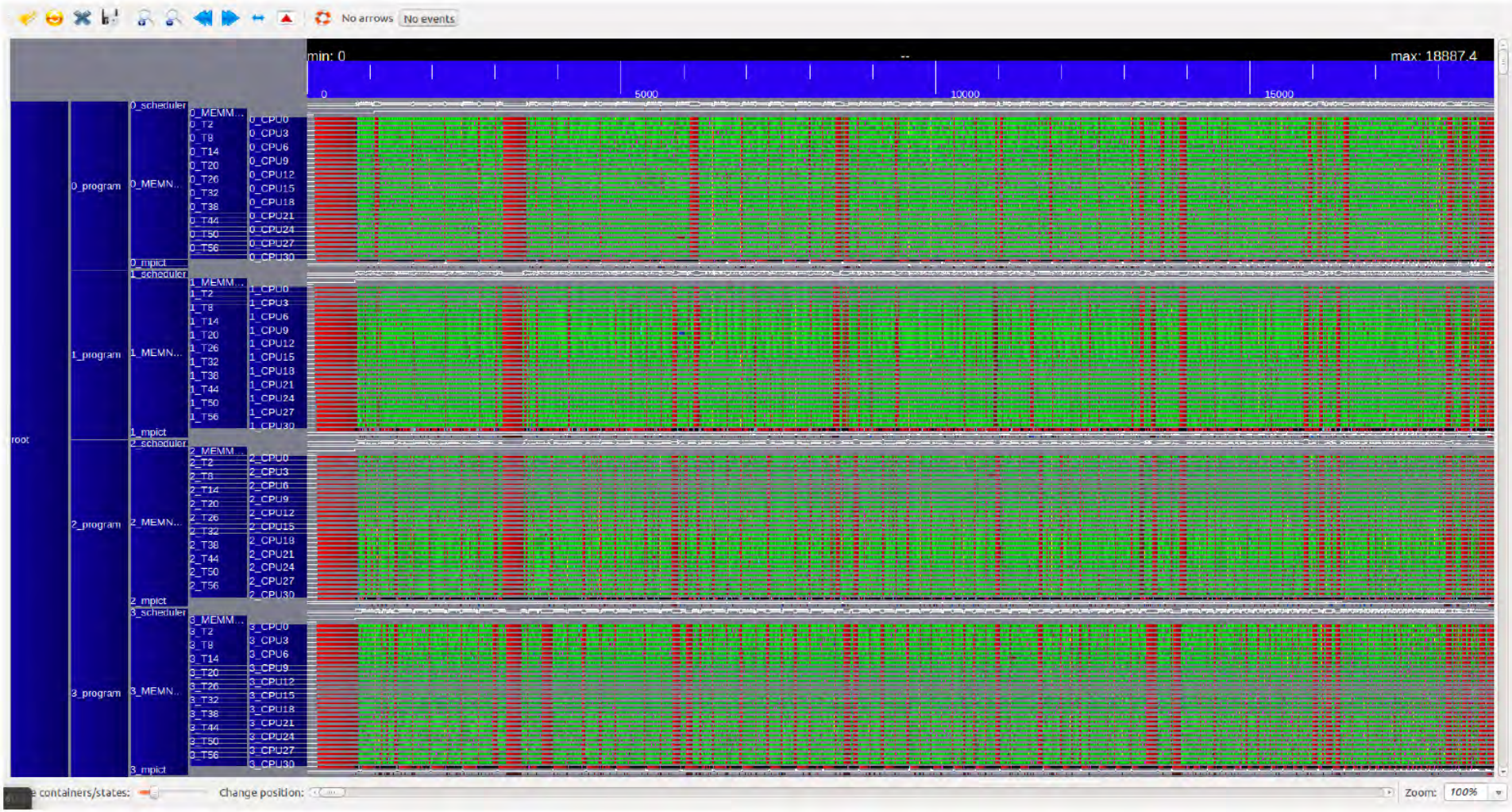


Cao, Pei, Akbudak, Mikhalev, Bosilca, Ltaief, K. & Dongarra, *Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications*. PASC '20 (ACM), 2020

Execution trace, dense DPOTRF

Chameleon: Dense DPOTRF time **18.1s**

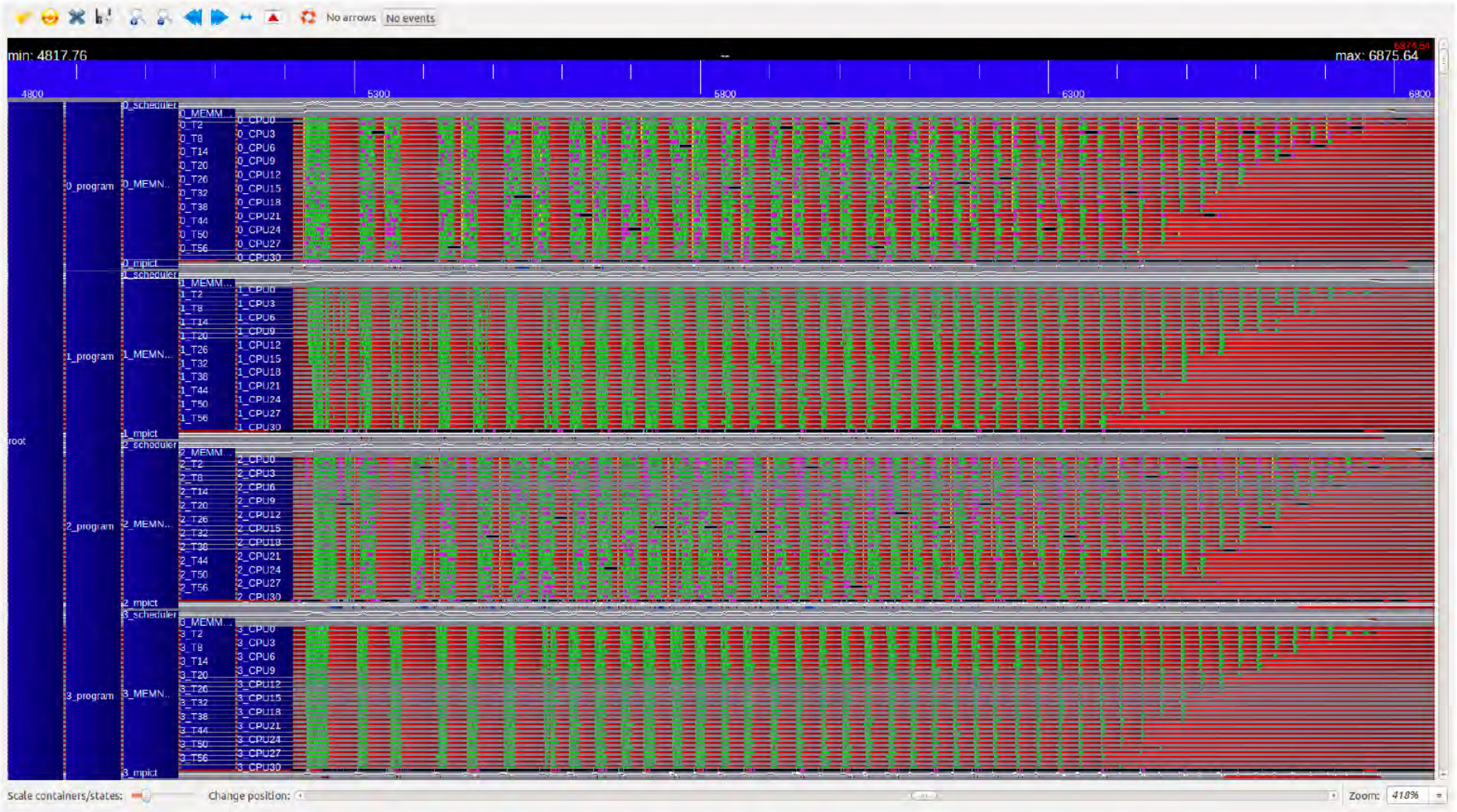
4 nodes of Shaheen with a matrix size of 54K using StarPU runtime



Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations,* Euro-Par 2018

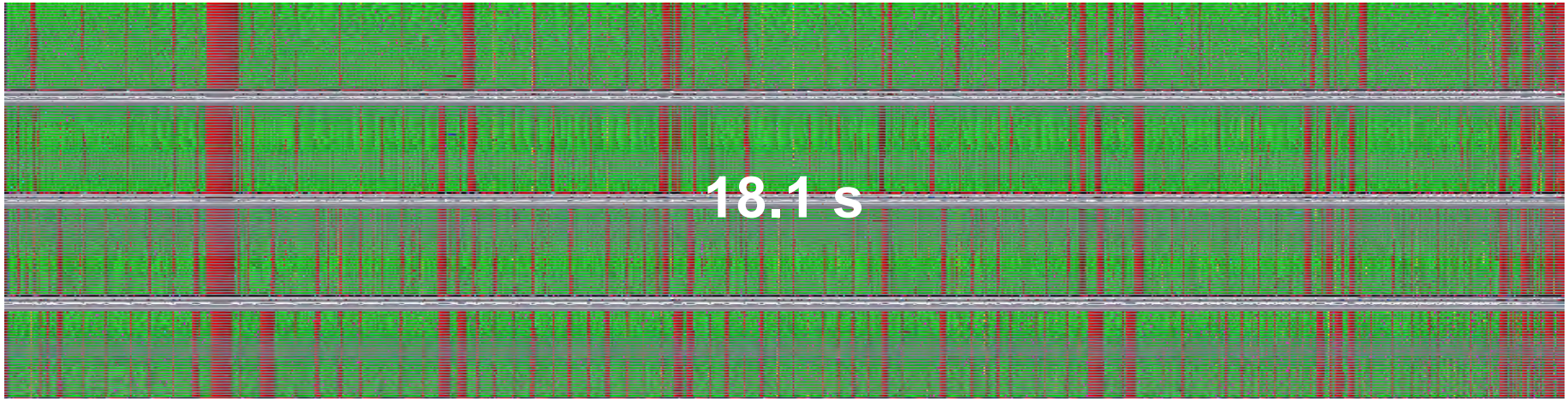
Execution trace, TLR DPOTRF

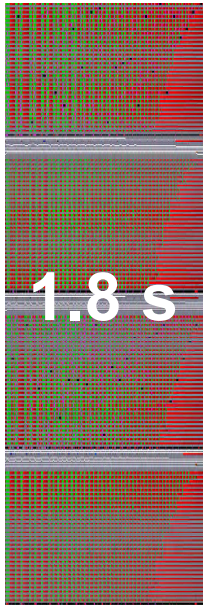
HiCMA: TLR DPOTRF time **1.8s** (10X faster)
4 nodes of Shaheen with a matrix size of 54K, using StarPU runtime



Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations,* Euro-Par 2018

Comparing the traces





- **Tile low rank has a higher percentage of idle time (red), relative to flop-intensive dense tile**
- **Scales less efficiently relative to itself**
- **However, for a acceptable accuracy tolerance, it is superior in time and energy**

Akbudak, Ltaief, Mikhalev, Charara & K., *Exploiting Data Sparsity for Large-scale Matrix Computations,* Euro-Par 2018

TLR (complex) LU factorization

Algorithm 1. Dense Tile *LU* factorization of a *N*-by-*N* matrix *A* composed of $nb \times nb$ tiles and solve.

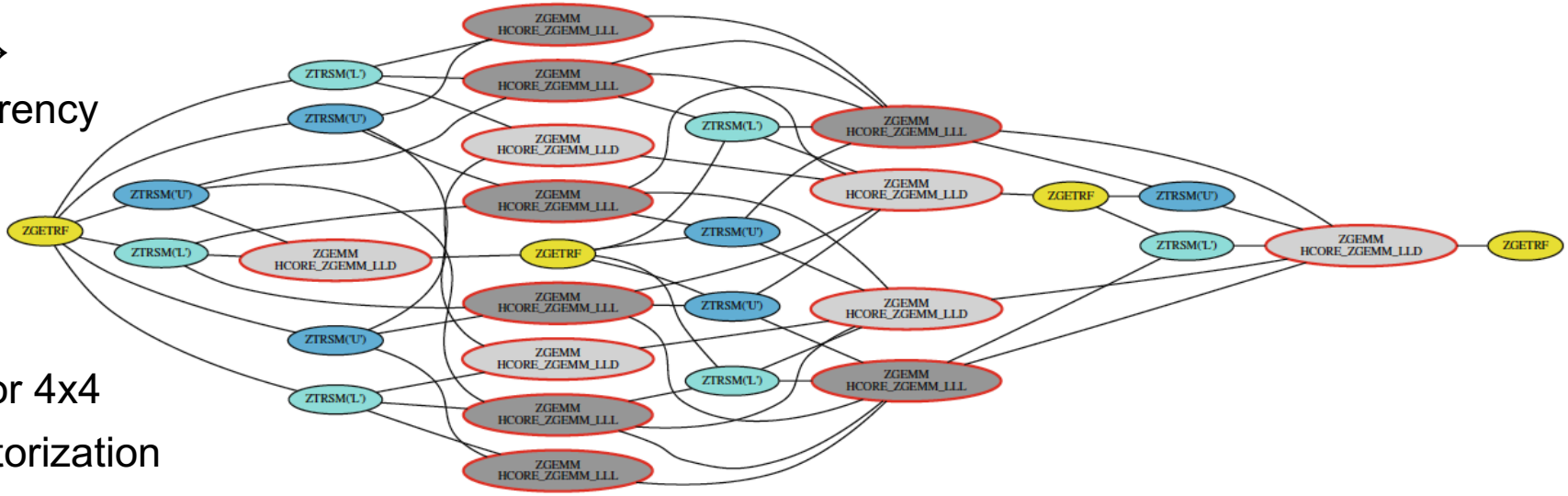
```
1: p = N / nb                                   ▷ number of tiles
2: for k = 1 to p do
3:     ZGETRF(A[k][k])
4:     for m = k+1 to p do
5:         ZTRSM('U', A[k][k], A[m][k])
6:     for n = k+1 to p do
7:         ZTRSM('L', A[k][k],A[k][n])
8:         for m = k+1 to p do
9:             ZGEMM(A[m][k],A[k][n],A[m][n])
```

```
for m = k+1 to p do
    if m == n then
        HCORE_ZGEMM_LLD(U[n][k],    V[n][k],
            U[k][n], V[k][n], D[n][n])
    else
        HCORE_ZGEMM_LLL(U[m][k],    V[m][k],
            U[k][n],    V[k][n],    U[m][n],
            V[m][n], rank, acc)
```

Conventional tile LU factorization (shown with complex data types, left) is converted to a TLR LU factorization with replacement of off diagonal blocks with low rank compressed

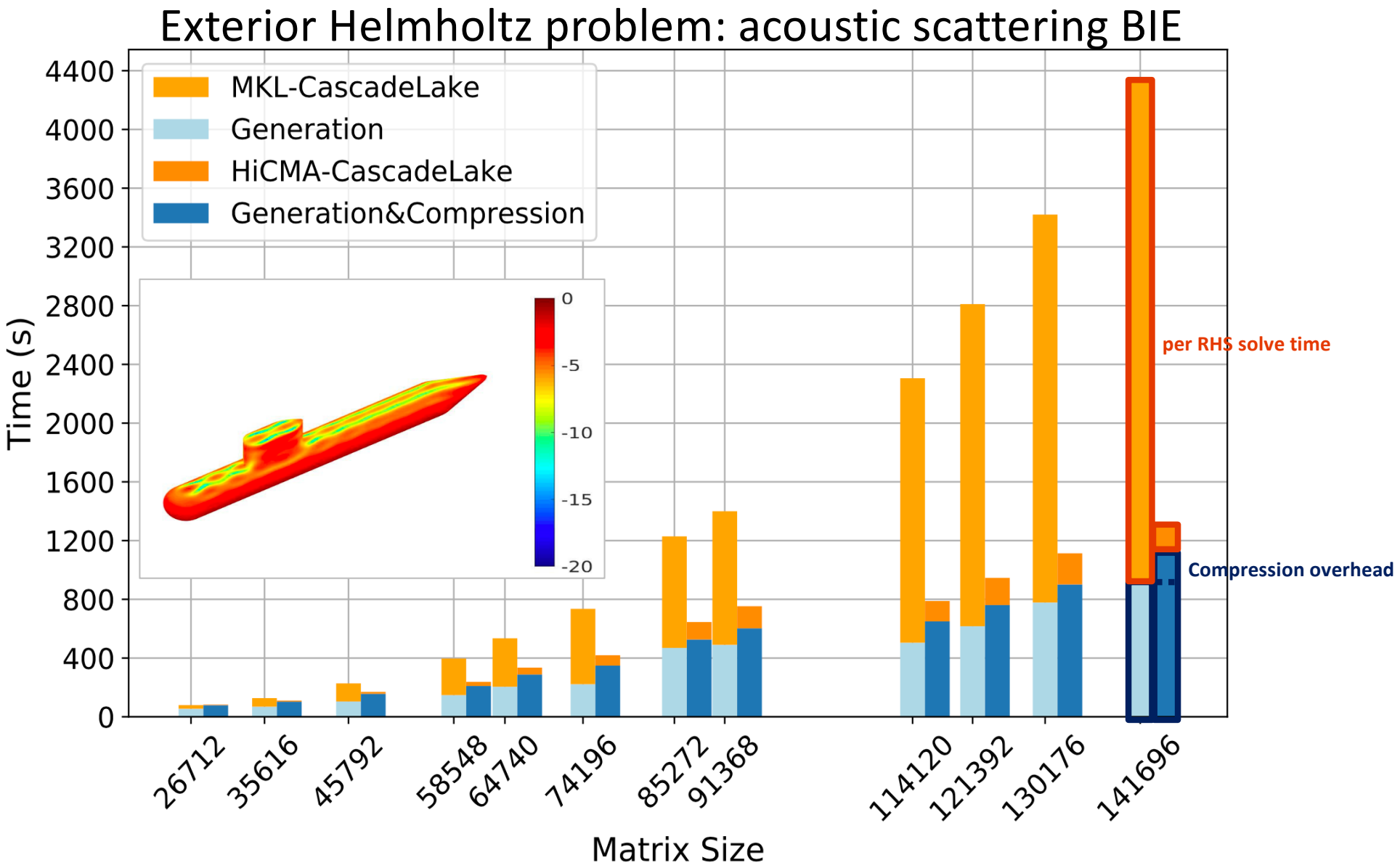
time →

concurrency ↓



DAG for 4x4 LU factorization

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



Al-Harthi, Alomairy, Akbudak, Chen, Ltaief, Bagci & K., *Solving Acoustic Boundary Integral Equations using High Performance Tile Low Rank LU Factorization*, Proceedings of ISC High Performance 2020

Rank distribution challenges with 3D exponential kernels

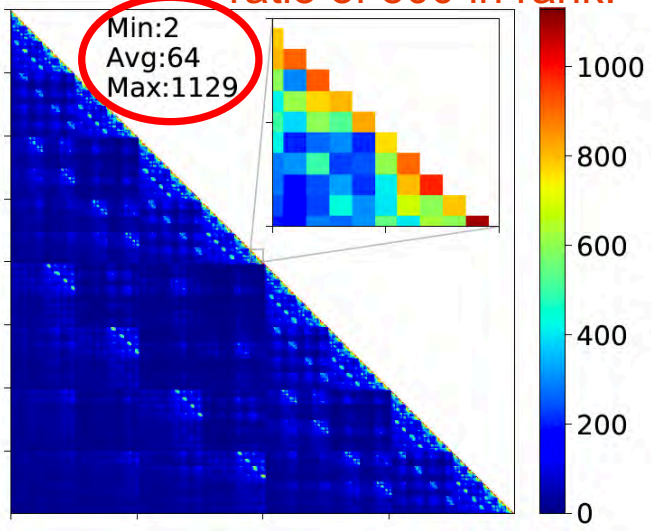
The simple exponential kernel:

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

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

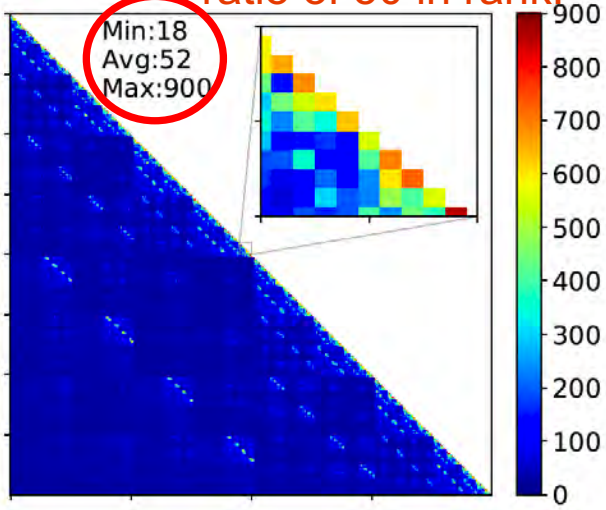
ratio of 500 in rank!

final ranks following Cholesky factorization



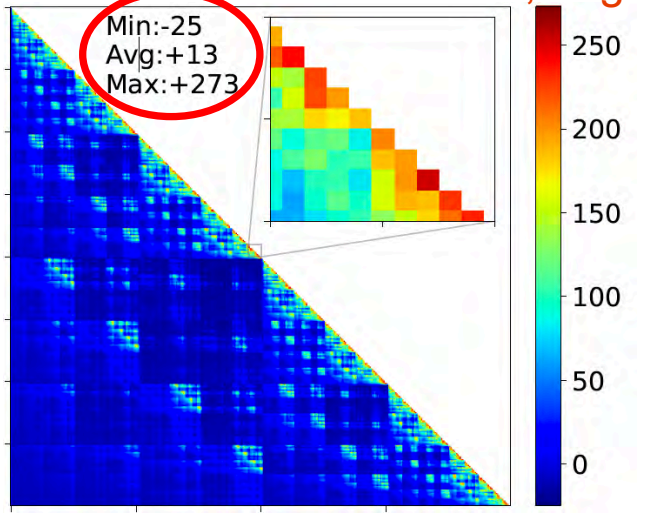
ratio of 50 in rank!

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



small→smaller, large→larger

rank variations before and after factorization



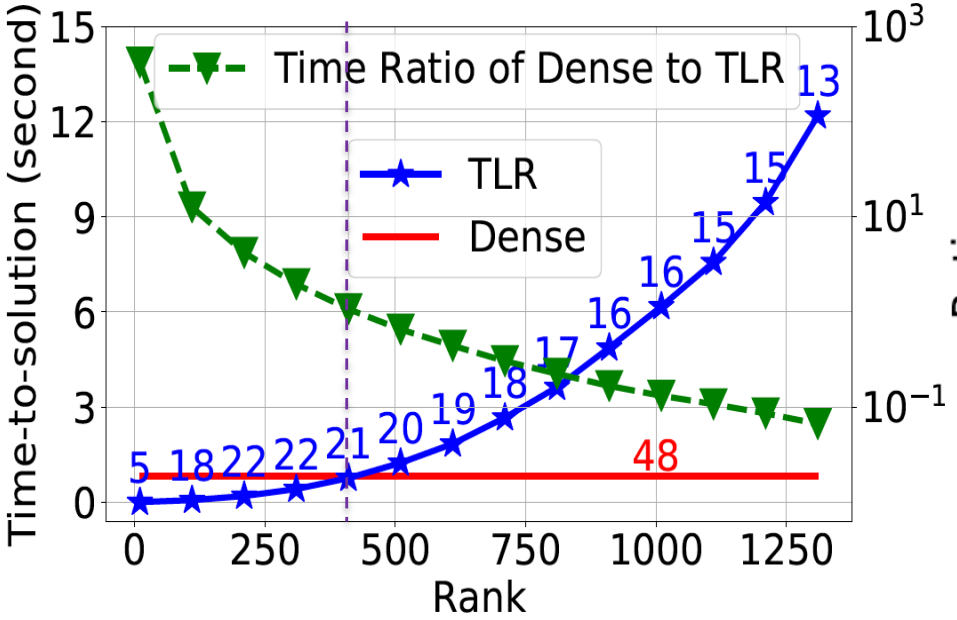
Cao, Pei, Akbudak, Bosilca, Ltaief, K. & Dongarra, *Leveraging PaRSEC Runtime Support to Tackle Challenging 3D Data-sparse Matrix Problems*. IPDPS (IEEE), 2021

Rank distribution challenges with 3D exponential kernels

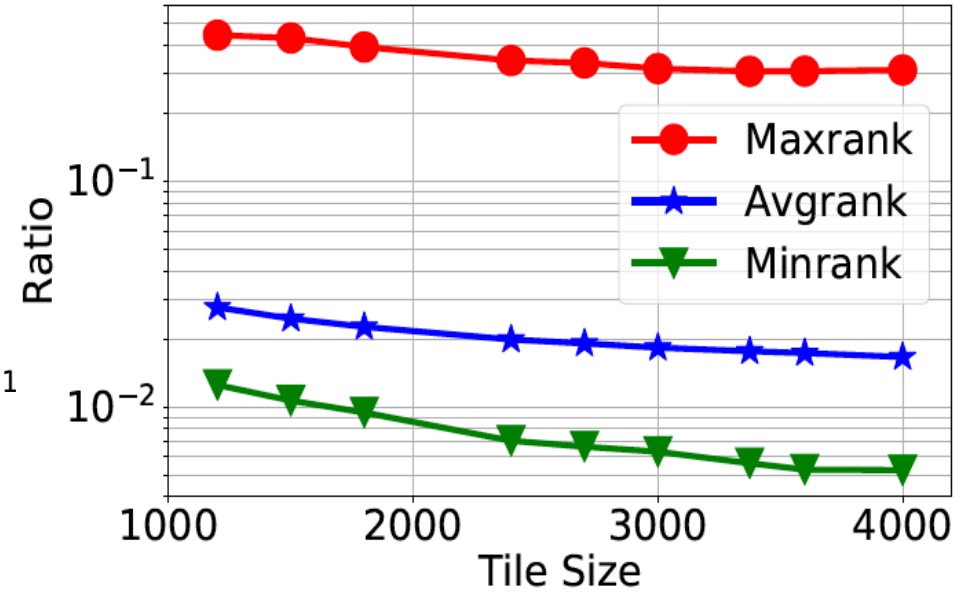
Threshold for treating an *individual* data-sparse tile as dense is relatively low (effectively about 15% of tile size)

May be lower considering *distribution* of ranks

NB: ratios in log scale



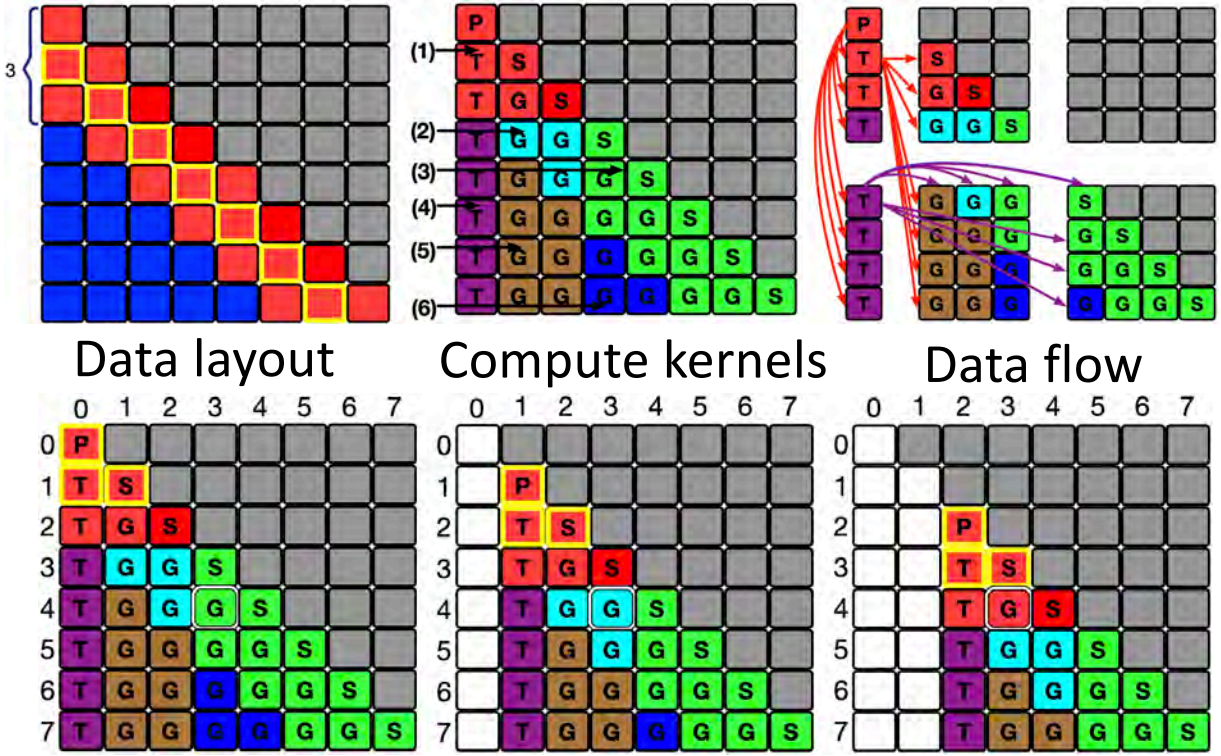
Time and annotated flop rates for low rank GEMM and dense GEMM (48 GF/s) for tiles size 2700, with ratios & threshold, as function of rank



Ratio of rank to tile size with matrix size 1.08M; rank is a mathematical object; tile size is an algorithmic parameter

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

Dynamic data structure adaption: treat near-diagonal high-rank tiles as fully dense



First, second and third panel factorization & update steps

P - POTRF, T - TRSM, S - SYRK, G – GEMM

Upper color: within tile band adaptively preserved as dense

Lower color: handled as low rank tile

ID	(Group)-Name	Complexity
0	(1)-POTRF	$\frac{1}{3} \times b^3$
1	(1)-TRSM	b^3
2	(4)-TRSM	$b^2 \times b$
3	(1)-SYRK	b^3
4	(3)-SYRK	$2 \times b^2 \times k + 4 \times b \times k^2$
5	(1)-GEMM	$2 \times b^3$
6	(2)-GEMM	$4 \times b^2 \times k$
7	(3)-GEMM	$2 \times b^2 \times k + 4 \times b \times k^2$
8	(5)-GEMM	$34 \times b \times k^2 + 157 \times k^3$
9	(6)-GEMM	$36 \times b \times k^2 + 157 \times k^3$

Algorithm 1: Algorithm for `BAND_SIZE` auto-tuning.

Input : Matrix data descriptor

1 Generate the matrix with `BAND_SIZE` $= 1$

2 Globalize the rank distribution to all the processes

3 Set $ID = 1$ and initialize $fluctuation$

4 **do**

5 $ID := ID + 1$

6 ops_dense = total `TRSM` and `GEMM` FLOPs of all tiles in sub-diagonal with `BAND_ID` $= ID$ if executing in dense format

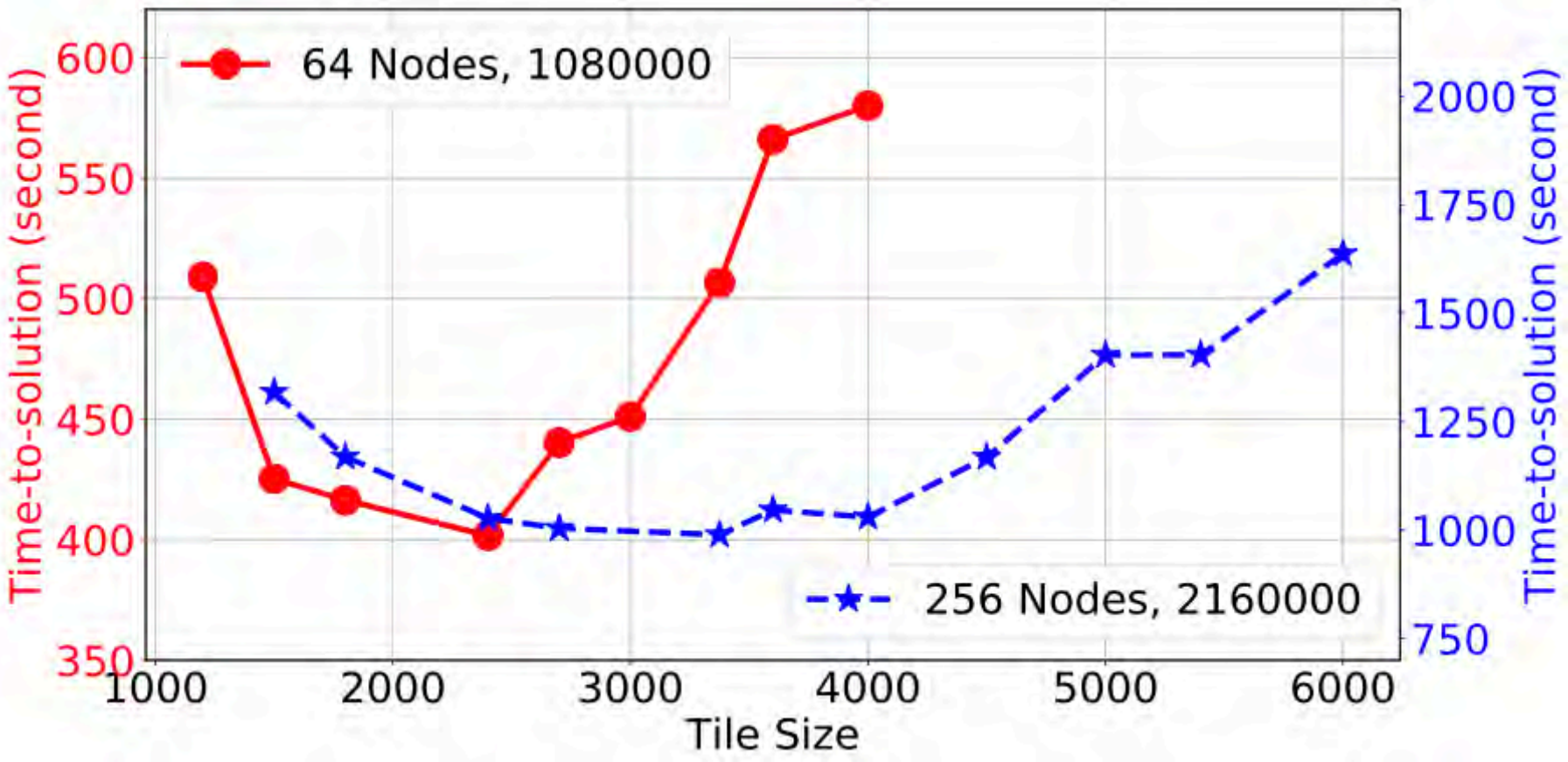
7 ops_tlr = total `TRSM` and `GEMM` FLOPs of all tiles in sub-diagonal with `BAND_ID` $= ID$ if executing in low-rank format

8 **while** $ops\_dense < fluctuation \times ops\_tlr$;

Output: `BAND_SIZE` $= ID - 1$

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

Sensitivity to tile size

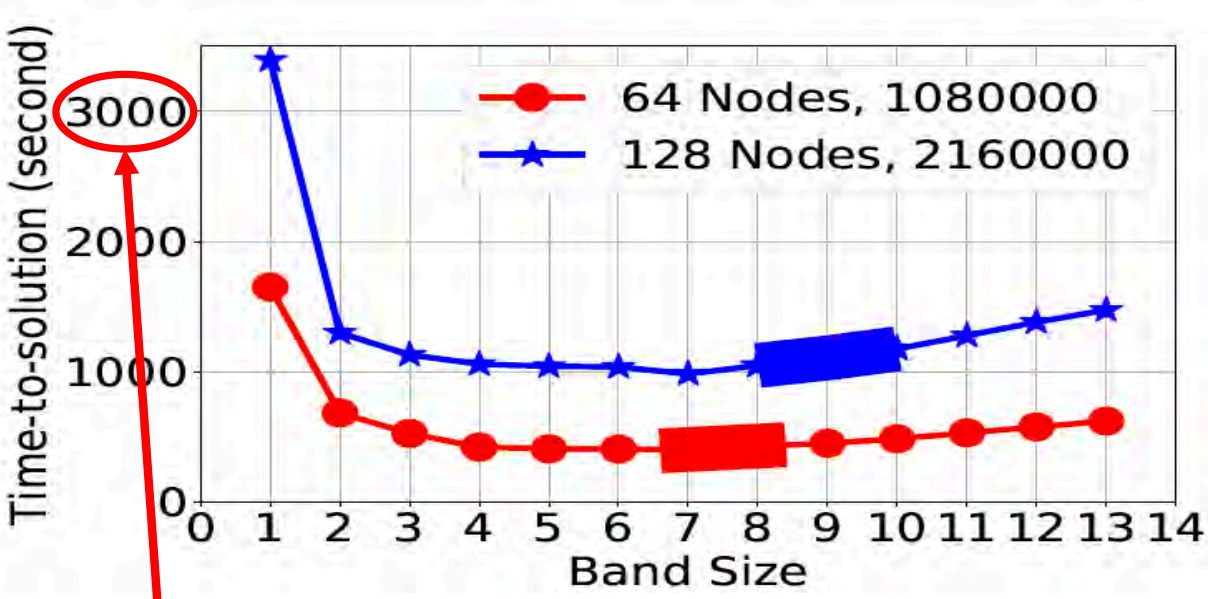


Auto-tuning is available based on a starting point for tile size $B = \sqrt{N}$

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

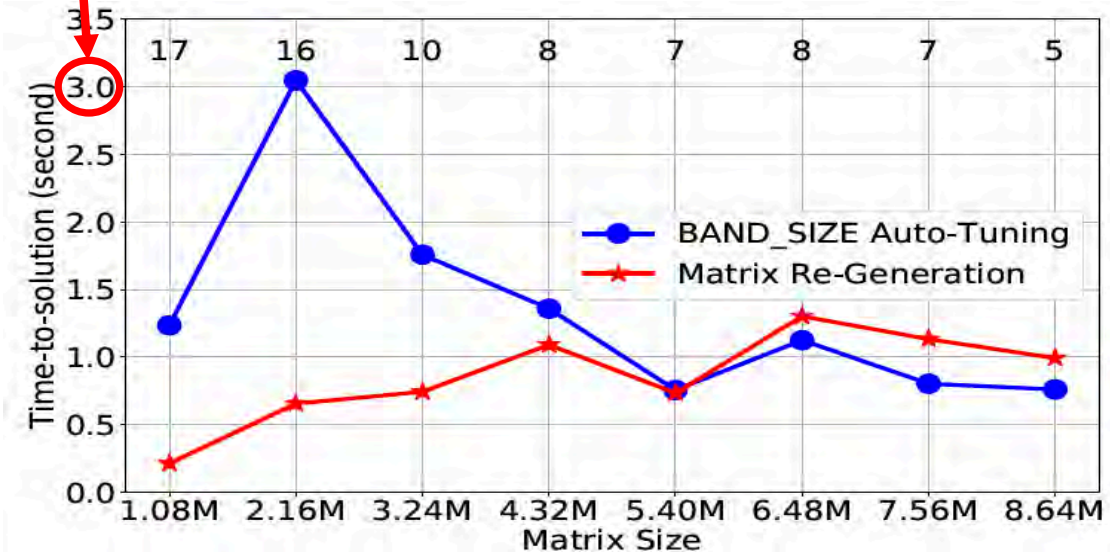
Dense tile bandsize auto-tuning

Time to Solution



Boxes indicate bandsize range in which TLR and dense flops for all GEMM and TRSM are within a small range around unity

Auto-tuning time on 512 nodes



Time spent autotuning for bandsize is about 0.1% of cost of solution

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

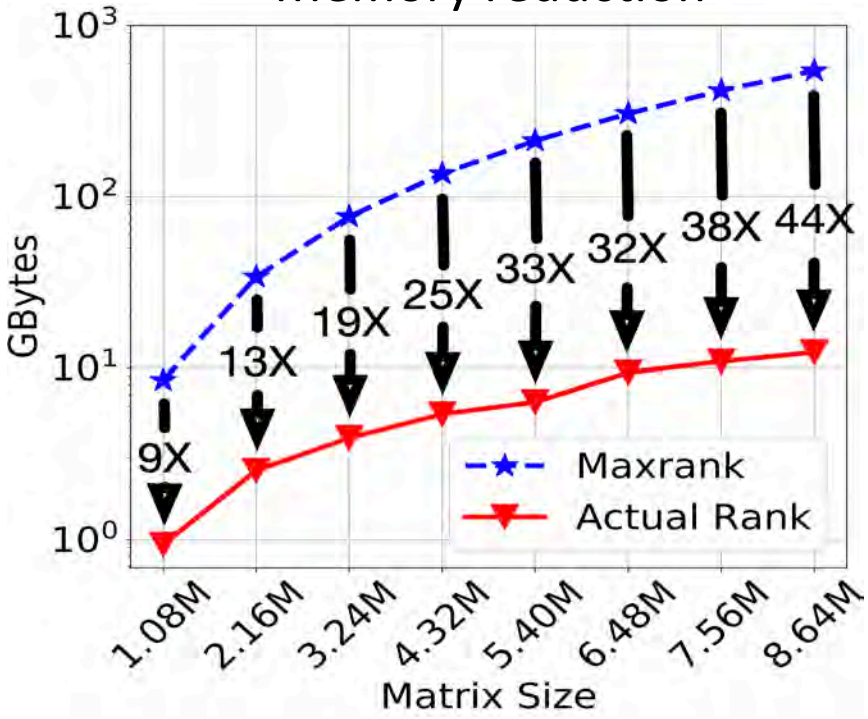
Memory reduction & load imbalance

Even after thresholding high-rank off-diagonal tiles as dense, a wide distribution of smaller ranks remains.

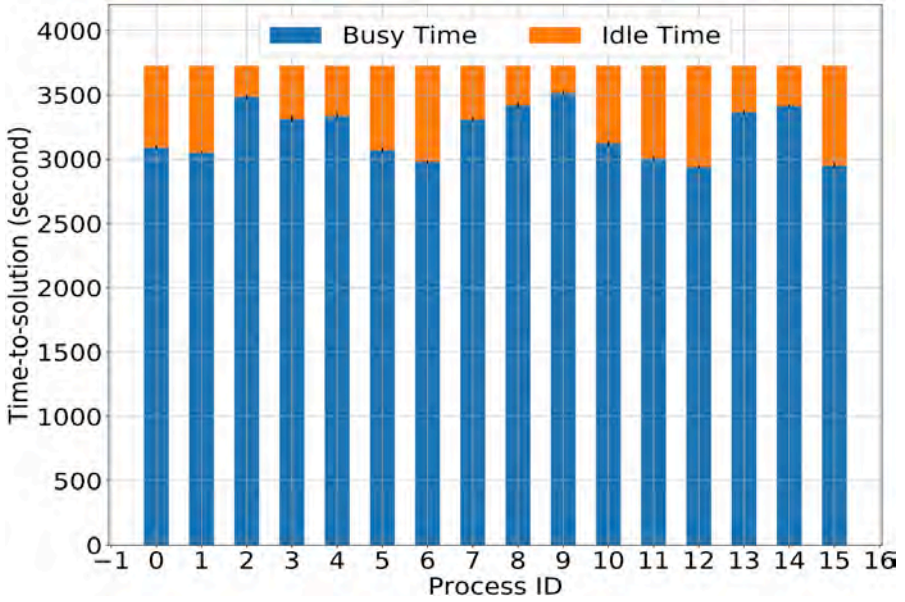
Memory allocations per tile are reduced to those actually needed with major reduction (up to 44x) in band adaptive TLR relative to earlier TLR versions.

Dynamic runtime system deals with tile load variations gracefully.

Memory reduction

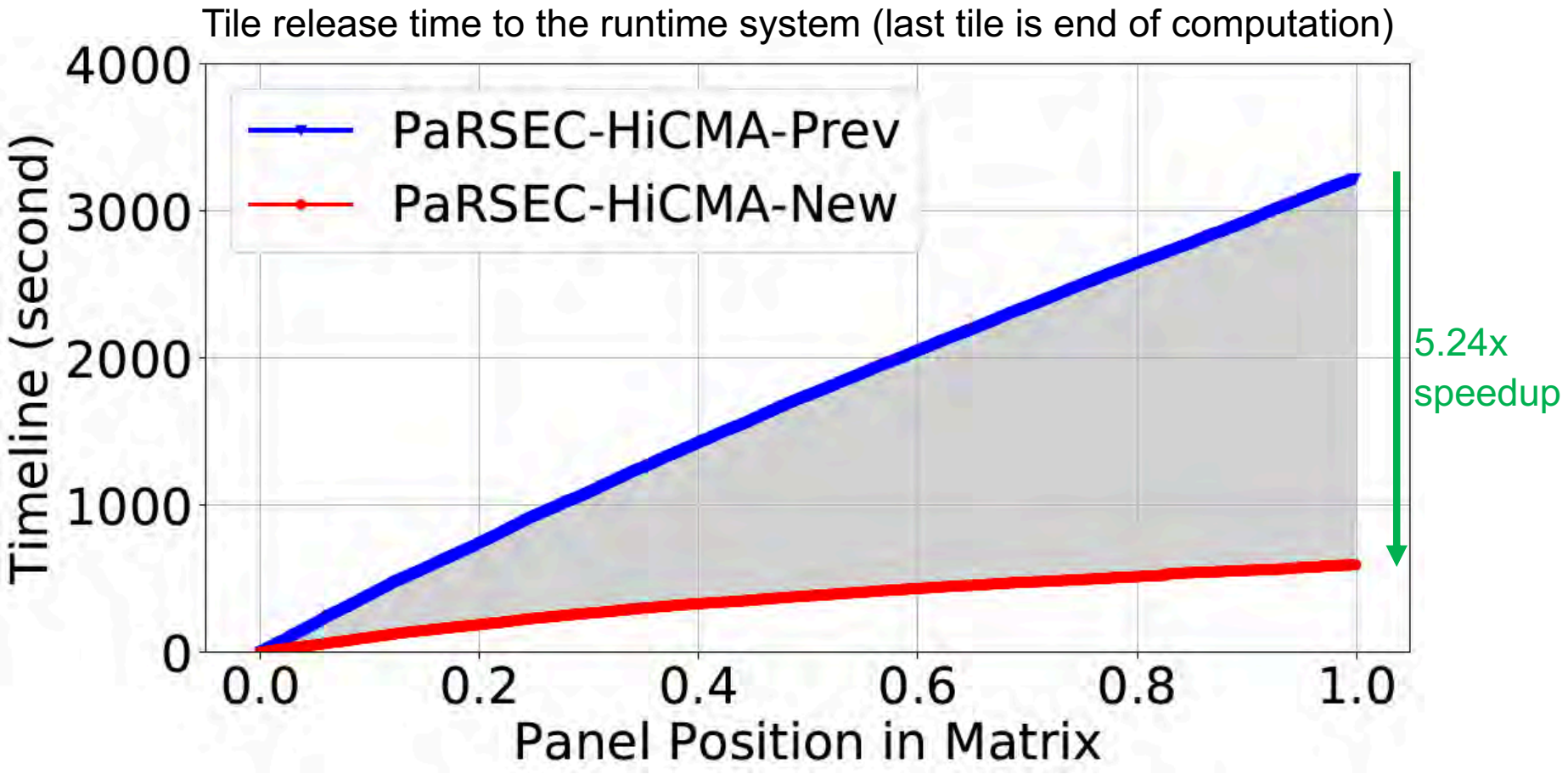


Imbalance idle times on 16 nodes



Comparison with prior state-of-the-art

For exponential kernel matrix of N = 2.16M on 256 nodes, new TLR optimizations to account for rank variation save factor of 5x to 7.5x across a range of relevant sizes for geospatial environmental statistics

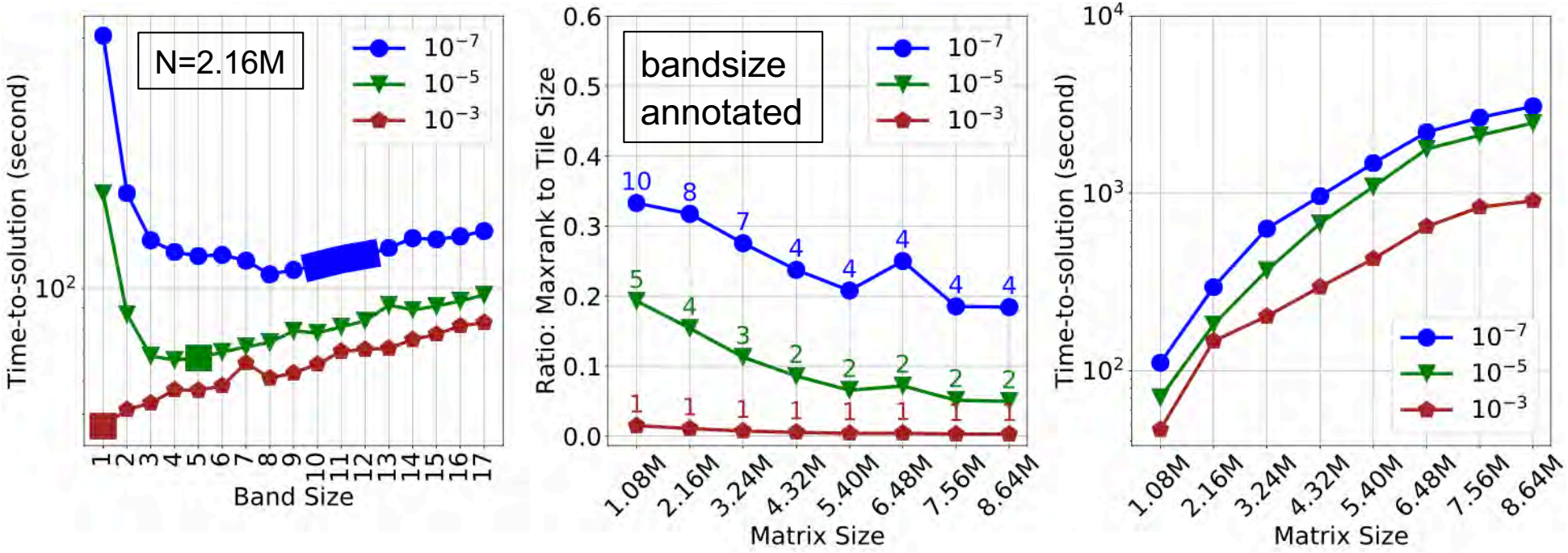


Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, *Extreme-scale Task-based Cholesky Factorization Toward Climate and Weather Prediction Applications*, PASC, 2020 (ACM)

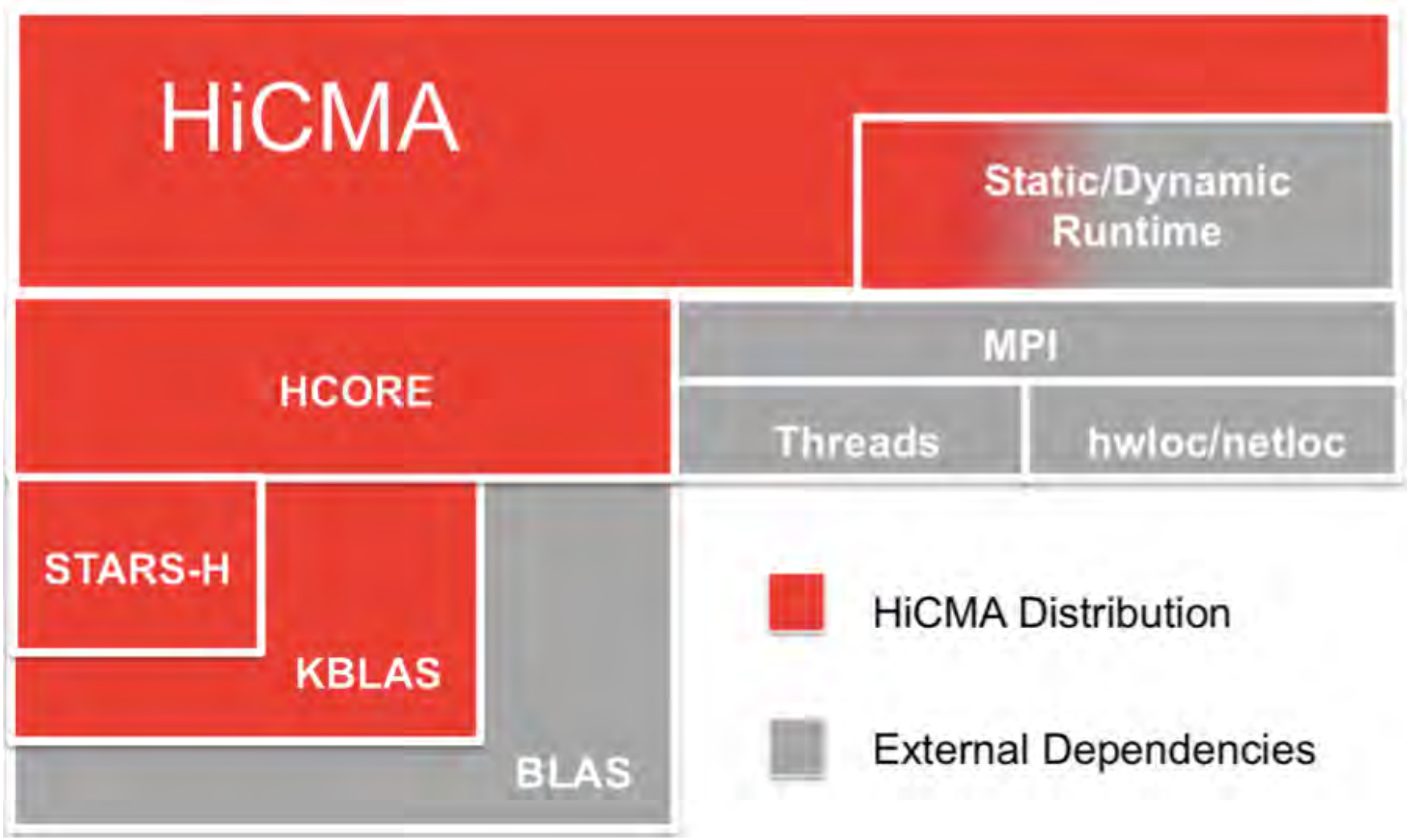
Sensitivity to accuracy thresholds

Effect of accuracy requirements on 512 nodes

Earlier results were for 10^{-9}; here are three looser tolerances



Hierarchical Computations on Manycore Architectures: HiCMA*

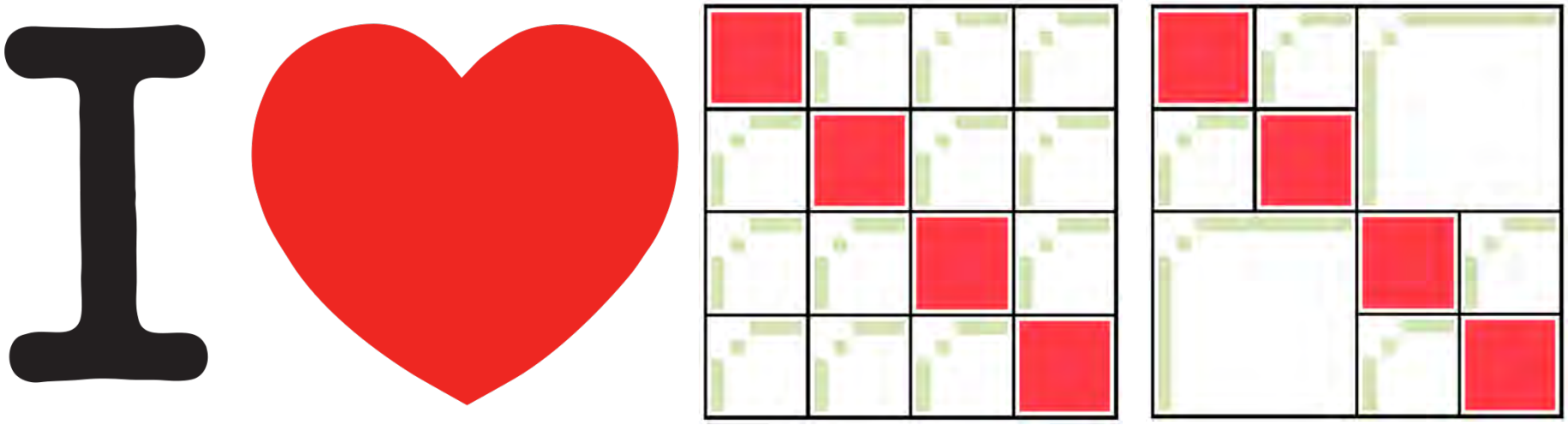


* appearing one thesis at a time at https://github.com/ecrc

Conclusions, recapped

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

Iconographic conclusion



Reference



Akbudak, Ltaief, Mikhalev, Charara, Esposito & K.

Lecture Notes in Computer Science 11014:811 (2018)

Exploiting Data Sparsity for Large-Scale Matrix Computations

Kadir Akbudak[1], Hatem Ltaief[1], Aleksandr Mikhalev[1](✉), Ali Charara[1], Aniello Esposito[2], and David Keyes[1]

[1] Extreme Computing Research Center, Division of Computer, Electrical, and Mathematical Sciences and Engineering, King Abdullah University of Science and Technology, Thuwal Jeddah 23955, Kingdom of Saudi Arabia
{kadir.akbudak,hatem.ltaief, aleksandr.mikhalev,ali.charara,david.keyes}@kaust.edu.sa

[2] Cray EMEA Research Lab, Bristol, UK
esposito@cray.com

Abstract. Exploiting data sparsity in dense matrices is an algorithmic bridge between architectures that are increasingly memory-austere on a per-core basis and extreme-scale applications. In this work, we leverage the Hierarchical matrix Computations on Manycore Architectures (HiCMA) library in order to tackle this challenging problem by achieving significant reductions in time to solution and memory footprint, while preserving a specified accuracy requirement of the application. We have extended HiCMA to provide a high-performance implementation on distributed-memory systems of one of the most widely used matrix factorization in large-scale scientific applications, i.e., the Cholesky factorization. It employs the tile low-rank data format to compress the dense data-sparse off-diagonal tiles of the matrix. It then decomposes the matrix computations into interdependent tasks and relies on the dynamic runtime system StarPU for asynchronous out-of-order scheduling, while allowing high user productivity. Performance comparisons and memory footprint on matrix dimensions up to eleven million show a performance gain and memory saving of more than an order of magnitude for both metrics on thousands of cores, against state-of-the-art open-source and vendor optimized numerical libraries. This represents an important milestone in enabling large-scale matrix computations toward solving big data problems in geospatial statistics for climate/weather forecasting applications.

1 Introduction

State-of-the-art dense linear algebra libraries are confronting memory capacity limits and/or are not able to produce solutions in reasonable times, when performing dense computations (e.g., matrix factorizations and solutions) on large

© Springer International Publishing AG, part of Springer Nature 2018

M. Aldinucci et al. (Eds.): Euro-Par 2018, LNCS 11014, pp. 721–734, 2018.
https://doi.org/10.1007/978-3-319-96983-1_51

Reference

Cao, Pei, Akbudak, Mikhalev, Bosilca, Ltaief, K. & Dongarra

Platform for Advanced Scientific Computing Conf. (ACM) (2020)

Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications

Qinglei Cao
qcao3@vols.utk.edu
University of Tennessee

Yu Pei
ypei2@vols.utk.edu
University of Tennessee

Kadir Akbudak
kadir.akbudak@kaust.edu.sa
King Abdullah University of Science and Technology

Aleksandr Mikhalev
aleksandr.mikhalev@kaust.edu.sa
King Abdullah University of Science and Technology

George Bosilca
bosilca@icl.utk.edu
University of Tennessee

Hatem Ltaief
hatem.ltaief@kaust.edu.sa
King Abdullah University of Science and Technology

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

Jack Dongarra
dongarra@icl.utk.edu
University of Tennessee, the Oak Ridge National Laboratory and the University of Manchester, UK

Abstract

Climate and weather can be predicted statistically via geospatial Maximum Likelihood Estimates (MLE), as an alternative to running large ensembles of forward models. The MLE-based iterative optimization procedure requires the solving of large-scale linear systems that performs a Cholesky factorization on a symmetric positive-definite covariance matrix—a demanding dense factorization in terms of memory footprint and computation. We propose a novel solution to this problem: at the mathematical level, we reduce the computational requirement by exploiting the data sparsity structure of the matrix off-diagonal tiles by means of low-rank approximations; and, at the programming-paradigm level, we integrate PaRSEC, a dynamic, task-based runtime to reach unparalleled levels of efficiency for solving extreme-scale linear algebra matrix operations. The resulting solution leverages fine-grained computations to facilitate asynchronous execution while providing a flexible data distribution to mitigate load imbalance. Performance results are reported using 3D synthetic datasets up to 42*M* geospatial locations on 130, 000 cores, which represent a cornerstone toward fast and accurate predictions of environmental applications.

CCS Concepts

• **Mathematics of computing**; • **Computing methodologies** → **Massively parallel algorithms**;

Keywords

Low-rank matrix computations, Dynamic runtime system, Asynchronous execution, Load balancing, High performance computing

ACM Reference Format:
Qinglei Cao, Yu Pei, Kadir Akbudak, Aleksandr Mikhalev, George Bosilca, Hatem Ltaief, David Keyes, and Jack Dongarra. 2020. Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications . In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '20), June 29-July 1, 2020, Geneva, Switzerland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394277.3401846

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PASC '20, June 29-July 1, 2020, Geneva, Switzerland

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7993-9/20/06…$15.00
https://doi.org/10.1145/3394277.3401846

1 Introduction

Massive parallelism is the dominant force behind the increased capabilities of scientific computing. Given the structural constraints on technology and hardware design, high-performance computing (HPC) architecture development, which is striving to satisfy application needs and achieve new levels of performance, has to deal with unprecedented increases in concurrency, non-uniform hardware designs, and changing performance capabilities. In this unfriendly landscape, application developers face unfamiliar challenges at all levels, from the increase in the number of nodes to the highly complex architectural capabilities on each node, and from the lack of portability between different architectures to a lack of compatibility across different versions of the same hardware. Faced with such daunting challenges, combining existing programming paradigms (i.e., the so-called "MPI+X model") often backfires. The application programmer is exposed to the complexity of handling the non-uniform system explicitly, while the composition of multiple programming paradigms encourages a static distribution of the computation between different logical domains. As the systems grow increasingly complex, static assumptions about synchrony, deterministic scheduling, and predictable runtime of computation and communication alike, no longer bear out; and even a minor amount of system noise and small delays introduce significant slack in large-scale synchronous applications [16, 28, 50].

Reference

Cao, Pei, Akbudak, Bosilca, Ltaief, K. & Dongarra

Int. Par. & Distr. Proc. Symp. (IEEE) (2021, to appear)

Leveraging PaRSEC Runtime Support to Tackle Challenging 3D Data-Sparse Matrix Problems

Qinglei Cao[1], Yu Pei[1], Kadir Akbudak[3], George Bosilca[1],
Hatem Ltaief[2], David Keyes[2], and Jack Dongarra[1]

[1]Innovative Computing Laboratory, University of Tennessee, US
[2]King Abdullah University of Science and Technology, KSA
[3]ASELSAN Research Center, Turkey

***Abstract*—The task-based programming model associated with dynamic runtime systems has gained popularity for challenging problems because of workload imbalance, heterogeneous resources, or extreme concurrency. During the last decade, low-rank matrix approximations, where the main idea consists of exploiting data sparsity typically by compressing off-diagonal tiles up to an application-specific accuracy threshold, have been adopted to address the curse of dimensionality at extreme scale. In this paper, we create a bridge between the runtime and the linear algebra by communicating knowledge of the data sparsity to the runtime. We design and implement this synergistic approach with high user productivity in mind, in the context of the PaRSEC runtime system and the HiCMA numerical library. This requires to extend PaRSEC with new features to integrate rank information into the dataflow so that proper decisions can be taken at runtime. We focus on the tile low-rank (TLR) Cholesky factorization for solving 3D data-sparse covariance matrix problems arising in environmental applications. In particular, we employ the 3D exponential model of Matérn matrix kernel, which exhibits challenging nonuniform high ranks in off-diagonal tiles. We first provide a dynamic data structure management driven by a performance model to reduce extra floating-point operations. Next, we optimize the memory footprint of the application by relying on a dynamic memory allocator, and supported by a rank-aware data distribution to cope with the workload imbalance. Finally, we expose further parallelism using kernel recursive formulations to shorten the critical path. Our resulting high-performance implementation outperforms existing data-sparse TLR Cholesky factorization by up to 7-fold on a large-scale distributed-memory system, while minimizing the memory footprint up to a 44-fold factor. This multidisciplinary work highlights the need to empower runtime systems beyond their original duty of task scheduling for servicing next-generation low-rank matrix algebra libraries.**

***Index Terms*—Low-rank matrix computations, Task-based programming model, Dynamic runtime system, Asynchronous executions and load balancing, High-performance computing, User productivity, Environmental applications.**

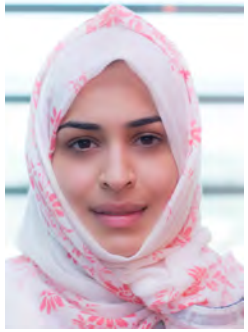
I. Introduction

The HPC community has enjoyed an order of magnitude performance improvement every five years [1] thanks to hardware innovations and technology scaling. At the dawn of exascale, this means systems with tens of millions of concurrent threads. This rapidly evolving hardware landscape requires new software paradigms, and perhaps equivalently important, advanced algorithmic responsibilities [2].

The coupling of task-based programming models with dynamic runtime systems corresponds to one of the most critical paradigm shifts embraced in order to replace the traditional bulk synchronous parallel model in favor of the asynchronous execution model [3]–[5]. This versatile software solution has shown performance superiority by overlapping expensive data movement with fine-grained computations and ultimately achieving higher occupancy on the underlying hardware architecture. Moreover, runtime systems are inherently designed with the abstraction of the hardware complexity. This latter feature enhances user-productivity and permits fast code deployment on massively parallel systems. At the same time, existing numerical methods have displayed limitations in addressing big data problems, due to high algorithmic complexity and large memory footprint. Low-rank matrix approximations may overcome the curse of dimensionality [6]. The main idea consists of approximating off-diagonal tiles up to an application-specific accuracy threshold and carrying out the matrix algorithm on the newly obtained data structures. This compression step may sacrifice numerical accuracy, so it results in a tunable tradeoff. By exploiting the rank structure naturally embedded in the data-sparse operator, lower complexity may be obtained in storage, data motion, and arithmetic operations, compared to traditional dense algorithms.

We propose a synergistic bridge between the runtime and linear algebra communities in the context of dealing with large-scale covariance matrices in geospatial statistics. We employ the HiCMA Tile Low-Rank (TLR) numerical library and the PaRSEC dynamic runtime system to showcase the mutual benefits of this approach. The objective is to propagate the rank information to PaRSEC so that it can take proper runtime decisions before HiCMA operates on the computational kernels. In particular, we focus on the challenging exponential model of Matérn matrix kernel for 3D environmental applications [7]. This model results in heterogeneous rank distribution with high-rank tiles located outside of the diagonal tiles. We extend PaRSEC with new functionality that takes into account the rank information in the task dataflow: (1) a dynamic data structure management driven by a performance model to reduce extra floating-point operations, (2) a dynamic memory allocator to further optimize memory footprint, (3) a rank-aware data distribution to cope with the workload imbalance, and (4) a recursive formulation of computational kernels to expose concurrency during the critical path. We use these features to leverage the performance of the TLR Cholesky factorization at the heart of the Maximum Likelihood Estimation (MLE) [8].

Reference


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

Lecture Notes in Computer Science 12151:209 (2020)

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

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

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

Abstract. We design and develop a new high performance implementation of a fast direct LU-based solver using low-rank approximations on massively parallel systems. The LU factorization is the most time-consuming step in solving systems of
analyzing acoustic scattering from larg
is obtained by discretizing the boundar
problem using a higher-order Nyström
the inherent data sparsity of the matri
centric approximations while still capt
tion. In particular, the proposed LU-b
Rank (TLR) data compression format
cal Computations on Manycore Archit
the complexity of "classical" dense dir
order. We taskify the underlying bo
fine-grained computations. We then e
StarPU to orchestrate the scheduling of
distributed-memory systems. The resu
mits to compensate for the load imbala
while mitigating the overhead of data
our TLR LU-based solver and study th
ferent numerical accuracies. The new
the state-of-the-art dense factorizatio
on various parallel systems, for analysi
synthetic and real geometries.

Keywords: Tile low-rank LU-based
Equations · Acoustic scattering · Tas
Dynamic runtime systems

© The Author(s) 2020

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

GCS
Gauss Centre for Supercomputing
SUPERCOMPUTING AT THE LEADING EDGE
ISC 2020

The Board of Directors of the Gauss Centre for Supercomputing (GCS) is pleased to award the

GCS AWARD 2020

to

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

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

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

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

Berlin, June 22, 2020

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

Dr. Claus Axel Müller
Managing Director, GCS

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

Very special thanks to…

Hatem Ltaief
Principal Research Scientist
Extreme Computing Research Center
KAUST


Closing haiku
Curse of dimension,
Can you be mitigated
By low rank's blessing?


Thank you!
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology
شكرا
david.keyes@kaust.edu.sa