

Algorithmic Adaptations to Extreme Scale Computing

David Keyes, Applied Mathematics & Computational Science
Director, Extreme Computing Research Center (ECRC)
King Abdullah University of Science and Technology
david.keyes@kaust.edu.sa

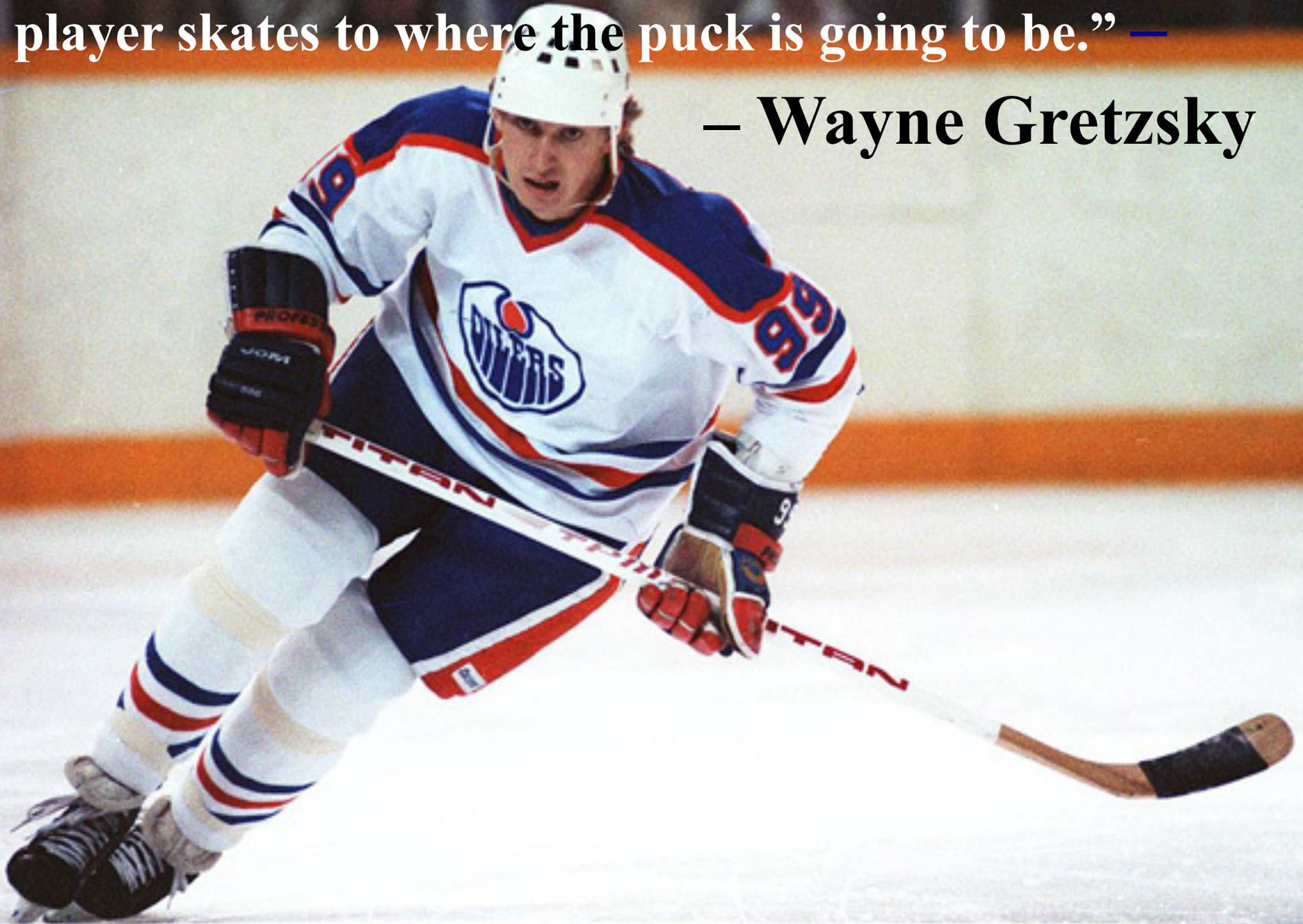


Tie-ins to other ATPESC'17 presentations

- **Numerous!**
 - ◆ architecture, applications, algorithms, programming models & systems software, etc., form an *interconnected ecosystem*
 - ◆ algorithms/software are in the middle – spanning diverging requirements in architectures (which “want” more uniformity) and applications (which “want” more irregularity)
- **To Rick Stevens’ dinner talk last night on convergence**
- **To architecture presentations:**
 - ◆ Intel, NVIDIA, IBM
- **To programming models presentations:**
 - ◆ MPI, OpenMP, Open ACC, OCCA, UPC++, Legion, kokkos, etc.
- **To other “Track 4” algorithms presentations:**
 - ◆ Almgren, Demmel, Dongarra, FASTMath team, CEED team

“A good player plays where the puck is, while a great player skates to where the puck is going to be.” —

– Wayne Gretzky



Aspiration for this talk

To paraphrase Gretzky:

**“Algorithms for where
architectures are going to be”**

Outline

- **Architectural and applications trends**
 - ◆ **limitations of our current software infrastructure for numerical simulation at exascale**
- **Four algorithmic imperatives**
 - ◆ **for extreme scale, tomorrow *and today***
- **Four sets of “bad news, good news”**
- **Four widely applicable strategies**
- **Four sample “points of light”**
 - ◆ **contributions to a new algorithmic infrastructure**



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***
 - **Mean time between hardware errors shortens**
-

→ 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 ROADMAP 1.0
SOFTWARE PROJECT



The International Exascale Software Roadmap

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

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
Mitsuhsa Sato
Ed Seidel
John Shalf
David Skinner
Marc Snir
Thomas Sterling
Rick Stevens
Fred Streit

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



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**
 - **Processor clocks may be slowed and speeded**
 - ◆ **may be scheduled, based on phases with different requirements, or may be dynamic, from power capping or thermal monitoring**
 - ◆ **makes per-node performance rate unreliable**
 - **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**
 - ◆ **more errors may need to be caught and corrected in software**
-

Today's power costs per operation

Operation	approximate energy cost
DP floating point multiply-add	100 pJ
DP DRAM read-to-register	4800 pJ
DP word transmit-to-neighbor	7500 pJ
DP word transmit-across-system	9000 pJ

2 orders of magnitude energy cost (worse ratio for latency)

A *pico* (10^{-12}) of something done *exa* (10^{18}) times per second is a *mega* (10^6)-somethings per second

- ◆ 100 pJ at 1 Eflop/s is 100 MW (for the flop/s only!)
- ◆ 1 MW-year costs about \$1M ($\$0.12/\text{KW-hr} \times 8760 \text{ hr/yr}$)
 - We “use” 1.4 KW continuously, so 100MW is 71,000 people



Why exa- is different

Dennard's MOSFET scaling (1972) ends before Moore's Law (1965) ends

Table 1
Scaling Results for Circuit Performance

Device or Circuit Parameter	Scaling Factor
Device dimension t_{ox}, L, W	$1/\kappa$
Doping concentration N_a	κ
Voltage V	$1/\kappa$
Current I	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit VC/I	$1/\kappa$
Power dissipation/circuit VI	$1/\kappa^2$
Power density VI/A	1

Table 2
Scaling Results for Interconnection Lines

Parameter	Scaling Factor
Line resistance, $R_L = \rho L/Wt$	κ
Normalized voltage drop IR_L/V	κ
Line response time $R_L C$	1
Line current density I/A	κ



Robert Dennard, IBM
(inventor of DRAM, 1966)

Eventually processing is limited by transmission, as known for 4.5 decades

Architectural resources to balance

- **Processing cores**
 - ◆ heterogeneous (CPUs, MICs, GPUs, FPGAs,...)
- **Memory**
 - ◆ hierarchical (registers, caches, DRAM, flash, stacked, ...)
 - ◆ partially reconfigurable
- **Intra-node network**
 - ◆ nonuniform bandwidth and latency
- **Inter-node network**
 - ◆ nonuniform bandwidth and latency

**For performance tuning:
Which resource is limiting, as a function of time?**

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
-

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

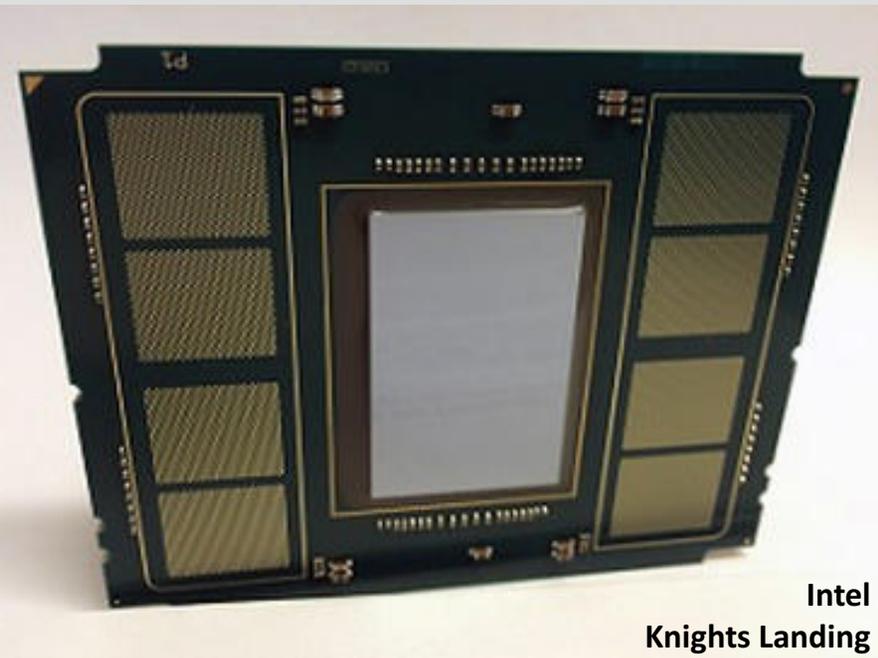
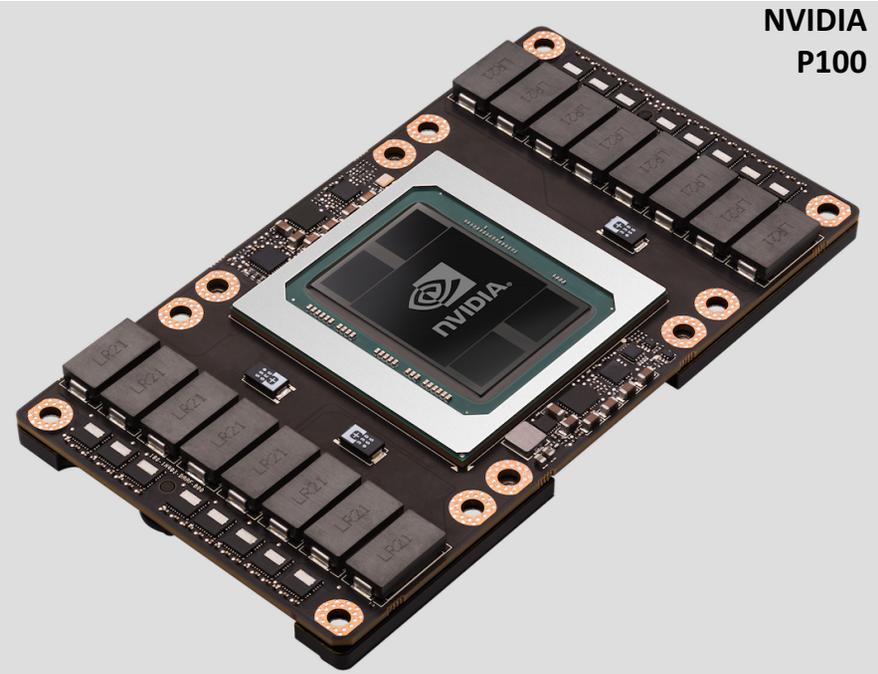
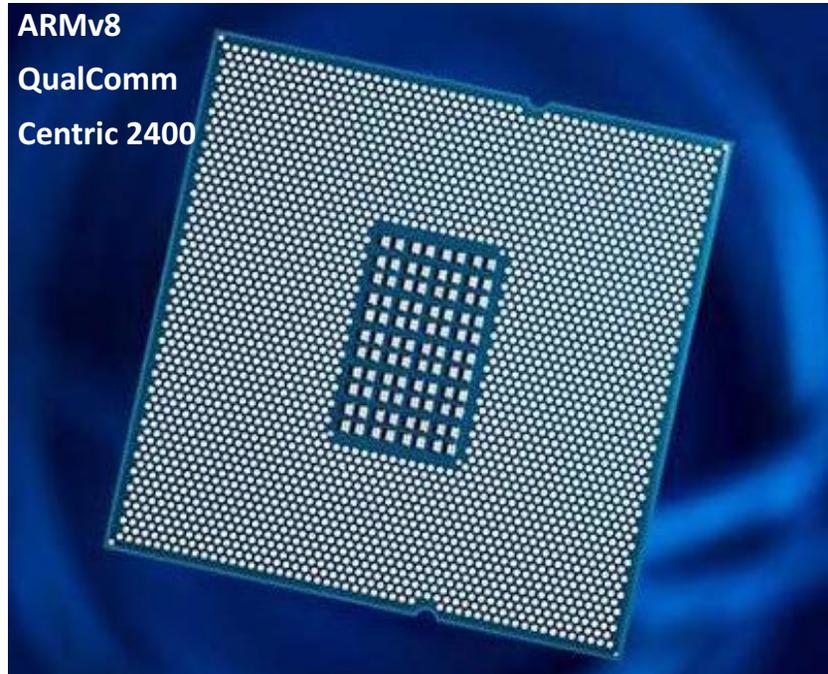


Sequoia

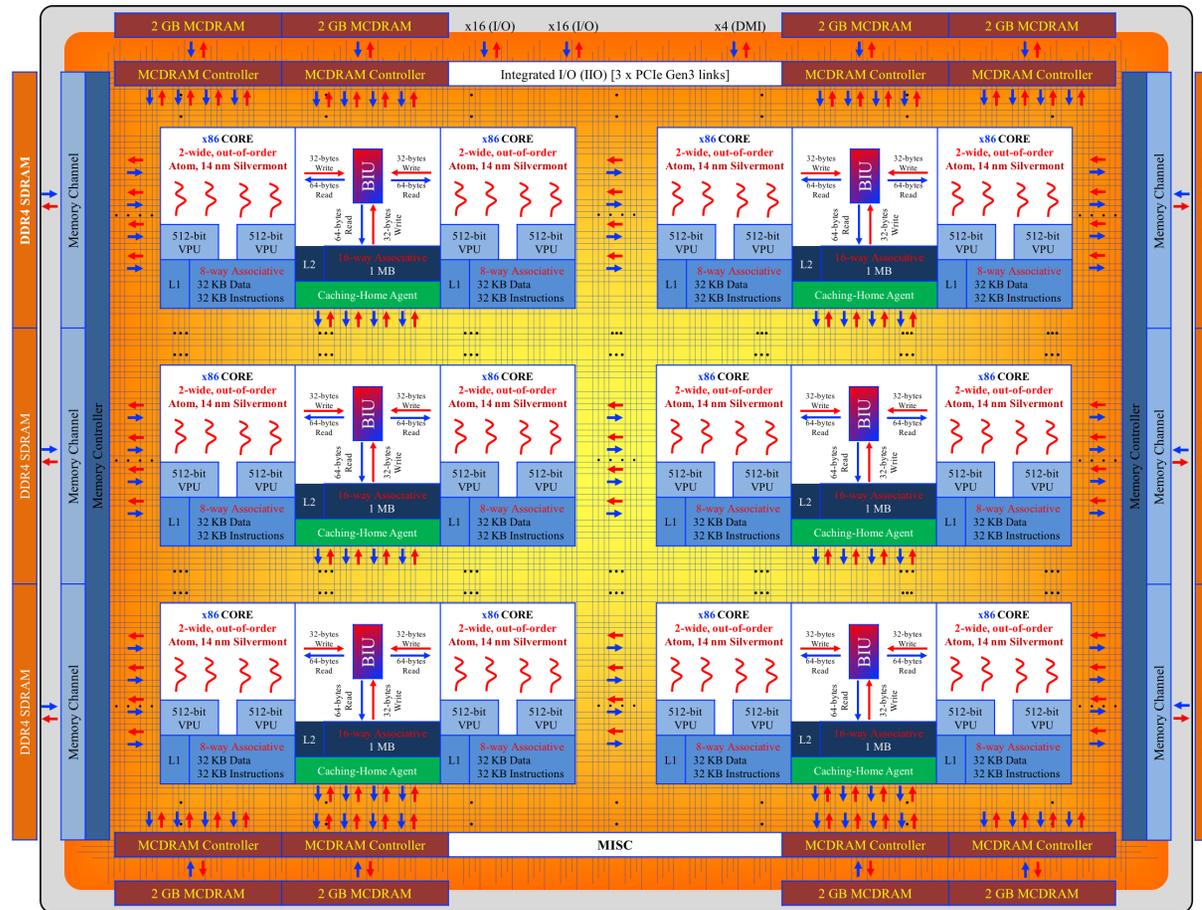


K

The challenge



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



Schematic of Intel Xeon Phi KNL by M. Farhan, KAUST

The main contest is already being waged on individual shared-memory nodes

Two decades of evolution

1997

2017



ASCI Red at Sandia

1.3 TF/s, 850 KW

Cavium ThunderX2

~ 1.1 TF/s, ~ 0.2 KW

3.5 orders of
magnitude



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	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	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

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

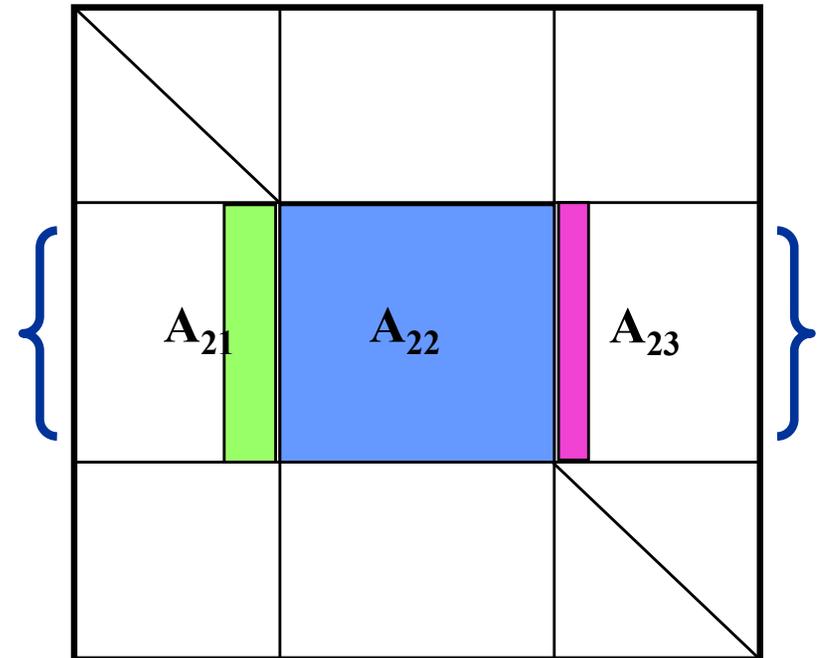
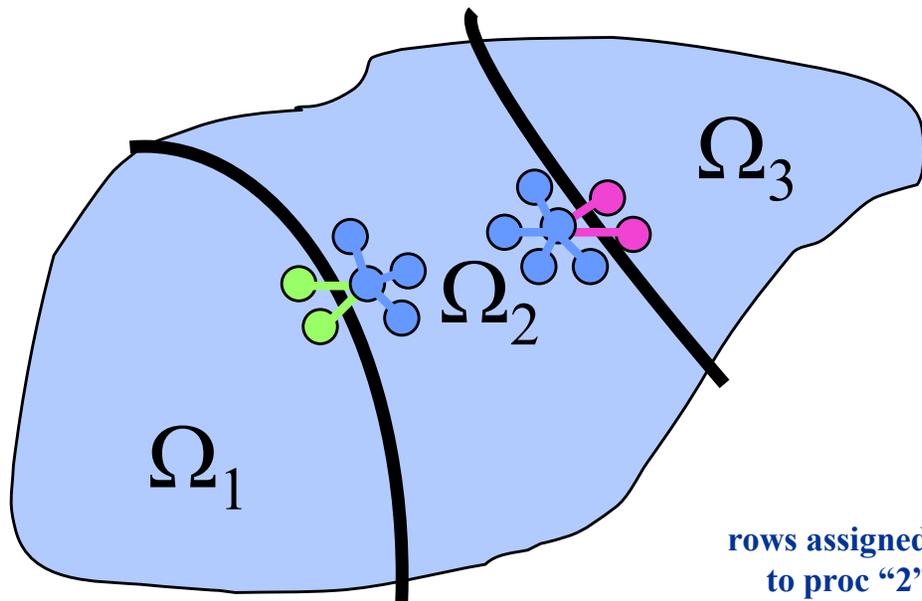
A. Bridging Model ^{for} parallel Computation

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



**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	Gigaflop/s delivered to applications
Year	
1988	1
1998	1,020
2008	1,350,000

Gordon Bell Prize: Price Performance	Cost per delivered Gigaflop/s
Year	
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**
-

Extrapolating exponentials eventually fails

- **Exa- is qualitatively different and looks more difficult**
 - ◆ but we once said that about message passing
 - **Core numerical analysis and scientific computing will confront exascale to maintain relevance**
 - ◆ potentially big gains in colonizing exascale for science and engineering
 - ◆ not a “distraction,” but an intellectual stimulus
 - ◆ the journey will be as fun as the destination 😊
-

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**
-

A close-up photograph of a hand holding a red baton. The hand is positioned on the right side of the frame, with the baton extending towards the left. The background is a soft, out-of-focus sky with light clouds. The text 'Energy-aware generation' is overlaid on the left side of the image, and 'BSP generation' is overlaid on the right side.

**Energy-aware
generation**

**BSP
generation**

Applications background

www.exascale.org/bdec

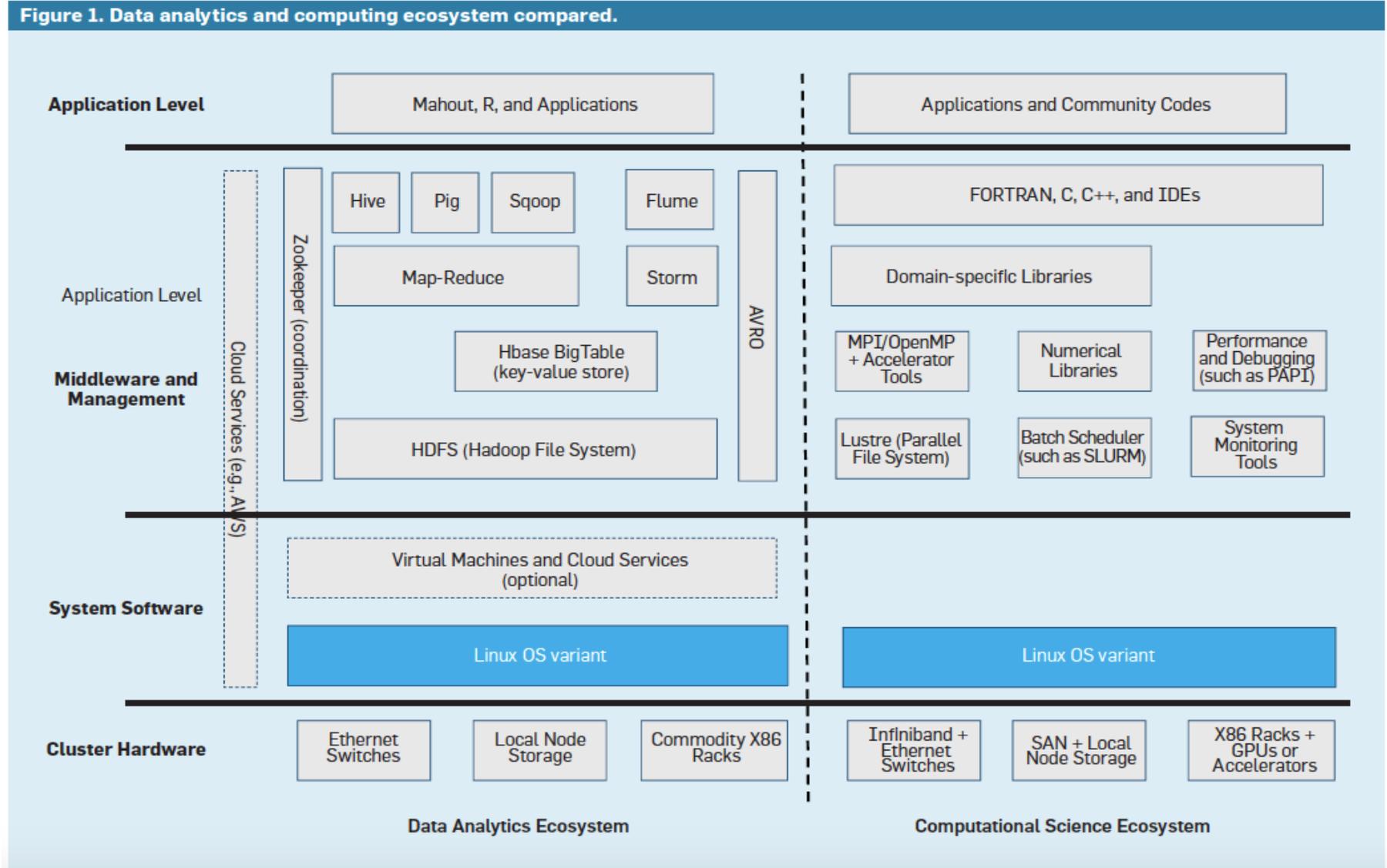


Big Data and Extreme Computing: Pathways to Convergence

J. Dongarra, P. Beckman, et al., *downloadable in draft form at URL above*

Successor to *The International Exascale Software Roadmap*, by many of the same authors and new authors from big data

Challenge for applications: merging software for 3rd and 4th paradigms



c/o Reed & Dongarra, Comm. ACM, July 2015

Interactions between application archetypes

Increasingly, there is scientific opportunity in pipelining

→ *Convergence is ripe*

		To Simulation	To Analytics	To Learning
3 rd	Simulation provides	—		
4 th (a)	Analytics provides		—	
4 th (b)	Learning provides			—

Interactions between application archetypes

Increasingly, there is scientific opportunity in pipelining

→ *Convergence is ripe*

		To Simulation	To Analytics	To Learning
3 rd	Simulation provides	—		
4 th (a)	Analytics provides	Steering in high dimensional parameter space; <i>In situ</i> processing	—	
4 th (b)	Learning provides	Smart data compression; Replacement of models with learned functions		—

Interactions between application archetypes

Increasingly, there is scientific opportunity in pipelining

→ *Convergence is ripe*

	To Simulation	To Analytics	To Learning	
3 rd	Simulation provides	—	Physics-based “regularization”	Data for training, augmenting real-world data
4 th (a)	Analytics provides	Steering in high dimensional parameter space; <i>In situ</i> processing	—	
4 th (b)	Learning provides	Smart data compression; Replacement of models with learned functions	—	

Interactions between application archetypes

Increasingly, there is scientific opportunity in pipelining

→ *Convergence is ripe*

	To Simulation	To Analytics	To Learning	
3 rd	Simulation provides	—	Physics-based “regularization”	Data for training, augmenting real-world data
4 th (a)	Analytics provides	Steering in high dimensional parameter space; <i>In situ</i> processing	—	Feature vectors for training
4 th (b)	Learning provides	Smart data compression; Replacement of models with learned functions	Imputation of missing data; Detection and classification	—



Four algorithmic imperatives

- **Reduce synchrony (in frequency and/or span)**
 - **Reside “high” on the memory hierarchy**
 - ◆ **as close as possible to the processing elements**
 - **Increase SIMT/SIMD-style shared-memory concurrency**
 - **Build in resilience (“algorithm-based fault tolerance” or ABFT) to arithmetic/memory faults or lost/delayed messages**
-



Bad news/good news



- **Must explicitly control more of the data motion**
 - ◆ carries the highest energy and time cost in the exascale computational environment
 - **More opportunities to control the *vertical* data motion**
 - ◆ *horizontal* data motion under control of users already
 - ◆ but vertical replication into caches and registers was (until recently) mainly scheduled and laid out by hardware and runtime systems, mostly invisibly to users
-



Bad news/good news



- **Use of uniform high precision in nodal bases on dense grids may decrease, to save storage and bandwidth**
 - ◆ **representation of a smooth function in a hierarchical basis or on sparse grids requires fewer bits than storing its nodal values, for equivalent accuracy**
 - **We may compute and communicate “deltas” between states rather than the full state quantities**
 - ◆ **as when double precision was once expensive (e.g., iterative correction in linear algebra)**
 - ◆ **a generalized “combining network” node or a smart memory controller may remember the last address and the last value, and forward just the delta**
 - **Equidistributing errors properly to minimize resource use will lead to innovative error analyses in numerical analysis**
-



Bad news/good news



- **Fully deterministic algorithms may be regarded as too synchronization-vulnerable**
 - ◆ rather than wait for missing data, we may predict it using various means and continue
 - ◆ we do this with increasing success in problems without models (“big data”)
 - ◆ should be fruitful in problems coming from continuous models
 - ◆ “apply machine learning to the simulation machine”
 - **A rich numerical analysis of algorithms that make use of statistically inferred “missing” quantities may emerge**
 - ◆ future sensitivity to poor predictions can often be estimated
 - ◆ numerical analysts will use statistics, signal processing, ML, etc.
-



Bad news/good news



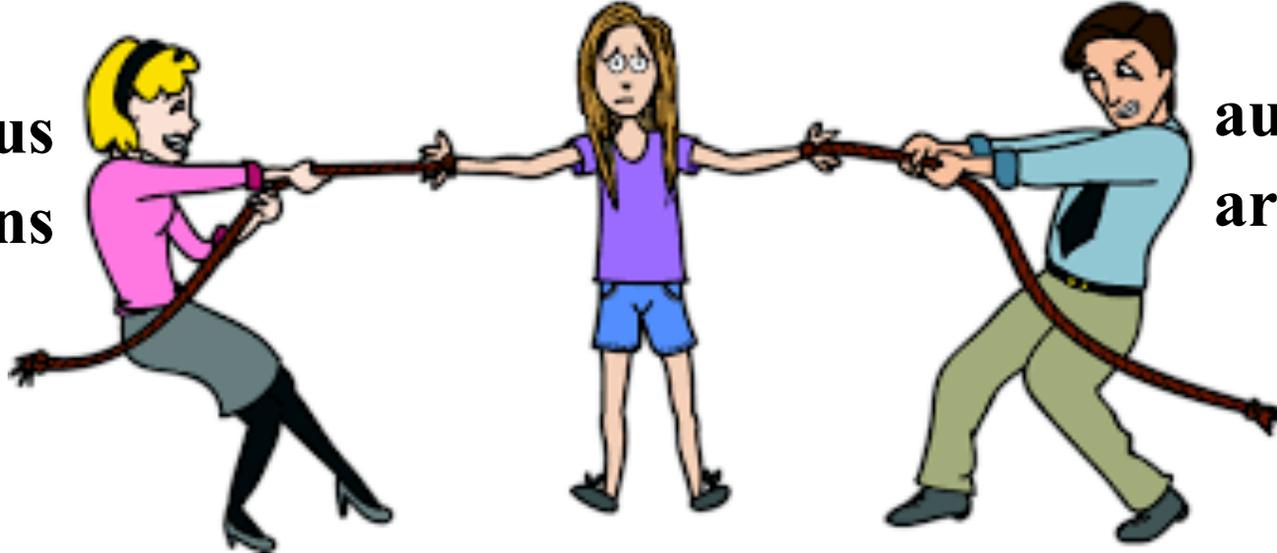
- **Fully hardware-reliable executions may be regarded as too costly**
 - **Algorithmic-based fault tolerance (ABFT) will be cheaper than hardware and OS-mediated reliability**
 - ◆ **developers will partition their data and their program units into two sets**
 - **a small set that must be done reliably (with today's standards for memory checking and IEEE ECC)**
 - **a large set that can be done fast and unreliably, knowing the errors can be either detected, or their effects rigorously bounded**
 - **Many examples in direct and iterative linear algebra**
 - **Anticipated by Von Neumann, 1956 (“Synthesis of reliable organisms from unreliable components”)**
-

Algorithmic philosophy

Algorithms must span a widening gulf ...

adaptive
algorithms

ambitious
applications



austere
architectures

A full employment program
for algorithm developers 😊

What will exascale algorithms look like?

- For weak scaling, must *start* with algorithms with optimal asymptotic order, $O(N \log^p N)$
- Some optimal hierarchical algorithms
 - ◆ Fast Fourier Transform (1960's)
 - ◆ Multigrid (1970's)
 - ◆ Fast Multipole (1980's)
 - ◆ Sparse Grids (1990's)
 - ◆ \mathcal{H} matrices (2000's)
 - ◆ Randomized algorithms (2010's)

“With great computational power comes great algorithmic responsibility.” – Longfei Gao

Required software

Model-related

- ◆ Geometric modelers
- ◆ Meshers
- ◆ Discretizers
- ◆ Partitioners
- ◆ Solvers / integrators
- ◆ Adaptivity systems
- ◆ Random no. generators
- ◆ Subgridscale physics
- ◆ Uncertainty quantification
- ◆ Dynamic load balancing
- ◆ Graphs and combinatorial algs.
- ◆ Compression

Development-related

- ◆ Configuration systems
- ◆ Source-to-source translators
- ◆ Compilers
- ◆ Simulators
- ◆ Messaging systems
- ◆ Debuggers
- ◆ Profilers

High-end computers come with little of this. Most is contributed by the user community.

Production-related

- ◆ Dynamic resource management
- ◆ Dynamic performance optimization
- ◆ Authenticators
- ◆ I/O systems
- ◆ Visualization systems
- ◆ Workflow controllers
- ◆ Frameworks
- ◆ Data miners
- ◆ Fault monitoring, reporting, and recovery

Midpoint: recap of algorithmic agenda

- **New formulations with**
 - ◆ **reduced synchronization and communication**
 - less frequent *and/or* less global
 - ◆ **reside high on the memory hierarchy**
 - greater arithmetic intensity (flops per byte moved into and out of registers and upper cache)
 - ◆ **greater SIMT/SIMD-style thread concurrency for accelerators**
 - ◆ **algorithmic resilience to various types of faults**
 - **Quantification of trades between limited resources**
 - ***Plus* all of the exciting analytical agendas that exascale is meant to exploit**
 - ◆ **“post-forward” problems: optimization, data assimilation, parameter inversion, uncertainty quantification, etc.**
-



Four widely applicable strategies

- **Employ dynamic runtime systems based on directed acyclic task graphs (DAGs)**
 - ◆ e.g., ADLB, Argo, Charm++, HPX, kokkos, Legion, OmpSs, Quark, STAPL, StarPU
 - **Exploit data sparsity of hierarchical low-rank type**
 - ◆ meet the “curse of dimensionality” with the “blessing of low rank”
 - **Employ high-order discretizations**
 - **Code to the architecture, but present an abstract API**
-

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**

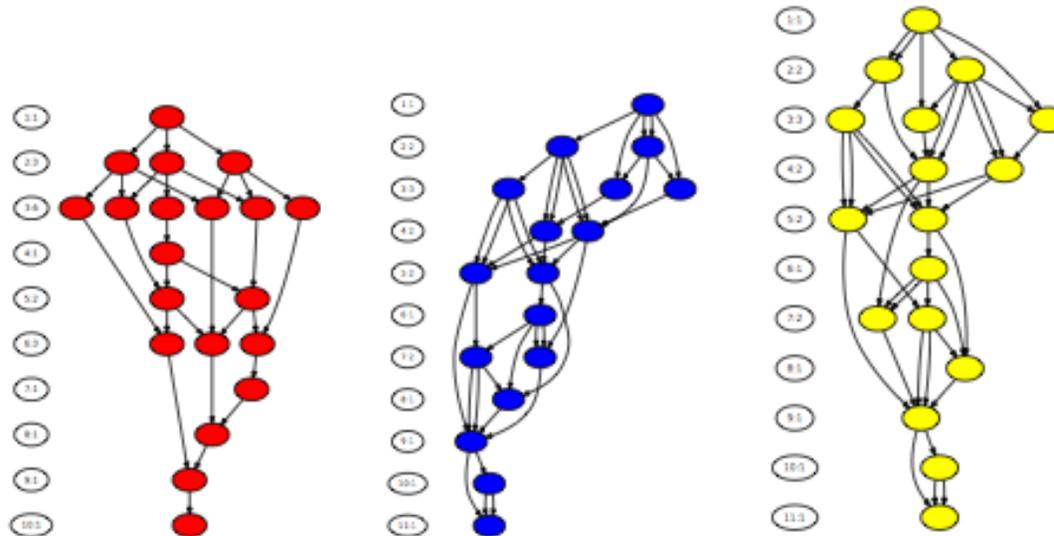
- **Disadvantages**

- ◆ **pay overhead of managing task graph**
 - ◆ **potentially lose some memory locality**
-

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

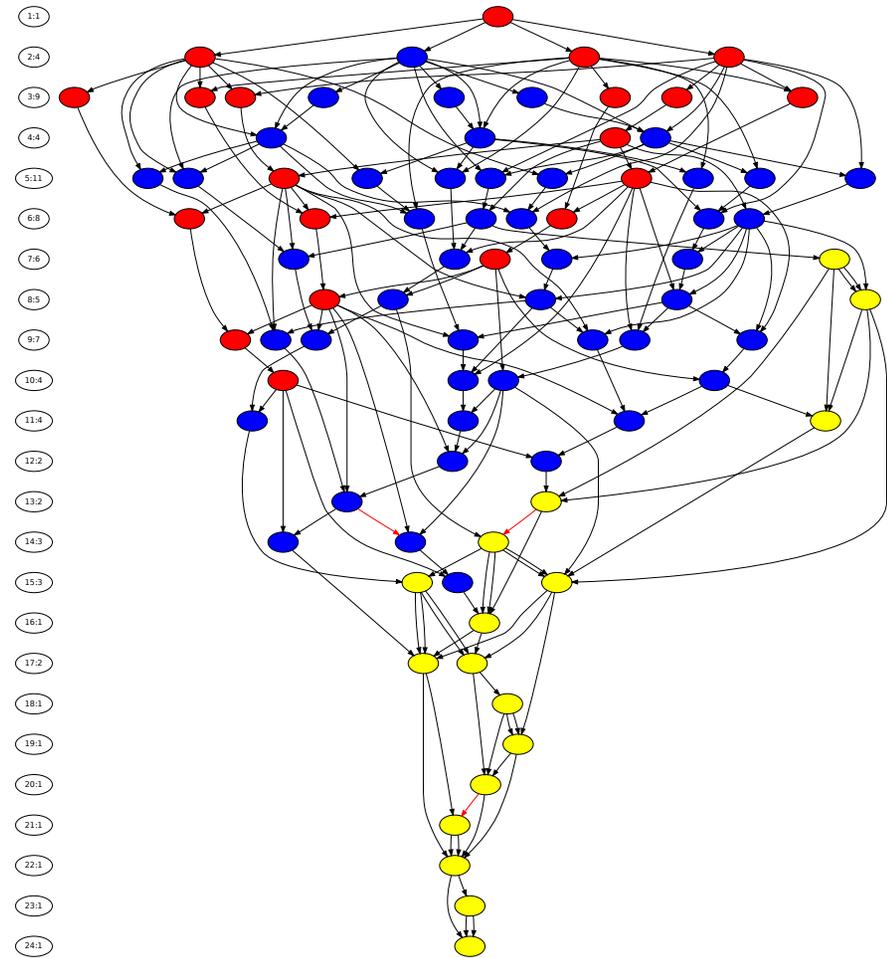
$$Ax = \lambda Bx$$

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



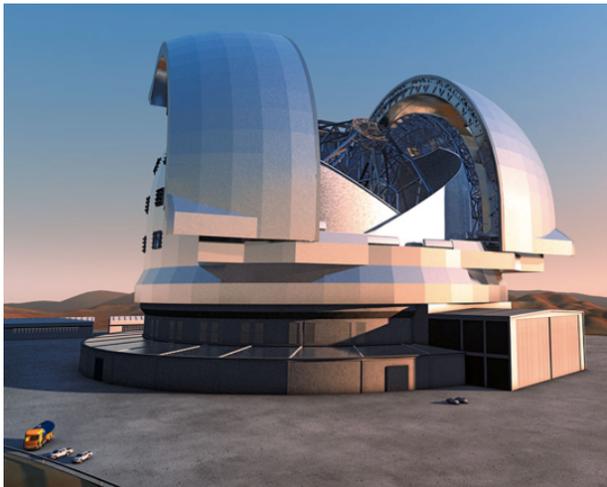
Loop nests and subroutine calls, with their over-orderings, can be replaced with DAGs

- Diagram shows a dataflow ordering of the steps of a 4×4 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

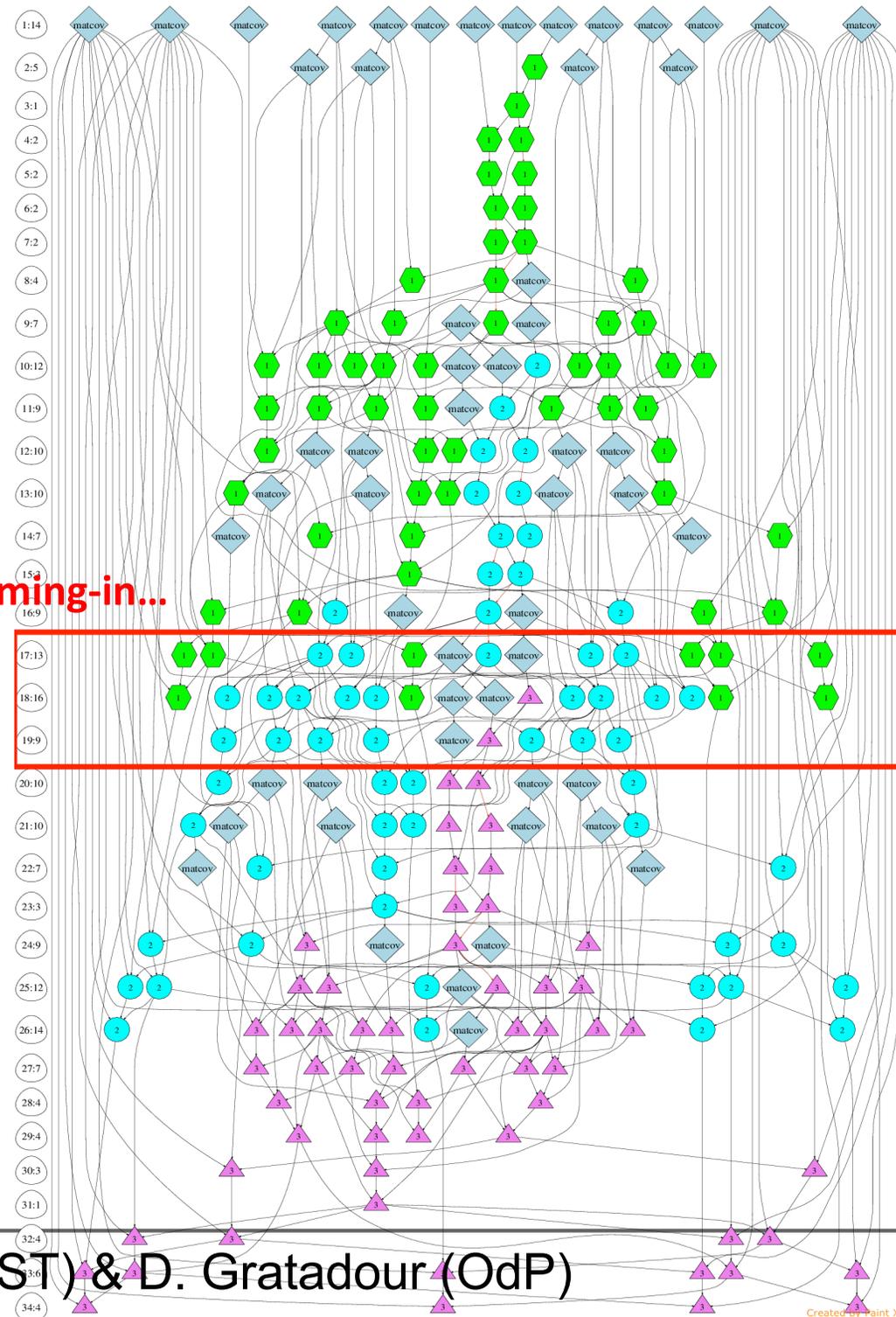


Loops can be overlapped in time

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



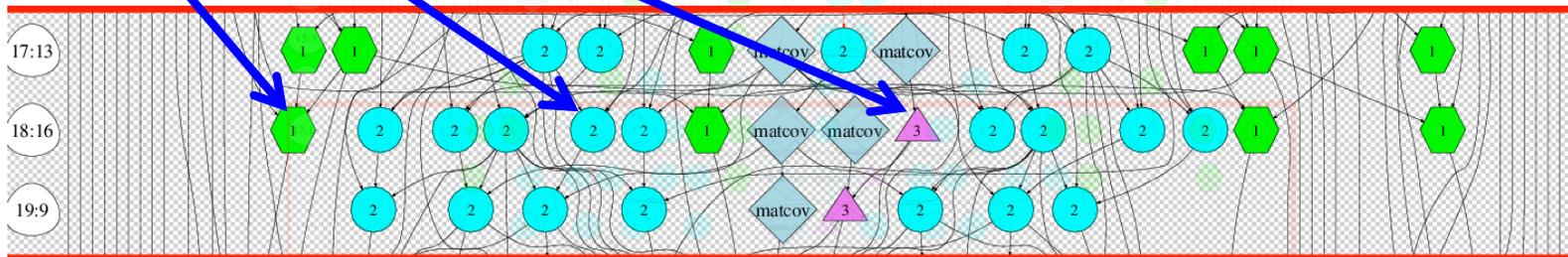
Zooming-in...



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

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)



Hierarchically low-rank operators

- **Advantages**

- ◆ **shrink memory footprints to live higher on the memory hierarchy**
 - higher means quick access
- ◆ **reduce operation counts**
- ◆ **tune work to accuracy requirements**
 - e.g., preconditioner versus solver

- **Disadvantages**

- ◆ **pay cost of compression**
 - ◆ **not all operators compress well**
-

Key tool: hierarchical matrices

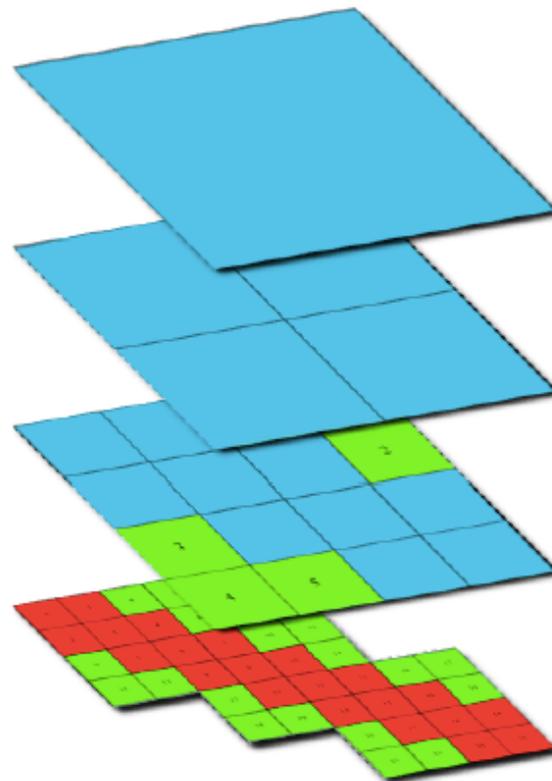
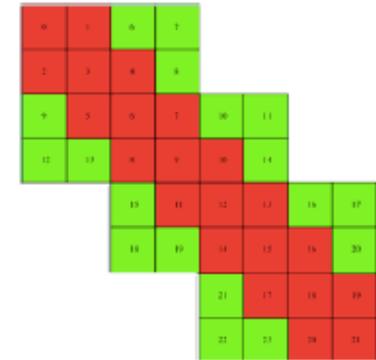
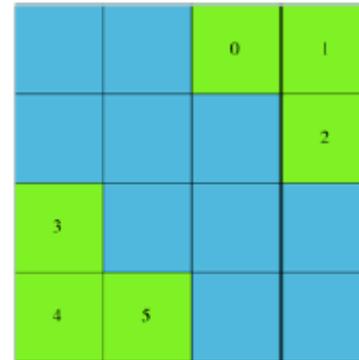
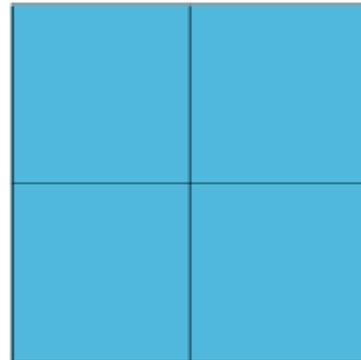
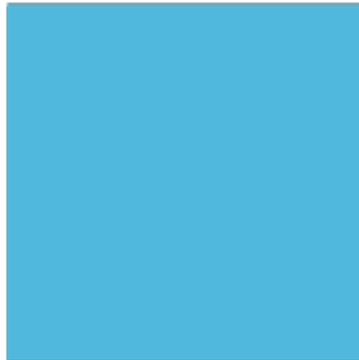
- [Hackbusch, 1999] : off-diagonal blocks of typical differential and integral operators have low effective rank
 - By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n :
 - polynomial in k
 - lin-log in n
 - constants carry the day
 - Such hierarchical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

Example: 1D Laplacian

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

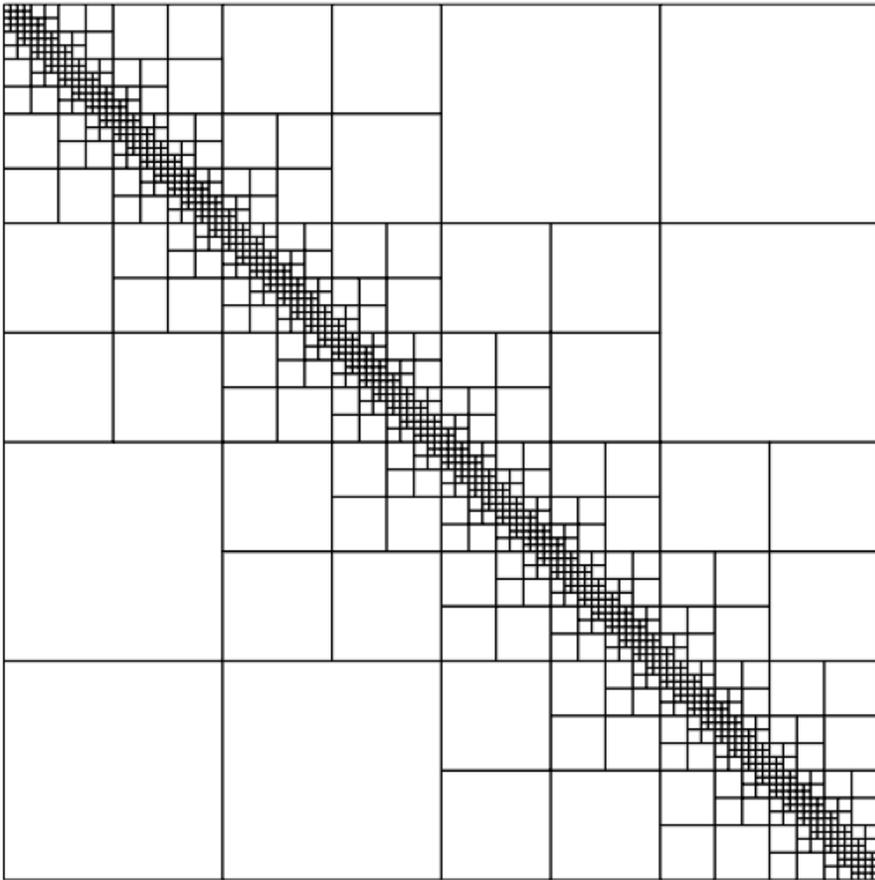
$$A^{-1} = \frac{1}{8} \times \left[\begin{array}{ccc|cccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 12 & 10 & 8 & 6 & 4 & 2 \\ 5 & 10 & 15 & 12 & 9 & 6 & 3 \\ \hline 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 15 & 10 & 5 \\ 2 & 4 & 6 & 8 & 10 & 12 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \right] \longleftrightarrow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

Recursive construction of an H -matrix

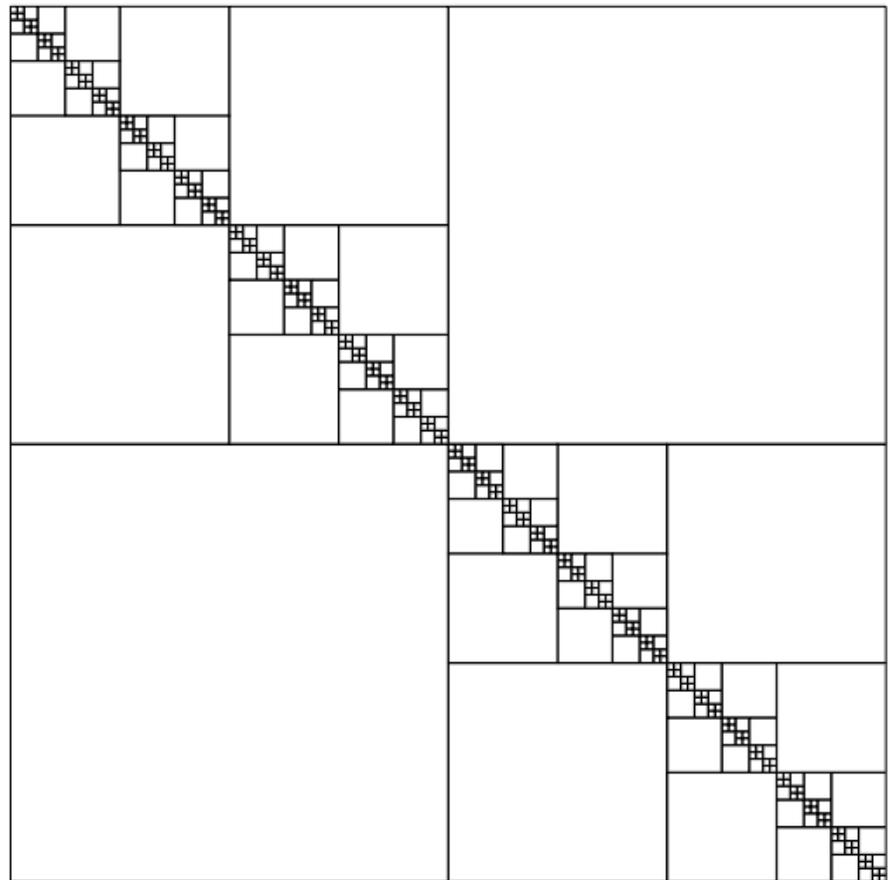


c/o W. Boukaram & G. Turkiyyah (KAUST)

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



strong admissibility



weak admissibility

After Hackbusch, et al., 2003

Employ high-order discretizations

- **Advantages**

- ◆ **shrink memory footprints to live higher on the memory hierarchy**
 - higher means shorter latency
- ◆ **increase arithmetic intensity**
- ◆ **reduce operation counts**

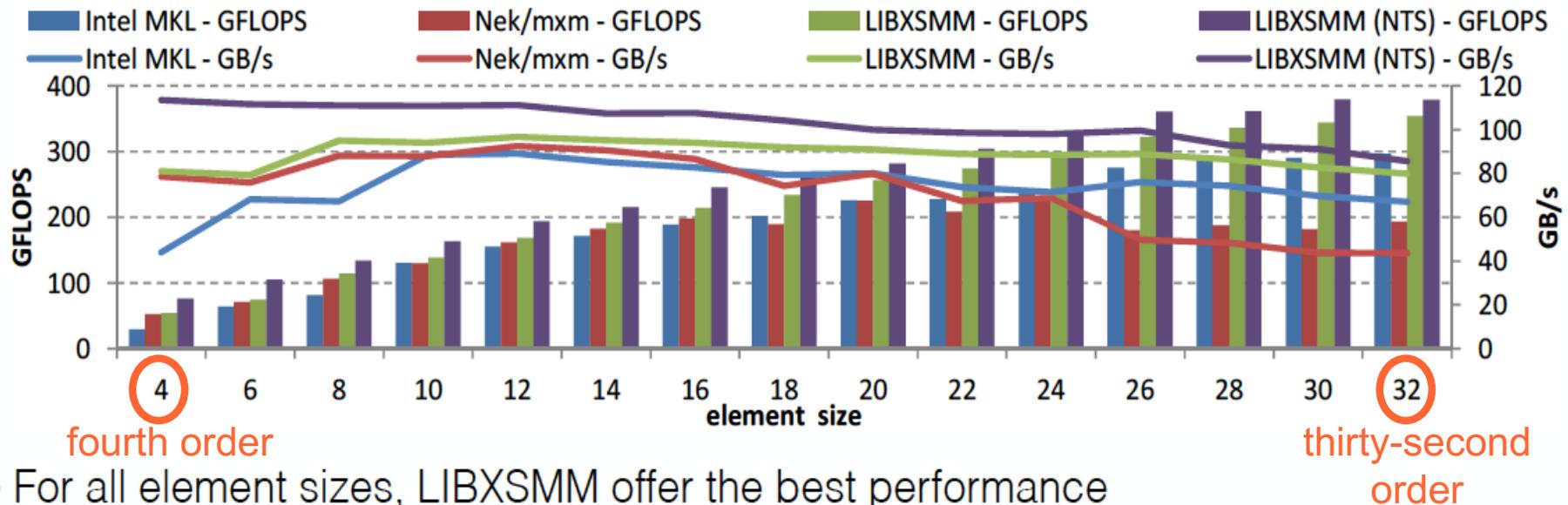
- **Disadvantages**

- ◆ **high-order operators less suited to some solvers**
 - e.g., algebraic multigrid, H -matrices*

* but see Gatto & Hesthaven, Dec 2016, on H for hp FEM

Performance effects of order in CFD

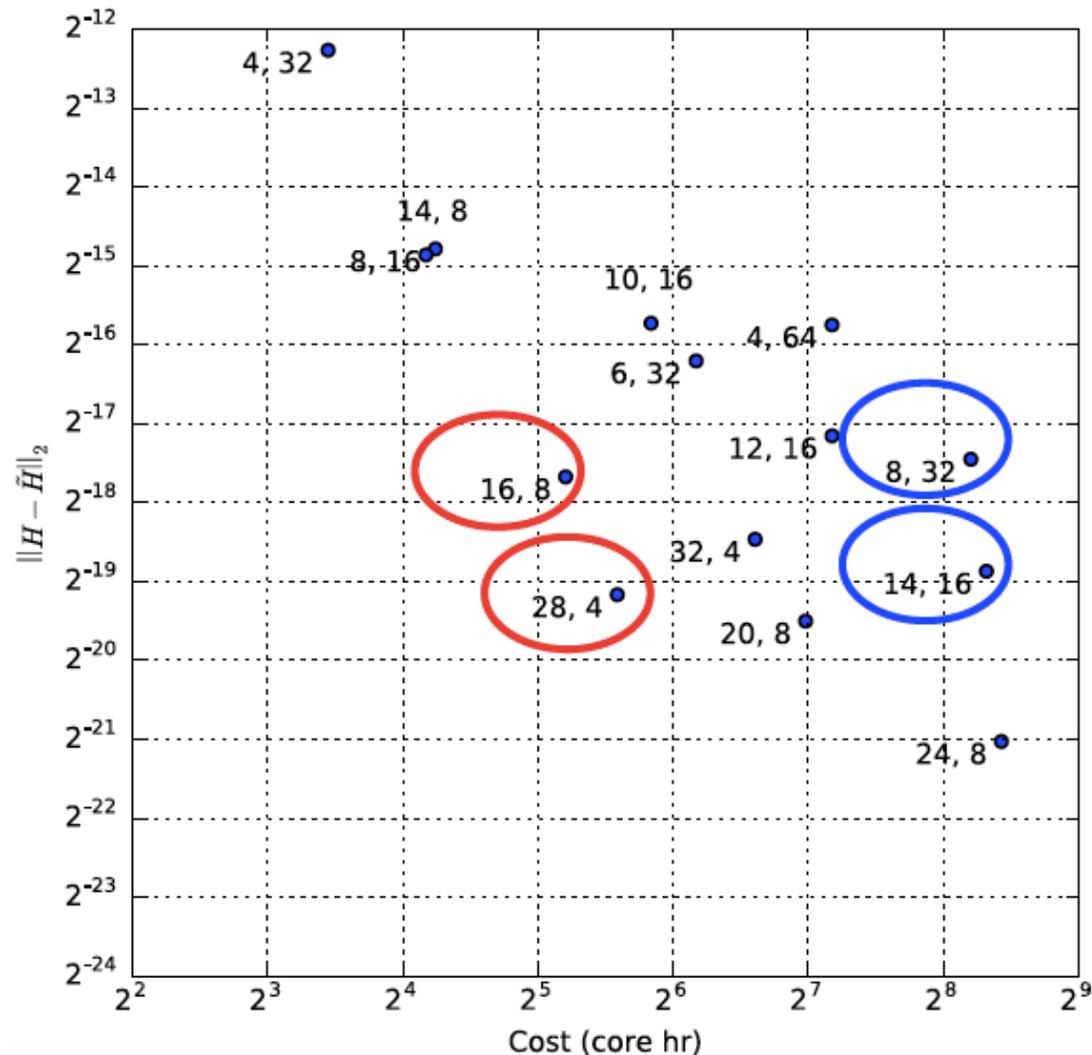
Helmholtz solve in spectral element code for incompressible Navier-Stokes



- For all element sizes, LIBXSMM offer the best performance
 - for order ≤ 16 , the difference is small because the computation are memory bandwidth bound
 - for for order ≤ 16 , a boost is possible with the non-temporal stores (101.6 GiB/s)
 - for order > 16 , LIBXSMM $\sim 2x$ is faster then Nek's `mxm_std` and up to 40% faster than Intel MKL

Runtime effects of order in CFD

Accuracy versus execution time as a function of order
Single-mode Rayleigh-Taylor instability



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**
-

Amdahl asks: where do the cycles go?

- **Dominant consumers in applications that occupy major supercomputer centers are:**
 - ◆ **Linear algebra on dense symmetric/Hermitian matrices**
 - Hamiltonians (Schroedinger) in chemistry/materials
 - Hessians in optimization
 - Schur complements in linear elasticity, Stokes & saddle points
 - covariance matrices in statistics
 - ◆ **Poisson solves**
 - highest order operator in many PDEs in fluid and solid mechanics, E&M, DFT, MD, etc.
 - diffusion, gravitation, electrostatics, incompressibility, equilibrium, Helmholtz, image processing – even analysis of graphs
-

Mapping algorithms to drivers

PhD thesis topics in the Extreme Computing Research Center at KAUST must address at least one of the four algorithmic drivers

Student	Algorithm/Kernel	Reduce Synchronization	Increase Intensity	Increase Concurrency	Algorithmic Resilience	New Capabilities
Abdelfattah	BLAS2		X	X		
Abduljabbar	FMM	X	X	X		
AlFarhan	Unstruct. PDEs	X		X		
AlHarthi	BEM	X	X	X		
AlOnazi	Multigrid	X		X	X	
Boukaram	H-BLAS		X	X		
Charara	BLAS2/3		X	X		
Chavez	H-Schur		X	X		
Ibeid	FMM precondition.	X	X	X		
Liu	Nonlinear precondition.	X			X	X
Malas	Stencil eval.		X	X		
Peng	Non-neg. mat. fact.			X		X
Sukkari	Eigen/SVD		X	X		

Examples being developed at KAUST's Extreme Computing Research Center

- **QDWH-SVD**, a 4-year-old SVD algorithm that performs more flops but beats state-of-the-art on MICs and GPUs and distributed memory systems
 - **KBLAS**, a library that improves upon or fills holes in L2/L3 BLAS for GPUs and MICs, including batched and hierarchically low-rank routines
 - **BDDC**, a linear preconditioner that performs extra local flops on interfaces for low condition number guarantee in high-contrast elliptic problems
 - **FMM(ϵ)**, a 31-year-old $O(N)$ solver for potential problems, used in low accuracy as a FEM preconditioner and scaled out on MICs and GPUs
 - **ACR(ϵ)**, a new spin on 52-year-old cyclic reduction that recursively uses \mathcal{H} matrices on Schur complements to reduce $O(N^2)$ complexity to $O(N \log^2 N)$
 - **M/ASPIN**, nonlinear preconditioners that replace most of the globally synchronized steps of Newton iteration with asynchronous local problems
 - **NekBox**, a MIC-optimized version of CFD code Nek5000 that uses extremely high-order schemes to minimize runtime to a given accuracy
-

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:

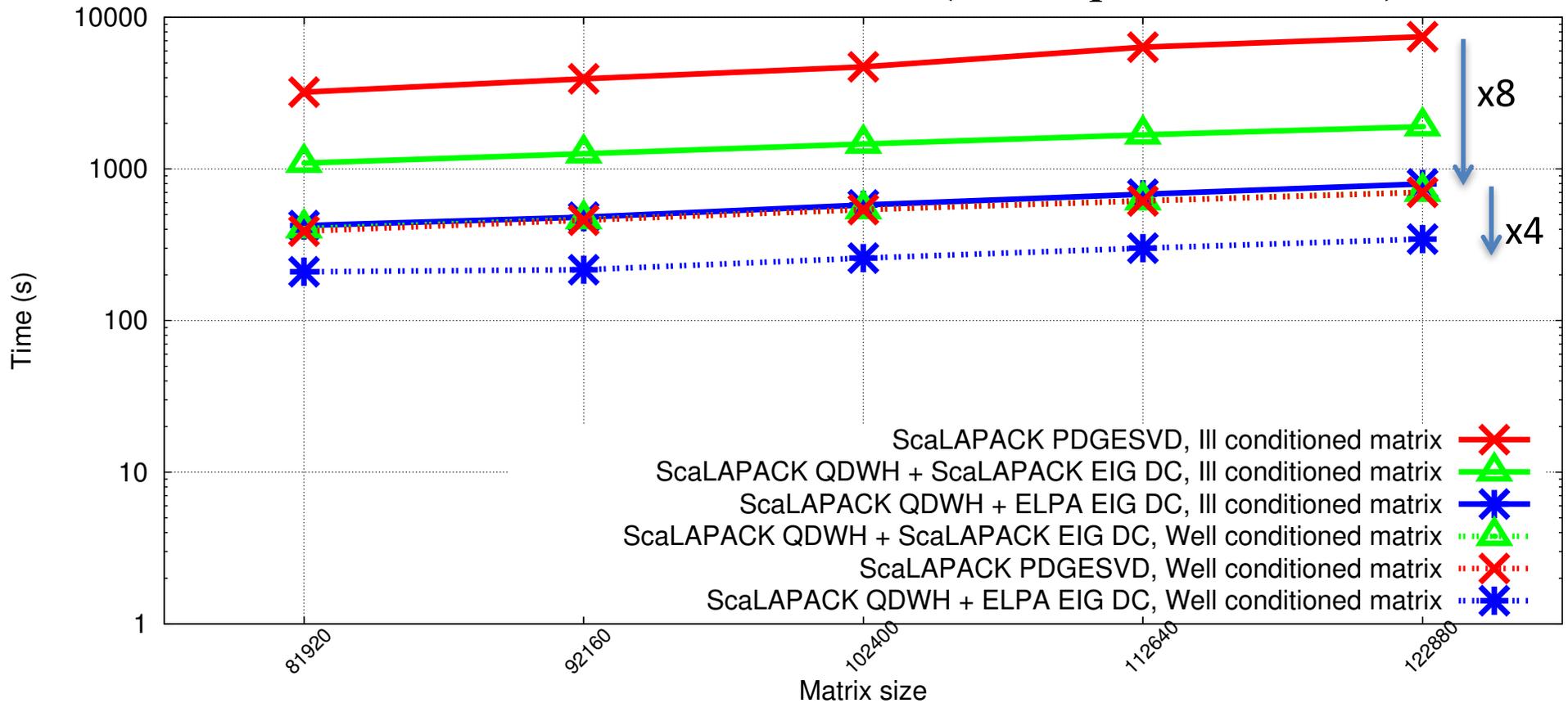
$$\begin{array}{cc} \text{polar} & \text{sym eigen} \\ A = U_p H & H = V \Sigma V^* \end{array}$$

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

- 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

576 nodes of 64-core Intel KNL (cache/quadrant mode)



fastest dense SVD

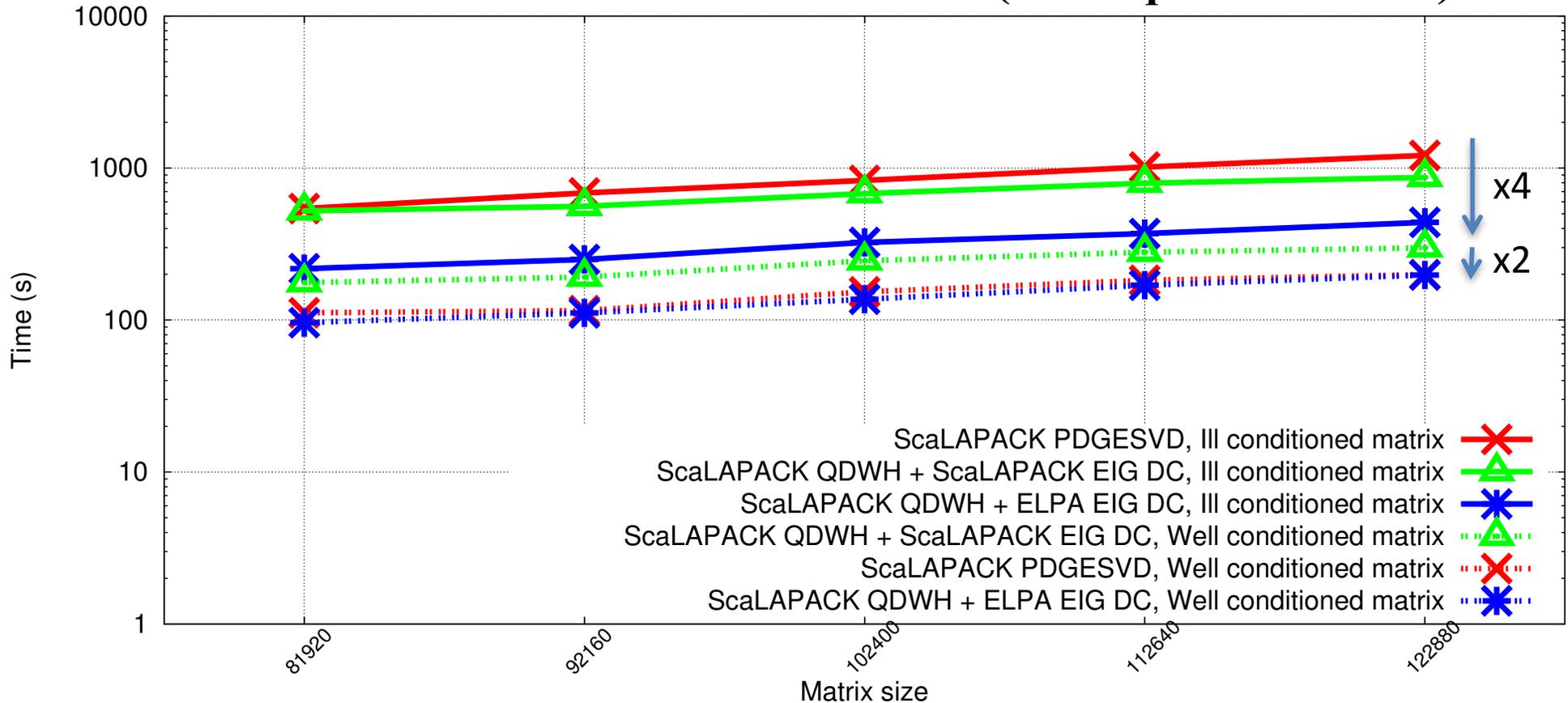
c/o D. Sukkari & H. Ltaief (KAUST)

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



QDWH-SVD

1152 nodes of 32-core Intel Haswell (cache/quadrant mode)



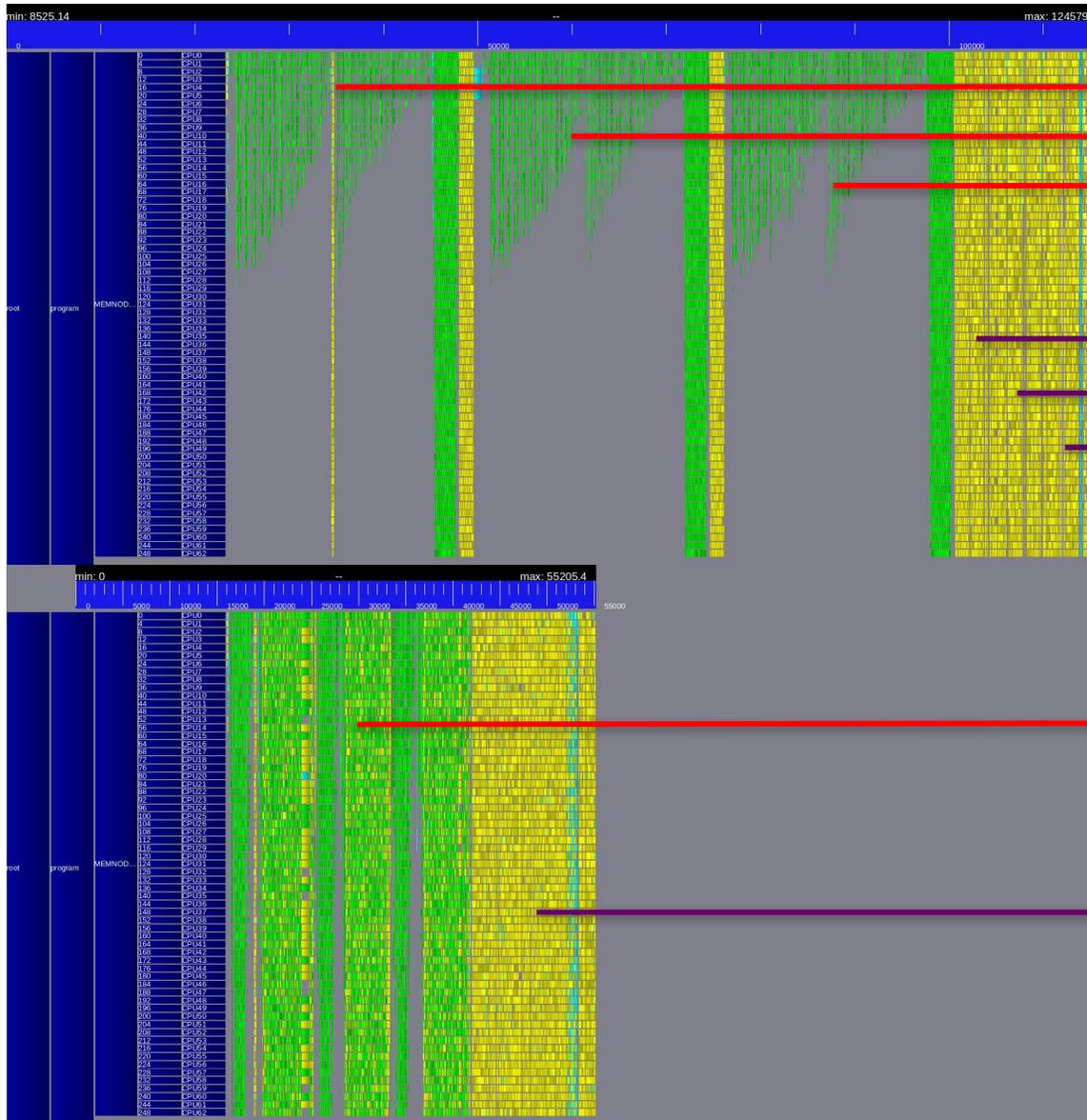
Is being integrated into Cray's LibSci w/A. Esposito (Cray)
Extensions underway to Zolotarev's method w/Y. Nakatsukasa (Oxford)



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

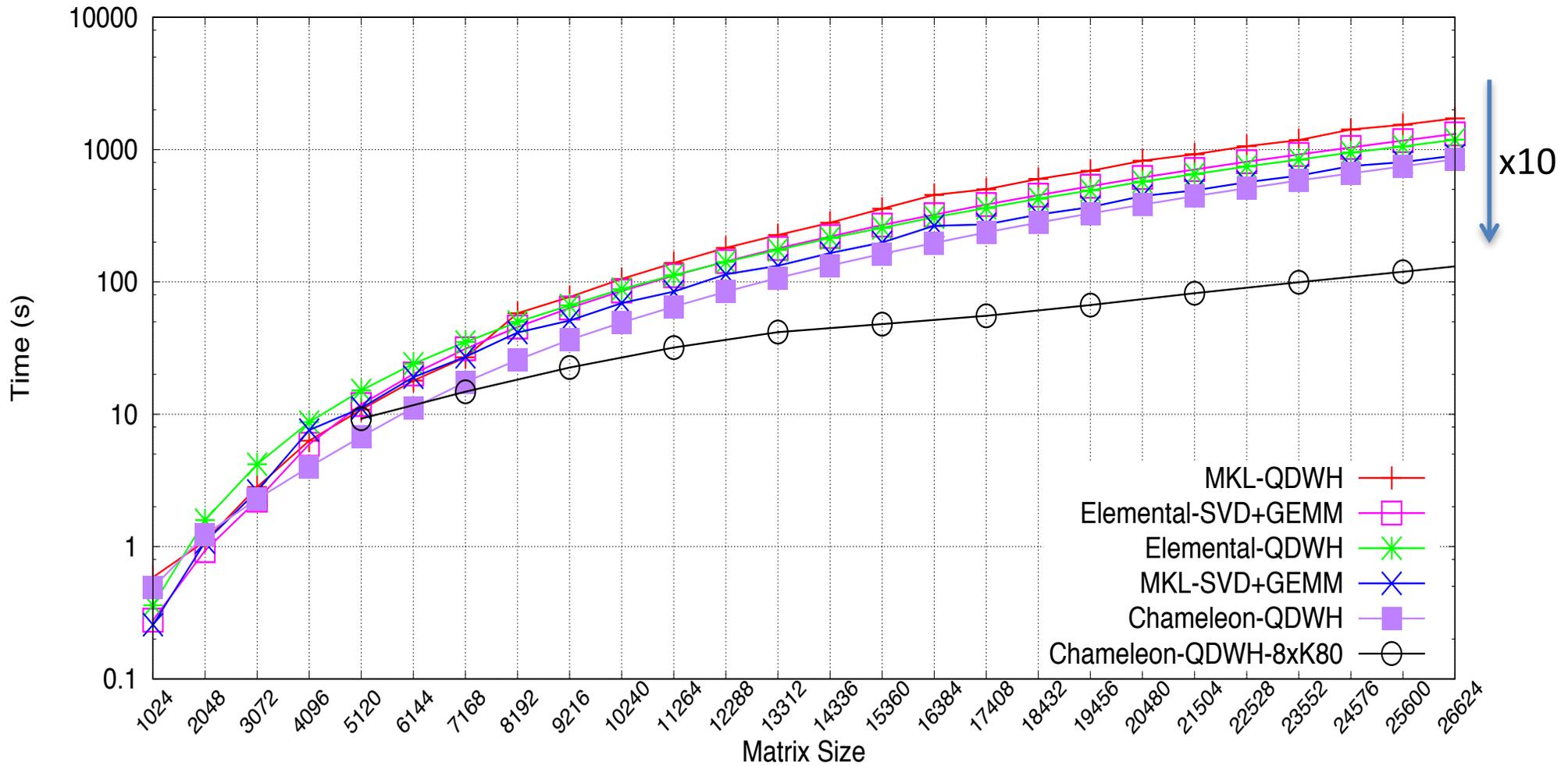
Sukkari et al., IEEE TDPS'17

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



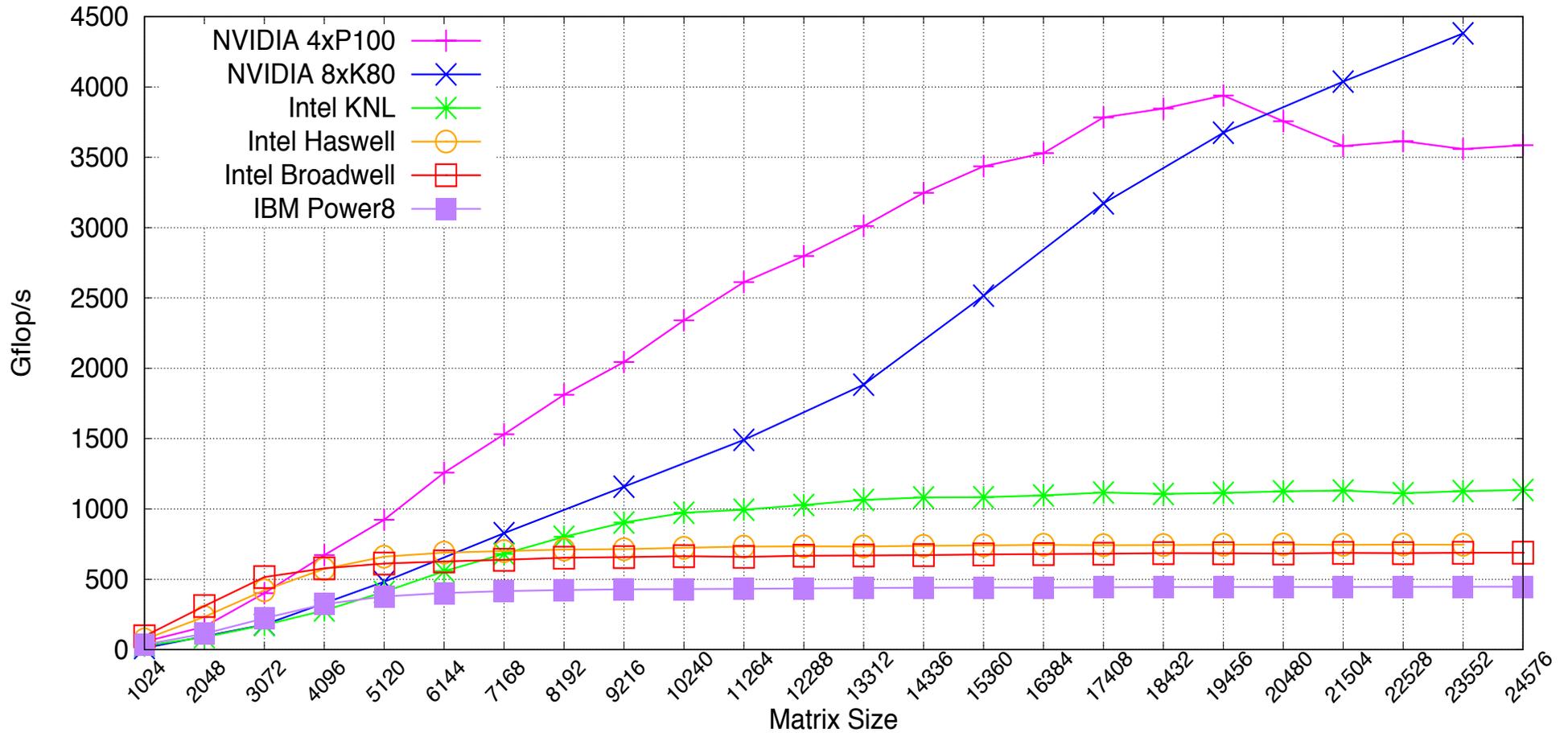
QDWH-SVD, taskified on hybrid architecture

32-cores Intel Intel Haswell + 8 NVIDIA K80s



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

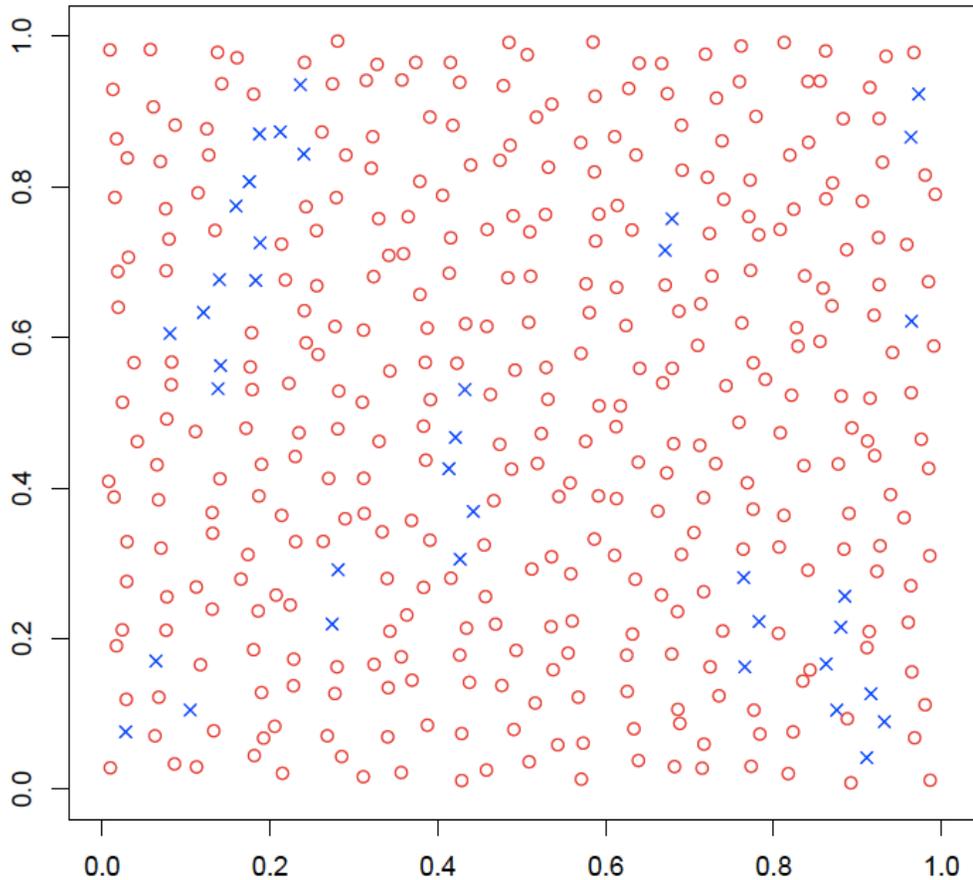
S. Abdulah, H. Ltaief, Y. Sun, M. Genton & D. Keyes
TDPS (2017, submitted)

Large dense symmetric systems arise as covariance matrices in spatial statistics

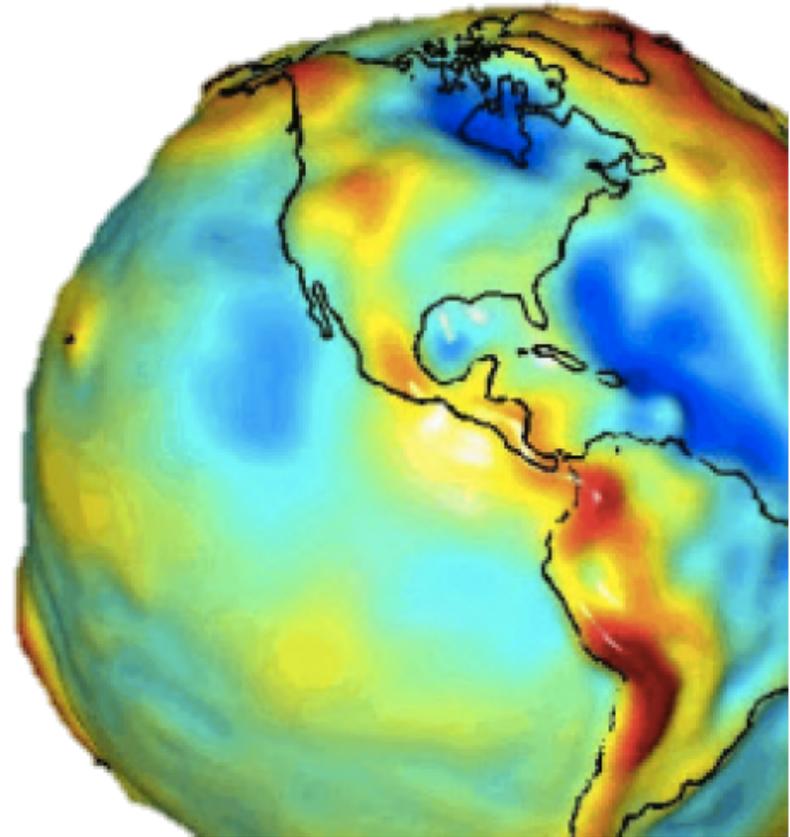
- **Climate and weather applications have many measurements located regularly or irregularly in a region; prediction is needed at other locations**
- **Modeled as realization of Gaussian or Matérn spatial random field, with parameters to be fit**
- **Leads to evaluating the log-likelihood function involving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



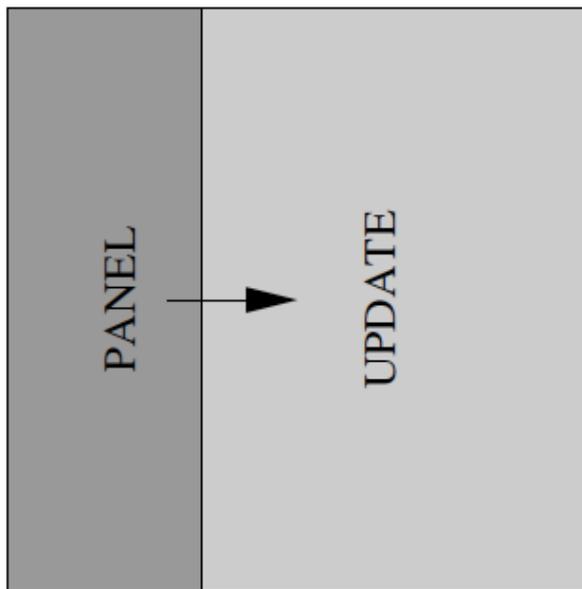
**362 measured points and
38 target points irregularly
distributed in unit square**



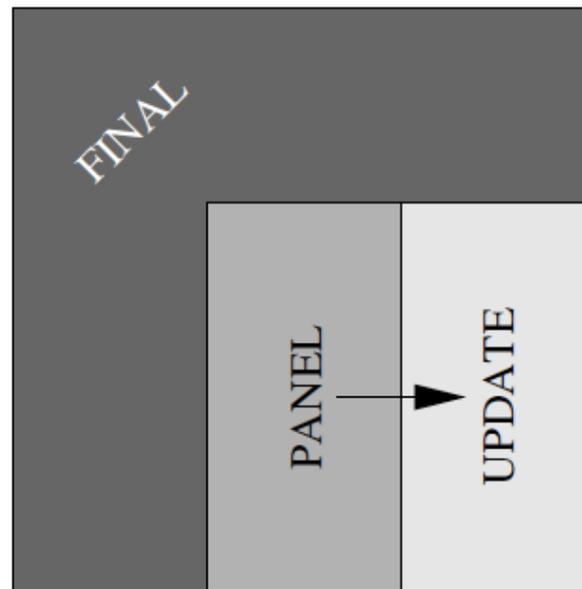
**Global temperature
data on sphere**

LAPACK DPOTRF

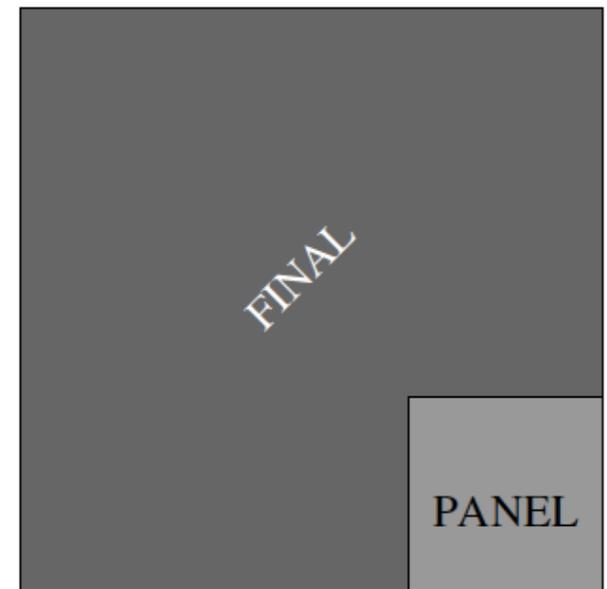
- **Classical algorithm (1990s) involves BLAS L2 panel updates and BLAS L3 trailing matrix updates**



(a) First step.



(b) Second step.

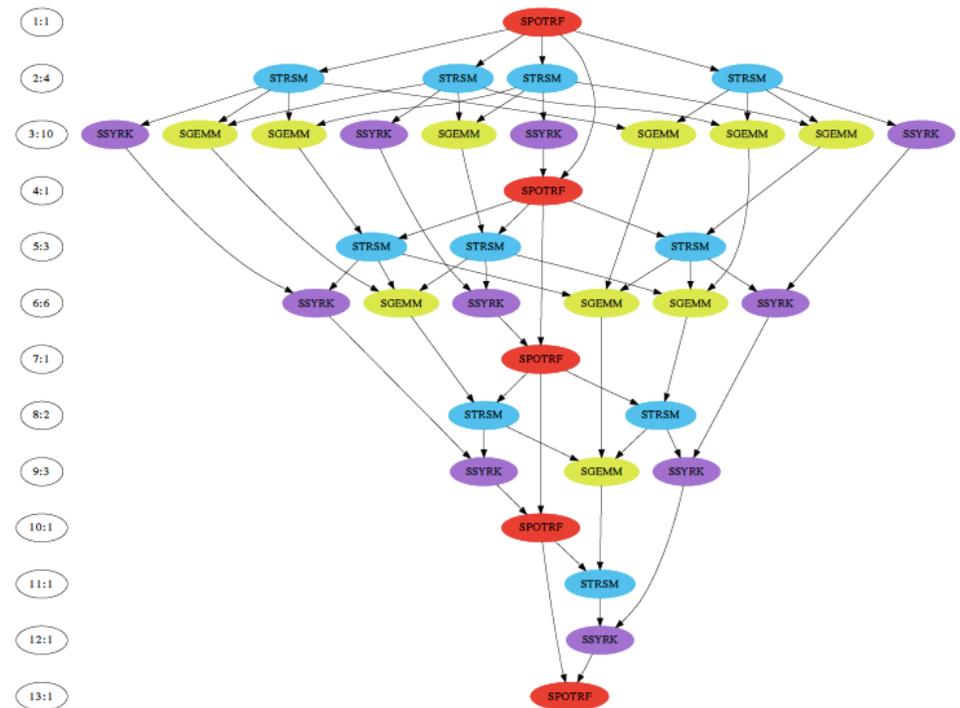
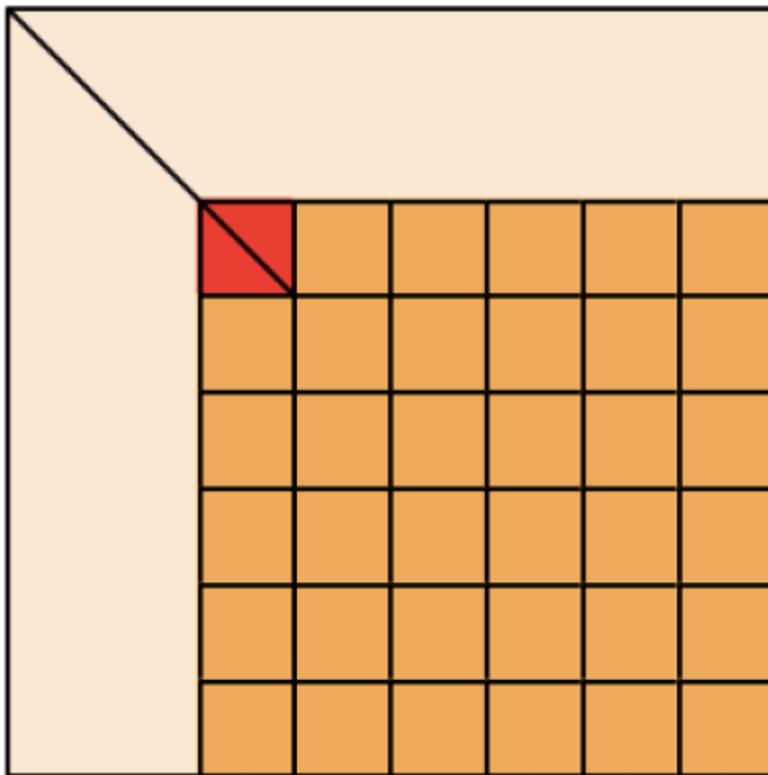


(c) Third step.



PLASMA/CHAMELEON DPOTRF

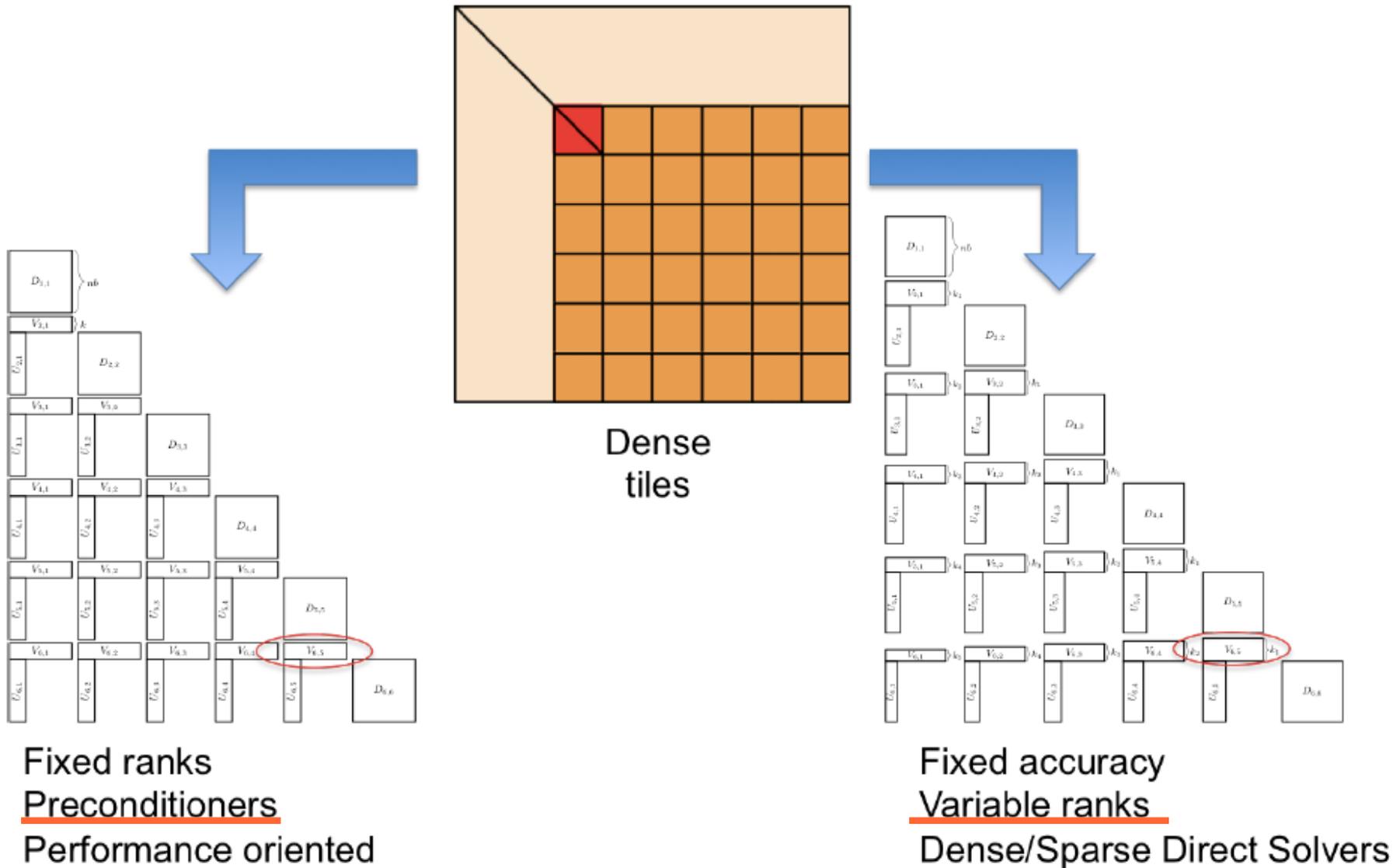
- Tile algorithm (PLASMA, FLAME, 2010s) involves mostly BLAS L3 operations within tiles scheduled with a DAG



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: $V_{(i,k)} = V_{(i,k)} \times D_{(k,k)}^{-1}$. 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: $D_{(j,j)} = D_{(j,j)} - (U_{(j,k)} \times V_{(j,k)}^T) \times (U_{(j,k)} \times V_{(j,k)}^T)^T$. 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.
-

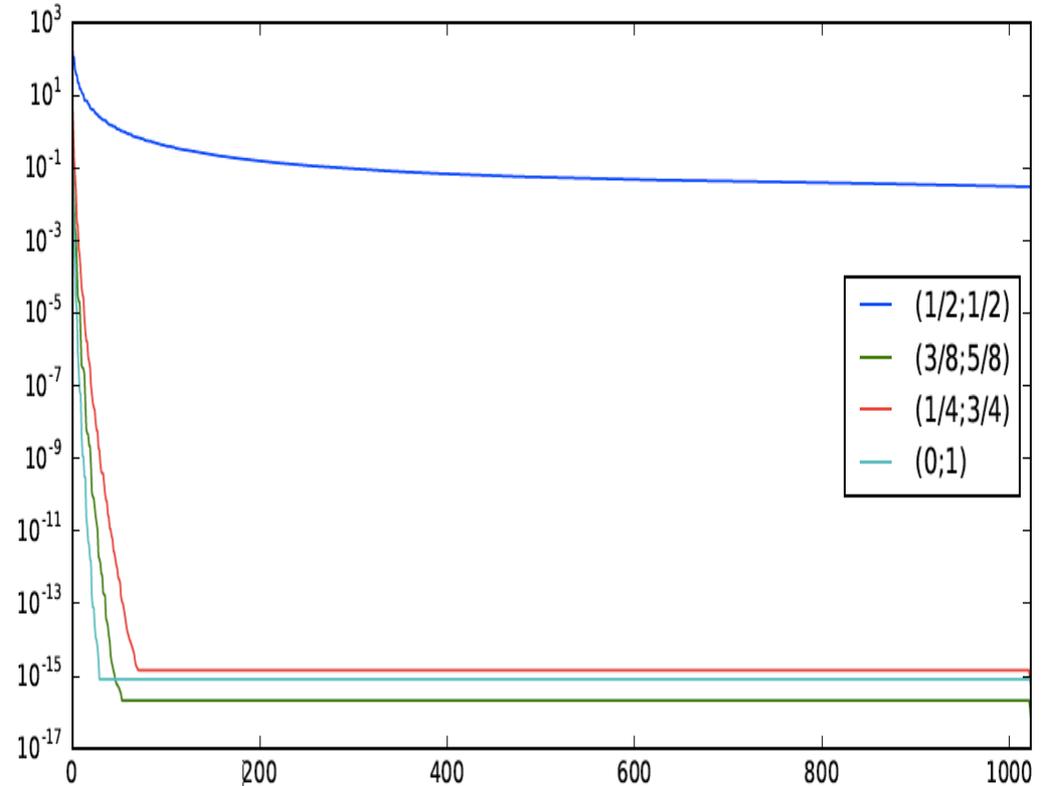
Data-sparse operations for Cholesky variants



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

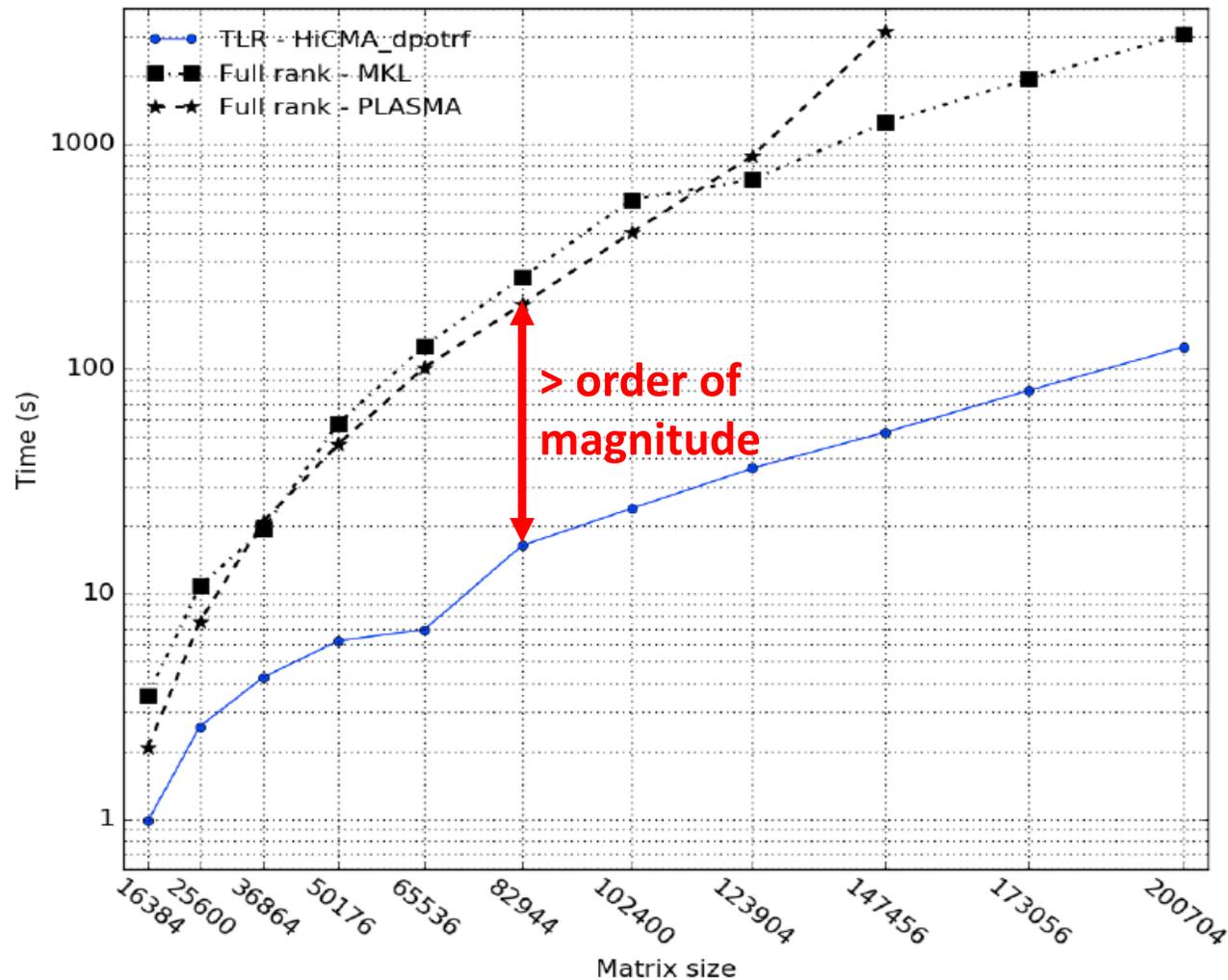


Compressibility of four typical blocks, for
Frobenius accuracy of 10^{-9}



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

Tile low-rank Cholesky, time per backsolve



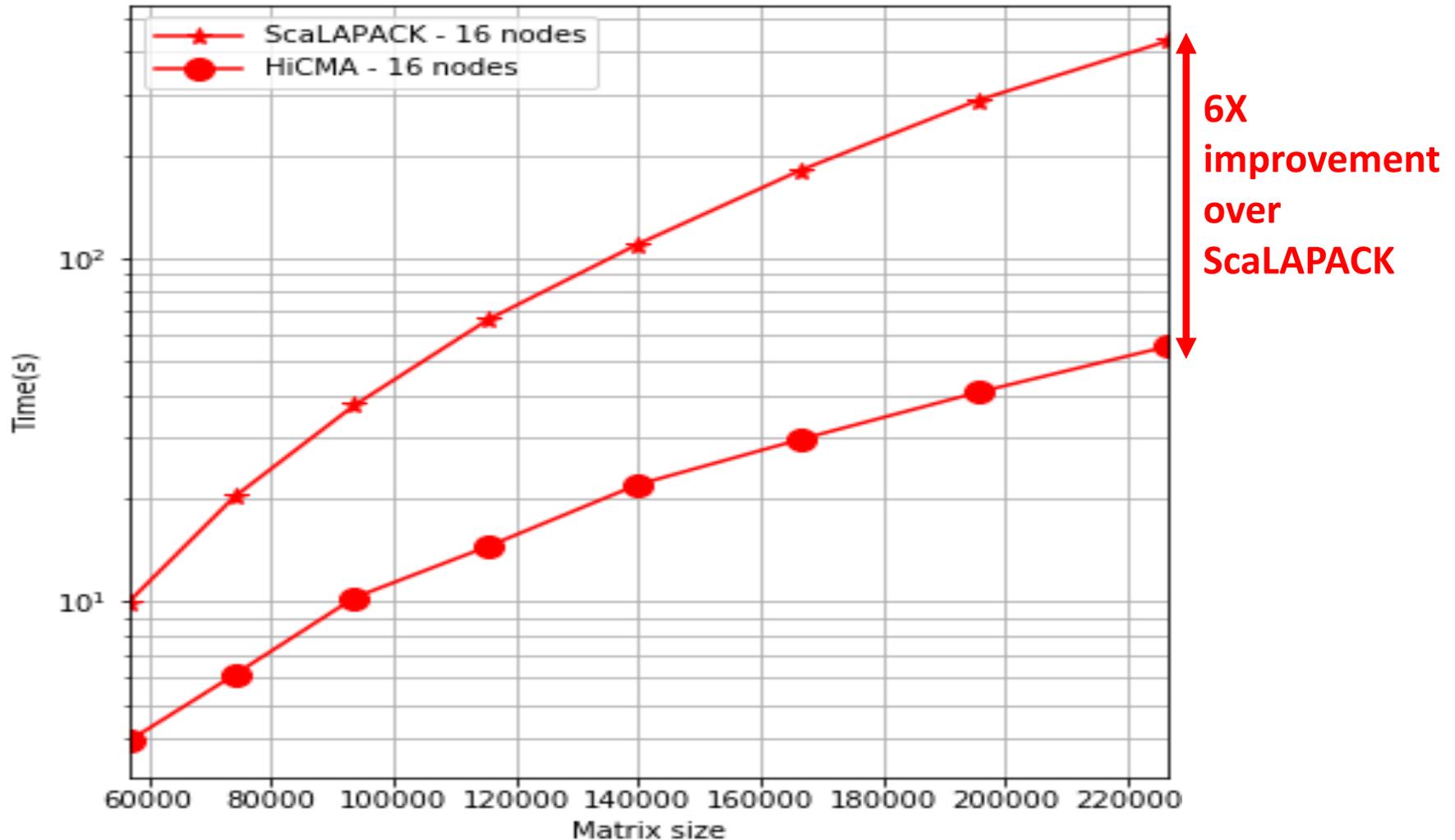
On 2-socket 18-core Intel Haswell @ 2.3GHz

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

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



Distributed memory TLR Cholesky (preliminary implementation)



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

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



KBLAS

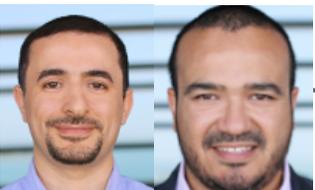
- ❖ **Subset of L2/L3 BLAS targeting GPU and Intel MIC**
 - ❖ GEMV, SYMV, TRSM, TRMM
- ❖ **Reduces communication and increases concurrency in these memory BW bound operations**
- ❖ **Batched BLAS for small sizes on GPUs**
 - ❖ TRSM, TRMM, SYRK, POTRF, POTRS, POSV, TRTRI, LAUUM, POTRI, POTI
- ❖ **Recursive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

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



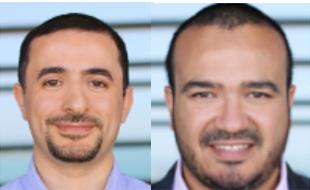
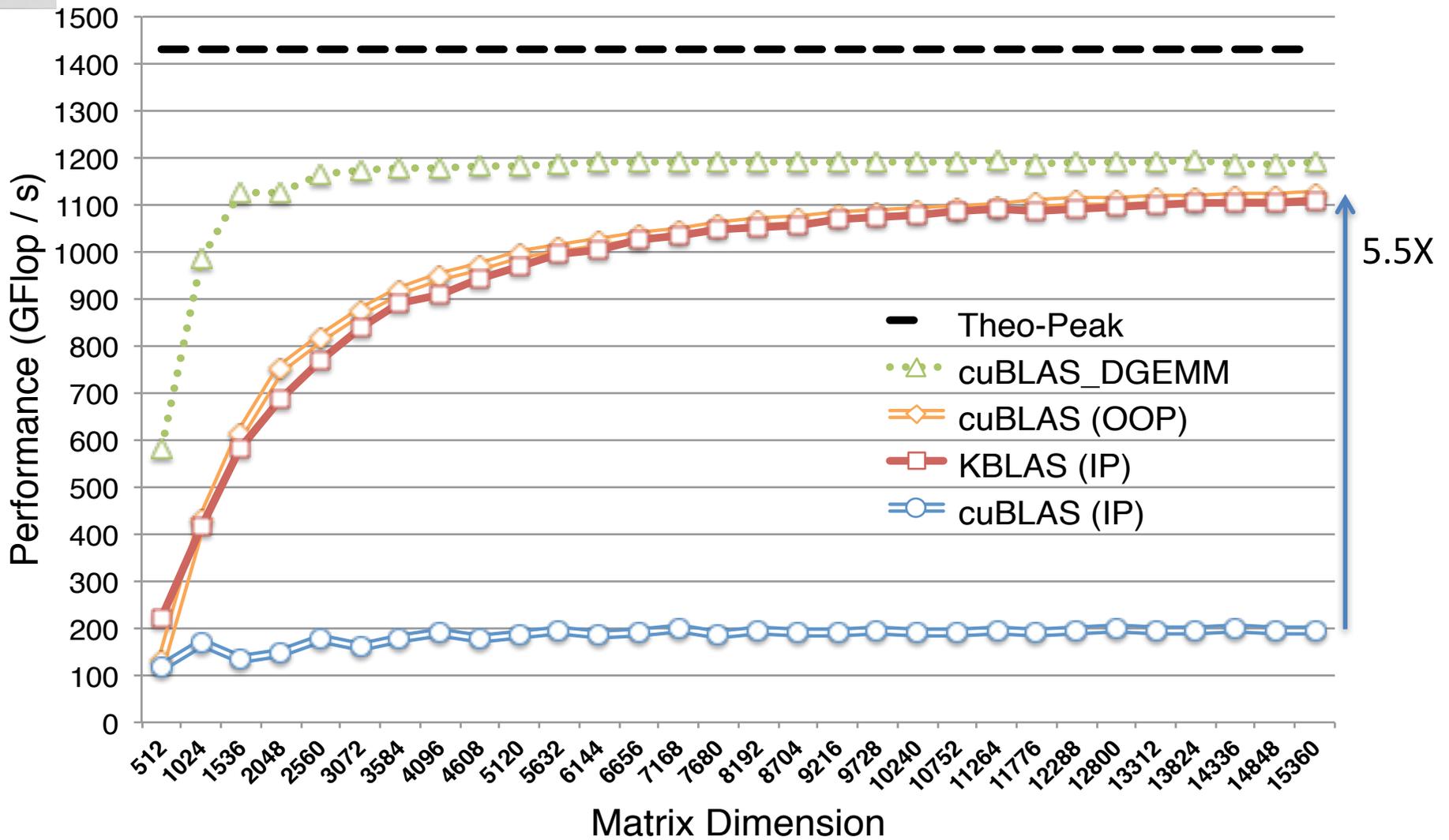
Recursively defined KBLAS operations for symmetric systems

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





KBLAS DTRMM

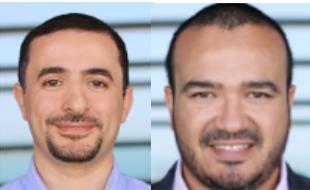
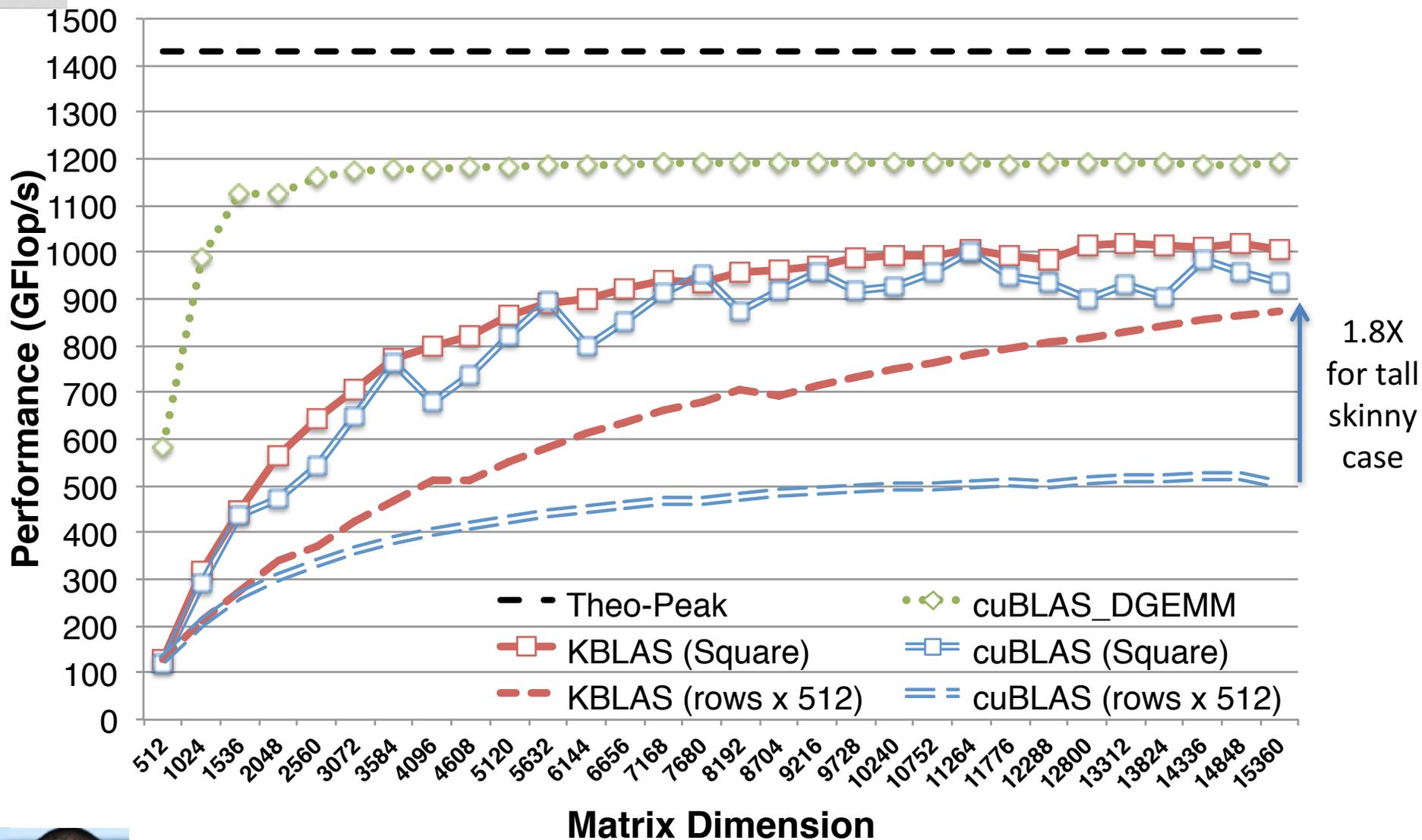


c/o A. Charara & H. Ltaief (KAUST)

Charara et al., Best papers, Europar'16
available: <https://github.com/ecrc/kblas>



KBLAS DTRSM

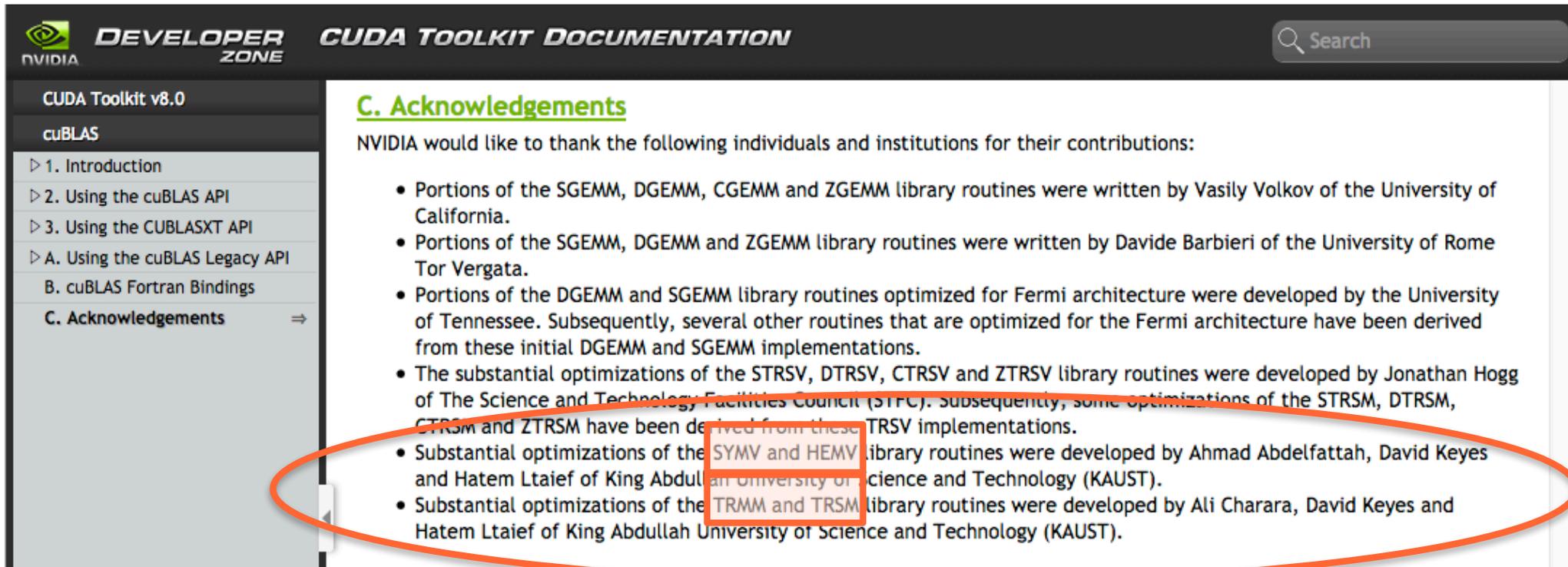


c/o A. Charara & H. Ltaief (KAUST)

Charara et al., Best papers, Europar'16
available: <https://github.com/ecrc/kblas>



KBLAS in cuBLAS (currently 8.0)



DEVELOPER ZONE NVIDIA **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).



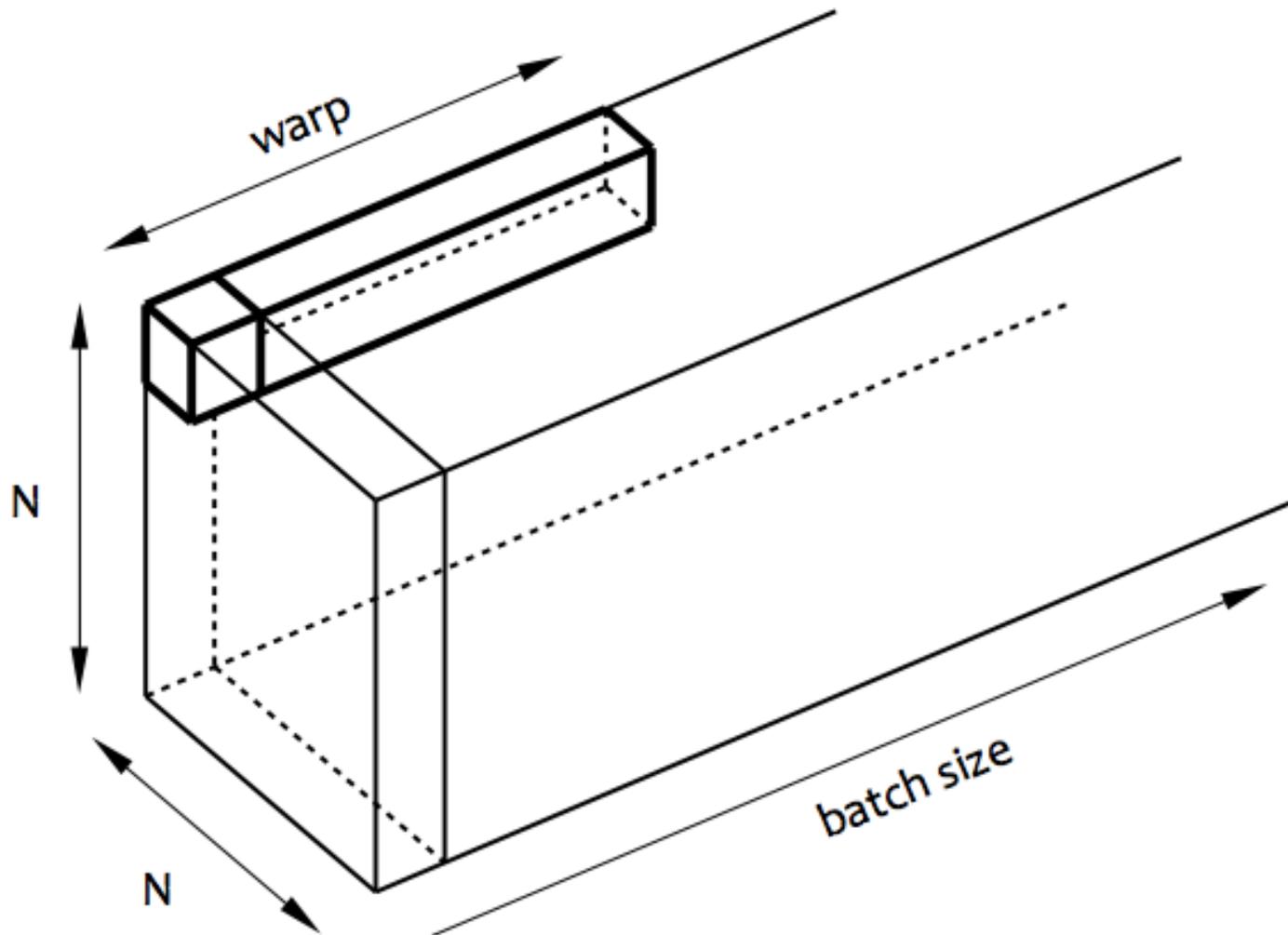
c/o A. Abdelfattah (ICL, KAUST'15), A. Charara & H. Ltaief (KAUST)



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



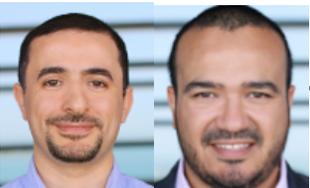
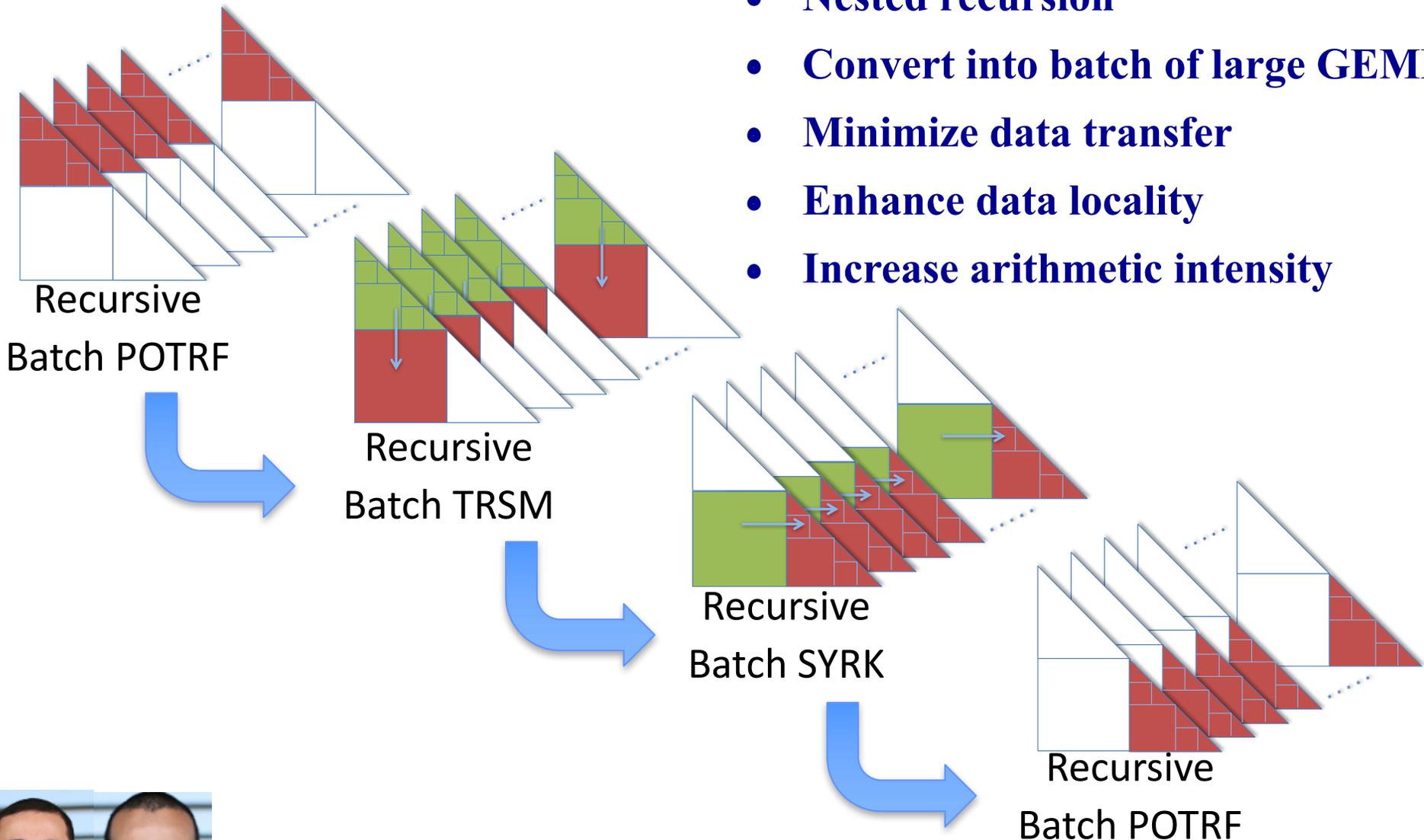
c/o Jacob Kurzak (ICL, U Tennessee)



KBLAS

Example: Batched POTRF

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

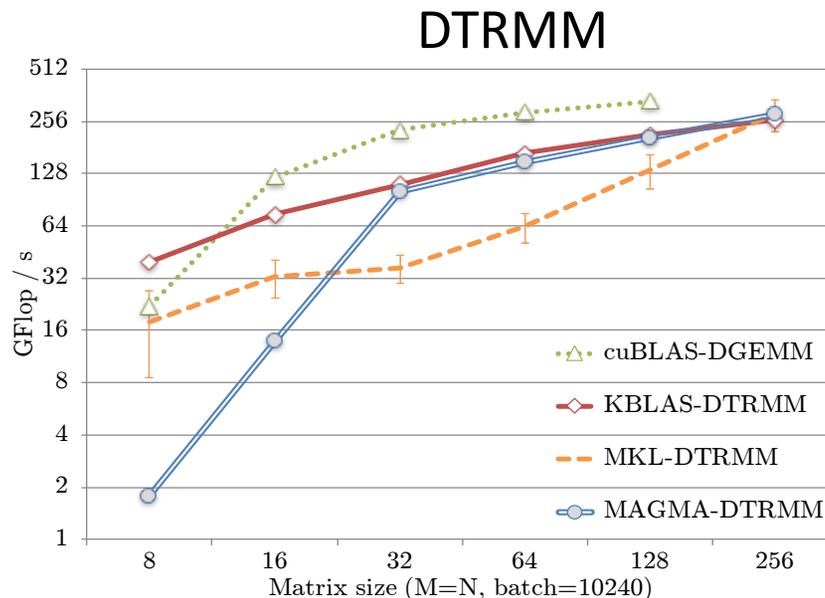
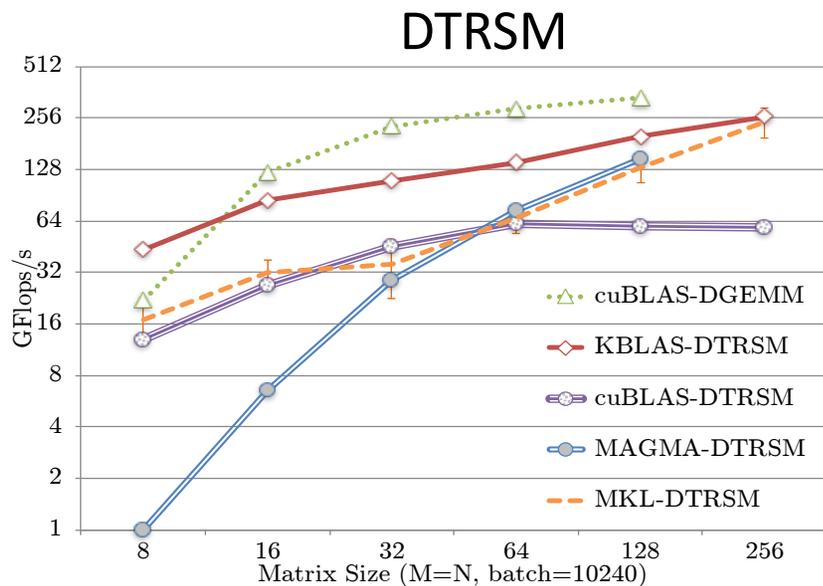


c/o A. Charara & H. Ltaief (KAUST)

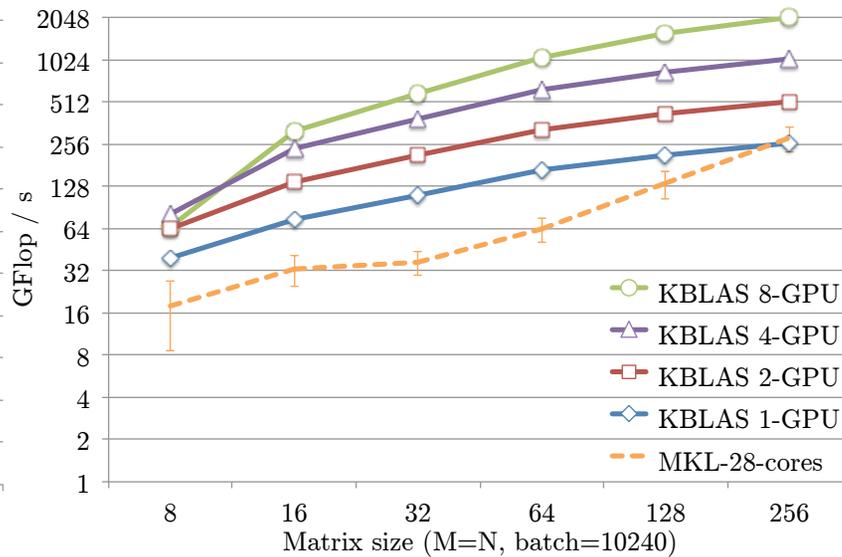
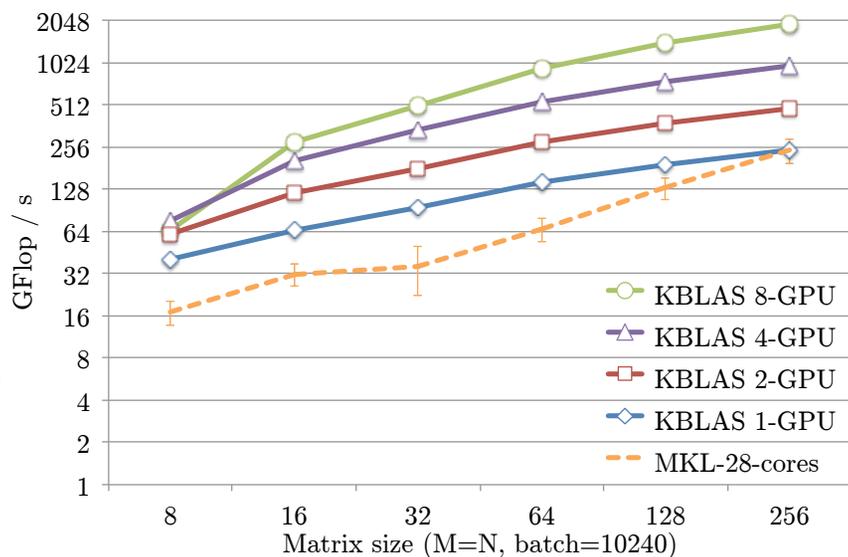


Batched KBLAS performance comparisons

Single K40 (MKL on 28-core Broadwell)



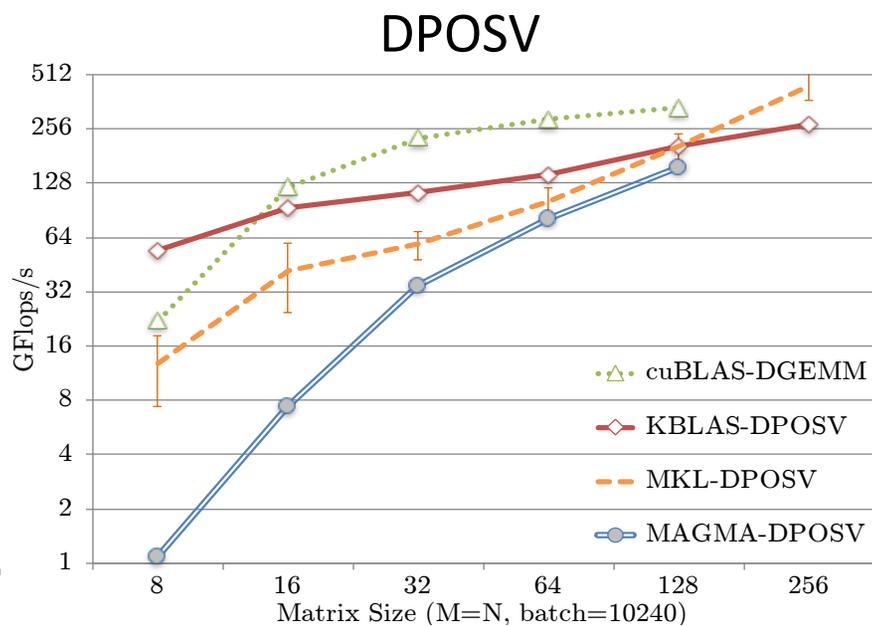
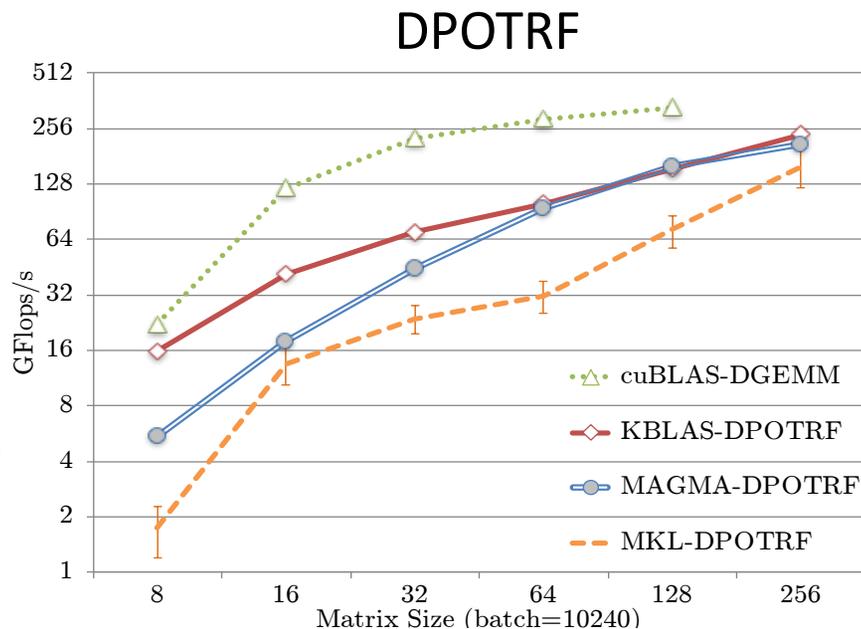
Multiple K40s (MKL on 28-core Broadwell)



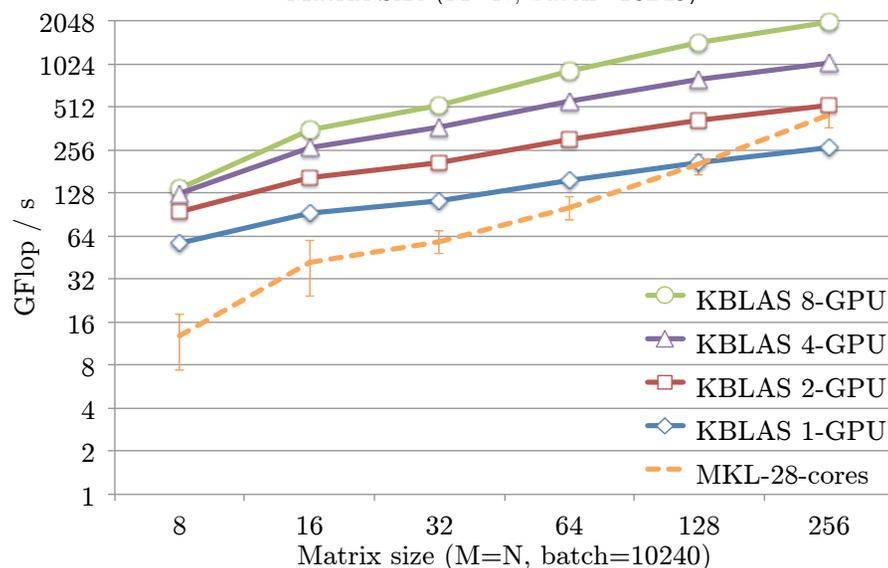
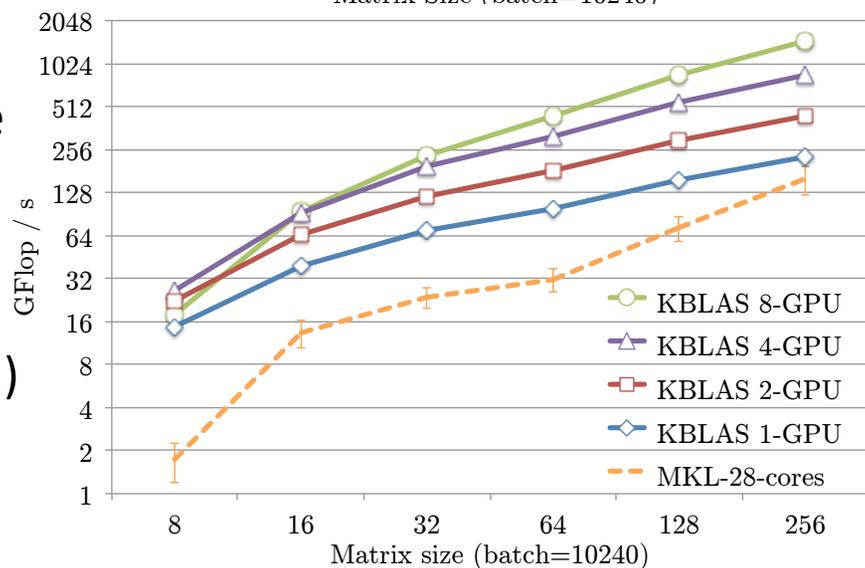


Batched KBLAS performance comparisons

Single K40 (MKL on 28-core Broadwell)

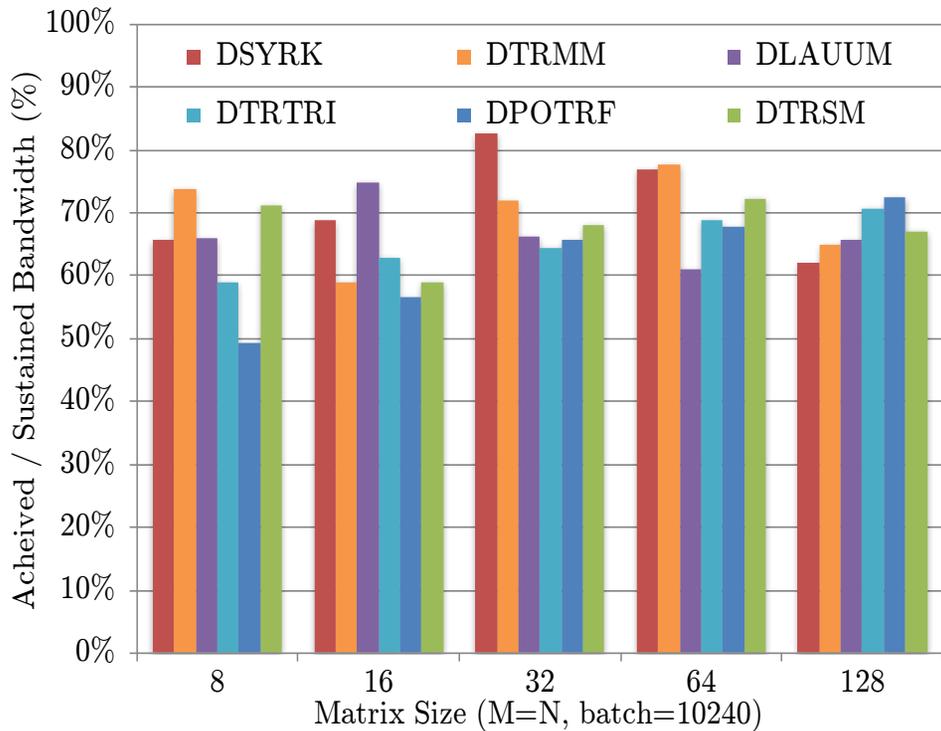


Multiple K40s (MKL on 28-core Broadwell)

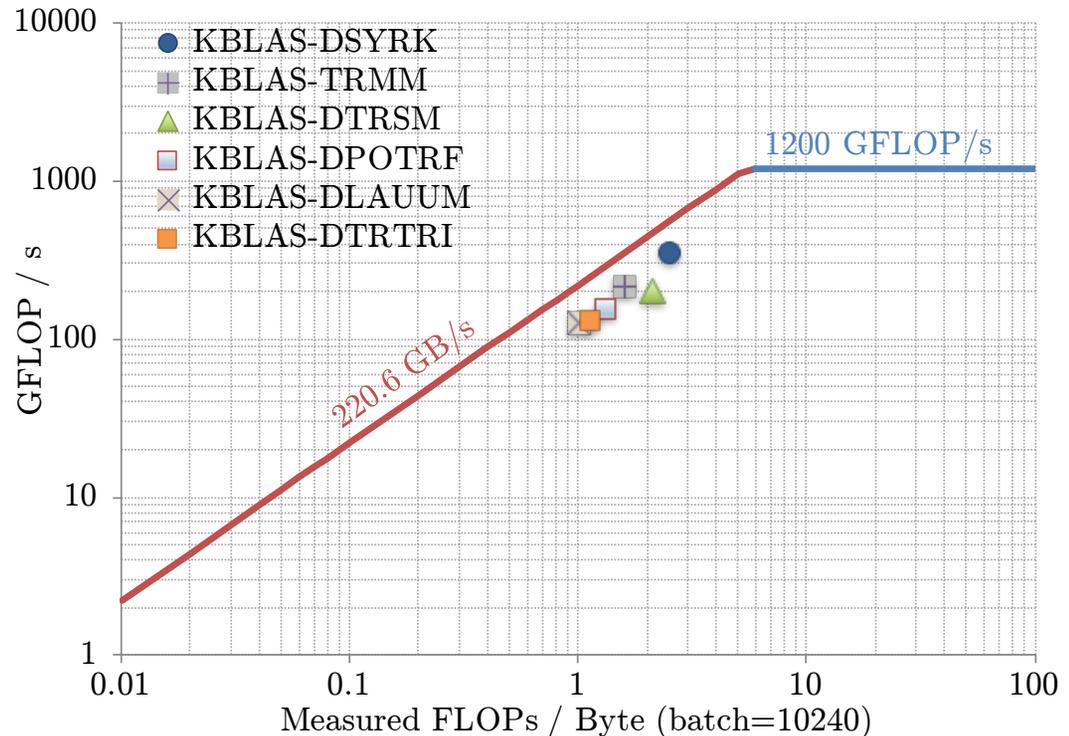




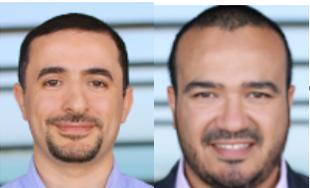
Batched KBLAS performance



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

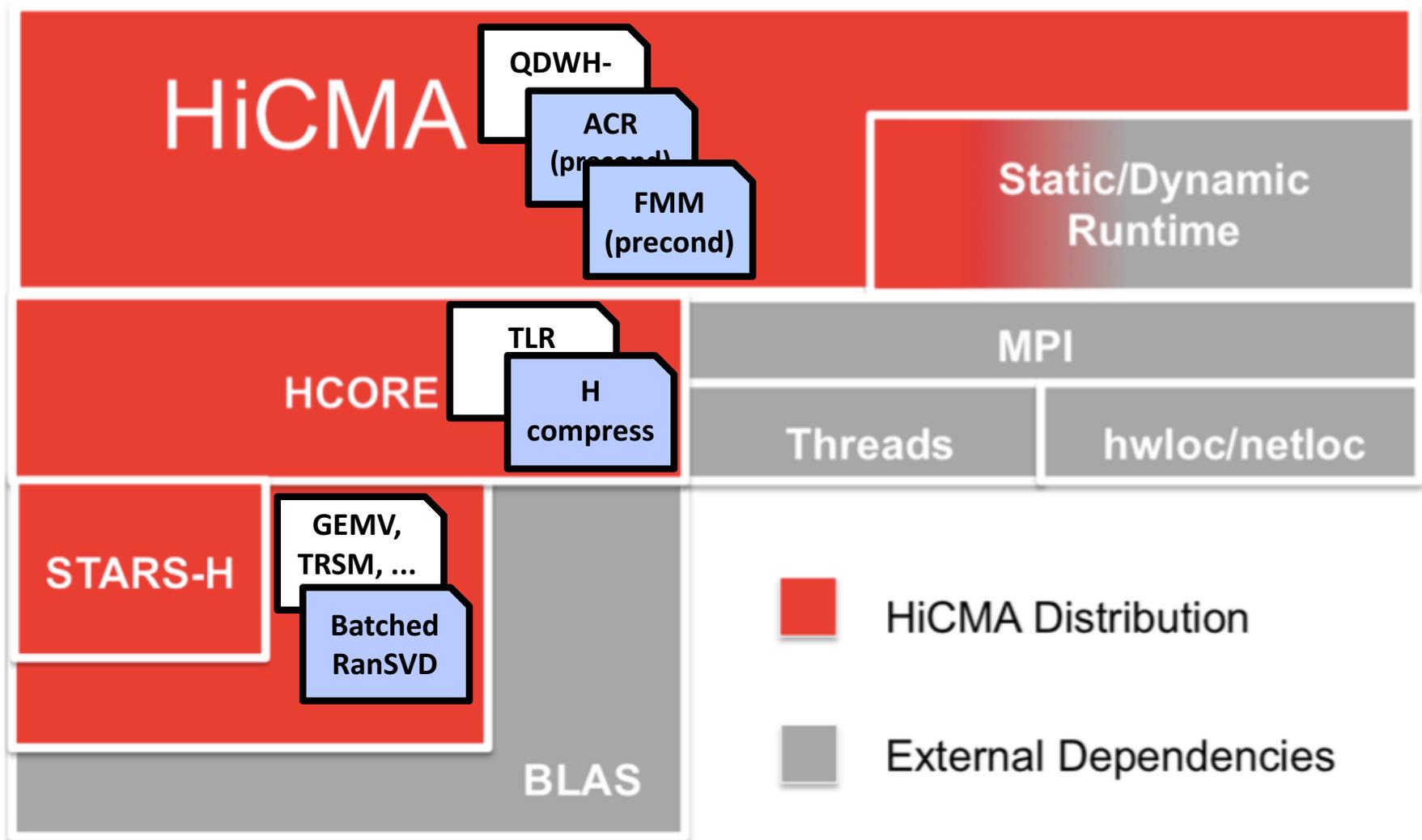


Roofline performance model of KBLAS batched operations in double precision and 10240 batched size running on NVIDIA K40 GPU, on square matrices of size 128.



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

Hierarchical Computations on Manycore Architectures: HiCMA*



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

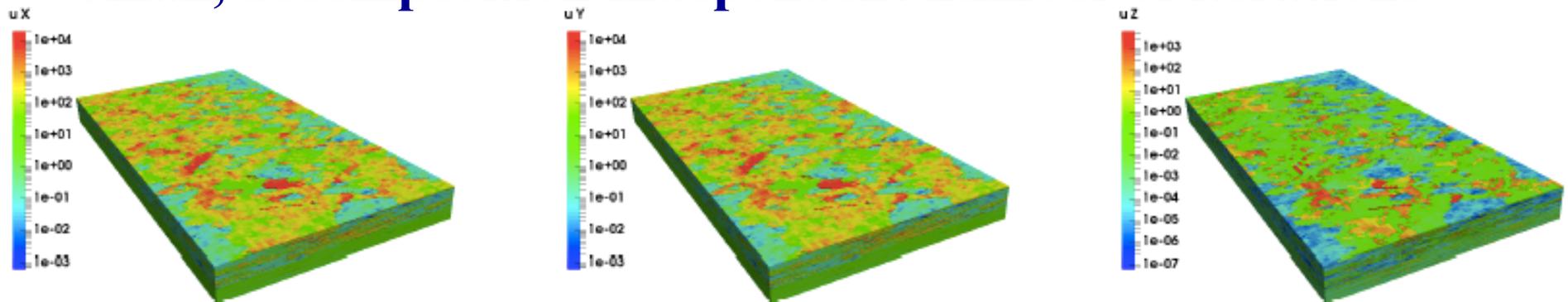
Balancing Domain Decomposition with Constraints (BDDC)

- ✧ **Reduce synchrony in Krylov solution to PDE problems by building an optimal preconditioner**
 - ✧ **convergence independent of mesh size, subdomain size, and alignment of subdomain with material interfaces**
- ✧ **For SPD problems, BDDC is built from Cholesky and symmetric eigensolvers**
 - ✧ **harness HiCMA**
 - ✧ **exploit well-known low-rank properties of Schur complements**

Math Comp (2017), SISC (2016, 2017)

BDDC: a very robust preconditioner

- Applied inside CG on the SPE10 benchmark
- Darcy flow, using H(div) finite elements
- 20M-45M DOFs, up to 8K subdomains
 - ◆ no alignment of subdomain faces with material jumps
- Small, decomposition-independent number of iterations



Condition number and number of iterations as a function of eigenvalue threshold λ and number of subdomains N .

$RT_0 \times P_0$ (20M dofs)

N	$\lambda = 10$	$\lambda = 5$	$\lambda = 2.5$	$\lambda = 1.5$
1024	15.9/25	7.77/17	3.57/11	1.77/6
2048	15.0/25	7.76/17	3.51/11	1.65/6
4096	15.4/25	8.19/18	3.41/11	1.63/6
8192	16.5/26	7.69/17	3.51/11	1.67/6

$BDM_1 \times P_0$ (45M dofs)

N	$\lambda = 10$	$\lambda = 5$	$\lambda = 2.5$	$\lambda = 1.5$
1024	16.1/24	7.51/16	3.49/10	1.60/6
2048	16.7/25	7.45/16	3.53/10	1.61/6
4096	15.5/24	7.53/16	3.57/10	1.58/6
8192	15.9/24	7.77/17	3.53/10	1.59/6



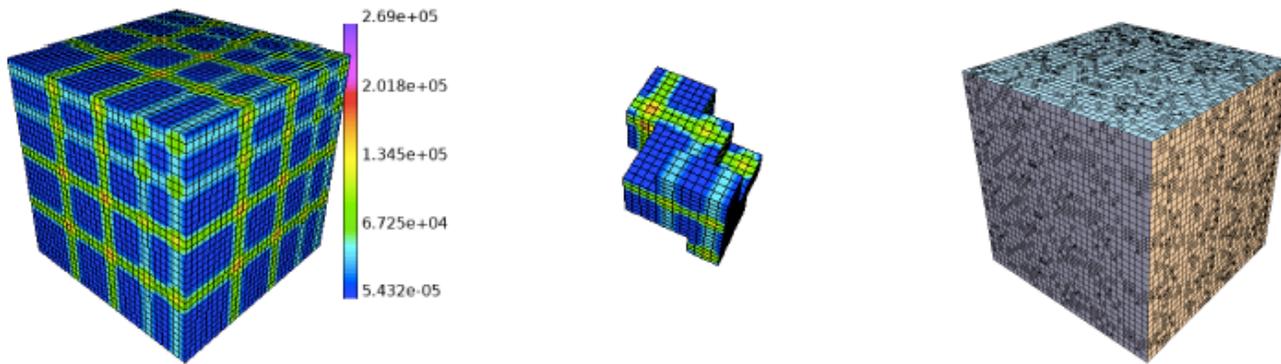
c/o S. Zampini (KAUST)

Zampini et al., Invited talk at DDM'24
to appear in Springer LNCSE

BDDC: a very robust preconditioner

- Maxwell equations, using H(curl) finite elements

κ , number of iterations, size of coarse problem (relative to Γ) for different eigenvalue thresholds. β as in figure. 40 subdomains.



Tetrahedral mesh

p=1 (200K dofs)					p=2 (1.2M dofs)				
	-	$\lambda=10$	$\lambda=5$	$\lambda=2.5$		-	$\lambda=10$	$\lambda=5$	$\lambda=2.5$
κ	150.2	7.5	4.6	2.2	κ	413.3	5.9	4.3	2.3
it	54	15	12	8	it	113	15	12	9
C/ Γ	0.01	0.05	0.06	0.09	C/ Γ	0.01	0.02	0.02	0.04

Hexahedral non-conforming mesh

p=1 (330K dofs)					p=2 (3.5M dofs)				
	-	$\lambda=10$	$\lambda=5$	$\lambda=2.5$		-	$\lambda=10$	$\lambda=5$	$\lambda=2.5$
κ	203.4	5.8	3.2	2.0	κ	330.8	5.1	3.4	2.0
it	62	13	10	7	it	97	14	11	8
C/ Γ	0.02	0.05	0.06	0.09	C/ Γ	0.01	0.01	0.02	0.04

c/o S. Zampini (KAUST) and P. Vassilevski (LLNL)



BDDC on the road to exascale

Adaptive BDDC satisfies 3 of the pillars for exascale algorithms [J. Dongarra, et al, Int. J. High Perf. Comp. Appl. 6, 2011]

- Reduces the synchronization steps and the number of MatVecs
- Increases arithmetic intensity of the preconditioning step
- Increases concurrency of the preconditioning step

Key features of the algorithm

- Tunable accuracy
- Cholesky based
- Local and coarse problem additively combined (overlap)
- Multilevel extensions with high F/C coarsening ratios $O(10^2)$ – $O(10^4)$

Note: BDDC is distributed in PETSc



c/o S. Zampini (KAUST)

Distributed data structures

Ω subdivided in N non-overlapping open subdomains

$$\bar{\Omega} = \bigcup_{i=1}^N \bar{\Omega}_i, \quad \Omega_j \cap \Omega_i = \emptyset, \quad \Gamma = \bigcup_{i \neq j} \partial\Omega_j \cap \partial\Omega_i.$$

Linear system's matrix A never assembled explicitly; mat-vec as

$$A = \begin{bmatrix} A_{//} & A_{/\Gamma} \\ A_{/\Gamma}^T & A_{\Gamma\Gamma} \end{bmatrix} = R^T A^* R, \quad A^* = \begin{bmatrix} A^{(1)} & & \\ & \ddots & \\ & & A^{(N)} \end{bmatrix},$$

with

$$A^{(i)} = \begin{bmatrix} A_{//}^{(i)} & A_{/\Gamma}^{(i)} \\ A_{/\Gamma}^{(i)T} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix}$$

the matrix of the FEM problem on Ω_i .



Condition number results

- If subdomains are solved exactly, overall condition number of the preconditioned system depends only on the Schur preconditioning
- Block factorization for A (I interior, Γ interface)

$$A^{-1} = \begin{bmatrix} I_{II} & -A_{II}^{-1}A_{I\Gamma} \\ & I_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} A_{II}^{-1} & \\ & S_{\Gamma}^{-1} \end{bmatrix} \begin{bmatrix} I_{II} & \\ -A_{I\Gamma}^T A_{II}^{-1} & I_{\Gamma\Gamma} \end{bmatrix},$$

with $S_{\Gamma} = A_{\Gamma\Gamma} - A_{I\Gamma}^T A_{II}^{-1} A_{I\Gamma}$.

- Block preconditioner

$$M^{-1} = \begin{bmatrix} I_{II} & -A_{II}^{-1}A_{I\Gamma} \\ & I_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} A_{II}^{-1} & \\ & M_{\Gamma}^{-1} \end{bmatrix} \begin{bmatrix} I_{II} & \\ -A_{I\Gamma}^T A_{II}^{-1} & I_{\Gamma\Gamma} \end{bmatrix},$$

$$\kappa(M^{-1}A) = \kappa(M_{\Gamma}^{-1}S_{\Gamma})$$



Global Schur complement is subassembled

$$M_{\Gamma}^{-1} = \tilde{R}_{D,\Gamma}^T \tilde{S}_{\Gamma}^{-1} \tilde{R}_{D,\Gamma},$$

Block Cholesky

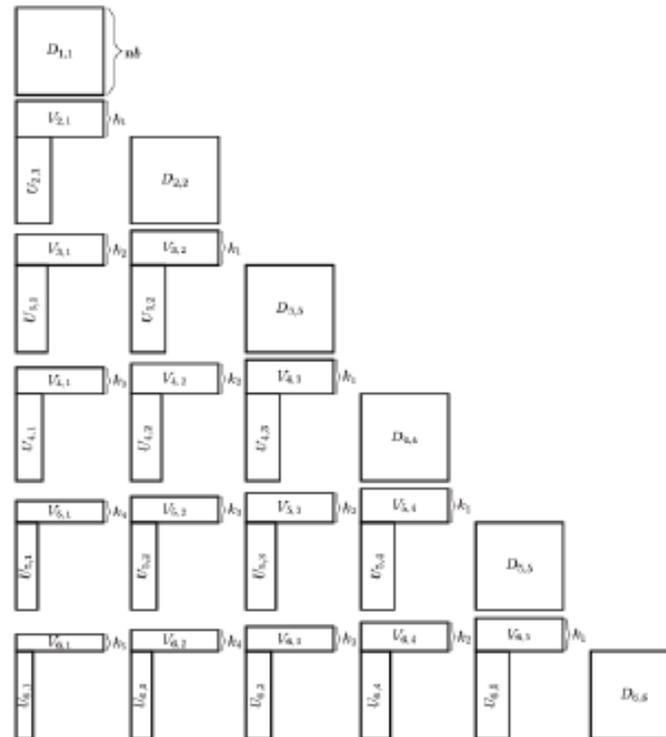
$$\tilde{S}_{\Gamma}^{-1} = R_{\Gamma\Delta}^T \left(\sum_{i=1}^N \begin{bmatrix} 0 & R_{\Delta}^{(i)T} \end{bmatrix} \begin{bmatrix} A_{II}^{(i)} & A_{I\Delta}^{(i)} \\ A_{I\Delta}^{(i)T} & A_{\Delta\Delta}^{(i)} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ R_{\Delta}^{(i)} \end{bmatrix} \right) R_{\Gamma\Delta} + \Phi S_{\Pi\Pi}^{-1} \Phi^T$$

- Cholesky is everywhere, in high concurrency for batching during both formation and application of the preconditioner
- Also, generalized symmetric eigenproblem on each interface where the “A, B” matrices are from Schur complements



BDDC with low rank Schur approximations

We use the block low rank (BLR) format as introduced by [Amestoy et al, SISC, 2015] (others are possible).



See Gatto & Hasthaven, Dec 2016, J Sci Comput on compressibility of Schur complements for hp finite elements

At full accuracy

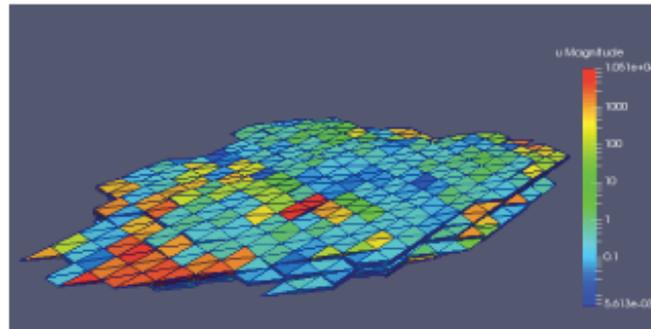
- memory complexity from $O(n^{4/3})$ to $O(n^{[0.93, 1.13]})$ [Poisson, Helmholtz].
- flops from $O(n^2)$ to $O(n^{[1.4, 1.7]})$.



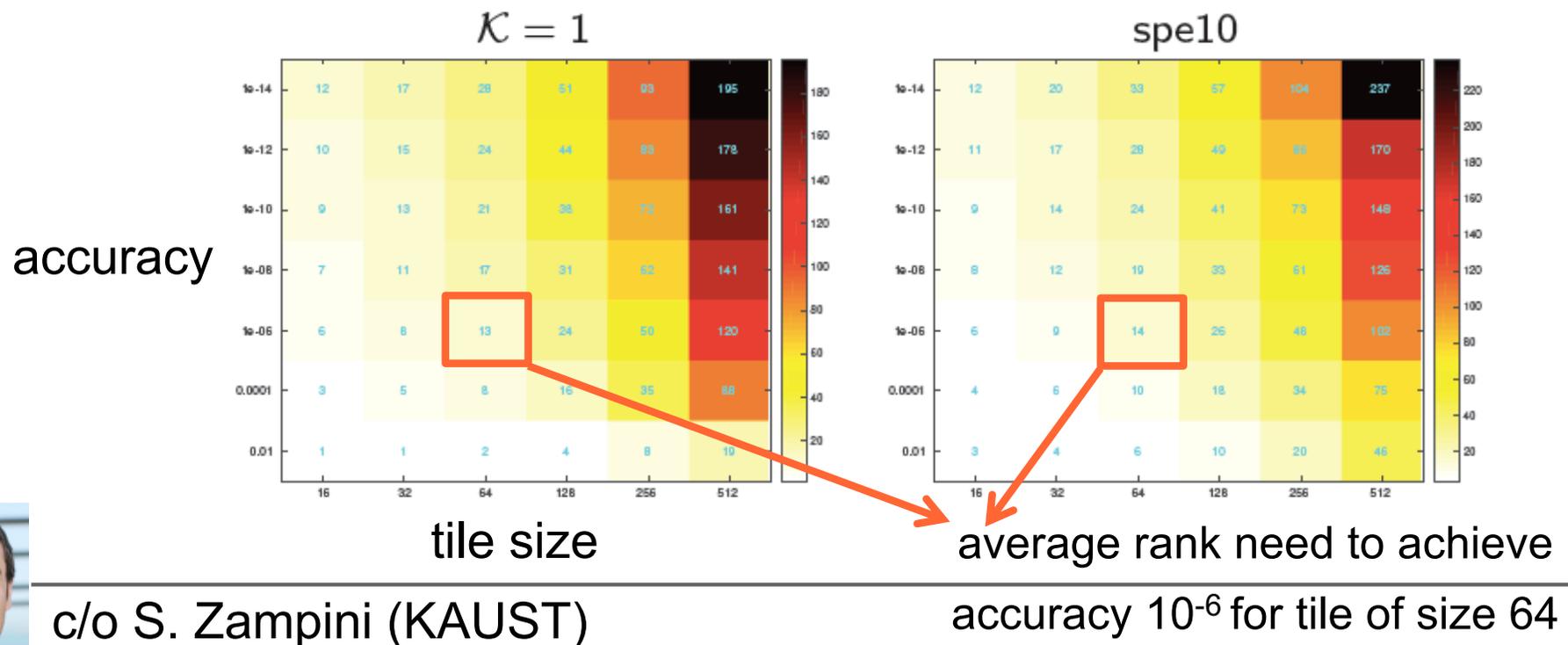
c/o S. Zampini (KAUST)

BDDC with low rank Schur approximations

Darcy problem, SPE10 benchmark. One representative subdomain



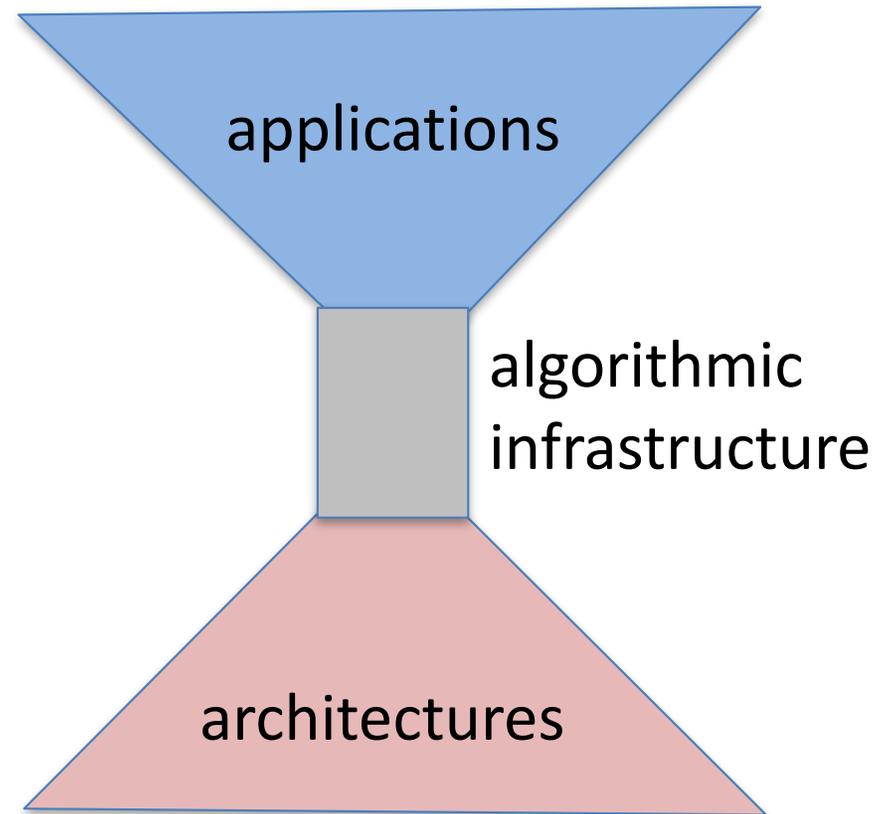
Heatmap of block ranks for a given subdomain for different accuracies.



c/o S. Zampini (KAUST)



“Hourglass” model for algorithms (traditionally applied to internet protocols)



How will complex PDE codes adapt?

- **Programming model will still be dominantly message-passing (due to large legacy code base), adapted to multicore or hybrid processors beneath a relaxed synchronization MPI-like interface**
 - **Load-balanced blocks, scheduled today with nested loop structures will be separated into critical and non-critical parts**
 - **Critical parts will be scheduled with directed acyclic graphs (DAGs) through dynamic languages or runtimes**
 - **Noncritical parts will be made available for NUMA-aware work-stealing in economically sized chunks**
-

Asynchronous programming styles

- **To take full advantage of such asynchronous algorithms, we need to develop greater expressiveness in scientific programming**
 - ◆ **create separate threads for logically separate tasks, whose priority is a function of algorithmic state, not unlike the way a time-sharing OS works**
 - ◆ **join priority threads in a directed acyclic graph (DAG), a task graph showing the flow of input dependencies; fill idleness with noncritical work or steal work**
-

Evolution of Newton-Krylov-Schwarz: breaking the synchrony stronghold

- Can write code in styles that do not require artificial synchronization
 - Critical path of a nonlinear implicit PDE solve is essentially
... `lin_solve`, `bound_step`, `update`; ...
 - However, we often insert into this path things that could be done less synchronously, because we have limited language expressiveness
 - ◆ Jacobian and preconditioner refresh
 - ◆ convergence testing
 - ◆ algorithmic parameter adaptation
 - ◆ I/O, compression
 - ◆ visualization, data analytics
-

Sources of nonuniformity

- **System**

- ◆ *Already* important: manufacturing, OS jitter, TLB/cache performance variations, network contention,
- ◆ *Newly* important: dynamic power management, more soft errors, more hard component failures, software-mediated resiliency, etc.

- **Algorithmic**

- ◆ physics at gridcell/particle scale (e.g., table lookup, equation of state, external forcing), discretization adaptivity, solver adaptivity, precision adaptivity, etc.

- **Effects of both types are similar when it comes to waiting at synchronization points**

- **Possible solutions for system nonuniformity will improve programmability for nonuniform problems, too 😊**

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**
 - ◆ **built-in resilience (“algorithm-based fault tolerance” or ABFT) to arithmetic/memory faults or lost/delayed messages**
 - **Programming models and runtimes may have to be stretched to accommodate**
 - **Everything should be on the table for trades, beyond disciplinary thresholds → “co-design”**
-

Thanks to:



Thank you!



شكرا

david.keyes@kaust.edu.sa