Hierarchical Computations on Manycore Architectures:

The HiCMA Library

David Keyes, Applied Mathematics & Computational Science

Director, Extreme Computing Research Center (ECRC)

King Abdullah University of Science and Technology

david.keyes@kaust.edu.sa

→ Abbreviated and updated version of the archived "SIAM Presents" plenary

https://www.pathlms.com/siam/courses/4150/sections/5818

from SIAM CSE, 28 Feb 2017:

"Algorithmic Adaptations to Extreme Scale"

Two decades of evolution in hardware

1997

2017

ASCI Red at Sandia

1.3 TF/s, 850 KW

Cavium ThunderX2 (ARM)

~ 1.1 TF/s, ~ 0.2 KW

3.5 orders of magnitude

Supercomputer in a node

System	Peak DP TFlop/s	Peak Power KW	Power Efficiency GFlop/s/Watt
ASCI Red	1.3	850	0.0015
ThunderX2 Cavium (ARM)	1.1	0.20	5.5

Supercomputer in a node

System	Peak DP TFlop/s	Peak Power KW	Power Efficiency GFlop/s/Watt
ASCI Red	1.3	850	0.0015
ThunderX2 Cavium (ARM)	1.1	0.20	5.5*
Knights Landing Intel	3.5	0.26	14
P100 Pascal NVIDIA	5.3	0.30	18
Taihu Light CAS	125,000	15,000	8.3
Exascale System (~2021)	1,000,000	20,000	50

* 8 memory channels in Cavium ARM vs. 6 for Intel KNL

Architectural trends

- **Clock rates cease to increase while arithmetic capability continues to increase through concurrency (flooding of cores)**
- **Memory storage capacity increases, but fails to keep up with arithmetic capability *per core***
- **Transmission capability – memory BW and network BW – increases, but fails to keep up with arithmetic capability *per core***

Well established resource trade-offs

- **Communication-avoiding algorithms**
  - **exploit extra memory to achieve theoretical lower bound on communication volume**
- **Synchronization-avoiding algorithms**
  - **perform extra flops between global reductions or exchanges to require fewer global operations**
- **High-order discretizations**
  - **perform more flops per degree of freedom (DOF) to store and manipulate fewer DOFs**

→ **Billions of**

$ € £ ¥

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

Architectural background

www.exascale.org/iesp

INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP 1.0

Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi

Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Yutaka Ishikawa, Fred Johnson

Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr

Matthias Mueller, Wolfgang Nagel, Hiroshi Nakashima, Michael E. Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz

Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad van der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, Kathy Yelick

SPONSORS

Office of Science U.S. Department of Energy, NSF, ANR, cea, CERFACS, CRAY THE SUPERCOMPUTER COMPANY, EDF, EPSRC Engineering and Physical Sciences Research Council, FUJITSU, INRIA, GENCI, NVIDIA, RIKEN, 東京大学 THE UNIVERSITY OF TOKYO

The International Exascale Software Roadmap

J. Dongarra, P. Beckman, et al., *International Journal of High Performance Computer Applications* **25**:3-60, 2011.

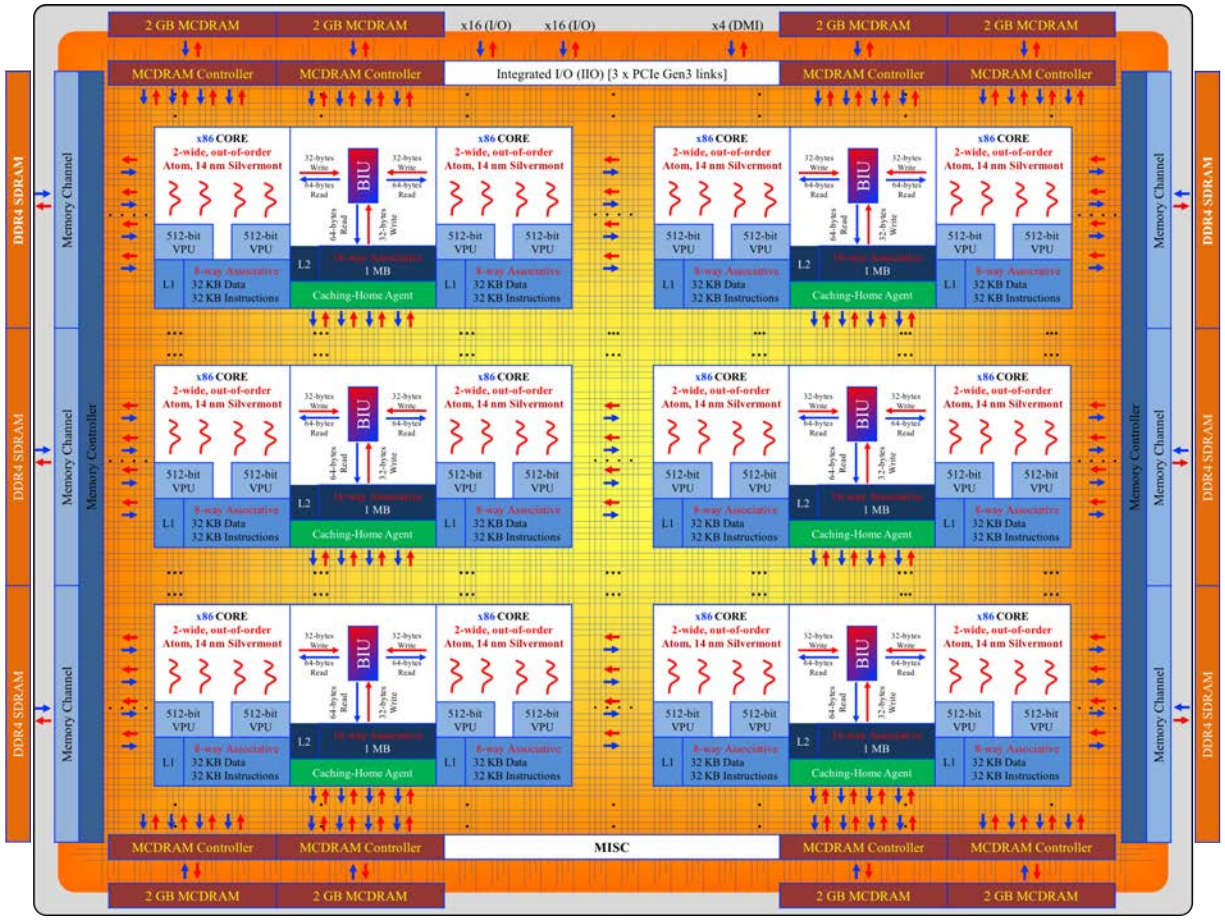
Uptake from IESP meetings

- While obtaining the next order of magnitude of performance, we need another order of performance efficiency
  - target: 50 Gigaflop/s/W, today typically ~ 5 Gigaflop/s/W
- Required reduction in power per flop and per byte may make computing and moving data less reliable
  - circuit elements will be smaller and subject to greater physical noise per signal, with less space redundancy and/or time redundancy for resilience in the hardware
- Power may be cycled off and on, or clocks slowed and speeded
  - may be scheduled, based on phases with different power requirements, or may be dynamic from thermal monitoring
- Performance rates less reliable

Node-based "weak scaling" is routine; thread-based "strong scaling" is the game

- An exascale configuration: 1 million 1000-way 1GHz nodes
- Expanding the number of nodes (processor-memory units) beyond 10^6 would *not* be a serious threat to algorithms that lend themselves to well-amortized precise load balancing
  - provided that the nodes are performance reliable
- Real challenge is usefully expanding the number of cores sharing memory on a node to 10^3
  - must be done while memory and memory bandwidth per node expand by (at best) ten-fold less (basically "strong" scaling)

⟶ Don't need to wait for full exascale systems to experiment in this regime…



The contest is being waged on individual shared-memory nodes today

The familiar

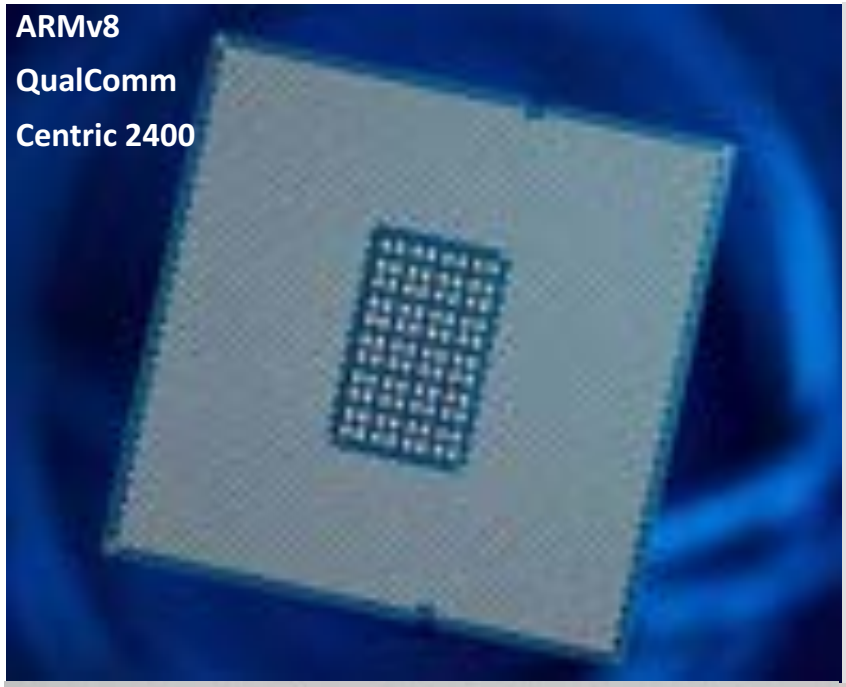
Taihu Light

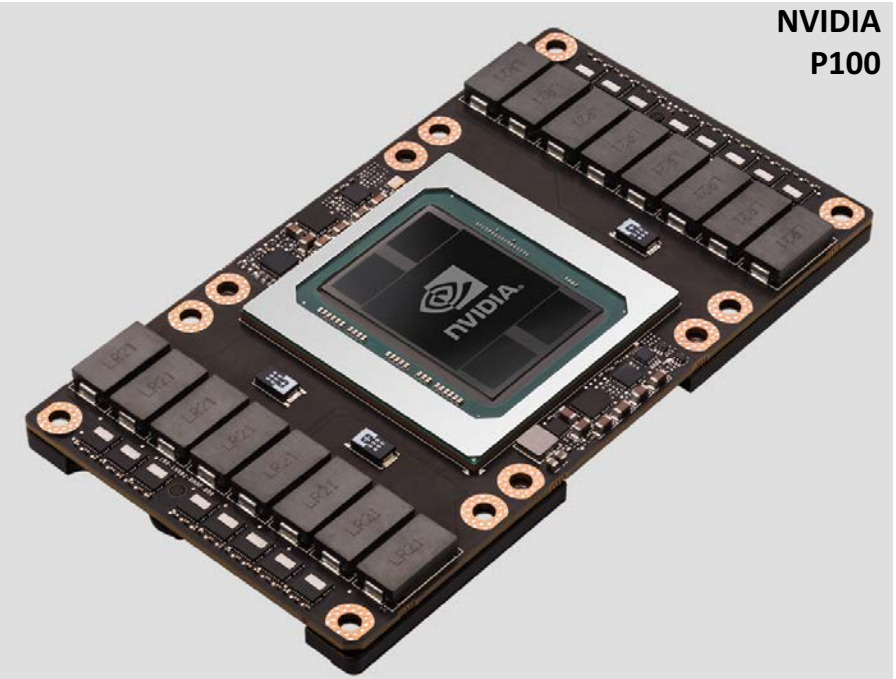
Shaheen

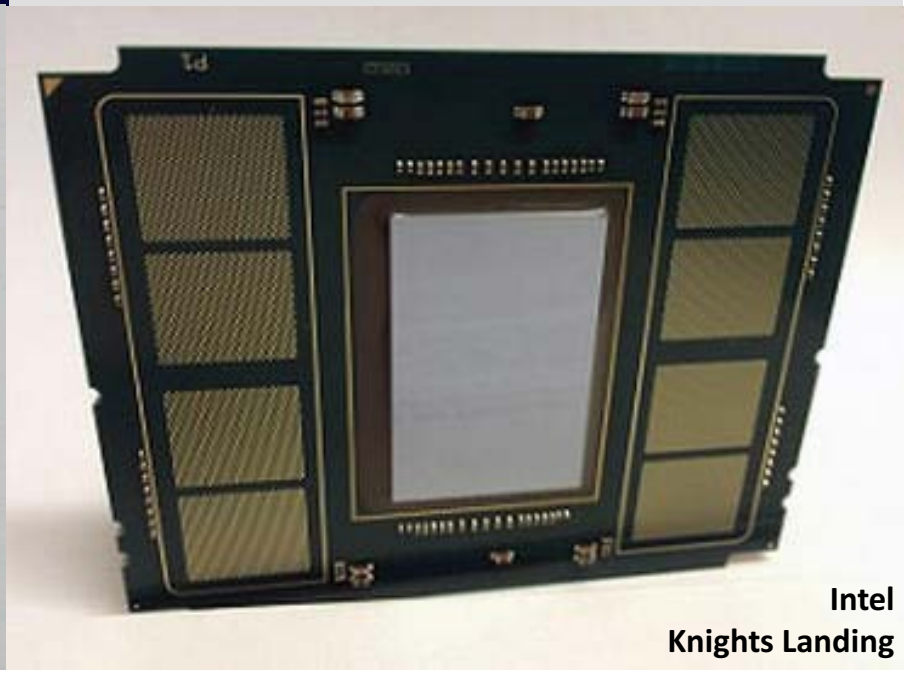
Sequoia

K

The challenge







How are most scientific simulations implemented at the petascale today?

- **Iterative methods based on data decomposition and message-passing**
  - data structures are distributed
  - each individual processor works on a subdomain of the original
  - exchanges information with other processors that own data with which it interacts causally, to evolve in time or to establish equilibrium
  - computation and neighbor communication are both fully parallelized and their ratio remains constant in weak scaling
- **The programming model is BSP/SPMD/CSP**
  - Bulk Synchronous Programming
  - Single Program, Multiple Data
  - Communicating Sequential Processes

Three decades of stability in programming model

Bulk Synchronous Parallelism

Leslie Valiant, F.R.S., N.A.S.
2010 Turing Award Winner

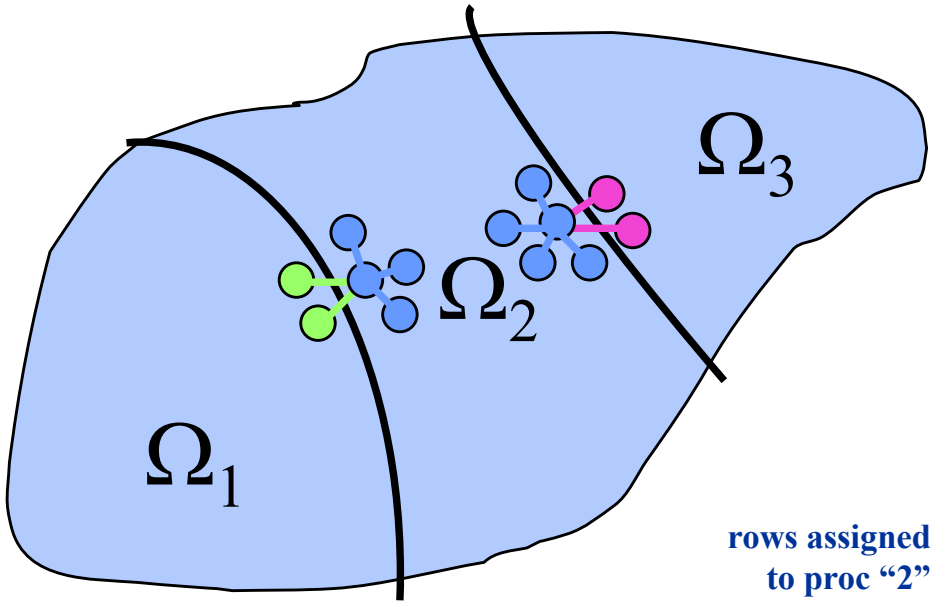
A Bridging Model for Parallel Computation

The success of the von Neumann model of sequential computation is attributable to the fact that it is an efficient bridge between software and hardware: high-level languages can be efficiently compiled on to this model; yet it can be efficiently implemented in hardware. The author argues that an analogous bridge between software and hardware is required for parallel computation if that is to become as widely used. This article introduces the bulk-synchronous parallel (BSP) model as a candidate for this role, and gives results quantifying its efficiency both in implementing high-level language features and algorithms, as well as in being implemented in hardware.

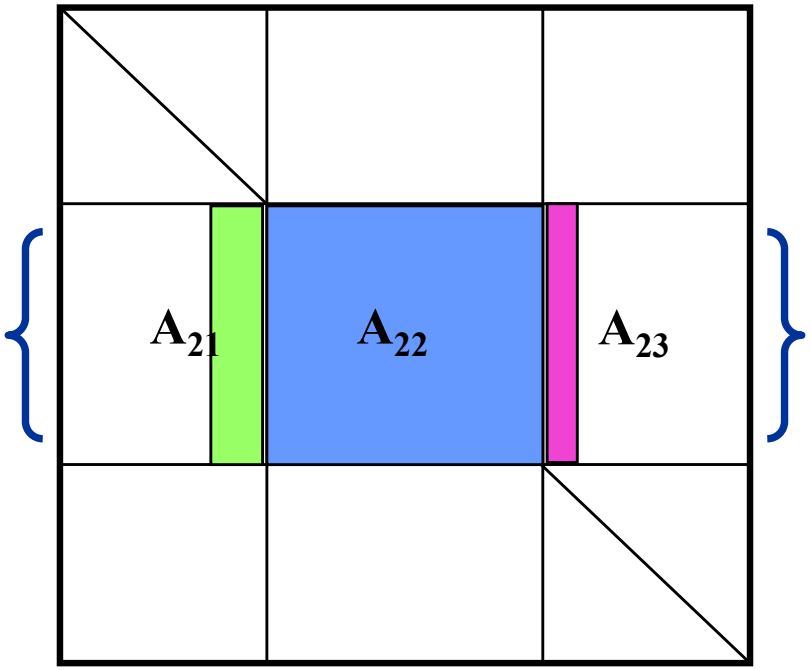
Leslie G. Valiant

Comm. of the ACM, 1990

BSP parallelism w/ domain decomposition



rows assigned
to proc "2"



Partitioning of the grid induces block structure on the system matrix (Jacobian)

BSP has an impressive legacy

By the Gordon Bell Prize, performance on *real applications* (e.g., mechanics, materials, petroleum reservoirs, etc.) has improved *more* than a million times in two decades. Simulation *cost per performance* has improved by nearly a million times.

Gordon Bell Prize: Peak Performance Year	Gigaflop/s delivered to applications
1988	1
1998	1,020
2008	1,350,000

Gordon Bell Prize: Price Performance Year	Cost per delivered Gigaflop/s
1989	$2,500,000
1999	$6,900
2009	$8

Riding exponentials

- **Proceeded steadily for decades from giga- (1988) to tera- (1998) to peta- (2008) with**
  - ***same* BSP programming model**
  - ***same* assumptions about who (hardware, systems software, applications software, etc.) is responsible for what (resilience, performance, processor mapping, etc.)**
  - ***same* classes of algorithms (*cf.* 25 yrs. of Gordon Bell Prizes)**
- **Scientific computing now at a crossroads with respect to extreme scale**

Main challenge going forward for BSP

- **Almost all "good" algorithms in linear algebra, differential equations, integral equations, signal analysis, etc., like to globally synchronize – and frequently!**
  - **inner products, norms, pivots, fresh residuals are "addictive" idioms**
  - **tends to hurt efficiency beyond 100,000 processors**
  - **can be fragile for smaller concurrency, as well, due to algorithmic load imbalance, hardware performance variation, etc.**
- **Concurrency is heading into the billions of cores**
  - **already 10 million on the most powerful system today**


Energy-aware generation
BSP generation

Some algorithmic imperatives

- **Reduce communication and synchrony**
  - **in frequency and/or span**
  - **see Grigori, Jolivet, Chow in this MS 7**
- **Reside "high" on the memory hierarchy**
  - **as close as possible to the processing elements**
  - **see Li, Schlatter, Dubey, Bader in this MS 7/16**
- **Increase SIMT/SIMD-style shared-memory concurrency**
  - **see many of the above**

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**
2) **Exploit data sparsity of hierarchically low-rank type**
   - **meet the "curse of dimensionality" with the "blessing of low rank"**
3) **Code to the architecture, but present an abstract API**

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 (↑ 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**

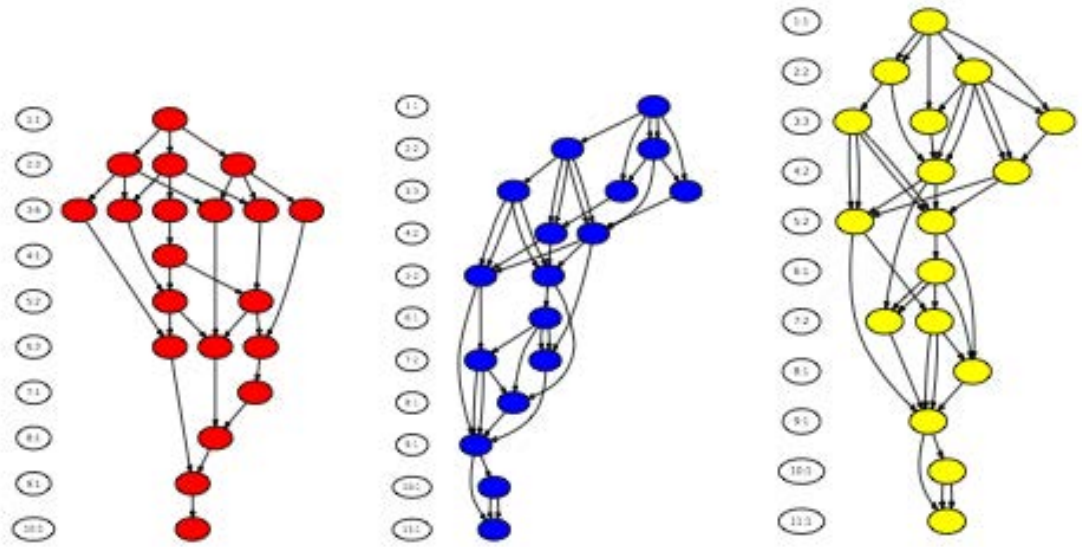
3) Code to the architecture

- **Advantages**
  - **tiling and recursive subdivision create large numbers of small problems suitable for batched operations on GPUs and MICs**
    - **reduce 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**

Reducing over-ordering and synchronization through DAGs, ex.: generalized eigensolver

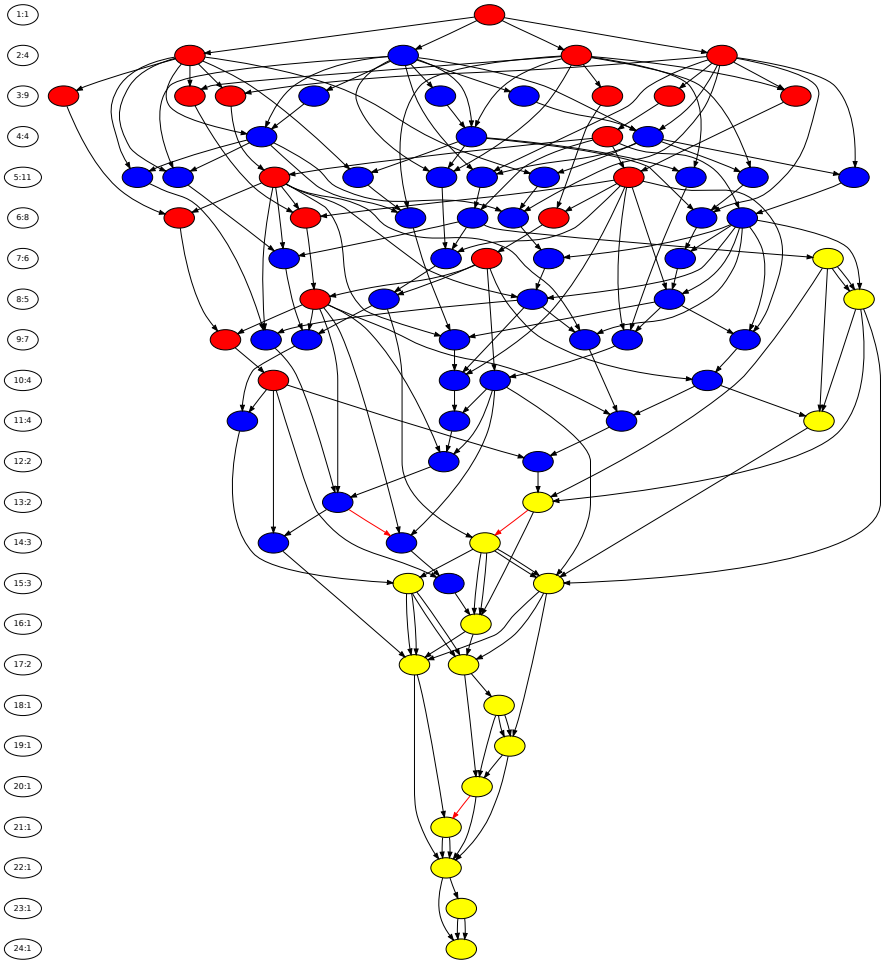
$$Ax = \lambda Bx$$

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



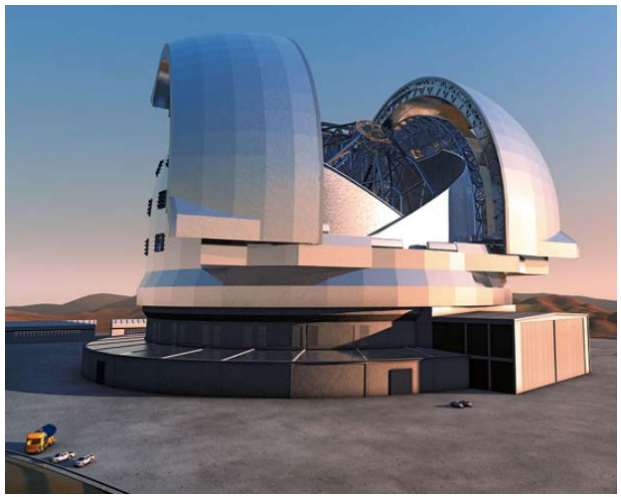
Loop nests and subroutine calls, with their over-orderings, can be replaced with DAGs

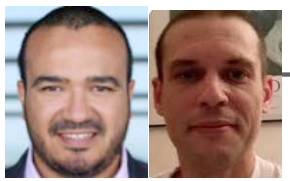
- Diagram shows a dataflow ordering of the steps of a 4×4 symmetric generalized eigensolver
- Nodes are tasks, color-coded by type, and edges are data dependencies
- Time is vertically downward
- Wide is good; short is good



Loops can be overlapped in time

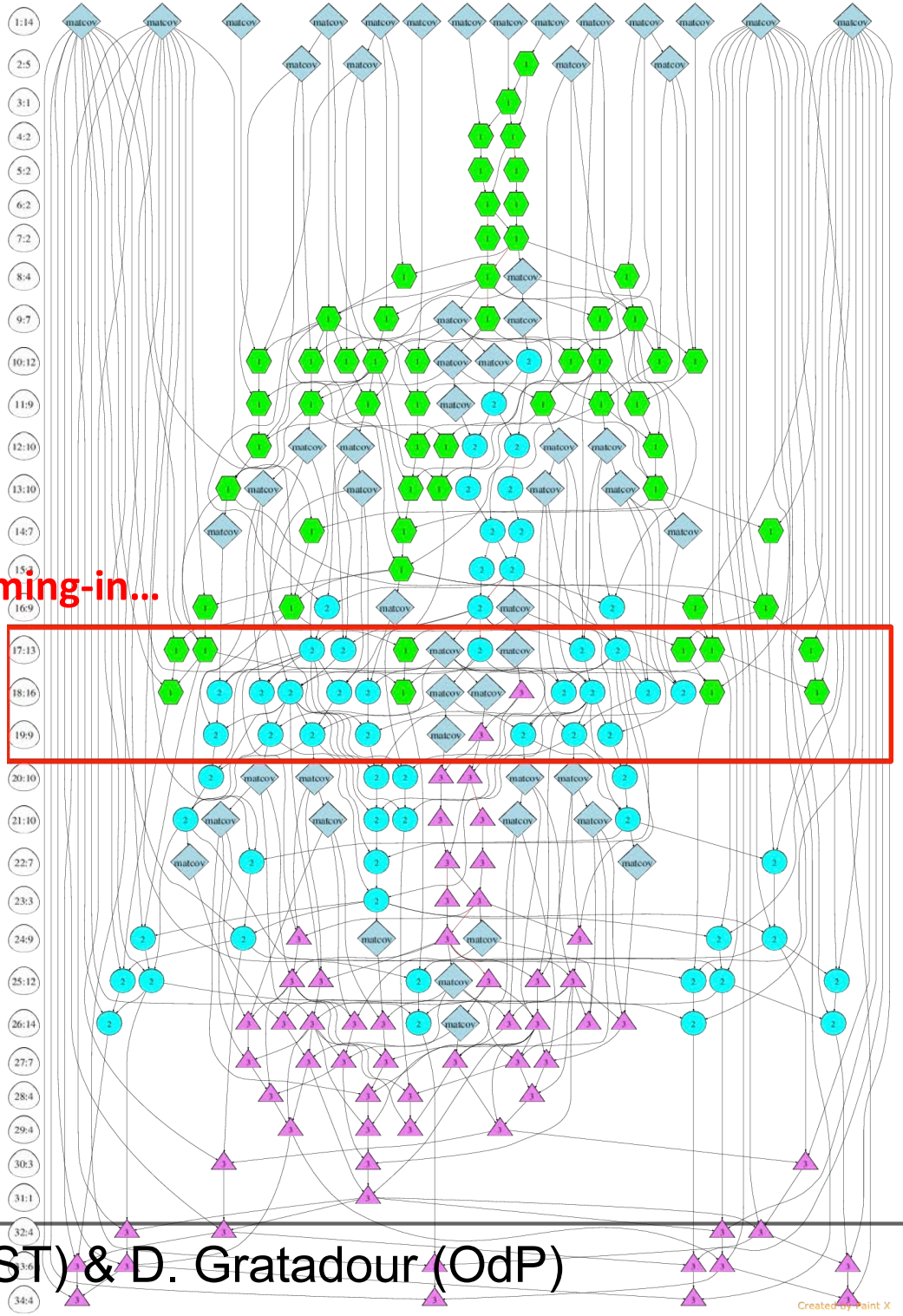
Green, blue and magenta symbols represent tasks in separate loop bodies with dependences from an adaptive optics computation





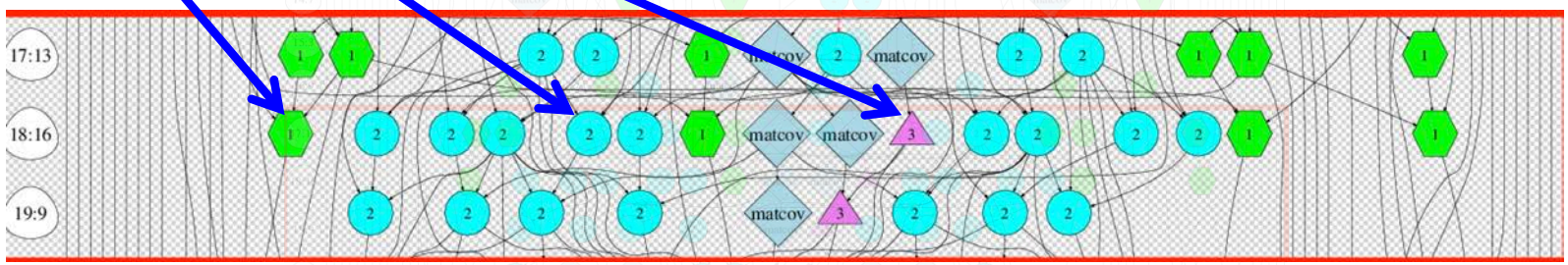
c/o H. Ltaief (KAUST) & D. Gratadour (OdP)

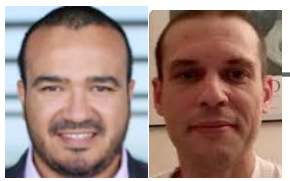
Zooming-in…



DAG-based safe out-of-order execution

Tasks from 3 loops of optical "reconstructor" pipeline are executed together





c/o H. Ltaief (KAUST) & D. Gratadour (OdP)

Reducing memory footprint and operation complexity with low rank

- When dense blocks arise in matrix operations, replace them with hierarchical representations
- Use high accuracy (high rank, but typically less than full) to build “exact” solvers
- Use low accuracy (low rank) to build preconditioners
- Block structure and rank provide useful tuning parameters for migration onto variety of hardware configurations

Key tool: hierarchical matrices

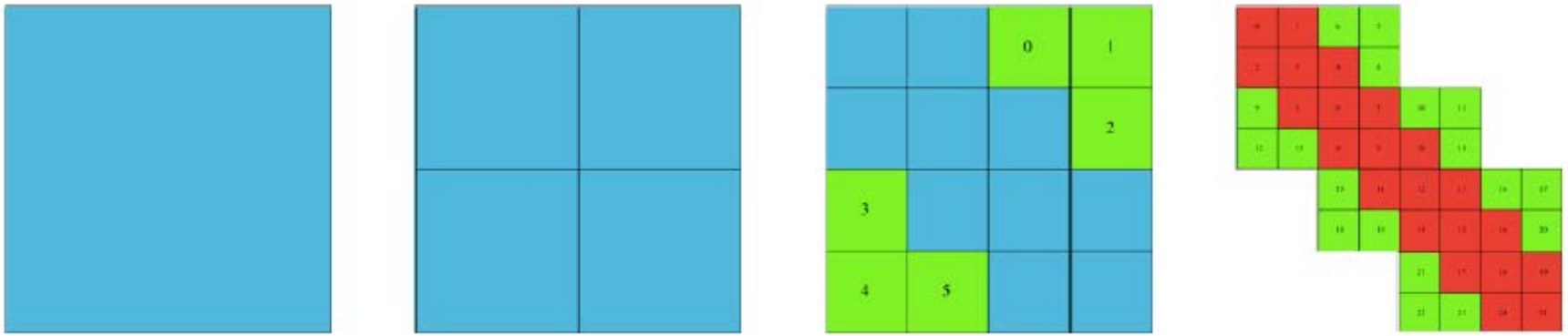
- [Hackbusch, 1999] **: off-diagonal blocks of typical differential and integral operators have low effective rank**
- **By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n:**
  - **polynomial in k**
  - **lin-log in n**
  - **constants carry the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

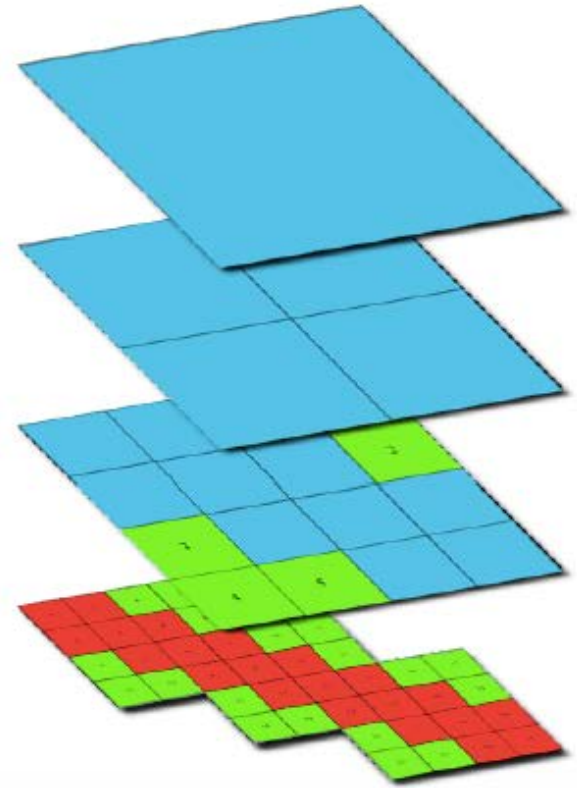
Example: 1D Laplacian

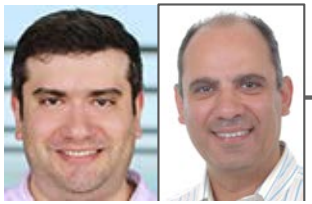
$$A = \left[\begin{array}{ccc|cccc} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ \hline & & & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{array}\right] \leftrightarrow = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0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

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

Recursive construction of an *H*-matrix

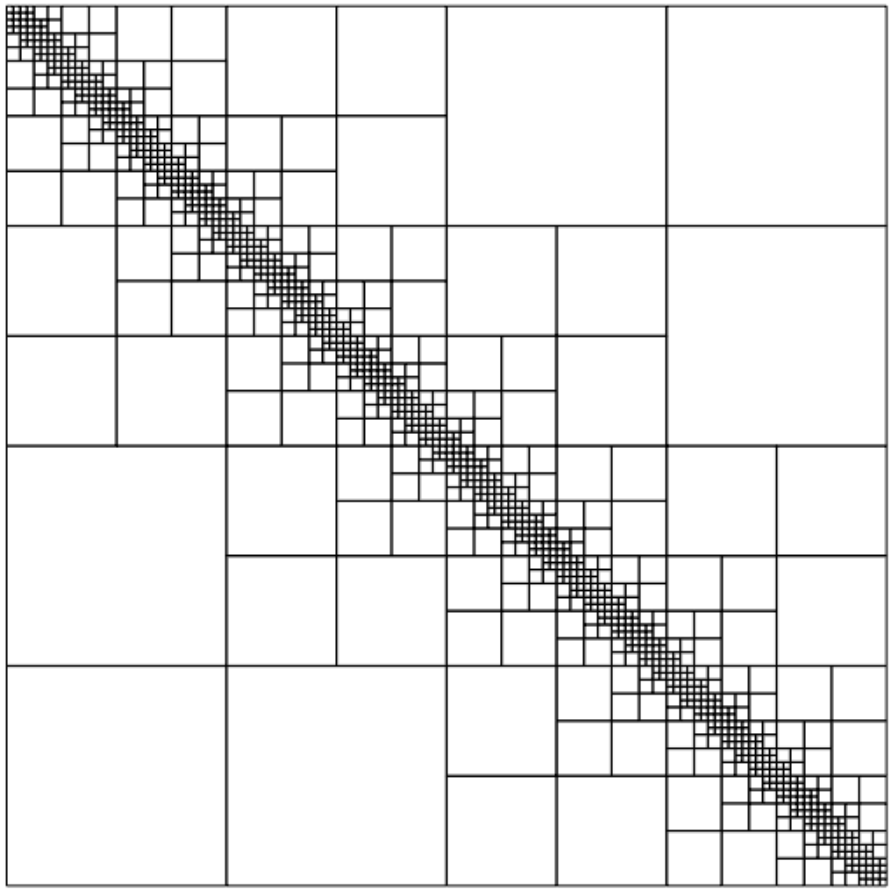




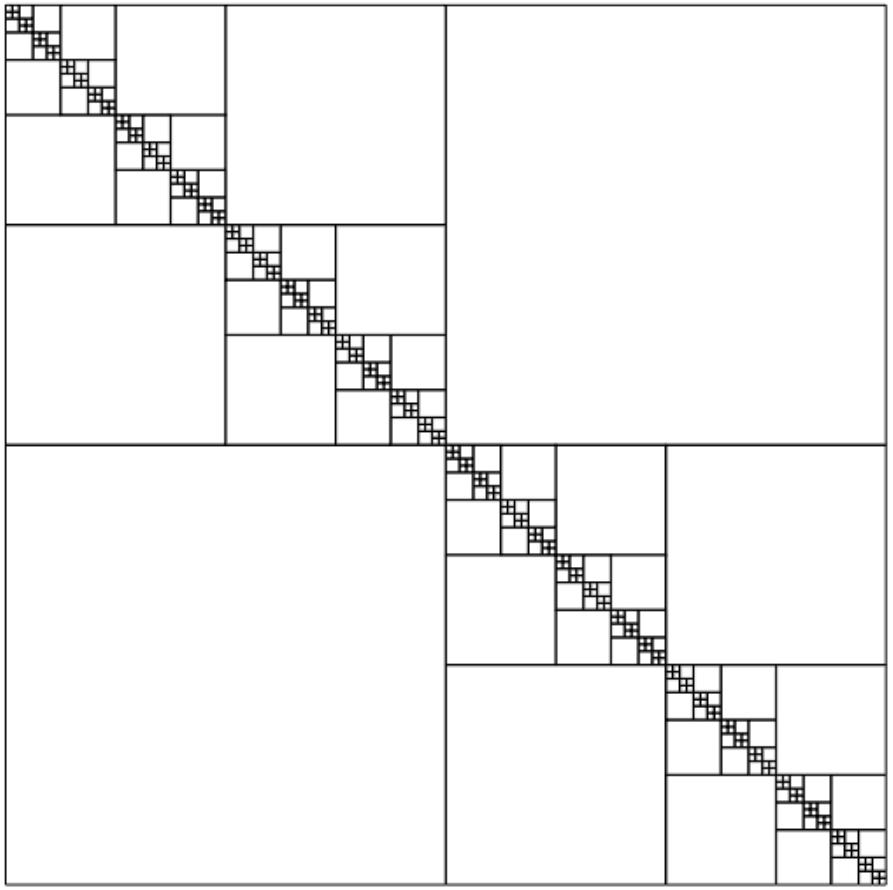


c/o W. Boukaram & G. Turkiyyah (KAUST)

“Standard (strong)” vs. “weak” admissibility



strong admissibility



weak admissibility

After Hackbusch, et al., 2003

Some solvers that leverage data sparsity

Package	Technique	Format	Contact
ACR	Cyclic reduction	$\mathcal{H}$	G. Chávez, G. Turkiyyah D. Keyes [118]
AHMED	$\mathcal{H}^{-1}$ & $\mathcal{H}$-LU	$\mathcal{H}$	M. Bebendorf [120]
BLR PaStiX	Supernodal	BLR	G. Pichon M. Faverge [44]
CE	$\mathcal{H}^2$-LU	$\mathcal{H}^2$	D. Sushnikova I. Oseledets [57]
DMHIF	Multifrontal	ID	Y. Li L. Ying [43]
DMHM	Newton-Schulz	$\mathcal{H}$	Y. Li L. Ying [121]
ExaFMM	Fast multipole	$\mathcal{H}$	H. Ibeid, R. Yokota D. Keyes [122]
H2Lib	$\mathcal{H}^{-1}$ & $\mathcal{H}^2$-LU	$\mathcal{H}^2$	S. Christophersen S. Böerm [123]
HLib	$\mathcal{H}^{-1}$ & $\mathcal{H}$-LU	$\mathcal{H}$	L. Grasedyck W. Hackbusch [124]
HLibPro	$\mathcal{H}^{-1}$ & $\mathcal{H}$-LU	$\mathcal{H}$	R. Kriemann W. Hackbusch [54]
LoRaSp	$\mathcal{H}^2$-LU	$\mathcal{H}^2$	H. Pouransari E. Darve [56]
MF-HODLR	Multifrontal	HODLR	A. Aminfar E. Darve [30]
MUMPS-BLR	Multifrontal	BLR	T. Mary P. R. Amestoy [22]
Structured CHOLMOD	Supernodal	BLR	J. Chadwick D. Bindel [46]
STRUMPACK-Sparse	Multifrontal	HSS	F.-H. Rouet, P. Ghysels X.S. Li [36]

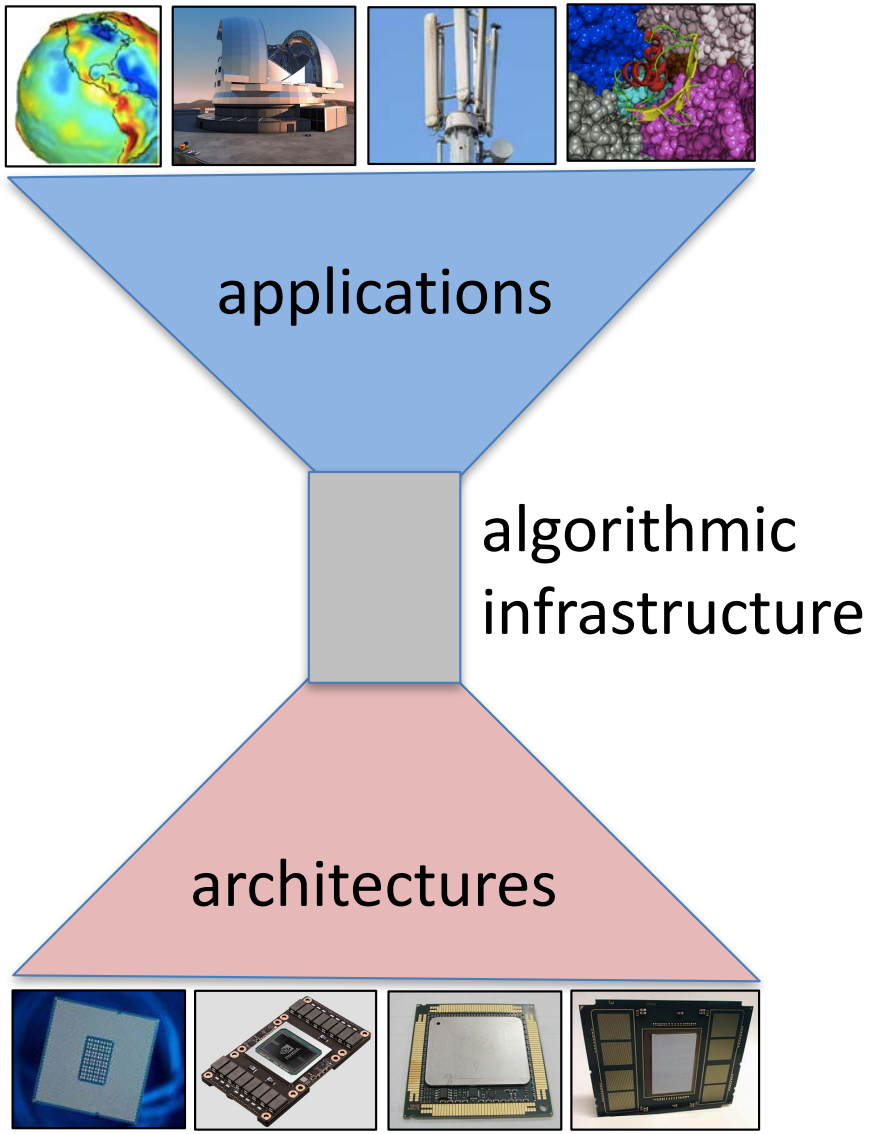
Please notify if you have released one that is not here:

gustavo.chavez

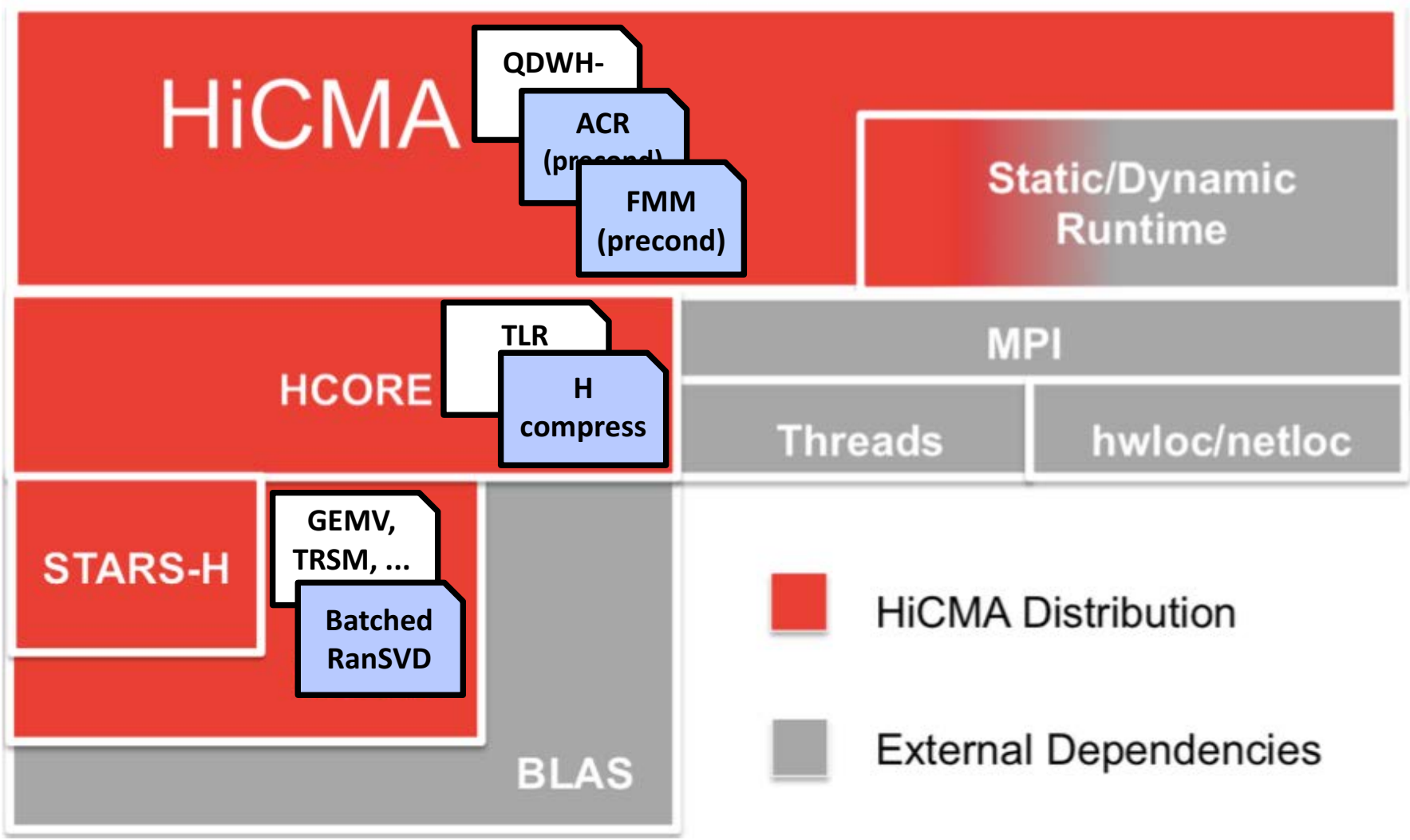
@kaust.edu.sa

c/o G. Chavez (KAUST)

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



Hierarchical Computations on Manycore Architectures: HiCMA*



* "Hikmah" is the Arabic word for wisdom

QDWH*-EVD/SVD

- **DAG-based dataflow tile algorithms for (eigen- and) singular value decomposition**
- **Reduces synchrony**
- **Increases SIMT-style concurrency through recursion**
- **Employs Chameleon tile library and StarPU dynamic runtime system**

*QR-based Dynamically Weighted Halley iteration from

Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD,
Y. Nakatsukasa & N. Higham, SISC (2013)

Asynchronous Task-Based Polar Decomposition on Massively Parallel Systems,
D. Sukkari, H. Ltaief, M. Faverge & D. Keyes, IEEE TPDS (2017)

QDWH-SVD

- **Obtain SVD from a polar decomposition:**

polar

$$A = U_p H$$

sym eigen

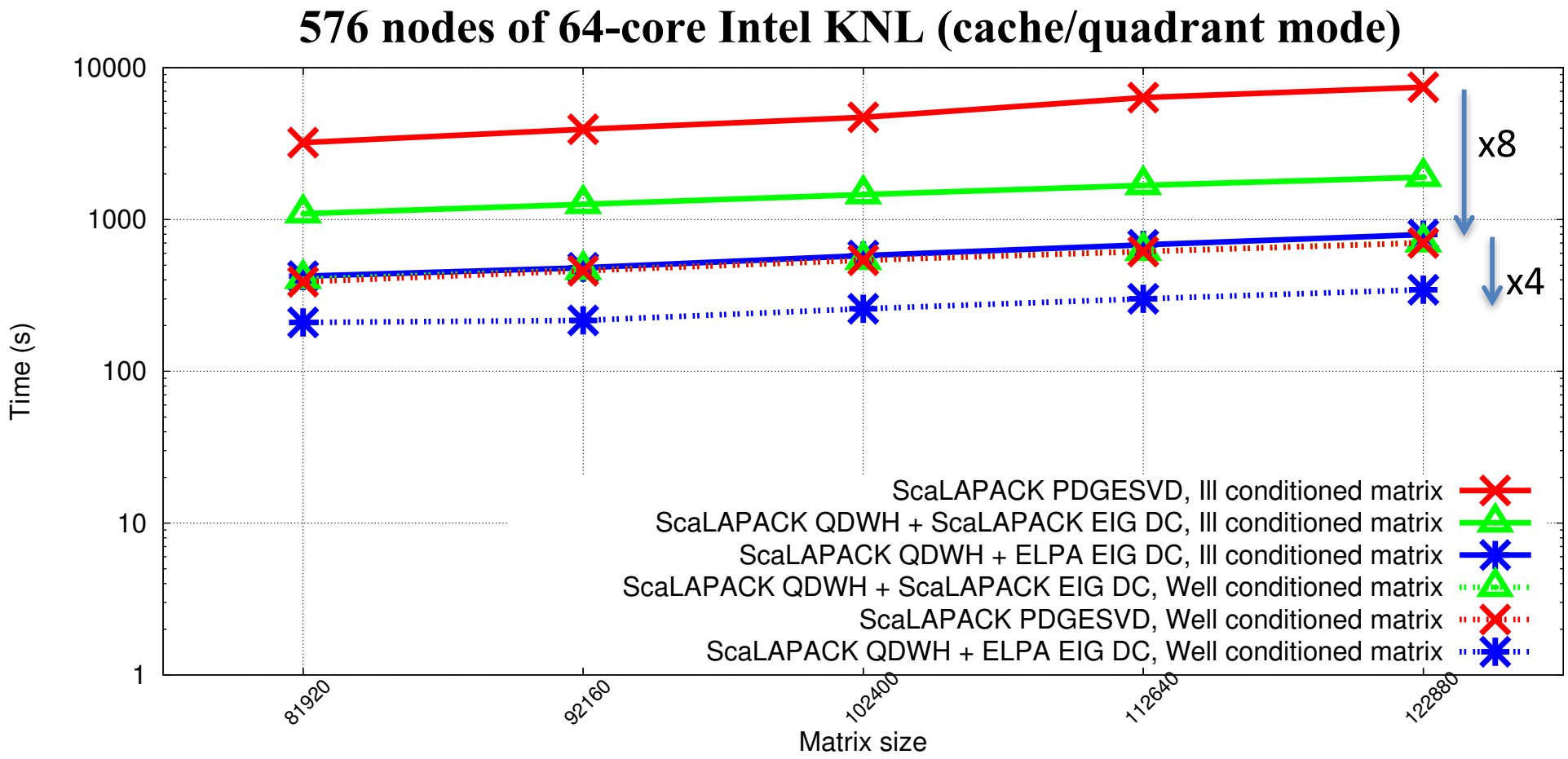
$$H = V \Sigma V^*$$

$$\rightarrow A = U_p V \Sigma V^* = U \Sigma V^*$$

- **QDWH iteration is a recursive divide-and-conquer method, backward stable**
- **Based on vendor-optimized kernels, i.e., Cholesky/QR factorizations and GEMM**
- **Complexity:**

  (10+2/3) n^3 for well-conditioned system, $43n^3$ for ill

QDWH-SVD



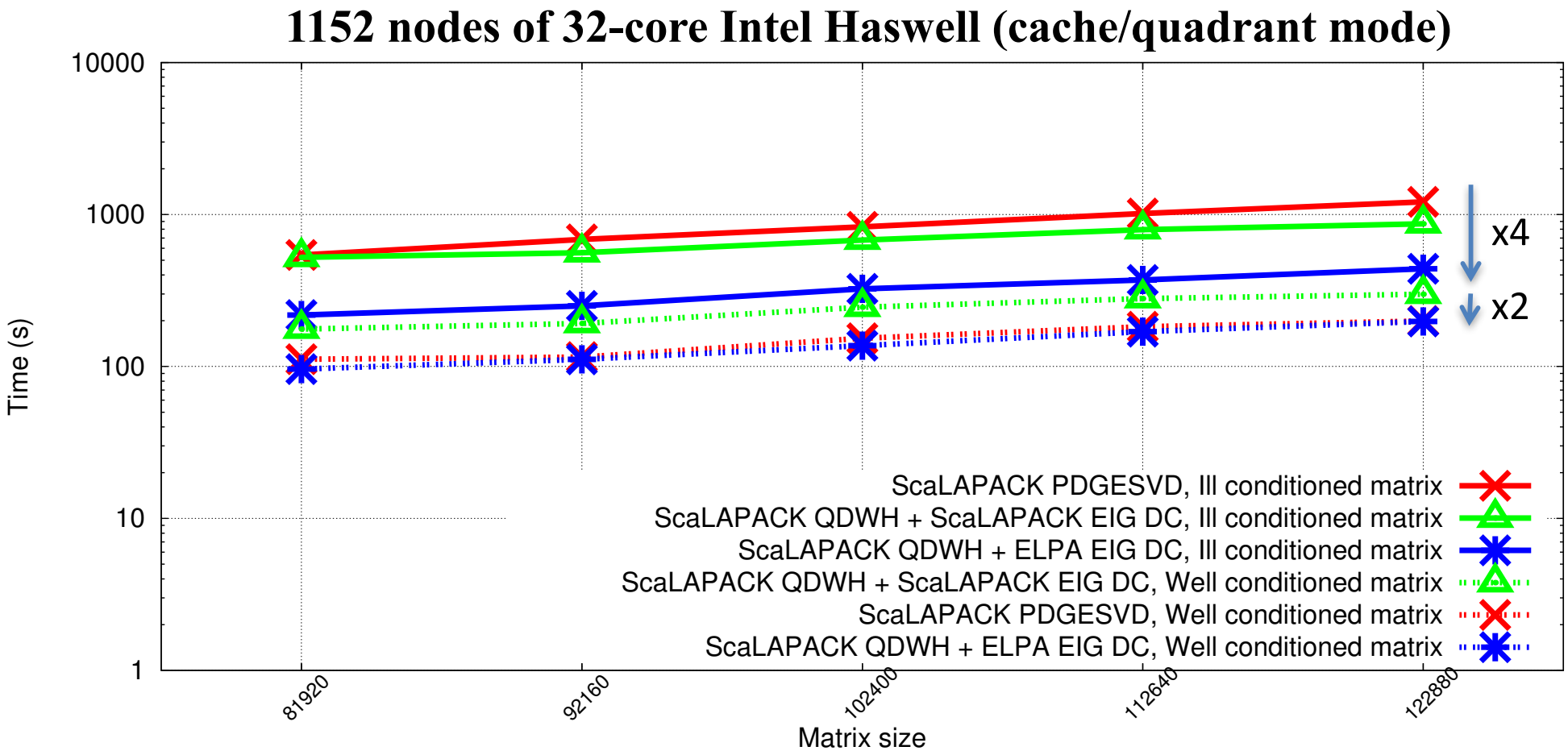
fastest dense SVD

c/o D. Sukkari & H. Ltaief (KAUST)

Sukkari et al., Best papers, Europar'16

available: https://github.com/ecrc/qdwh.git

QDWH-SVD

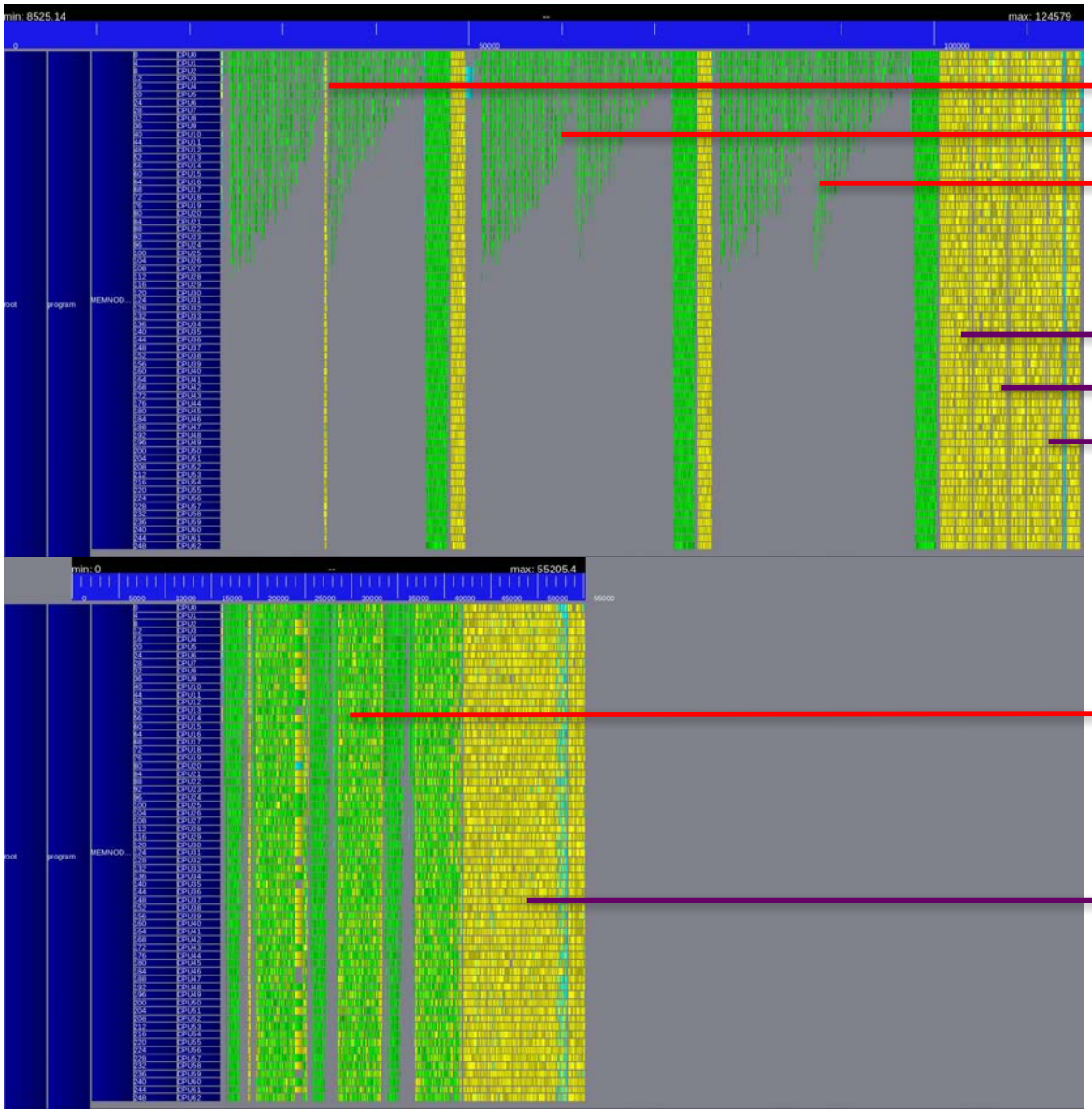


Is being integrated into Cray's LibSci w/A. Esposito (Cray)
Extensions underway to Zolotarev's method w/Y. Nakatsukasa (Oxford)

c/o D. Sukkari & H. Ltaief (KAUST)

Sukkari et al., Best papers, Europar'16
available: https://github.com/ecrc/qdwh.git

QDWH-SVD, taskified



1st QR iteration
2nd QR iteration
3rd QR iteration

1st Cholesky iteration
2nd Cholesky iteration
3rd Cholesky iteration

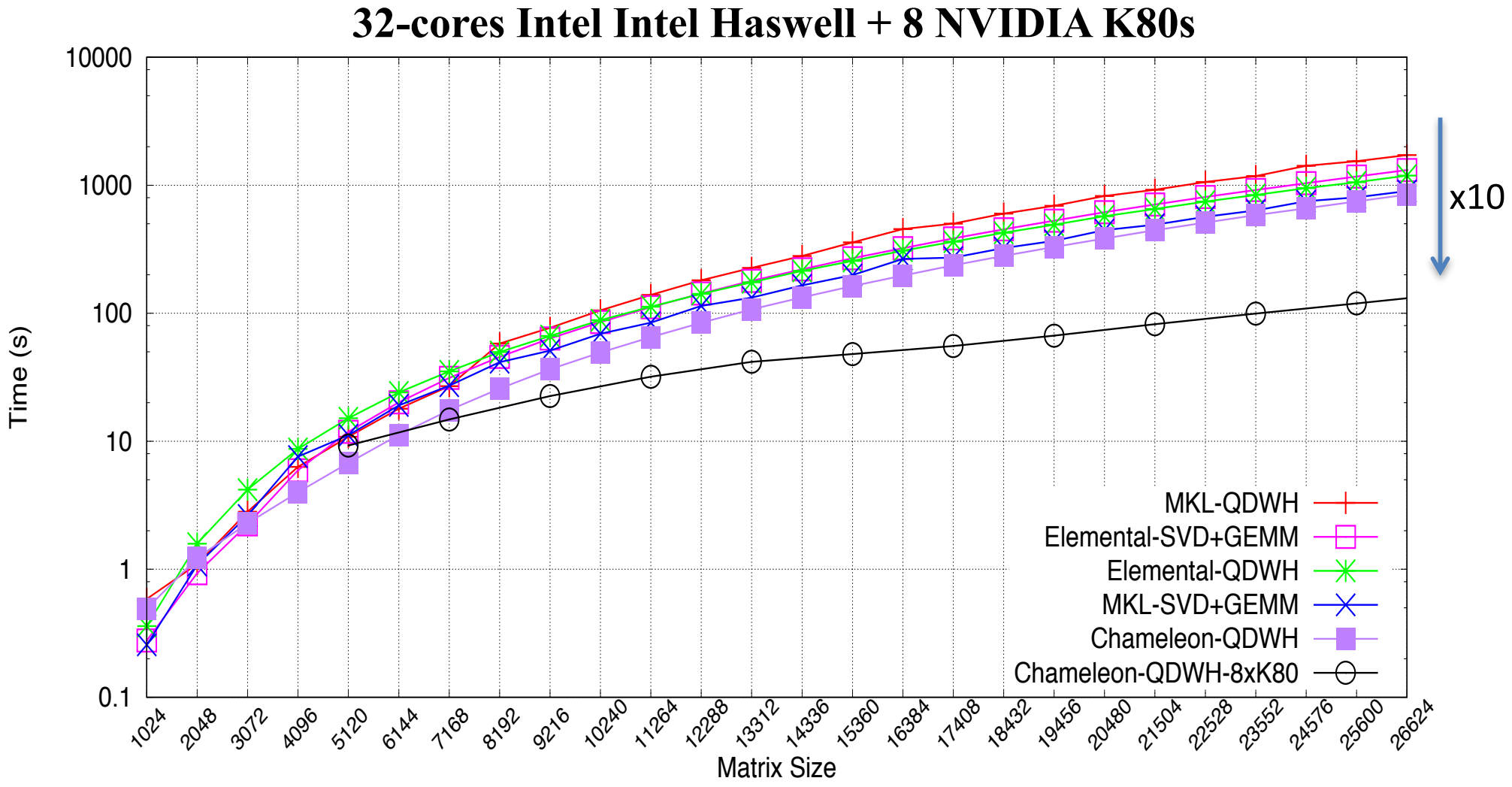
Three QR iterations

Three Cholesky iterations

Sukkari et al., IEEE TDPS'17

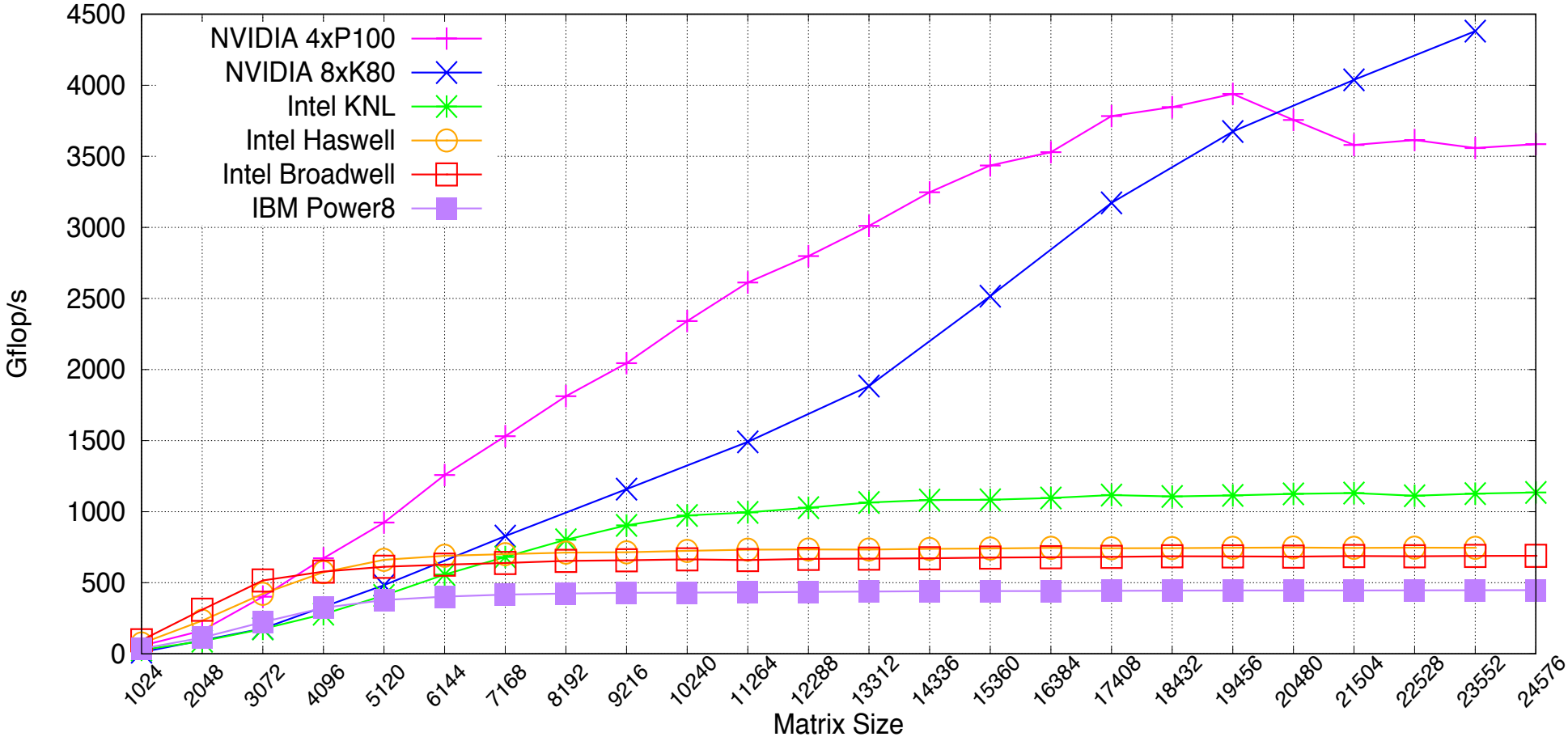
c/o D. Sukkari, H. Ltaief (KAUST) & M. Faverge (INRIA)

QDWH-SVD, taskified on hybrid architecture



c/o D. Sukkari, H. Ltaief (KAUST) & M. Faverge (INRIA)

QDWH-SVD, taskified on various architectures



c/o D. Sukkari, H. Ltaief (KAUST) & M. Faverge (INRIA)

Tile Low-rank Cholesky

- **A low-rank, but flat (not hierarchical) first step towards expanding capability for large dense symmetric problems, e.g., covariance matrices**
- **Reduces synchrony**
- **Increases SIMT-style concurrency**
- **Employs OpenMP taskification pragmas and HLibPro on individual tiles**

ExaGeoStat: A High Performance Unified Framework for Geostatistics on Manycore Systems

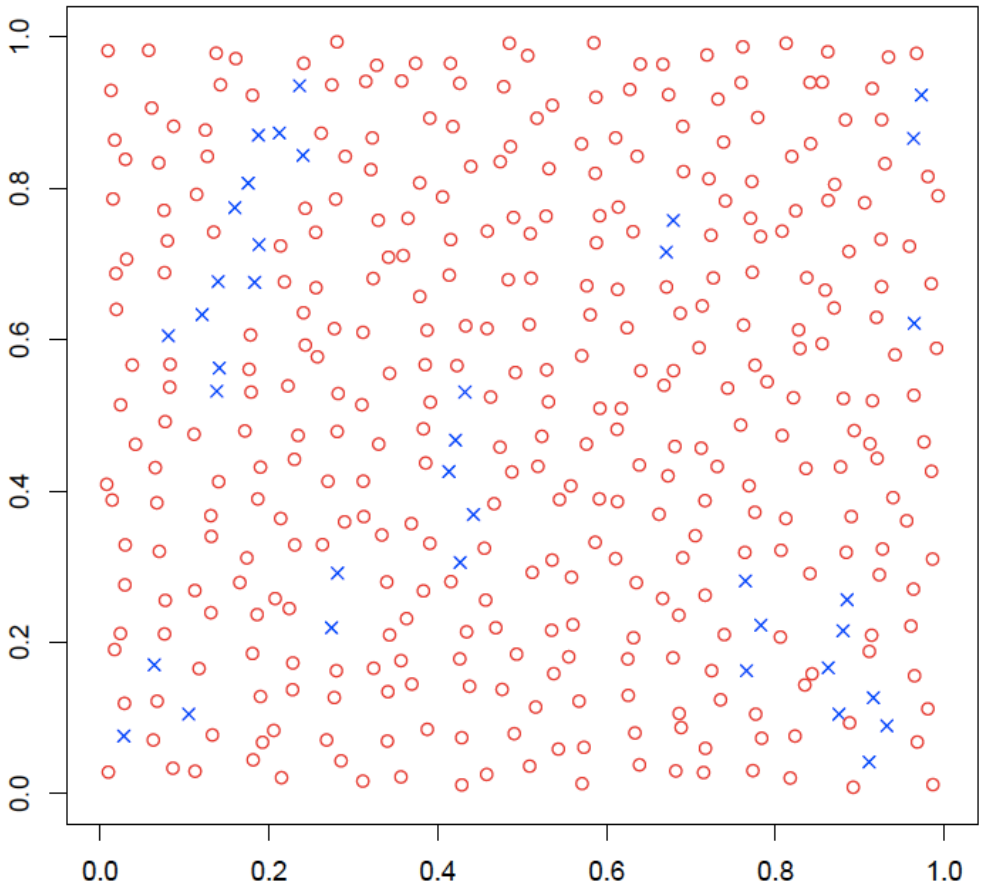
S. Abdulah, H. Ltaief, Y. Sun, M. Genton & D. Keyes
TDPS (2017, submitted)

Large dense symmetric systems arise as covariance matrices in spatial statistics

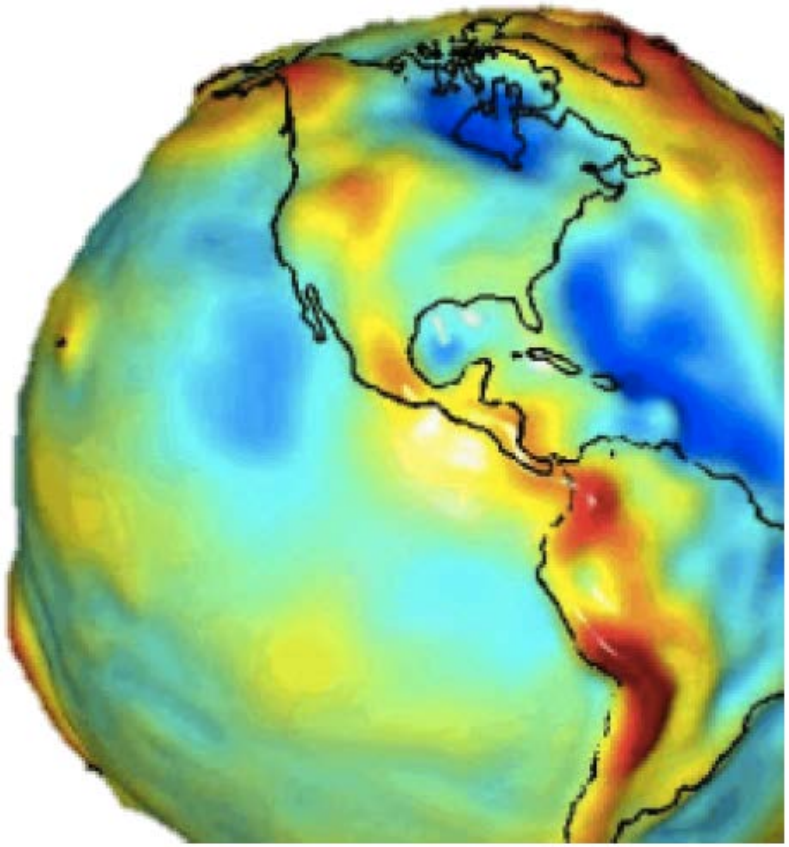
- Climate and weather applications have many measurements located regularly or irregularly in a region; prediction is needed at other locations
- Modeled as realization of Gaussian or Matérn spatial random field, with parameters to be fit
- Leads to evaluating the log-likelihood function involving a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

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

Synthetic and practical examples



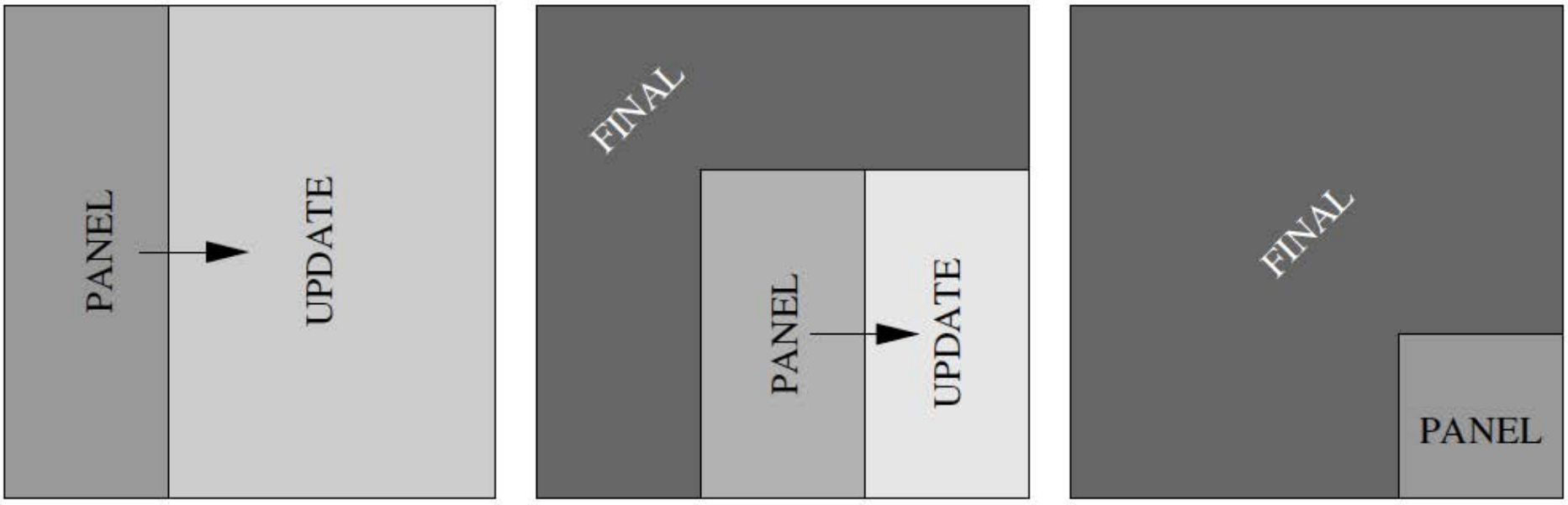
362 measured points and 38 target points irregularly distributed in unit square



Global temperature data on sphere

LAPACK DPOTRF

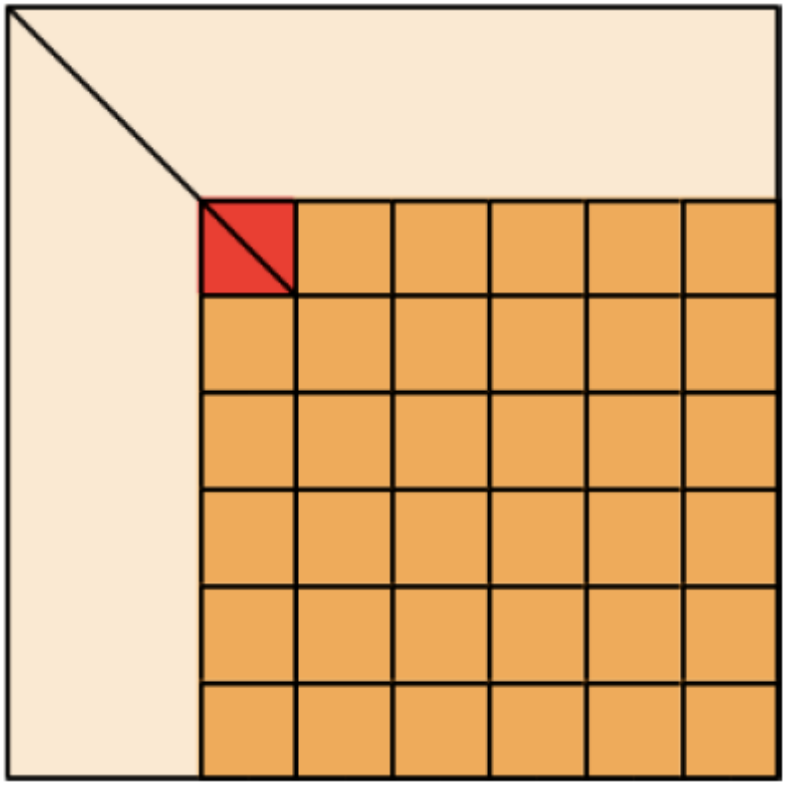
- **Classical algorithm (1990s) involves BLAS L2 panel updates and BLAS L3 trailing matrix updates**

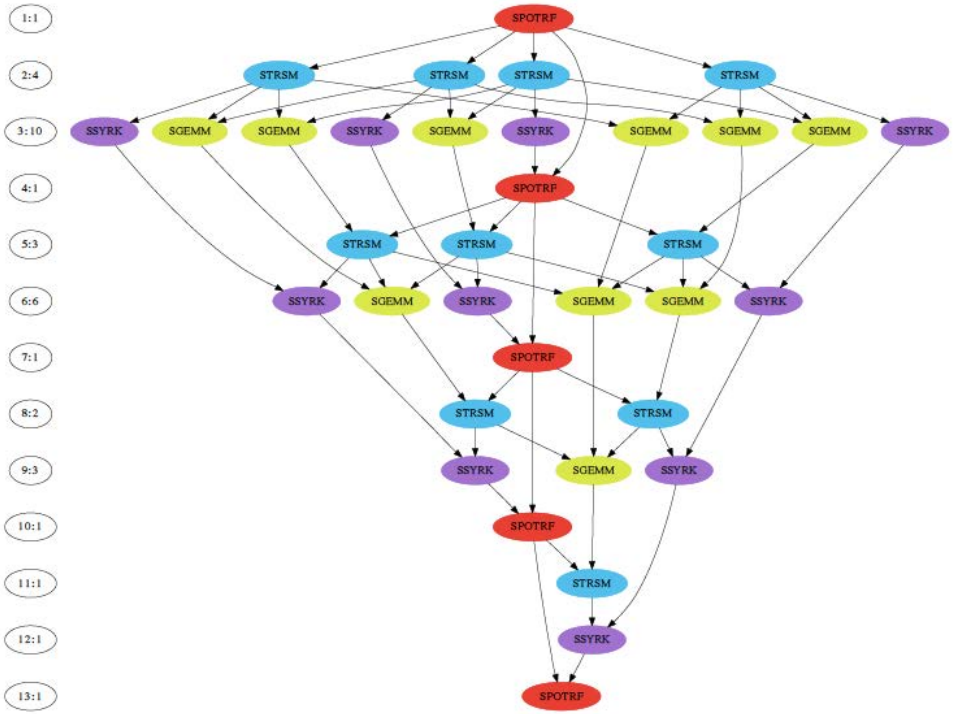


(a) First step. (b) Second step. (c) Third step.

PLASMA/CHAMELEON DPOTRF

- **Tile algorithm (PLASMA, FLAME, 2010s) involves mostly BLAS L3 operations within tiles scheduled with a DAG**

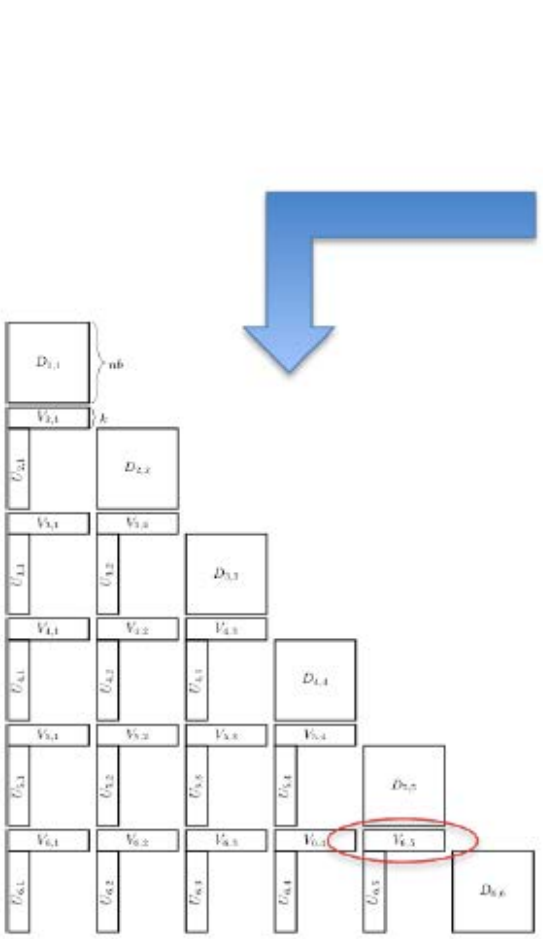


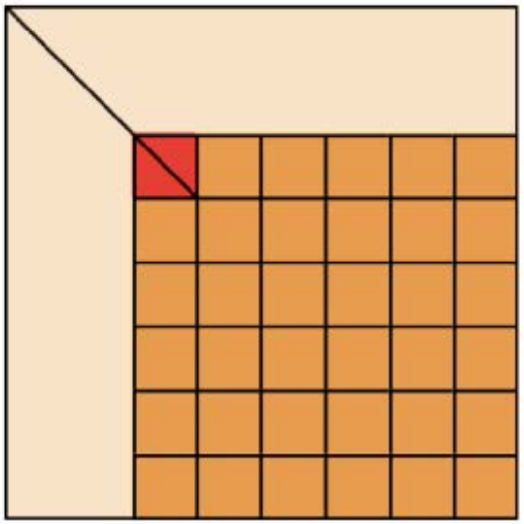


Tile operations for TLR version of Cholesky

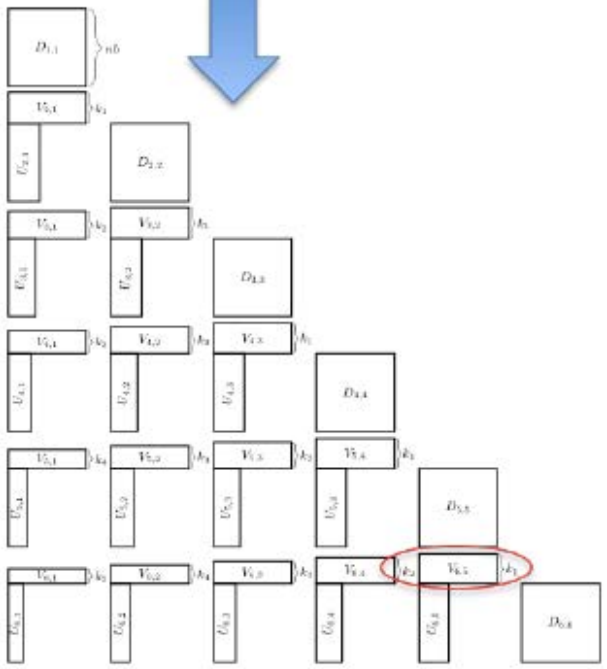
- DPOTRF: The kernel performs the Cholesky factorization of a diagonal (lower triangular) tile. It is similar to DPOTRF since the diagonal tiles are dense.
- DTRSM: The operation applies an update to an off-diagonal low-rank tile of the input matrix, resulting from factorization of the diagonal tile above it and overrides it with the final elements of the output matrix: $V_{(i,k)} = V_{(i,k)} \times D^{-1}_{(k,k)}$. The operation is a triangular solve.
- DSYRK: The kernel applies updates to a diagonal (lower triangular) tile of the input matrix, resulting from factorization of the low-rank tiles to the left of it: $D_{(j,j)} = D_{(j,j)} - (U_{(j,k)} \times V^T_{(j,k)}) \times (U_{(j,k)} \times V^T_{(j,k)})^T$. The operation is a symmetric rank-k update.
- DGEMM: The operation applies updates to an off-diagonal low-rank tile of the input matrix, resulting from factorization of the low-rank tiles to the left of it. The operation involves two QR factorizations, one reduced SVD (depending on the rank and/or the accuracy parameter) and two matrix-matrix multiplications.

Data-sparse operations for Cholesky variants





Dense
tiles



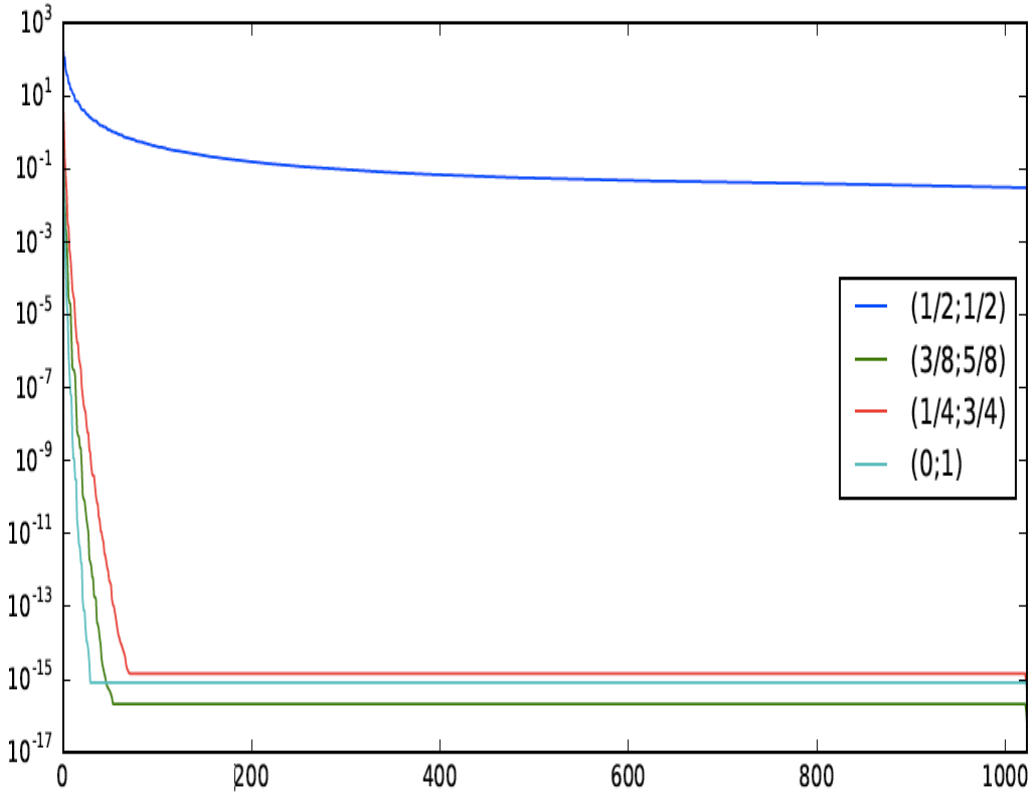
Fixed ranks
Preconditioners
Performance oriented

Fixed accuracy
Variable ranks
Dense/Sparse Direct Solvers

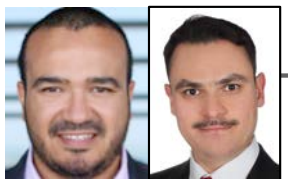
Even “brute force” tilings pay off (block low-rank without hierarchy)

1024	175	174	77	42	28	37	28	42	37	28	28	30	26	26	17
175	1024	78	174	174	42	76	38	37	42	28	29	37	31	28	25
174	78	1024	173	37	27	42	27	173	76	42	37	37	28	30	24
77	174	173	1024	77	37	173	42	77	174	37	42	76	38	37	31
42	174	37	77	1024	174	173	77	30	37	26	27	42	37	28	28
28	42	27	37	174	1024	78	175	25	30	23	26	37	42	28	29
37	76	42	173	173	78	1024	174	37	77	30	37	174	77	42	38
28	38	27	42	77	175	174	1024	24	38	24	30	78	175	38	43
42	37	173	77	30	25	37	24	1024	174	174	77	42	28	37	28
37	42	76	174	37	30	77	38	174	1024	77	175	174	42	76	38
28	28	42	37	26	23	30	24	174	77	1024	174	37	28	42	29
28	29	37	42	27	26	37	30	77	175	174	1024	77	37	173	42
30	37	37	76	42	37	174	78	42	174	37	77	1024	175	174	76
26	31	28	38	37	42	77	175	28	42	28	37	175	1024	77	174
26	28	30	37	28	28	42	38	37	76	42	173	174	77	1024	174
17	25	24	31	28	29	38	43	28	38	29	42	76	174	174	1024

Covariance Matrix of dimension 16384 in 16×16 blocks of 1024×1024 each

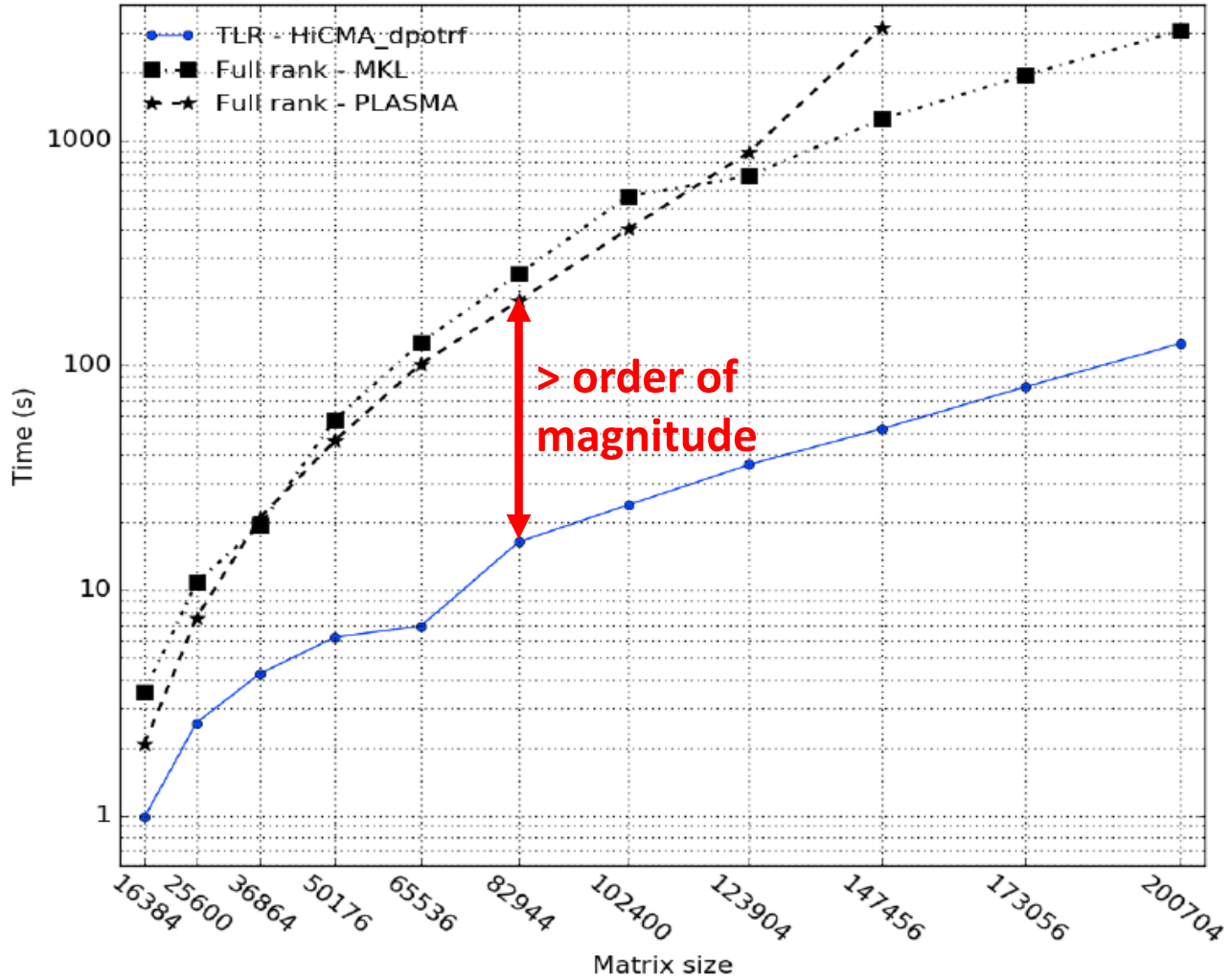


Compressibility of four typical blocks, for Frobenius accuracy of 10^{-9}



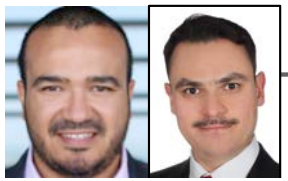
c/o H. Ltaief & K. Akbudak (KAUST)

Tile low-rank Cholesky, time per backsolve



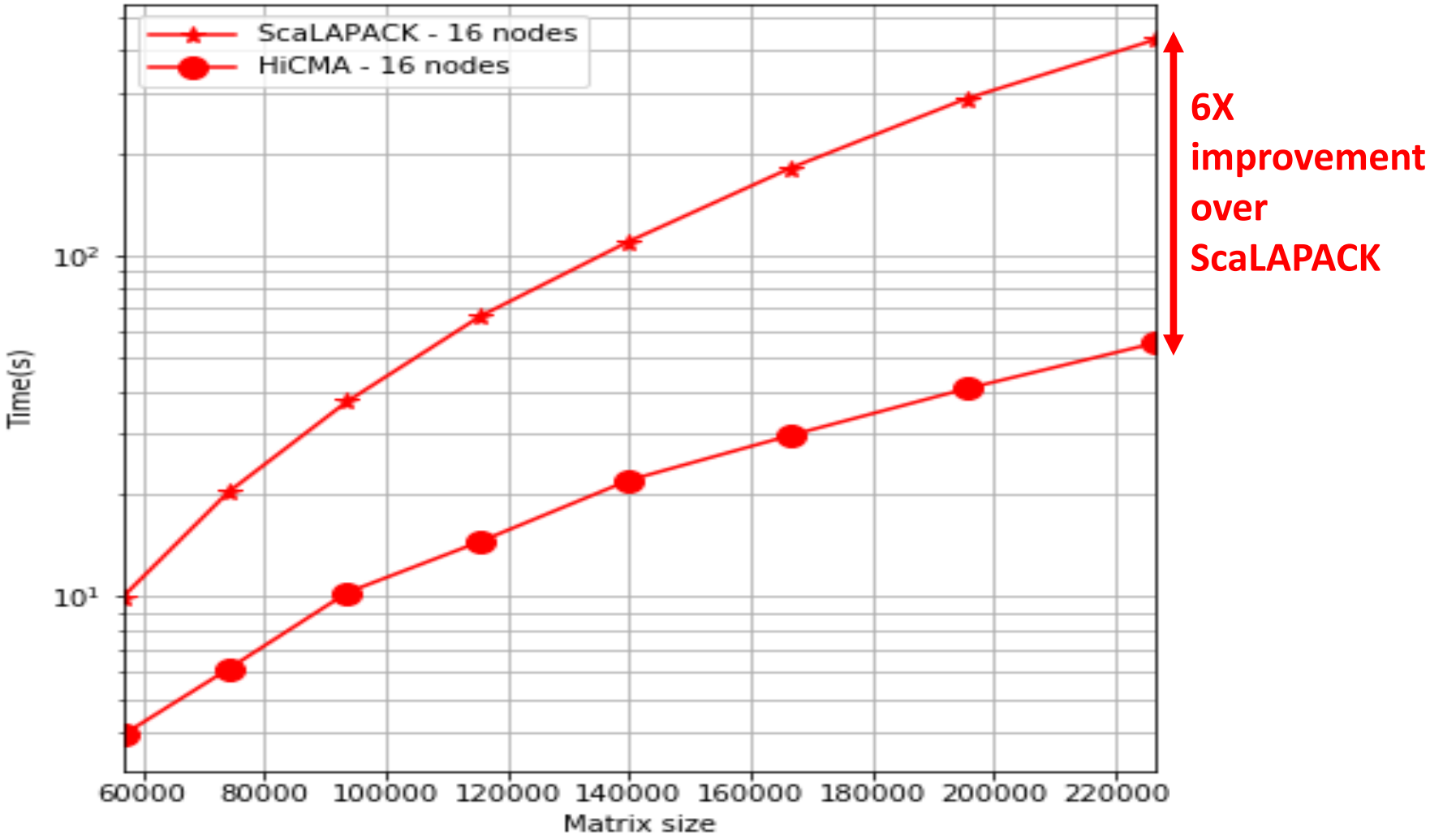
On 2-socket 18-core Intel Haswell @ 2.3GHz

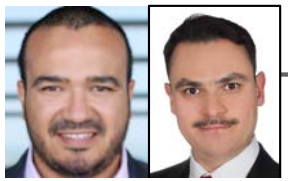
OpenMP pragmas for taskification and accuracy of 10^{-9}



c/o H. Ltaief & K. Akbudak (KAUST)

Distributed memory TLR Cholesky – preliminary





On 16 nodes of 2-socket 16-core Intel Haswell @ 2.3GHz

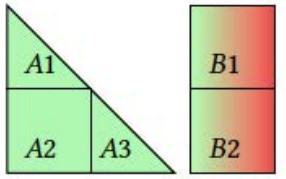
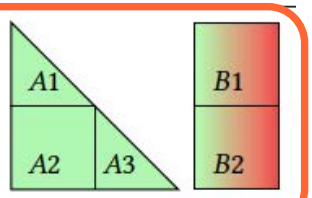
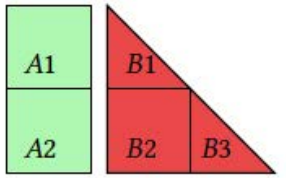
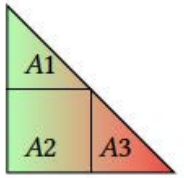
c/o H. Ltaief & K. Akbudak (KAUST)

KBLAS

- **Subset of L2/L3 BLAS targeting GPU and Intel MIC**
  - GEMV, SYMV, TRSM, TRMM
- **Reduces communication and increases concurrency in these memory BW bound operations**
- **Batched BLAS for small sizes on GPUs**
  - TRSM, TRMM, SYRK, POTRF, POTRS, POSV, TRTRI, LAUUM, POTRI, POTI
- **Recursive formulation**
- **Employs vendor-optimized L3 BLAS underneath**

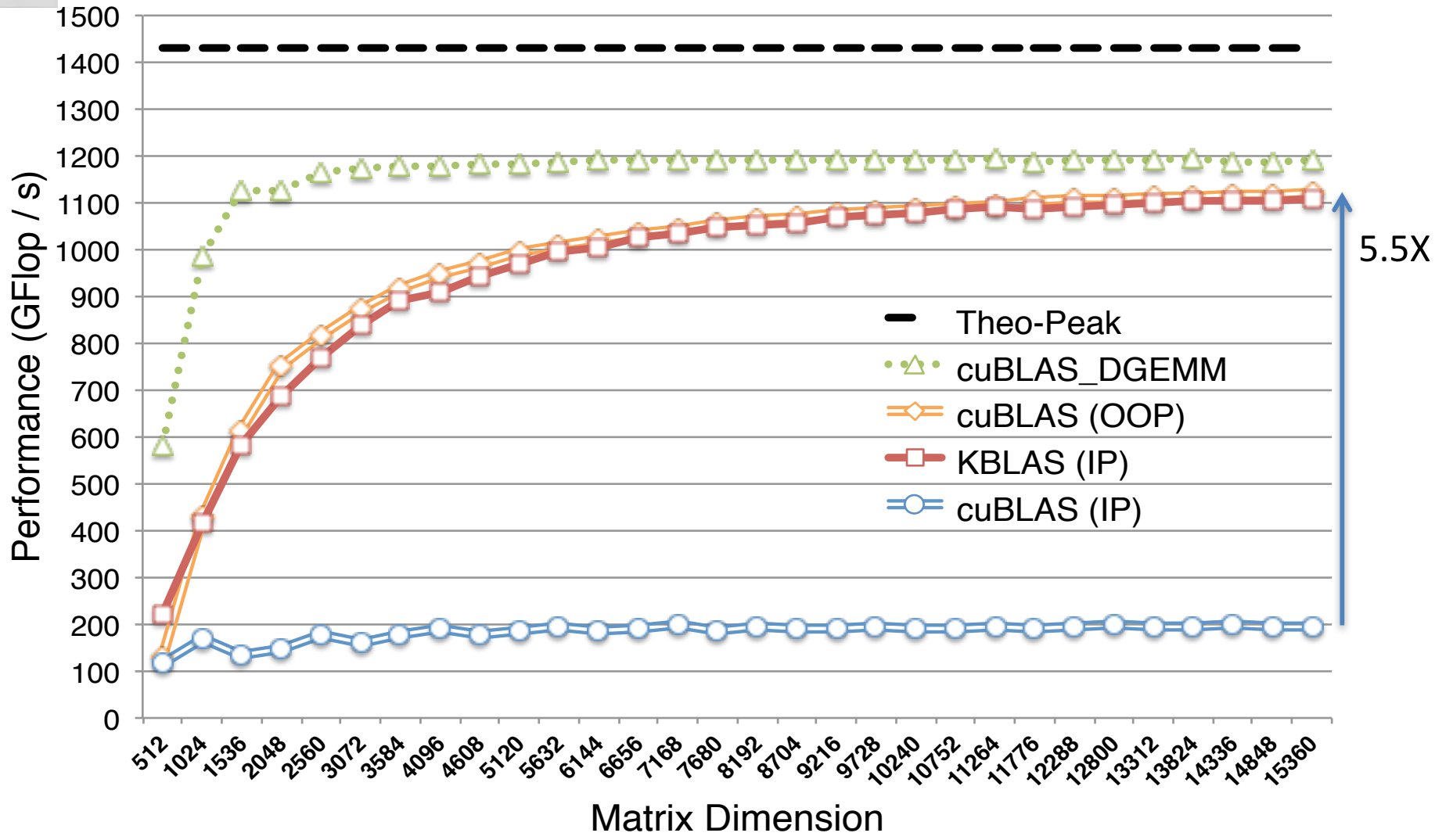
ACM TOMS (2016), CCPE (2016, 2017)

Recursively defined KBLAS operations for symmetric systems

$TRSM : A\,X = \alpha B$	$RecTRSM$:	$A_1\,X_1 = \alpha\,B_1$	$RecTRSM$	
		$B_2 = \alpha\,B_2 - A_2\,B_1$	$GEMM$	
		$A_3\,X_2 = B_2$	$RecTRSM$	
$TRMM : B = \alpha\,A^T\,B$	$RecTRMM$:	$B_1 = \alpha\,A_1^T\,B_1$	$RecTRMM$	
		$B_1 = \alpha\,A_2^T\,B_2 + B_1$	$GEMM$	
		$B_2 = \alpha\,A_3^T\,B_2$	$RecTRMM$	
$SYRK : B = \alpha AA^T + \beta B$	$RecSYRK$:	$B_1 = \alpha A_1 A_1^T + \beta B_1$	$RecSYRK$	
		$B_2 = \alpha A_2 A_1^T + \beta B_2$	$GEMM$	
		$B_3 = \alpha A_2 A_2^T + \beta B_3$	$RecSYRK$	
$POTRF : A = L\,L^T$	$RecPOTRF$:	$A_1 = L_1\,L_1^T$	$RecPOTRF$	
		$A_1\,X = A_2$	$RecTRSM$	
		$A_3 = -A_2\,A_2^T + A_3$	$RecSYRK$	
		$A_3 = L_3\,L_3^T$	$RecPOTRF$	

c/o A. Charara & H. Ltaief (KAUST)

KBLAS DTRMM

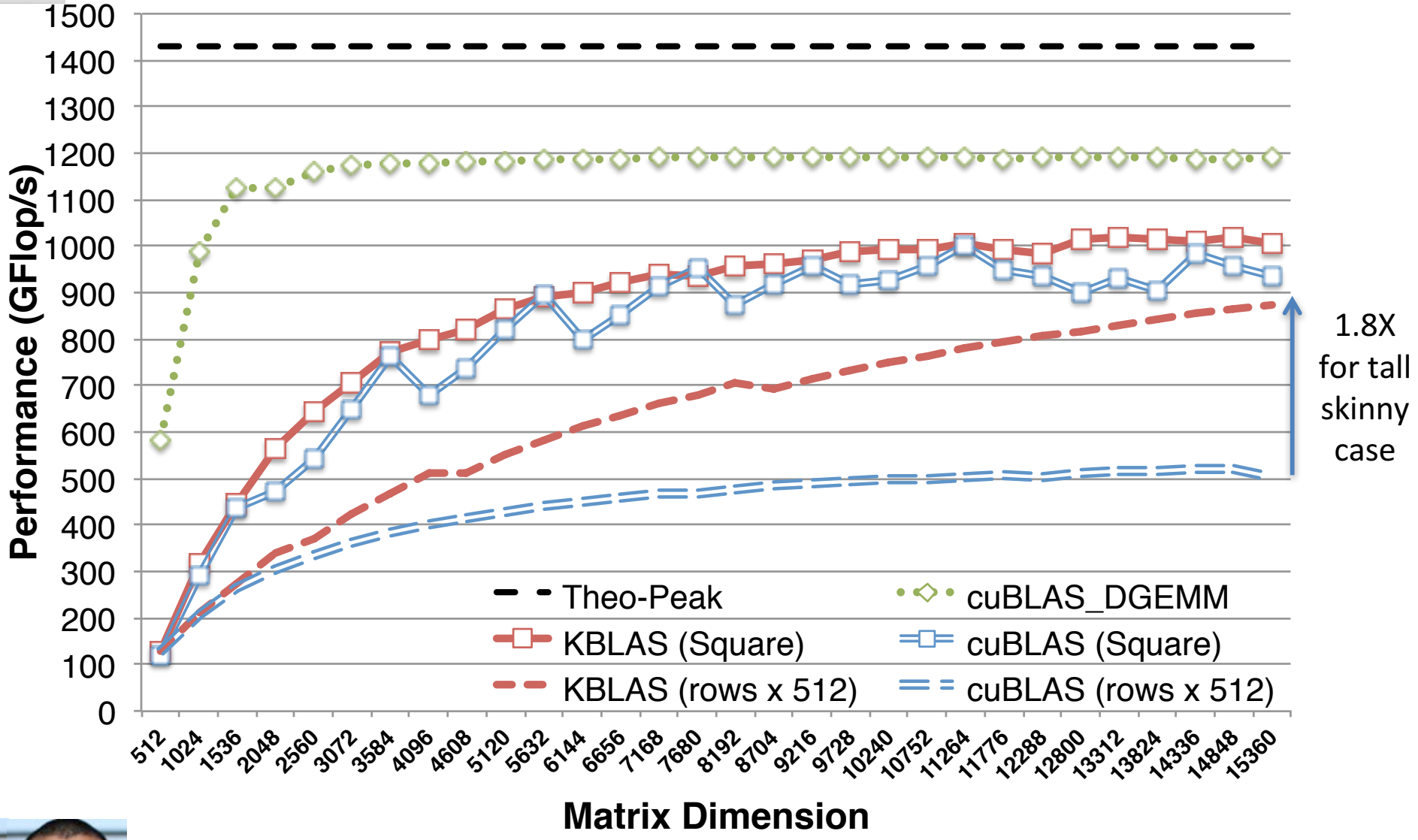


c/o A. Charara & H. Ltaief (KAUST)

Charara et al., Best papers, Europar'16

available: https://github.com/ecrc/kblas

KBLAS DTRSM



c/o A. Charara & H. Ltaief (KAUST)

Charara et al., Best papers, Europar'16

available: https://github.com/ecrc/kblas

KBLAS now in cuBLAS 8; will be in cuBLAS 9

NVIDIA DEVELOPER ZONE CUDA TOOLKIT DOCUMENTATION

Search

CUDA Toolkit v8.0

cuBLAS

- 1. Introduction
- 2. Using the cuBLAS API
- 3. Using the CUBLASXT API
- A. Using the cuBLAS Legacy API
- B. cuBLAS Fortran Bindings
- C. Acknowledgements

C. Acknowledgements

NVIDIA would like to thank the following individuals and institutions for their contributions:

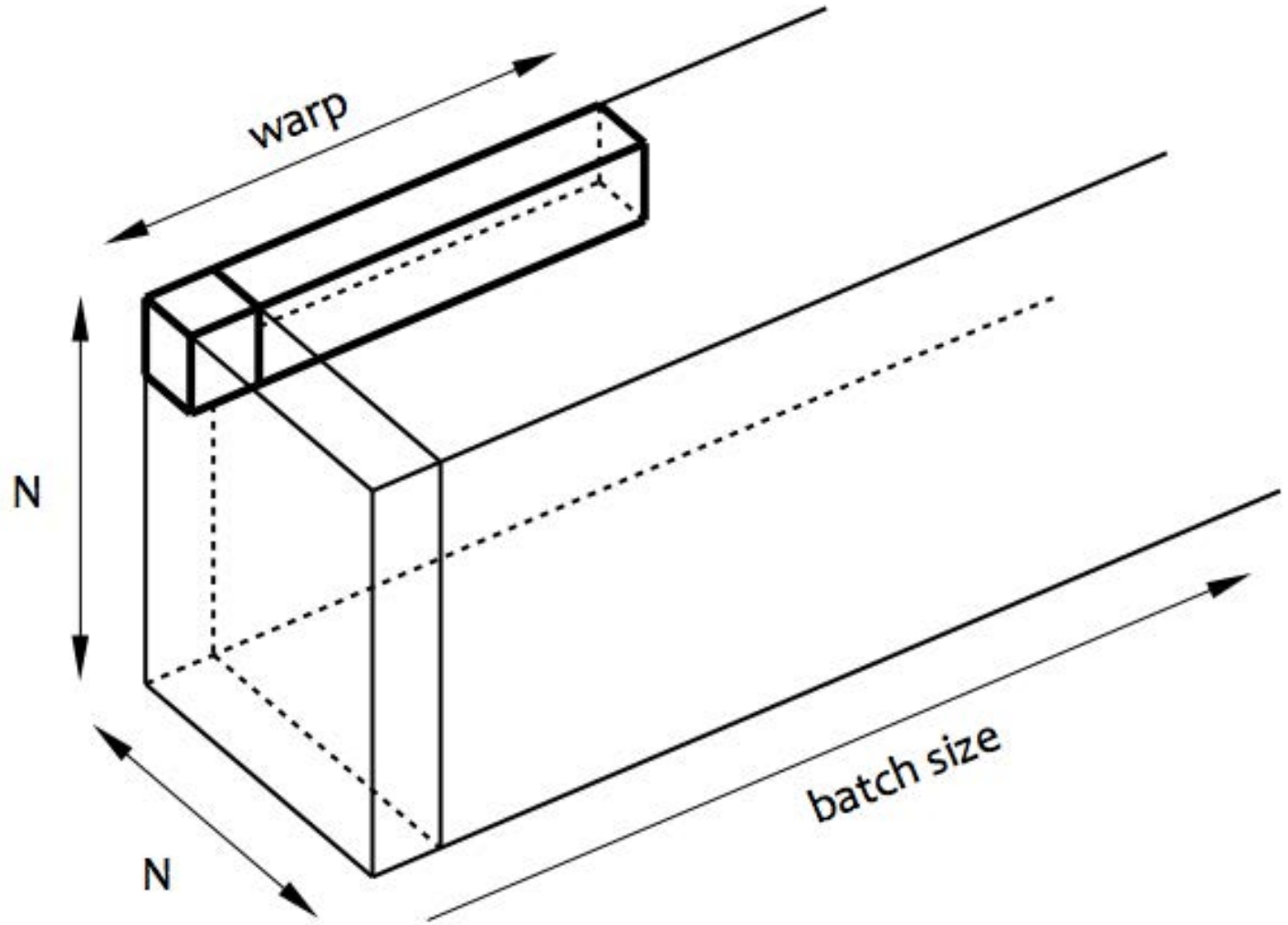
- Portions of the SGEMM, DGEMM, CGEMM and ZGEMM library routines were written by Vasily Volkov of the University of California.
- Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Davide Barbieri of the University of Rome Tor Vergata.
- Portions of the DGEMM and SGEMM library routines optimized for Fermi architecture were developed by the University of Tennessee. Subsequently, several other routines that are optimized for the Fermi architecture have been derived from these initial DGEMM and SGEMM implementations.
- The substantial optimizations of the STRSV, DTRSV, CTRSV and ZTRSV library routines were developed by Jonathan Hogg of The Science and Technology Facilities Council (STFC). Subsequently, some optimizations of the STRSM, DTRSM, CTRSM and ZTRSM have been derived from these TRSV implementations.
- Substantial optimizations of the SYMV and HEMV library routines were developed by Ahmad Abdelfattah, David Keyes and Hatem Ltaief of King Abdullah University of Science and Technology (KAUST).
- Substantial optimizations of the TRMM and TRSM library routines were developed by Ali Charara, David Keyes and Hatem Ltaief of King Abdullah University of Science and Technology (KAUST).

c/o A. Abdelfattah (ICL, KAUST'15), A. Charara & H. Ltaief (KAUST)

Extending KBLAS to batched execution

- **Batched BLAS workshop:**
  - http://bit.ly/Batch-BLAS-2017
- **Problem:**
  - L2 BLAS individually of low arithmetic intensity
  - memory latency overheads
- **Redesign the legacy BLAS API**
  - launch thousands of small BLAS kernels simultaneously
  - increase device occupancy
  - remove API/kernel launch overheads
  - extend the recursive formulation
- **Driven by scientific data-sparse applications**
  - computational statistics and astronomy
  - Schur complement in sparse direct solvers and BDDC preconditioning

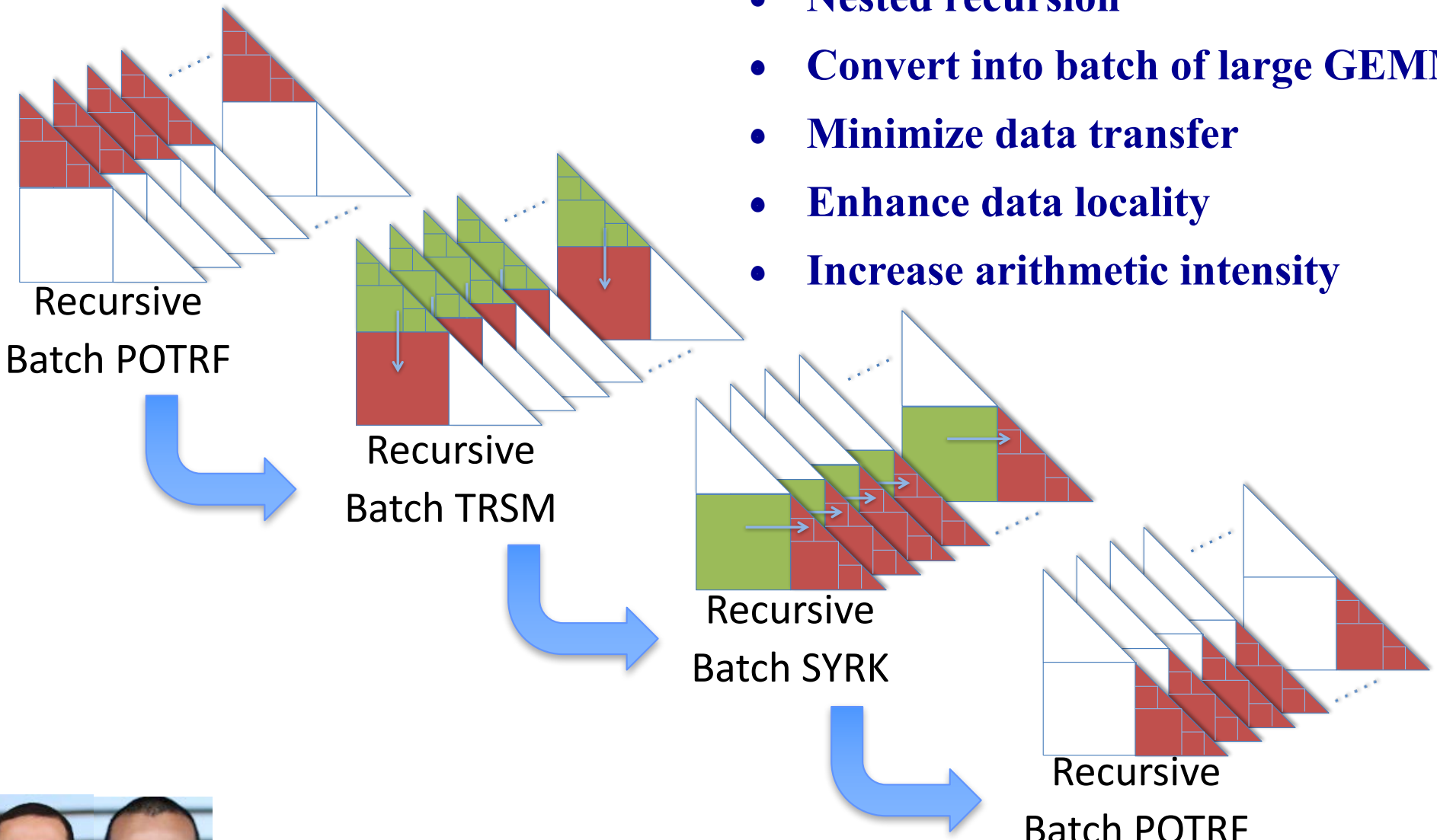
Batched operations

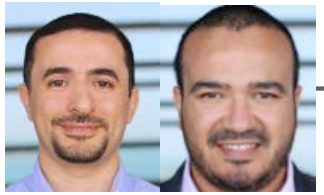


c/o Jacob Kurzak (ICL, U Tennessee)

KBLAS
Example: Batched POTRF

- Nested recursion
- Convert into batch of large GEMMs
- Minimize data transfer
- Enhance data locality
- Increase arithmetic intensity



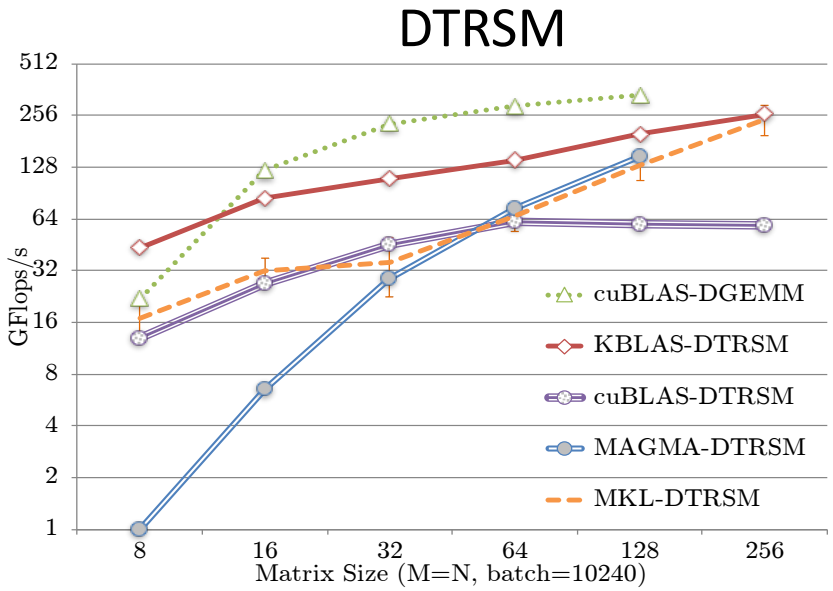


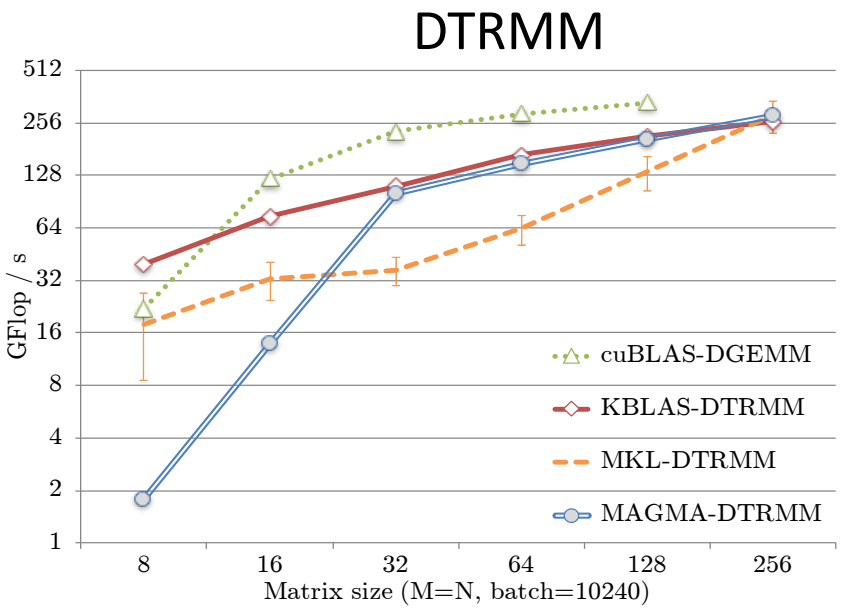
c/o A. Charara & H. Ltaief (KAUST)

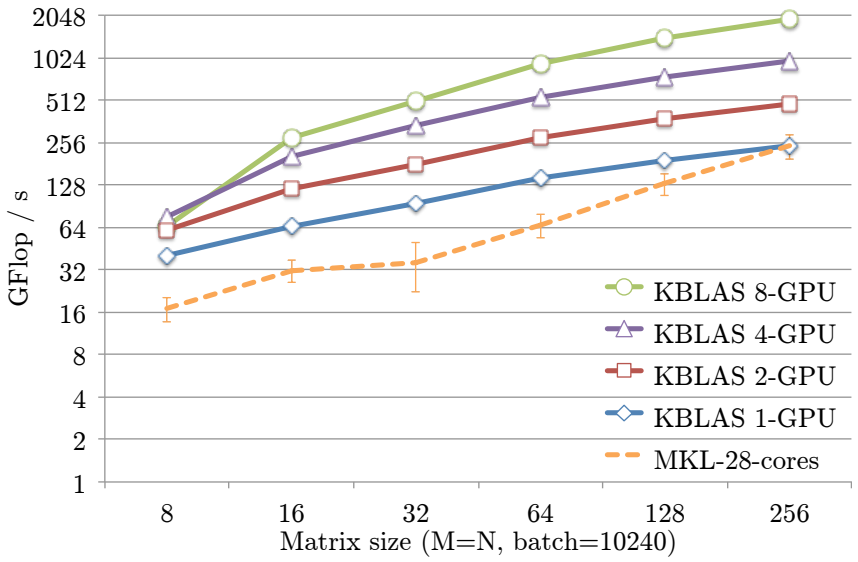
Batched KBLAS performance comparisons

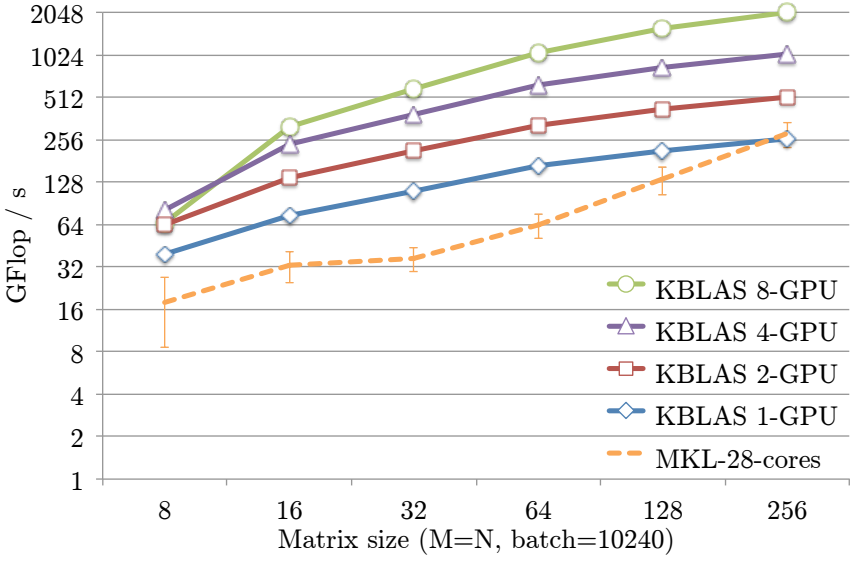
Single K40 (MKL on 28-core Broadwell)

Multiple K40s (MKL on 28-core Broadwell)







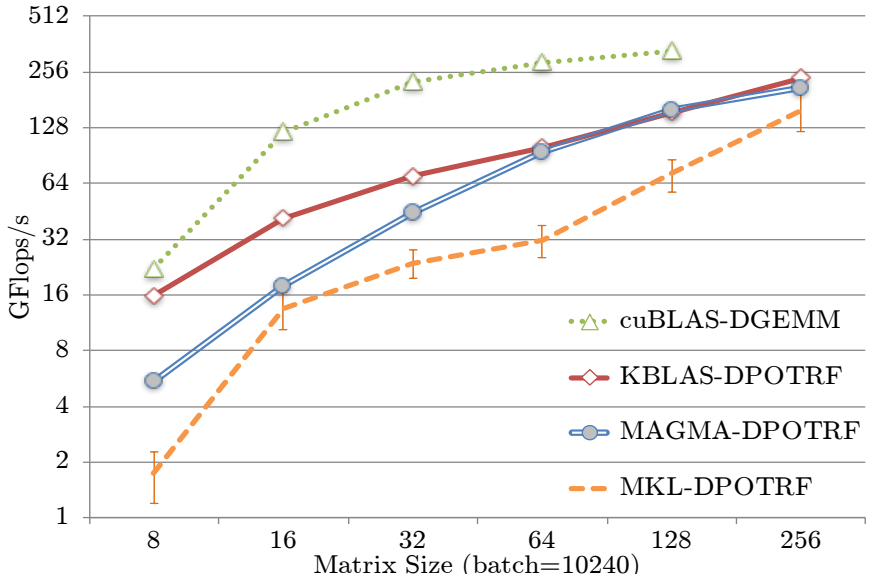


Batched KBLAS performance comparisons

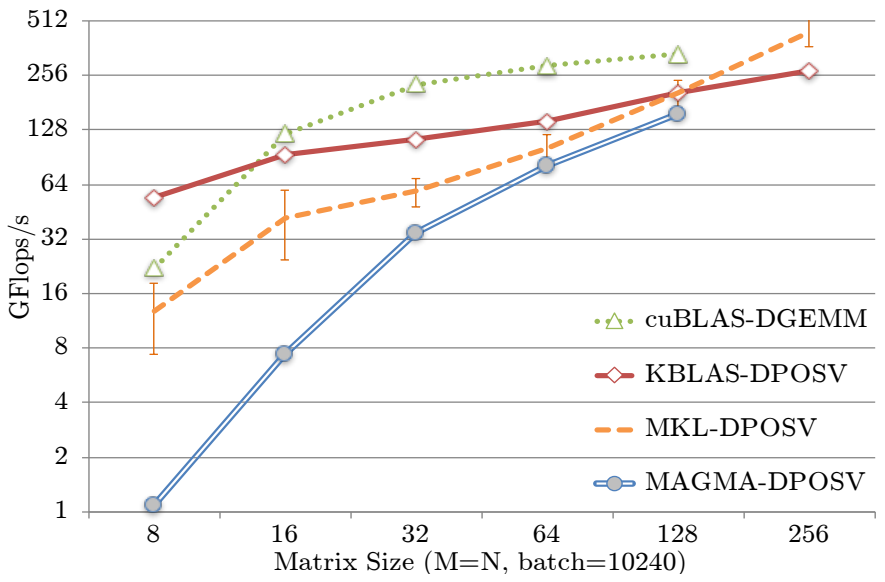
Single K40 (MKL on 28-core Broadwell)

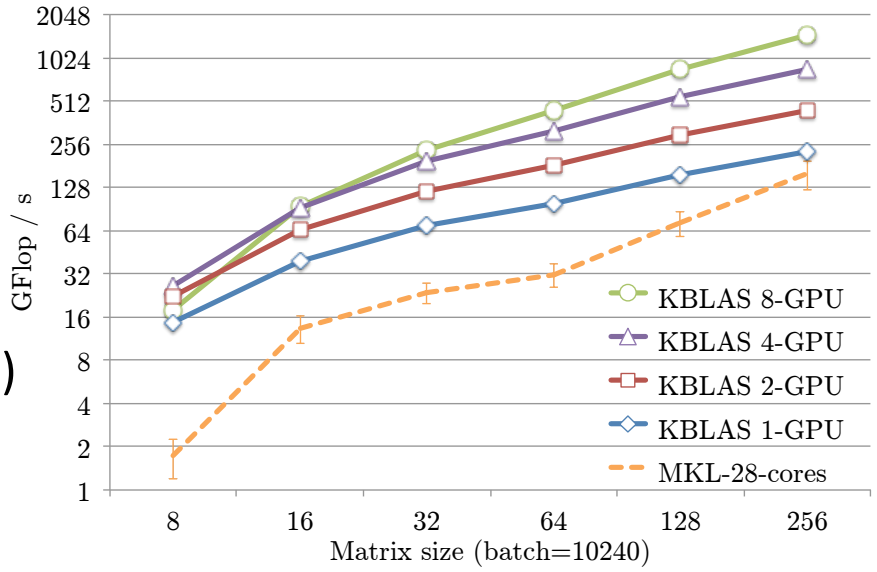
Multiple K40s (MKL on 28-core Broadwell)

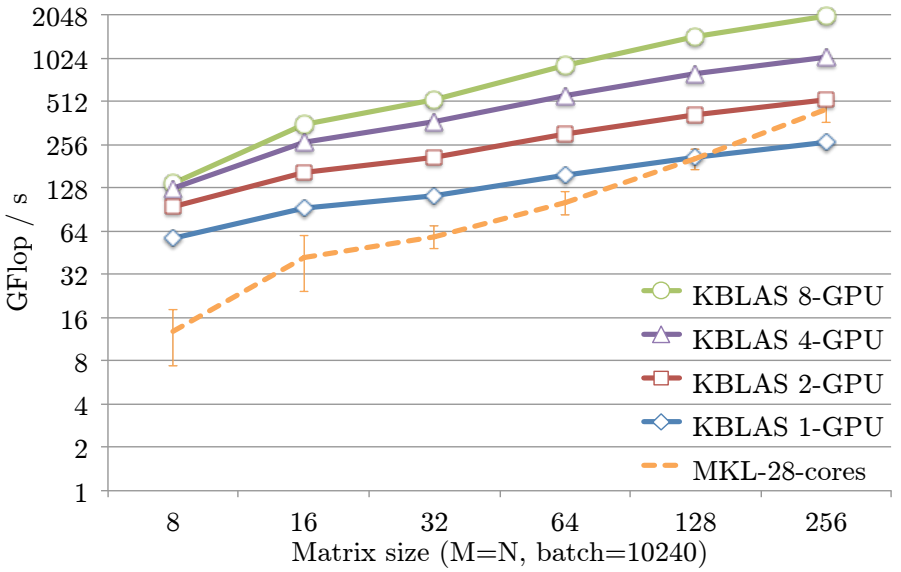
DPOTRF



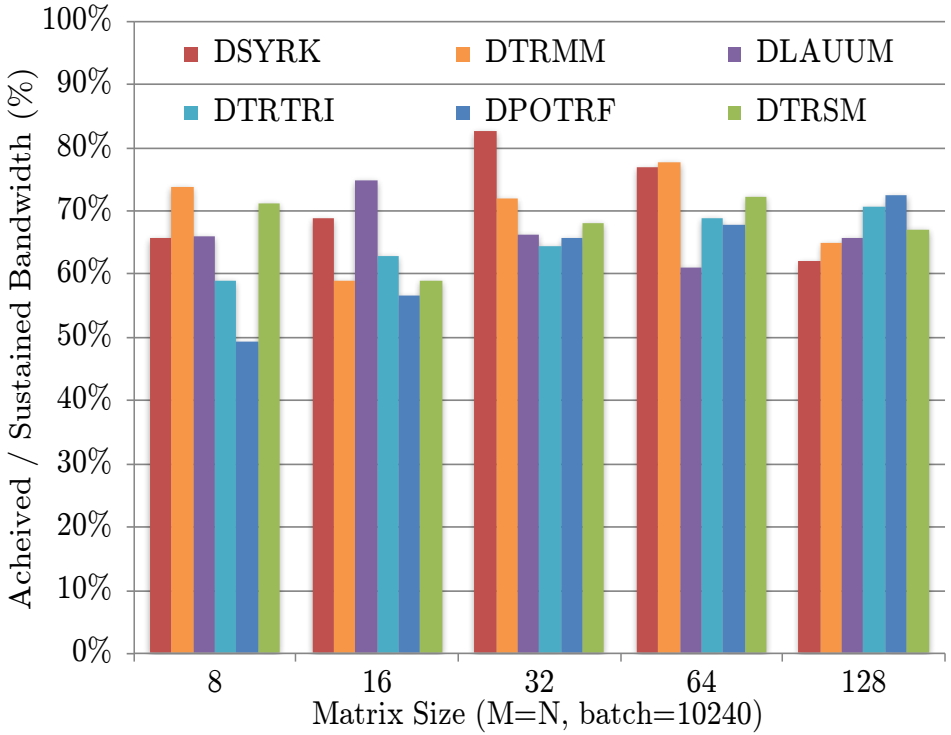
DPOSV



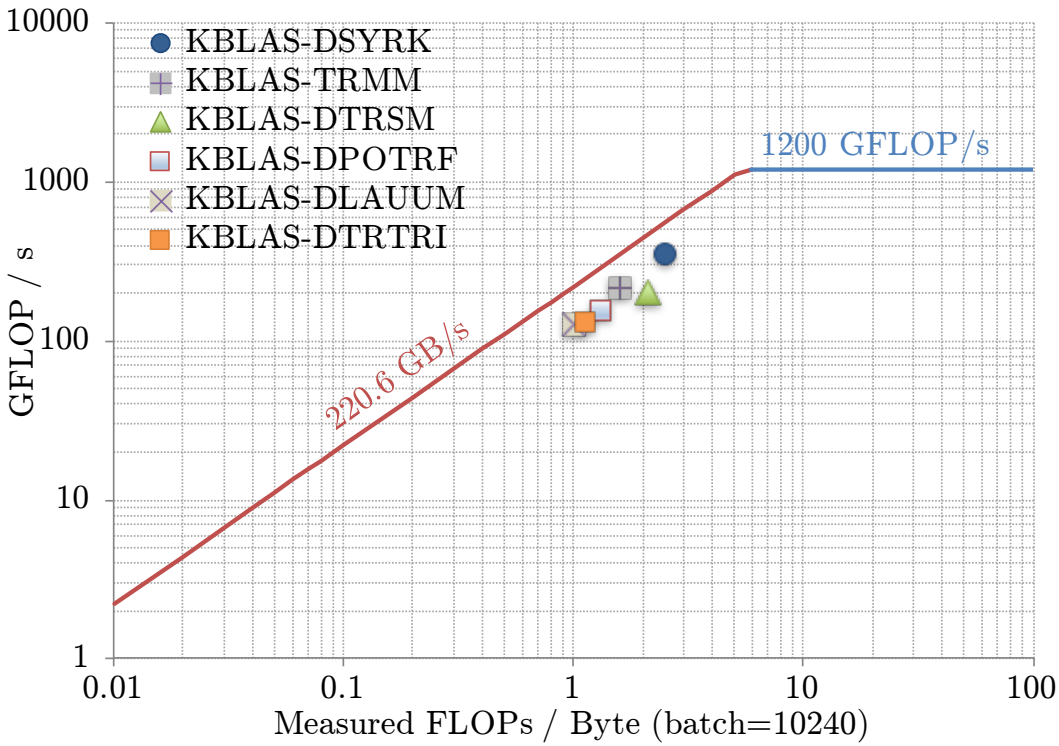




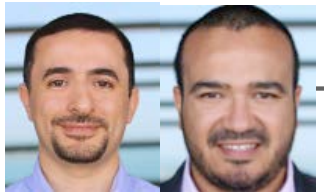
Batched KBLAS performance



Ratio of achieved to sustained bandwidth of various KBLAS batched operations in double precision on a K40 GPU with 10240 batch size.



Roofline performance model of KBLAS batched operations in double precision and 10240 batched size running on NVIDIA K40 GPU, on square matrices of size 128.

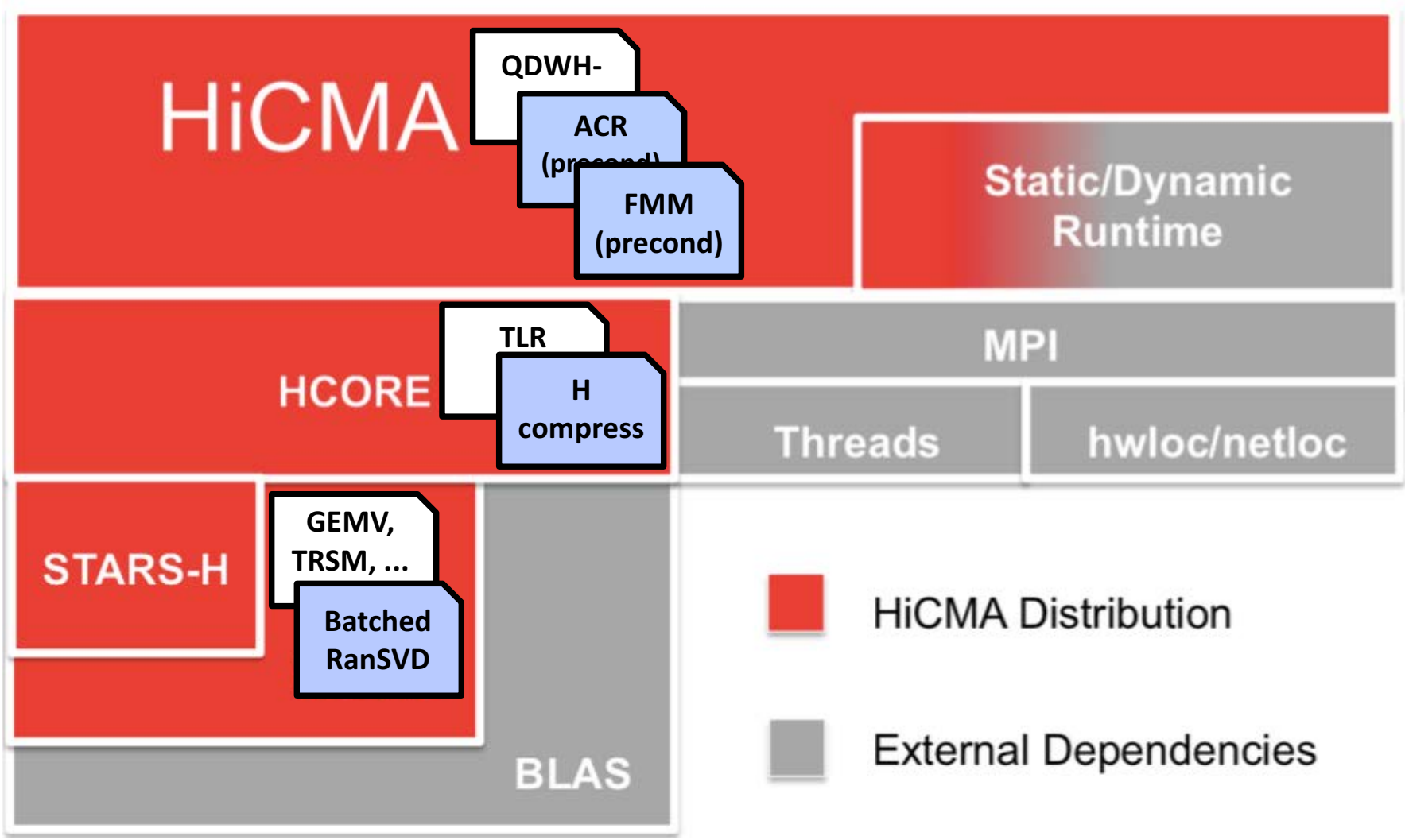


c/o A. Charara & H. Ltaief (KAUST)

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"**

Hierarchical Computations on Manycore Architectures: HiCMA*



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

Thanks to:

CENTER OF EXCELLENCE


Thank you!
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology
شكرا
https://github.com/ecrc/
david.keyes@kaust.edu.sa