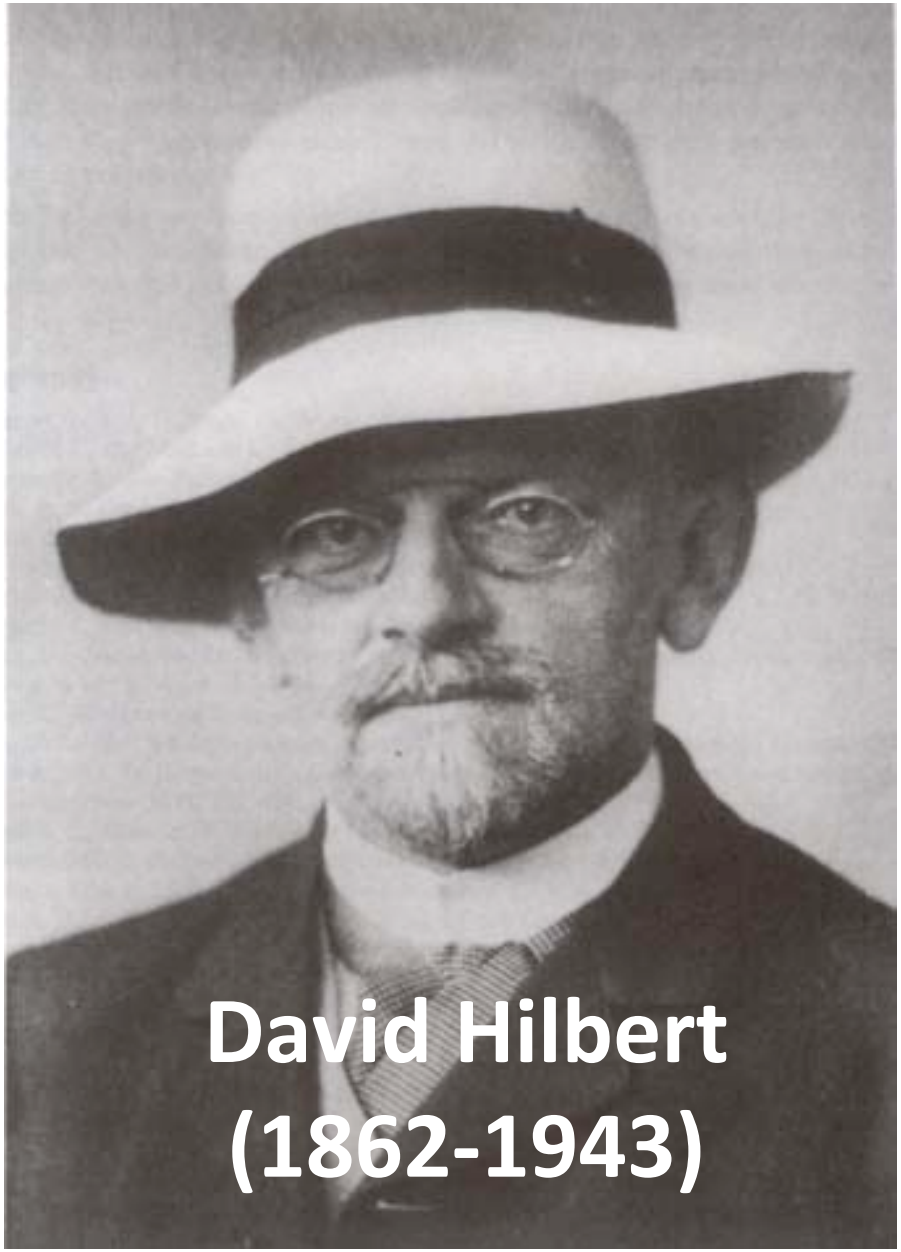
Algorithmic Adaptations to Extreme Scale

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

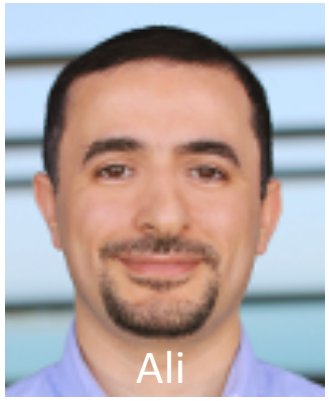


[at the International Mathematical Congress in Bologna, 1928, after the 1920 and 1924 Congresses had excluded mathematicians representing the countries defeated in WW1:]

"Mathematics knows no races or geographic boundaries; for mathematics, the whole cultural world is one country."

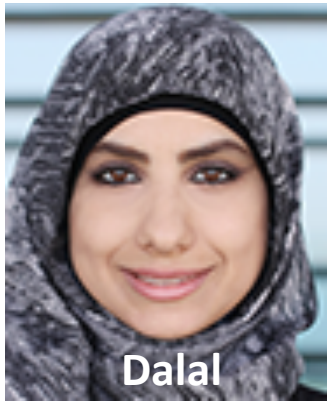
“It makes me very happy that after a long, hard time all the mathematicians of the world are represented here. That is as it should be and as it must be for the prosperity of our beloved science. […] **Mathematics knows no races; for mathematics the whole cultural world is a single country.**”

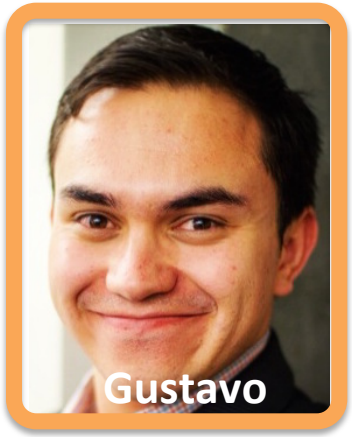

Ahmad

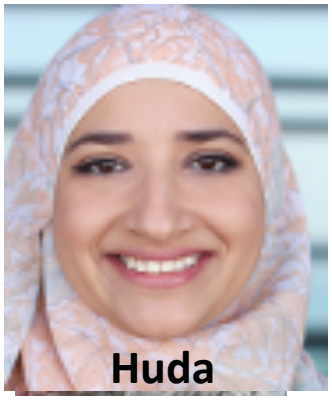

Ali


Amani


Chengbin


Dalal


Gustavo


Huda


Lulu


Mohammed


Mustafa


Noha


Rabab


Tareq


Wajih

Activities in the USA

Internships at:

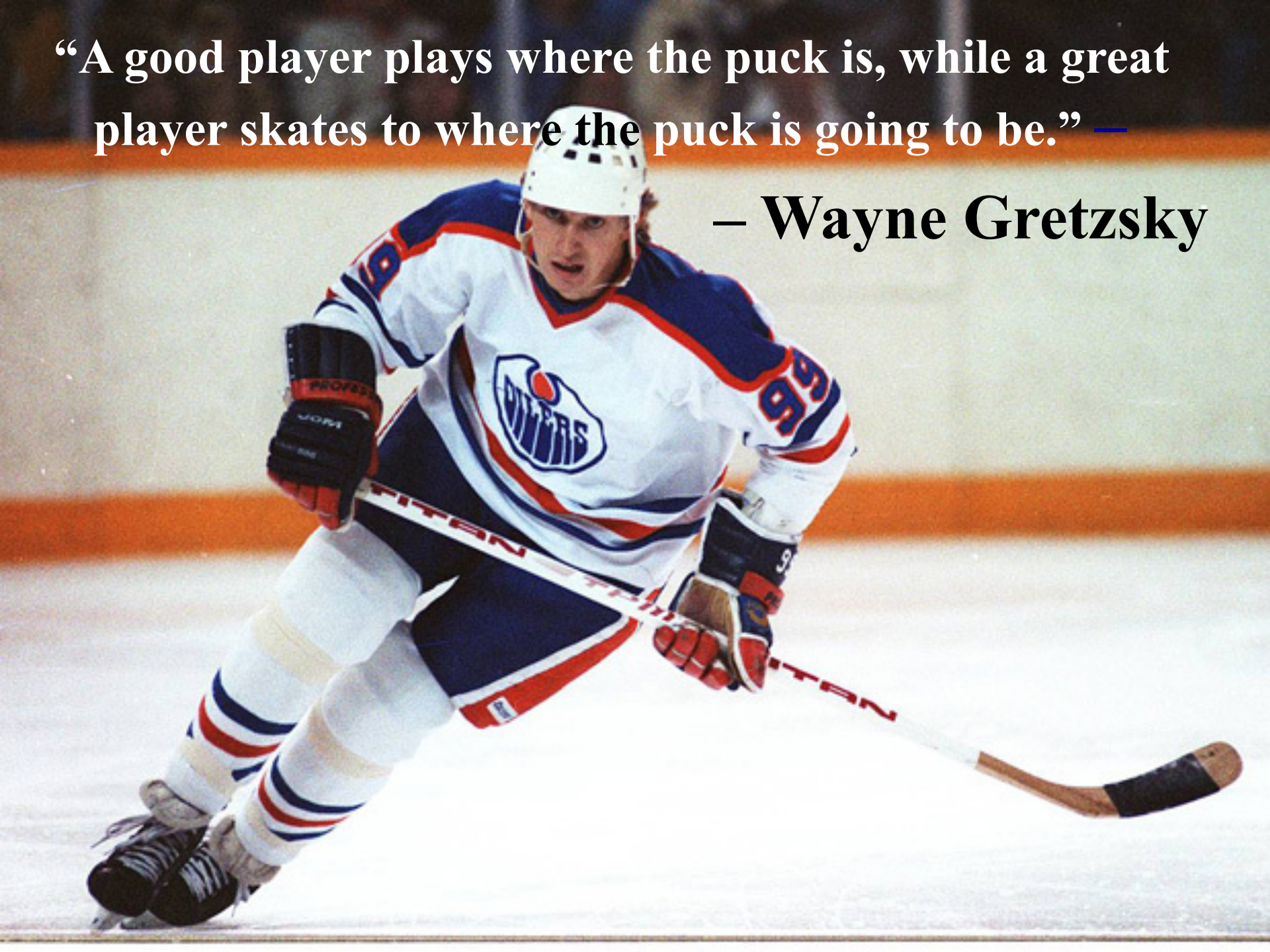
- Argonne
- IBM
- NVIDIA
- Sandia
- Texas A&M

Employment at:

- Intel
- ICL (U Tennessee)
- NERSC (Berkeley)
- XPACC (UIUC)

Presentations at:

SIAM, Supercomputing, GTC, ICS


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

Aspiration for this talk

To paraphrase Gretzsky:

"Algorithms for where architectures are going to be"

Such algorithms may *or may not* be the best today; however, hardware trends can be extrapolated to the new potential "sweet spots."

Outline

- **Four architectural trends**
  - **limits to the extension our current bulk synchronous software infrastructure**
- **Four algorithmic imperatives**
  - **for extreme scale, tomorrow *and today***
- **Four sets of "bad news, good news"**
- **Four widely applicable strategies**
- **Four "points of light" (work in progress)**
  - **others described in detail in minisymposia**

Four architectural trends

- Clock rates cease to increase while arithmetic capability continues to increase exponentially through concurrency
- Memory storage capacity diverges exponentially below arithmetic capability
- Transmission capability (memory BW and network BW) diverges exponentially below arithmetic capability
- Mean time between hardware interrupts shortens

➔ **Billions of**

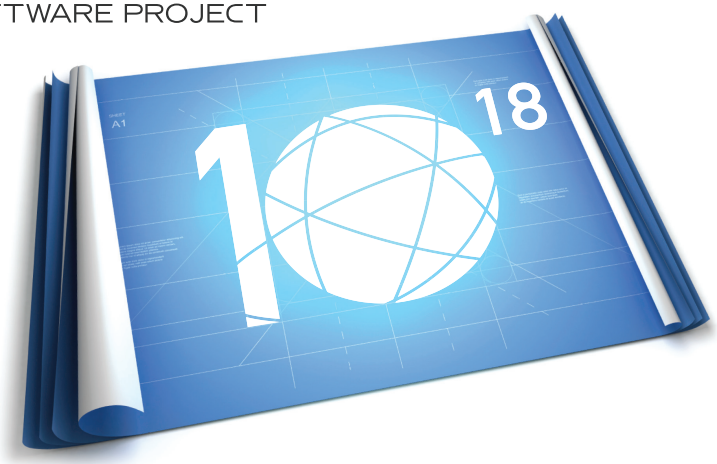
$ € £ ¥

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 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, CEPTILO, CRAY, EDF, EPSRC, 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
- 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
  - makes per-node performance rate unreliable
  - overprovisioned, specialized inhomogeneous nodes, sometimes dark
- 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

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/KW-hr × 8760 hr/yr)**
  - **We "use" 1.4 KW continuously, so 100MW is 71,000 people**

c/o J. Shalf (LBNL)

Why exa- is different

Moore's Law (1965) does not end but Dennard's MOSFET scaling (1972) does

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 enlist

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

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


Blue Waters

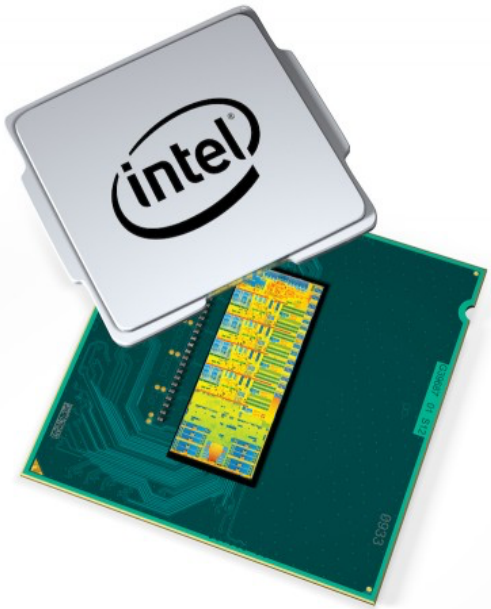
Shaheen

Sequoia

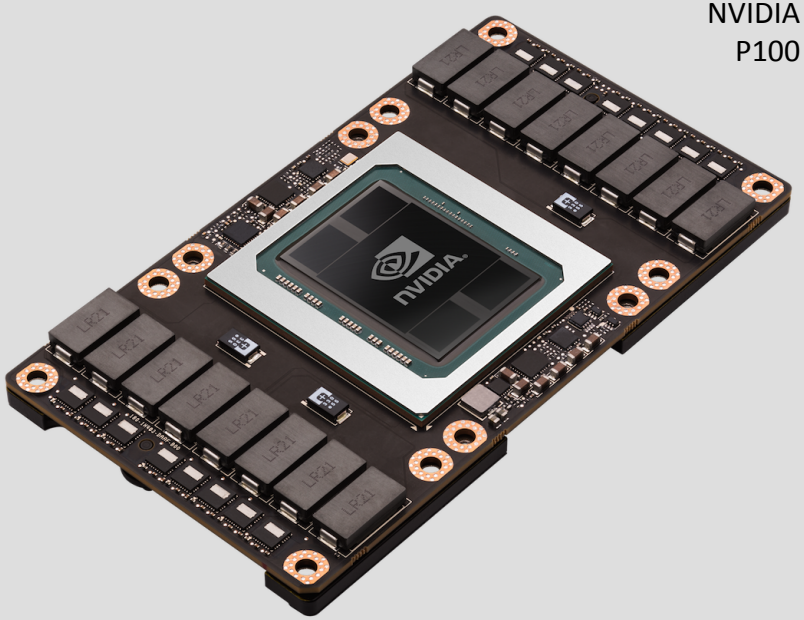
K

The challenge

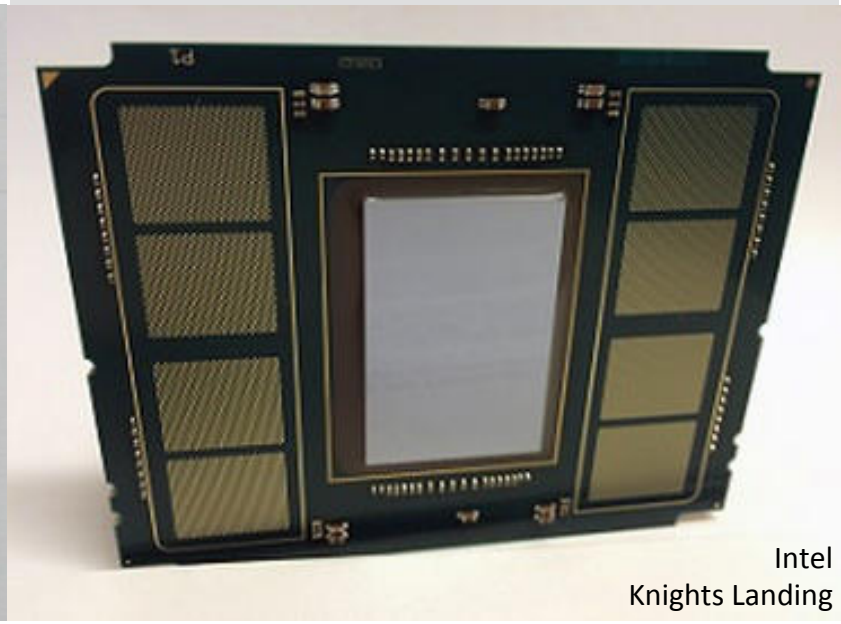
Intel
Broadwell



NVIDIA
P100



IBM
Power8



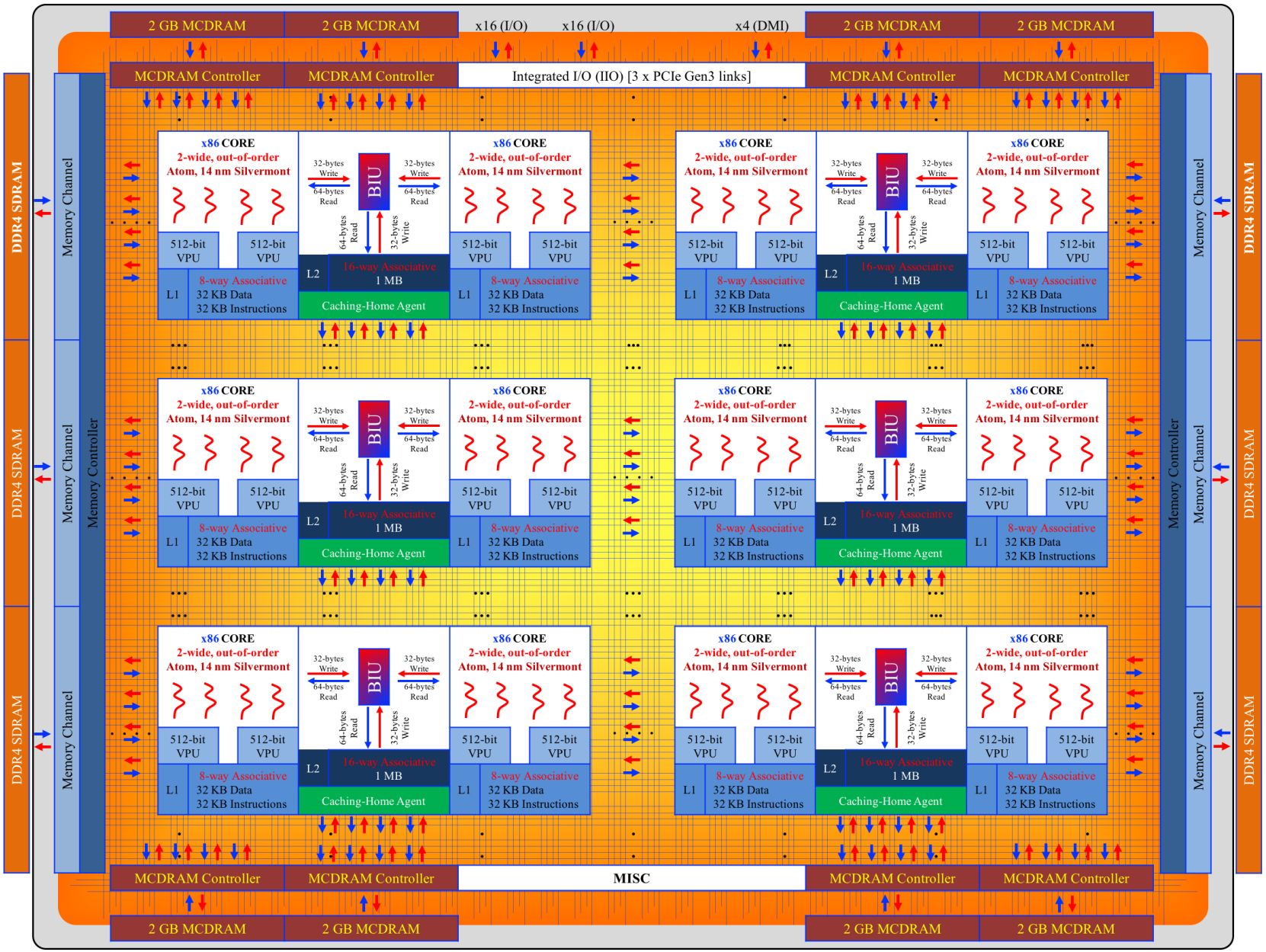
Intel
Knights Landing

Two decades of evolution

1997 ⇨ 2016

ASCI Red at Sandia by Intel
1.3 TF/s, 850 KW

Intel Xeon Phi MIC KNL
3.5 TF/s, 0.26 KW



c/o M. Al Farhan (KAUST)

Supercomputer in a node

System	Peak DP TFlop/s	Peak Power KW	Power Efficiency GFlop/s/Watt
ASCI Red (1997-2006)	1.3	850	0.0015

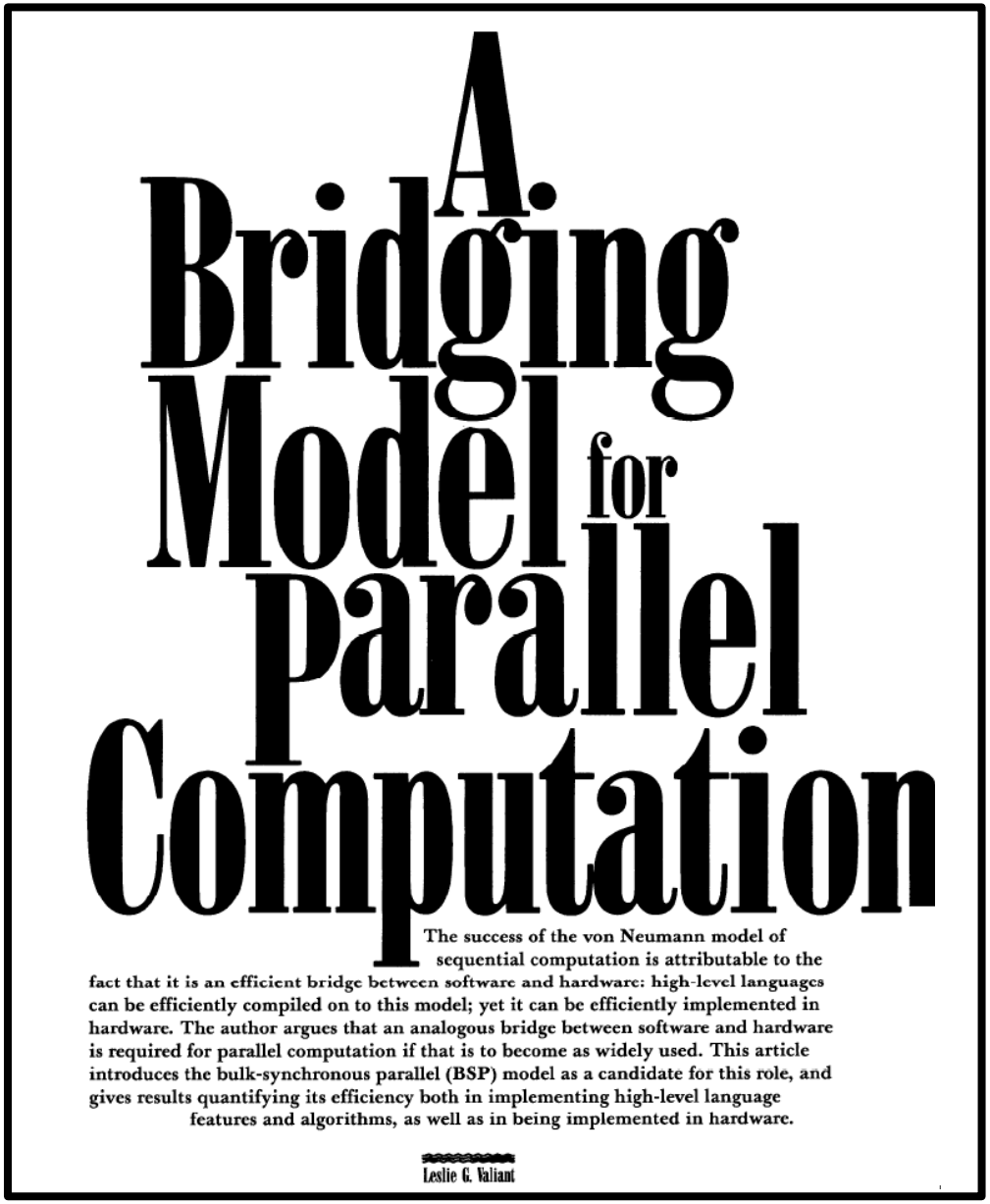
How are most scientific simulations implemented at the petascale today?

- **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, Harvard
2010 Turing Award Winner



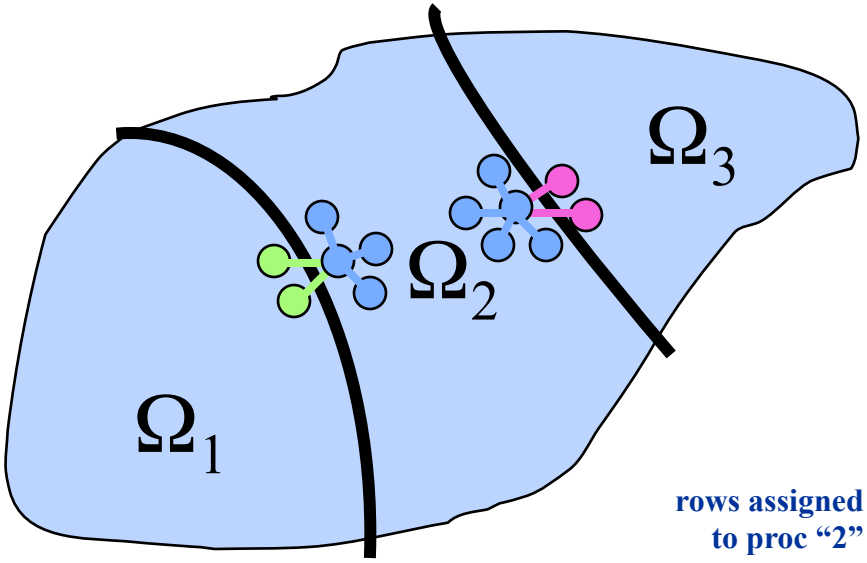
A Bridging Model for Parallel Computation

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.

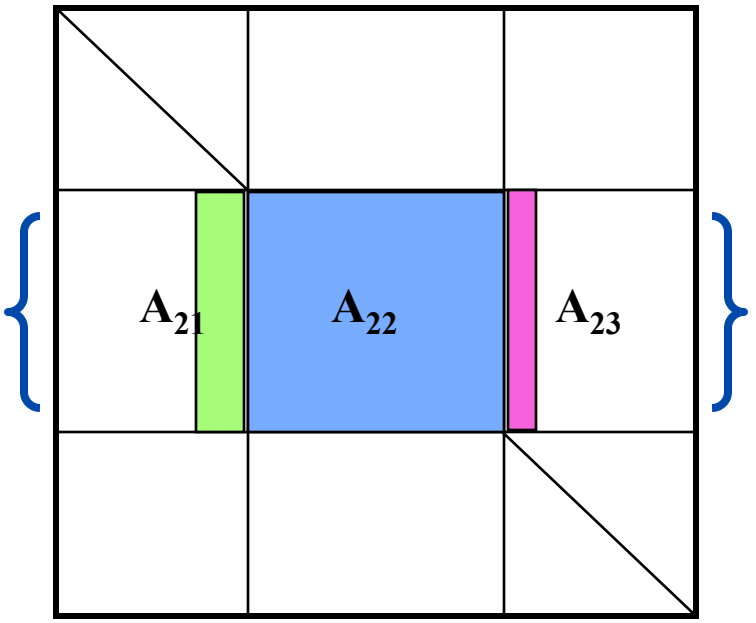
Leslie G. Valiant

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**

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**


Energy-aware generation
BSP generation

Four algorithmic imperatives

- **Reduce synchrony (in frequency and/or span)**
- **Increase arithmetic intensity**
- **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 (1)

- **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 (2)

- **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 (3)

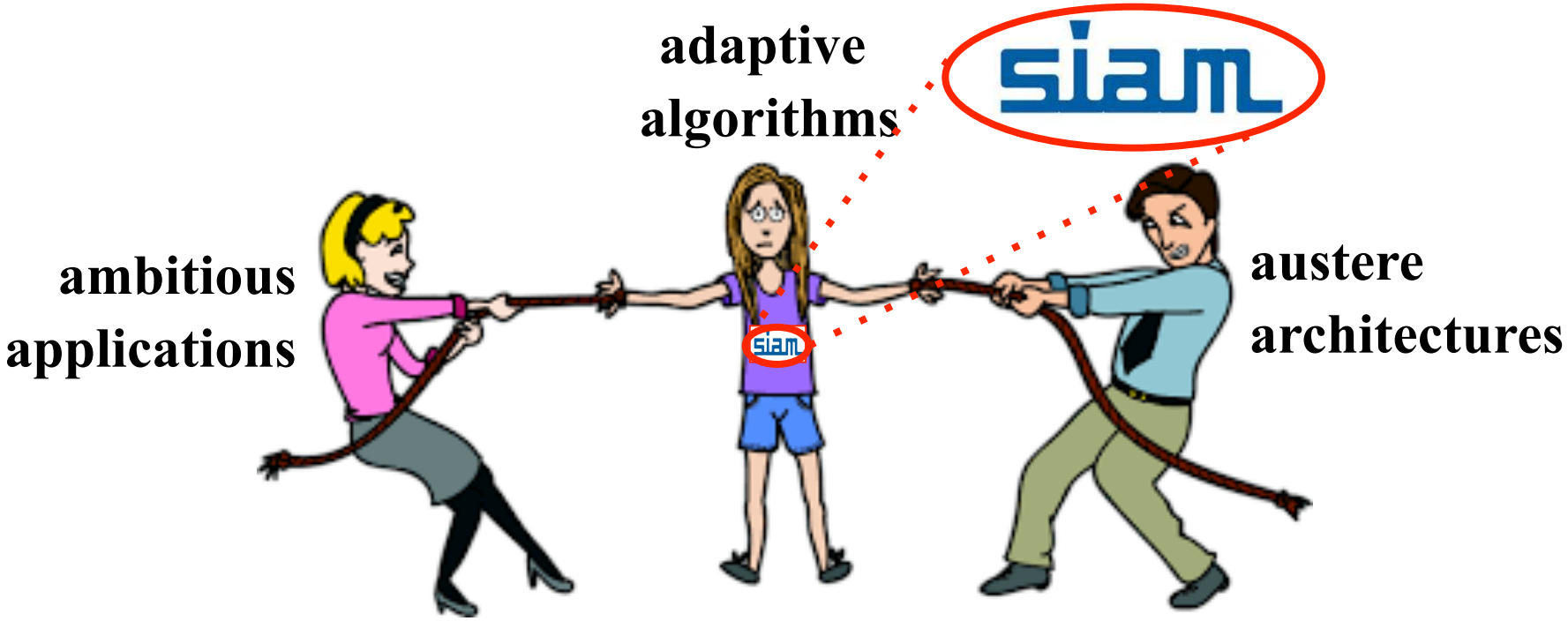
- **Fully deterministic algorithms may be regarded as too synchronization-vulnerable**
  - **rather than wait for missing data, we may predict it using various means and continue**
  - **we do this with increasing success in problems without models ("big data")**
  - **should be fruitful in problems 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 (4)

- 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 the widening gulf**



A full employment program for computational scientists and engineers ☺

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 (see DOE's new ECP)

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 stuff. Most has to be 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**
  - **greater arithmetic intensity (flops per byte moved into and out of registers and upper cache)**
    - **including assured accuracy with (adaptively) less floating-point precision**
  - **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., Charm++, Quark, StarPU, OmpSs, HPX, ADLB, Argo**
- **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**

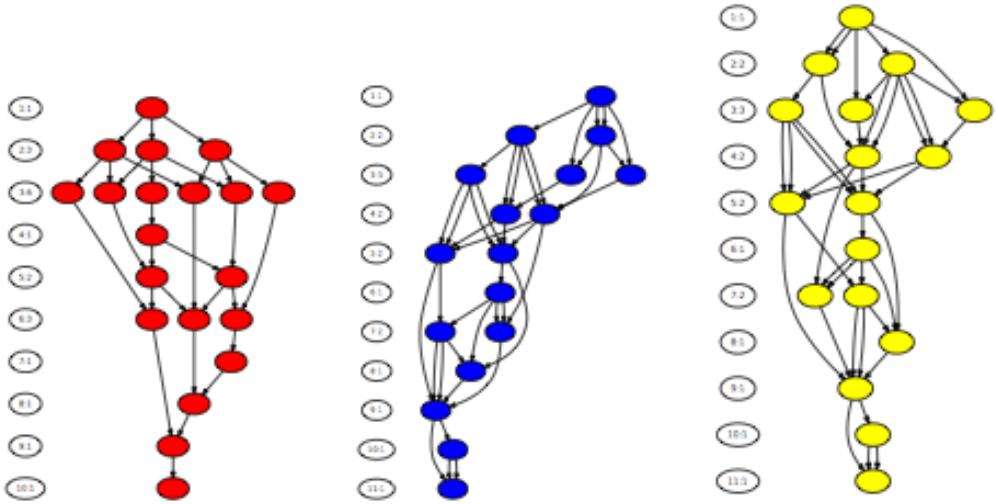
Taskification based on DAGs

- **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

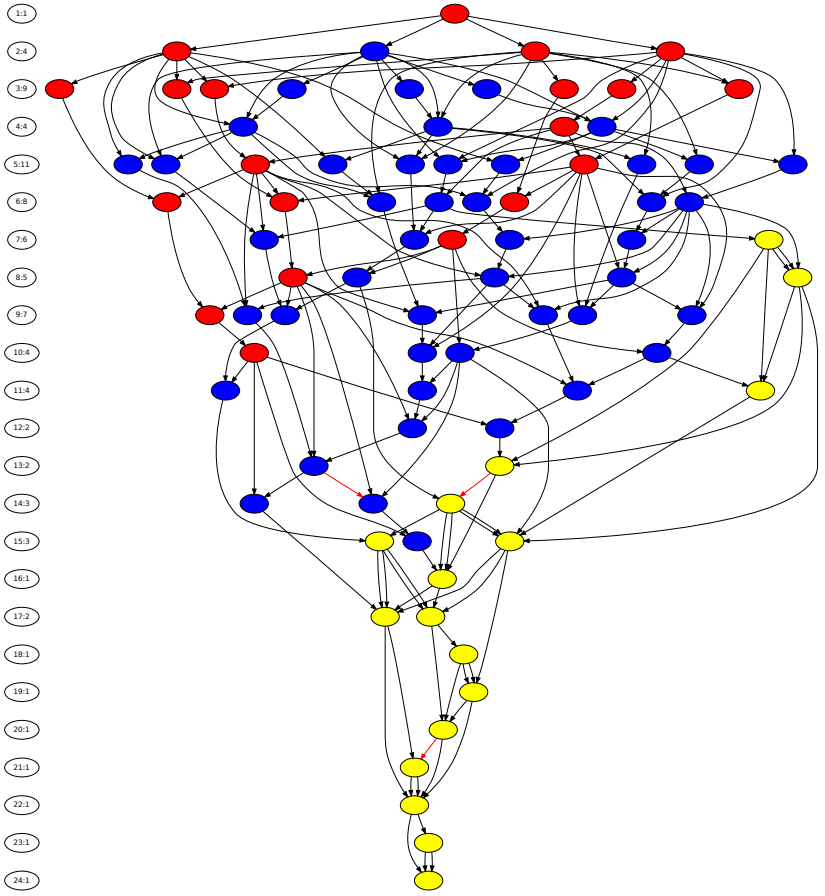
$$Ax = \lambda Bx$$

Operation	Explanation	LAPACK routine name
1. $B = L \times L^T$	Cholesky factorization	POTRF
2. $C = L^{-1} \times A \times L^{-T}$ 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



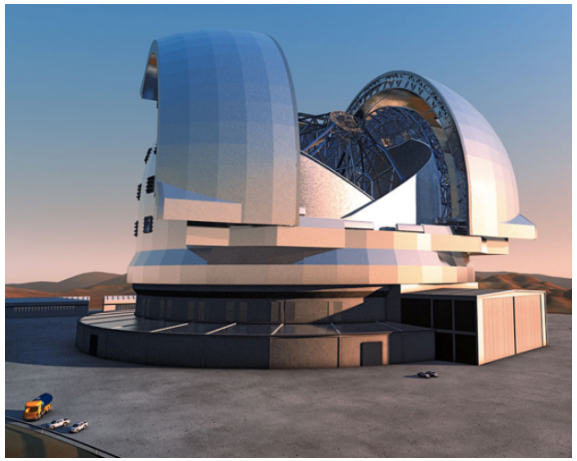
Loop nests and subroutine calls, with their over-orderings, can be replaced with DAGs

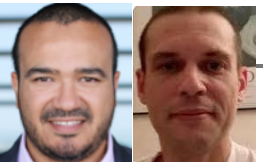
- Diagram shows a dataflow ordering of the steps of a 4×4 symmetric generalized eigensolver
- Nodes are tasks, color-coded by type, and edges are data dependencies
- Time is vertically downward
- Wide is good; short is good



Loops can be overlapped in time

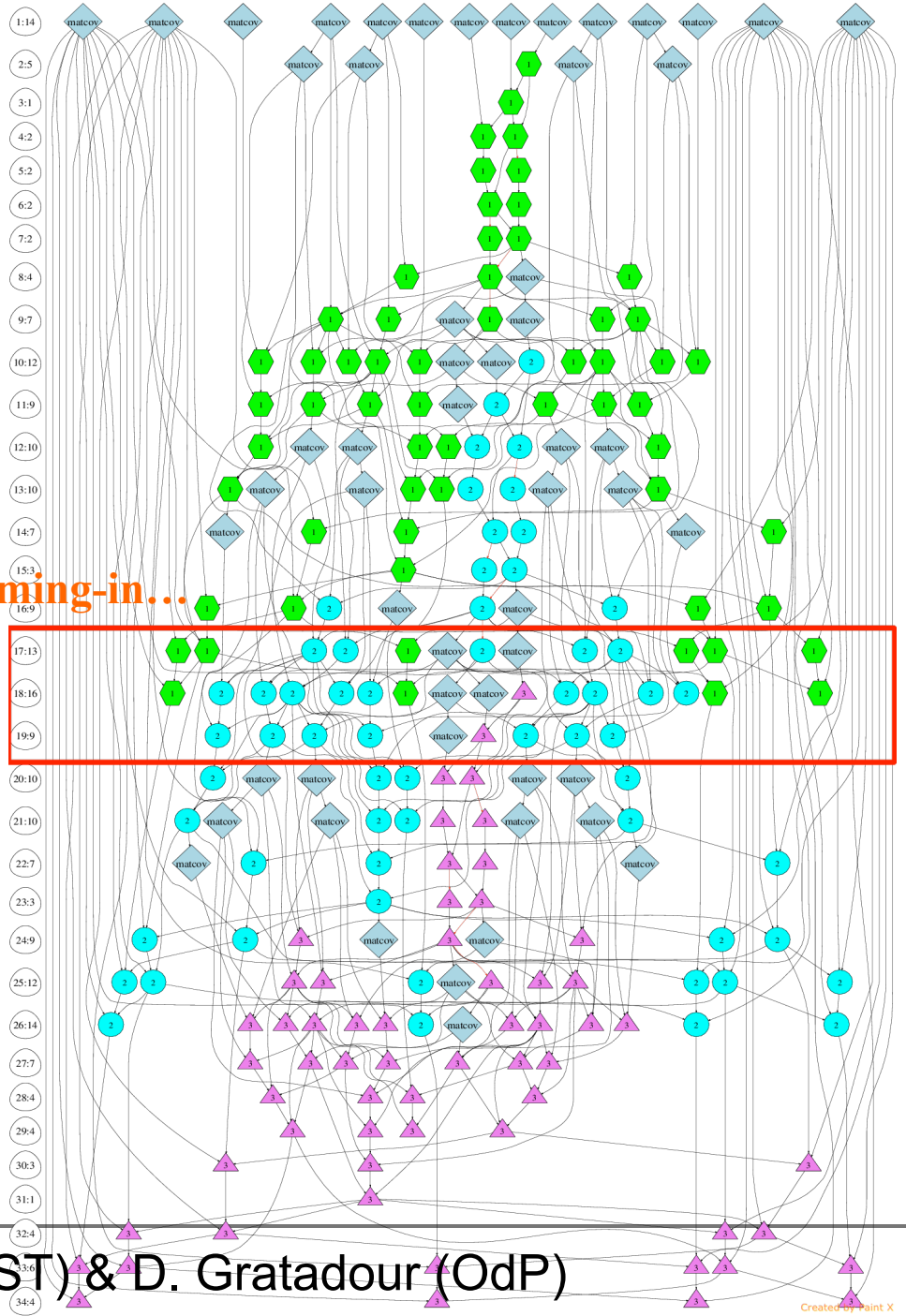
Green, blue and magenta symbols represent tasks in separate loop bodies with dependences from an adaptive optics computation





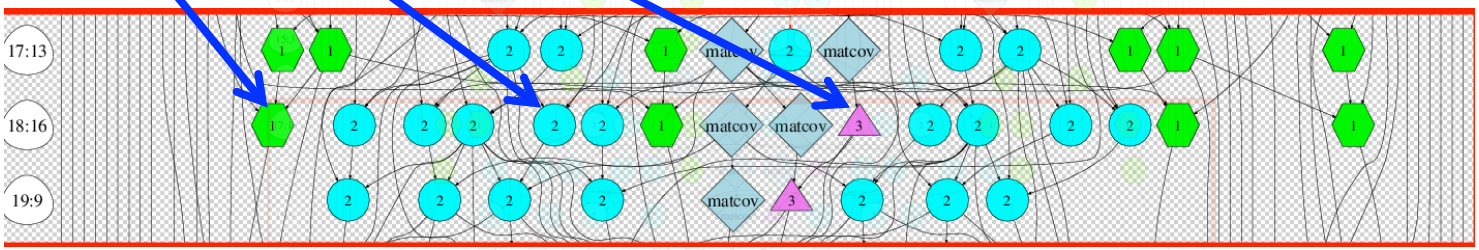
c/o H. Ltaief (KAUST) & D. Gratadour (OdP)

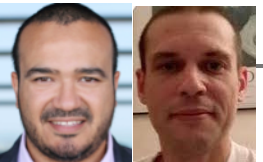
Zooming-in…



DAG-based safe out-of-order execution

Tasks from the 3 loops are scheduled 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

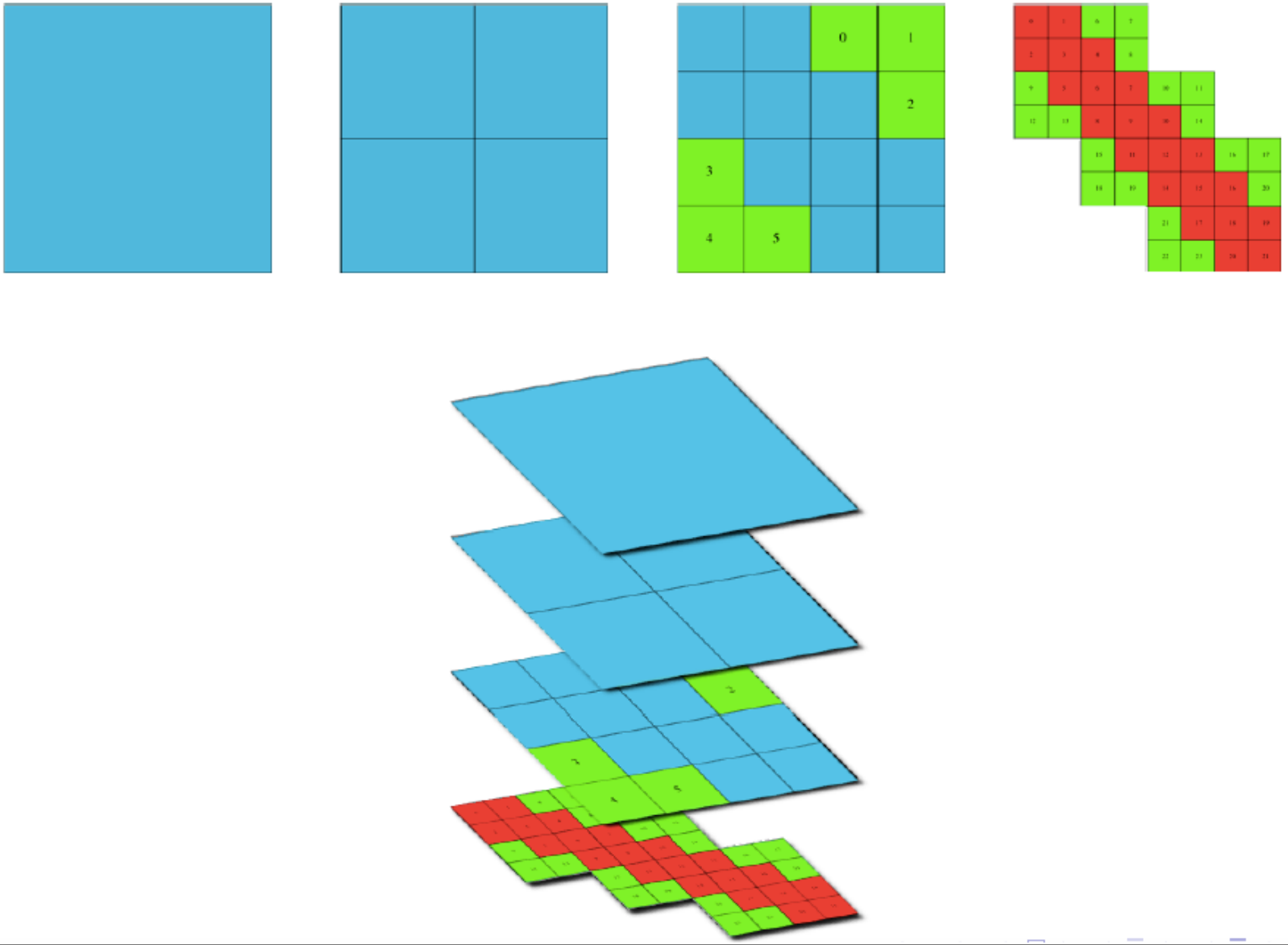
- [Hackbusch, 1999] **: off-diagonal blocks of typical differential and integral operators have low effective rank**
- **By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n:**
  - **polynomial in k**
  - **lin-log in n**
  - **constants carry the 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

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

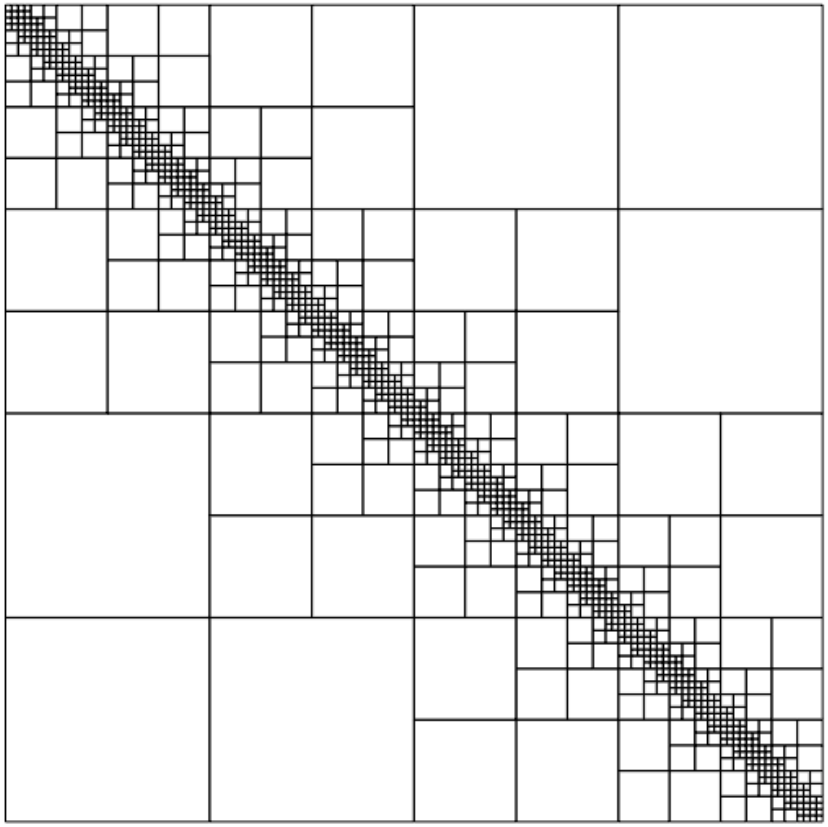
$$A^{-1} = \frac{1}{8} \times \left[\begin{array}{ccc|cccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 12 & 10 & 8 & 6 & 4 & 2 \\ 5 & 10 & 15 & 12 & 9 & 6 & 3 \\ \hline 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 15 & 10 & 5 \\ 2 & 4 & 6 & 8 & 10 & 12 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}\right] \leftrightarrow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

Recursive construction of an *H*-matrix

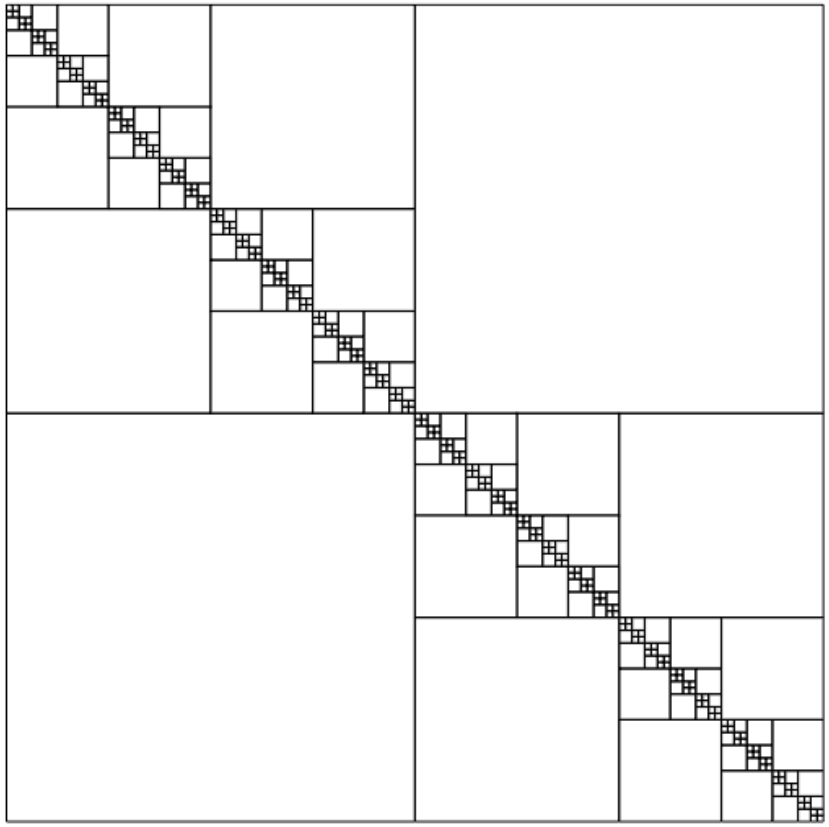


c/o W. Boukaram (KAUST)

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



strong admissibility



weak admissibility

After Hackbusch, et al., 2003

Employ high-order discretizations

- **Advantages**
  - **(also) 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 (?)**

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**

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*-SVD

- **DAG-based data flow tile algorithms for (eigen- and) singular value decomposition**
- **Reduce synchrony**
- **Increase SIMT-style concurrency**
- **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,* Nakatsukasa and Higham, SISC (2013)

QDWH-SVD

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

polar

$$A = U_p H$$

sym eigen

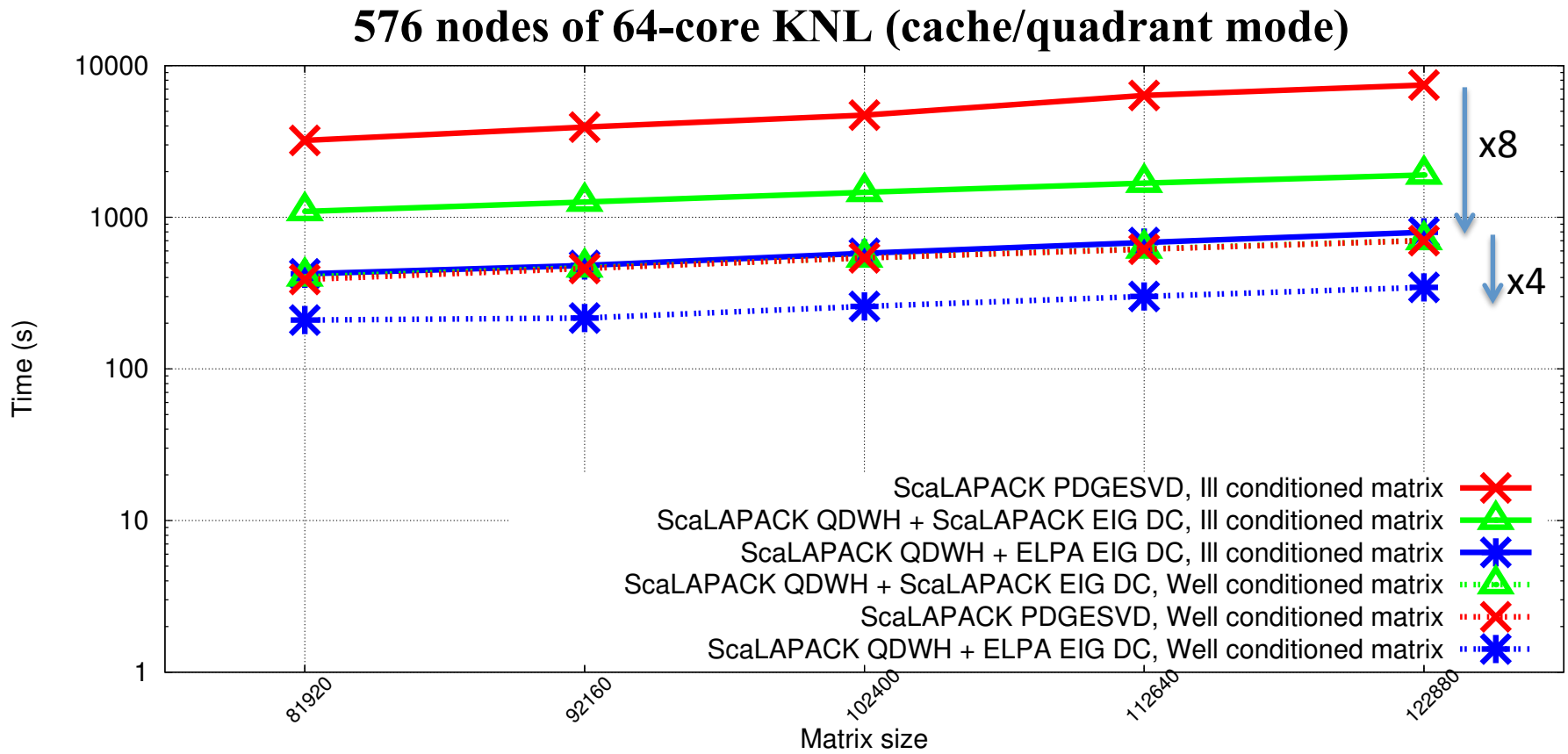
$$H = V \Sigma V^*$$

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

- **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



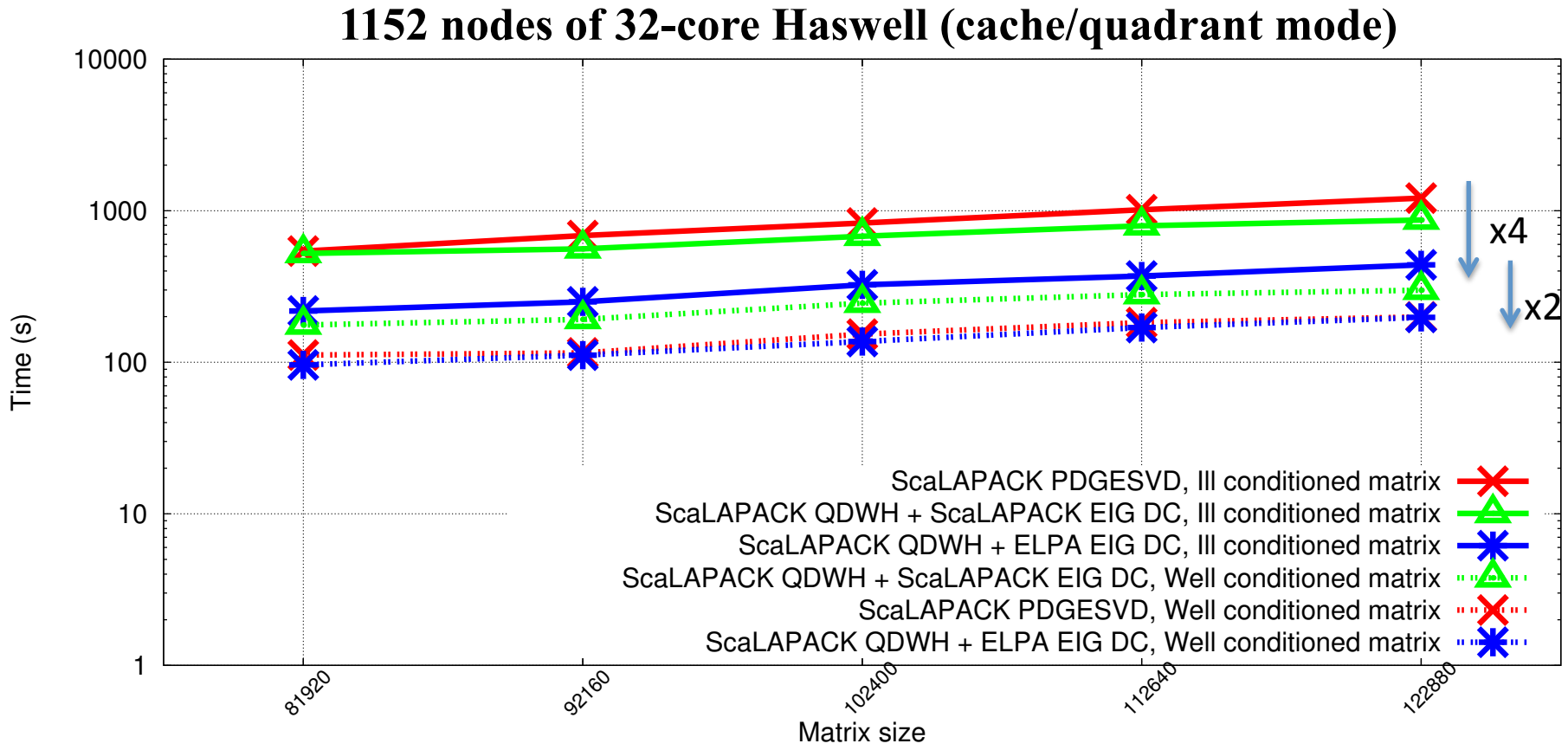
fastest dense SVD

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

Sukkari et al., Best papers, Europar'16

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

QDWH-SVD



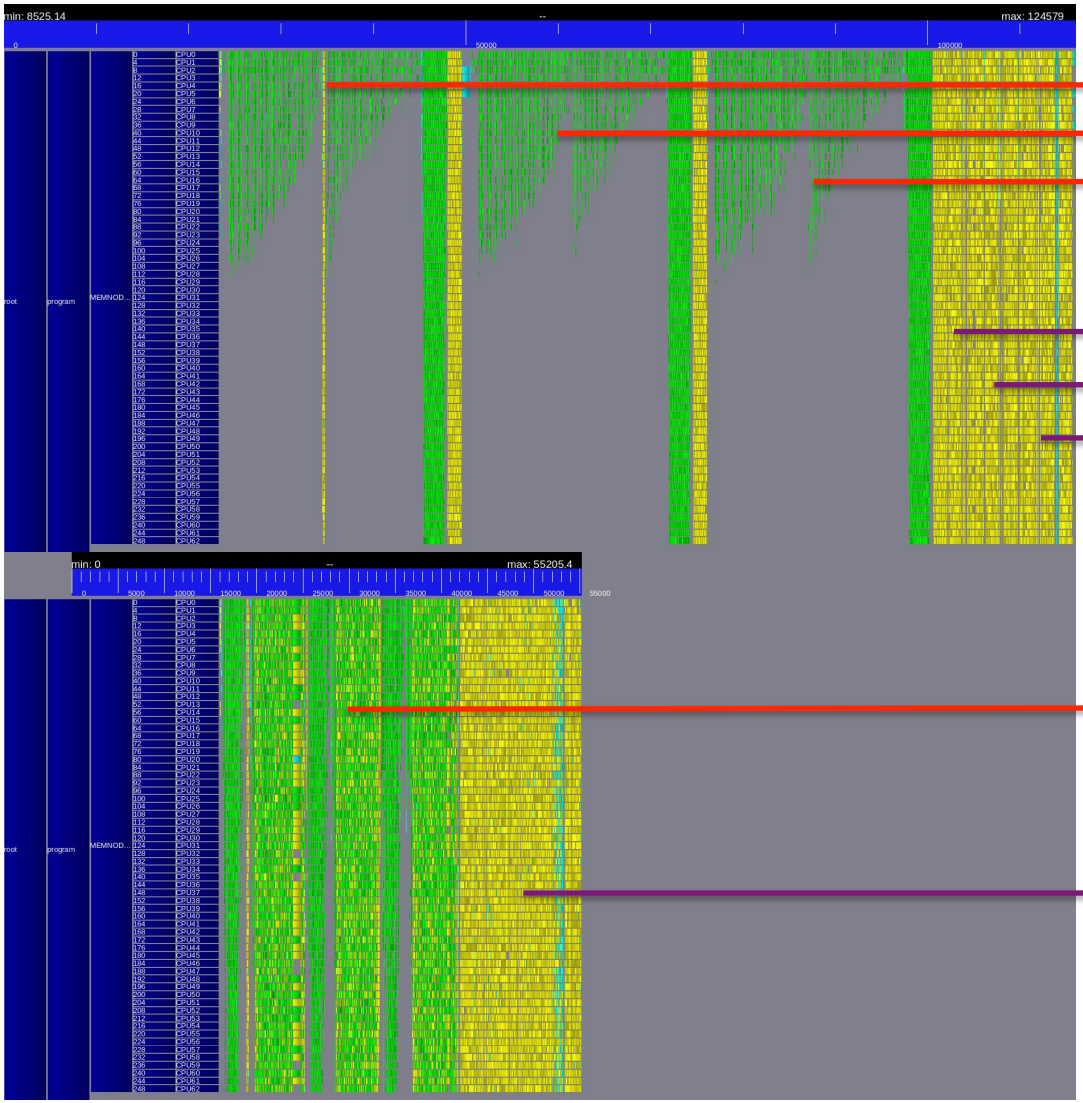
under consideration for Cray LibSci integration

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

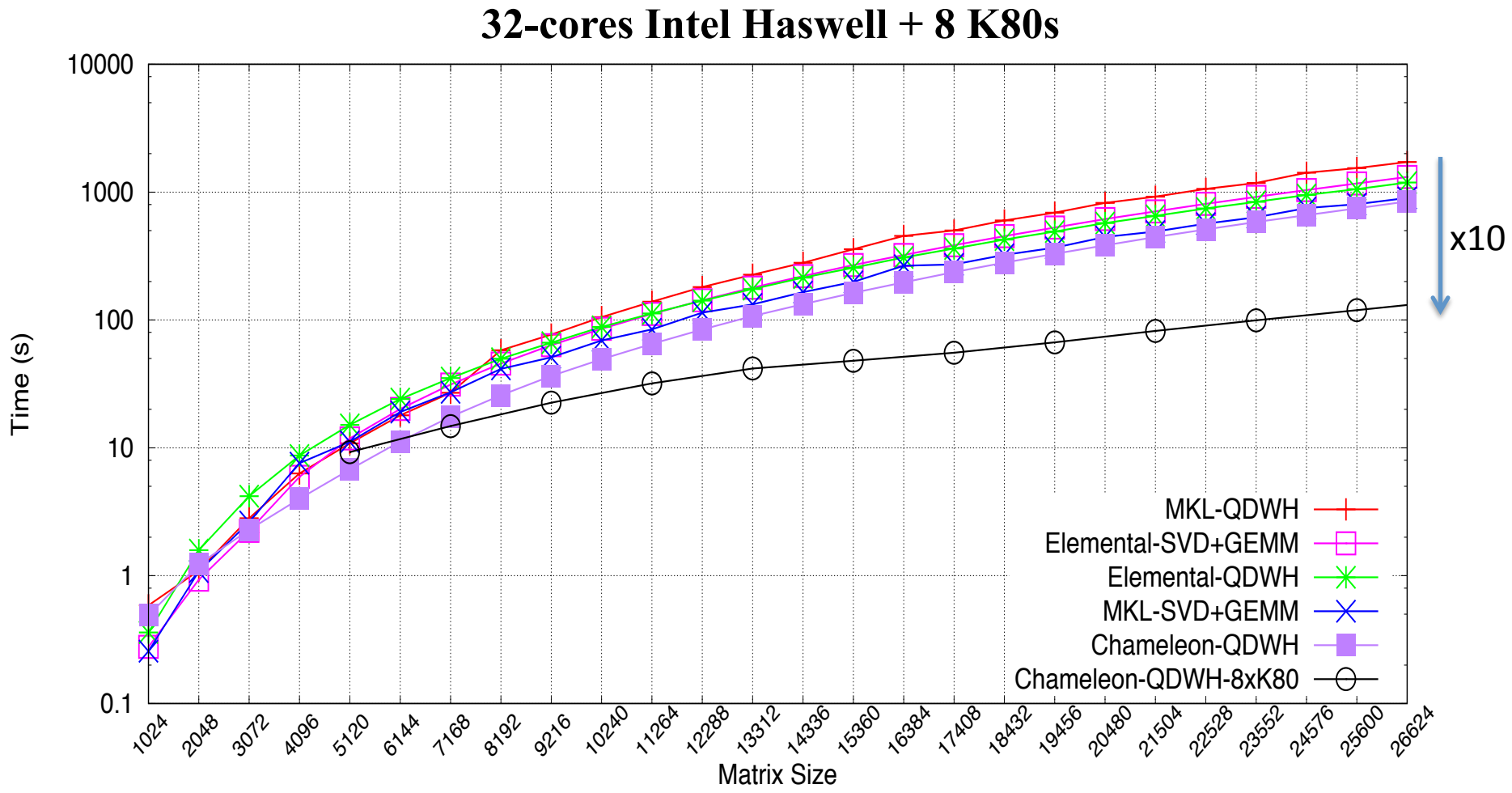
Three QR iterations

Three Cholesky iterations

Sukkari et al., submitted to IEEE TDPS'17

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

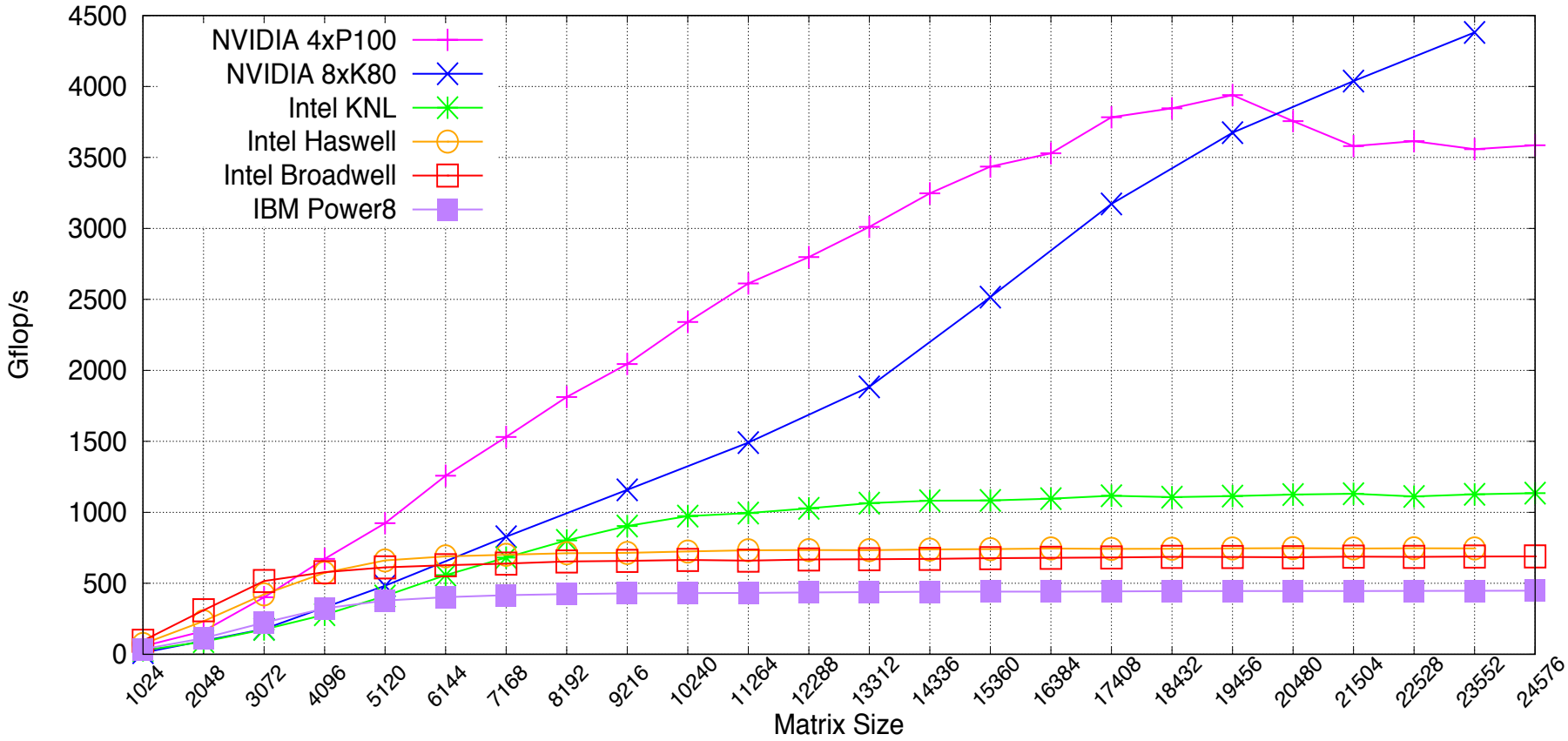
QDWH-SVD, taskified



Sukkari et al., submitted to IEEE TDPS'17

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

QDWH-SVD, taskified



Sukkari et al., submitted to IEEE TDPS'17

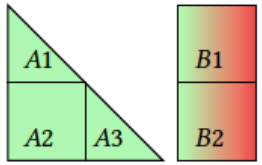
c/o D. Sukkari, H. Ltaief (KAUST) & M. Faverge (INRIA)

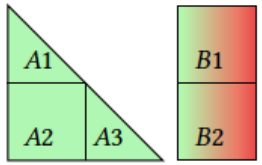
KBLAS

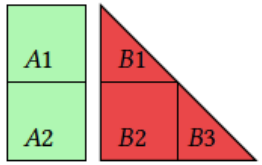
- **Subset of L2 and L3 BLAS targeting GPU and Intel MIC**
  - GEMV, SYMV, TRSM, TRMM
- **Batched BLAS for very small sizes on GPUs**
  - TRSM, TRMM, SYRK, POTRF, POTRS, POSV, TRTRI, LAUUM, POTRI, POTI
- **Recursive formulation**

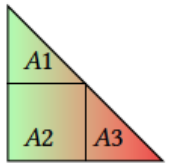
Sample recursively defined KBLAS operations

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



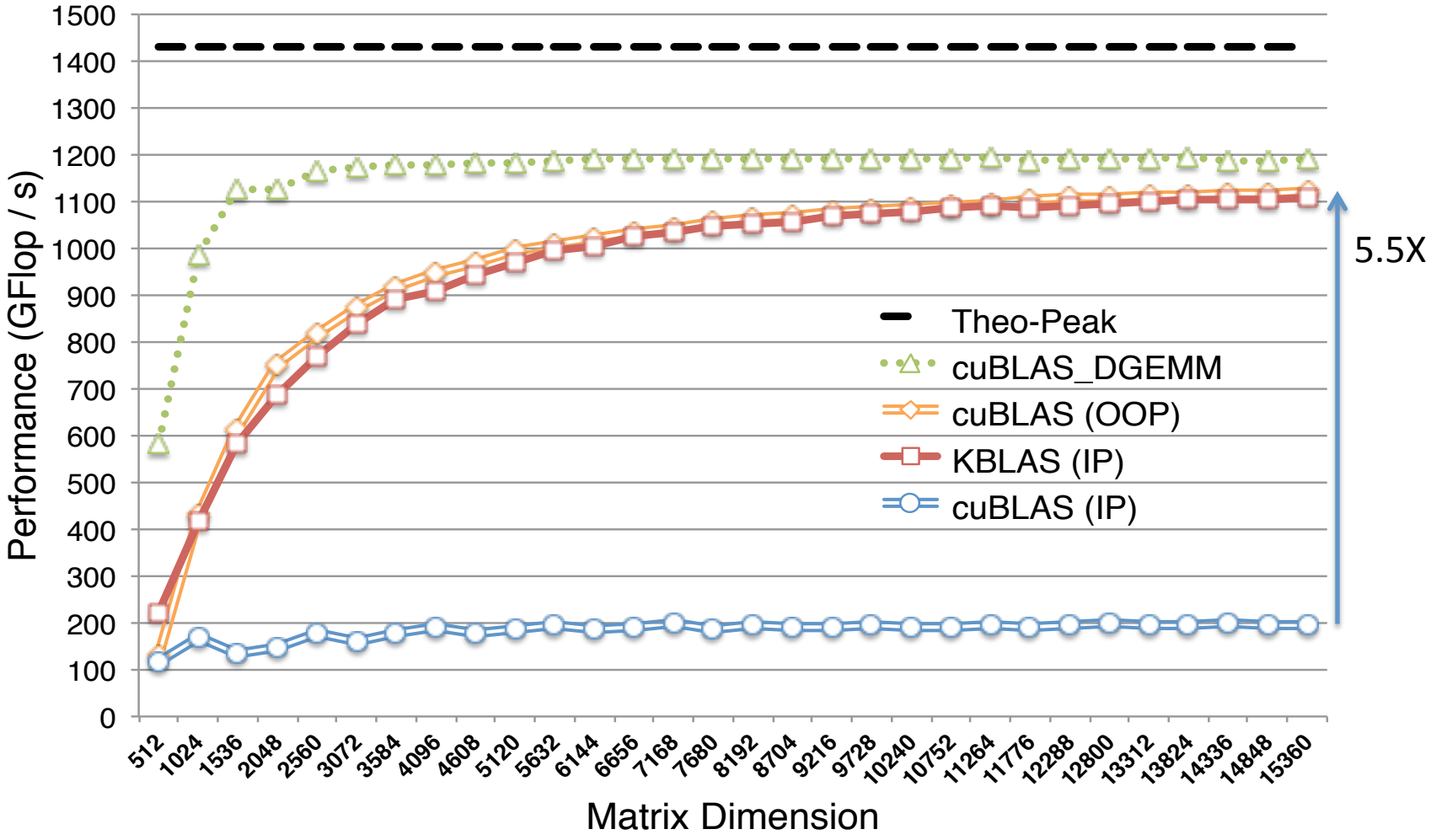






c/o A. Charara & H. Ltaief (KAUST)

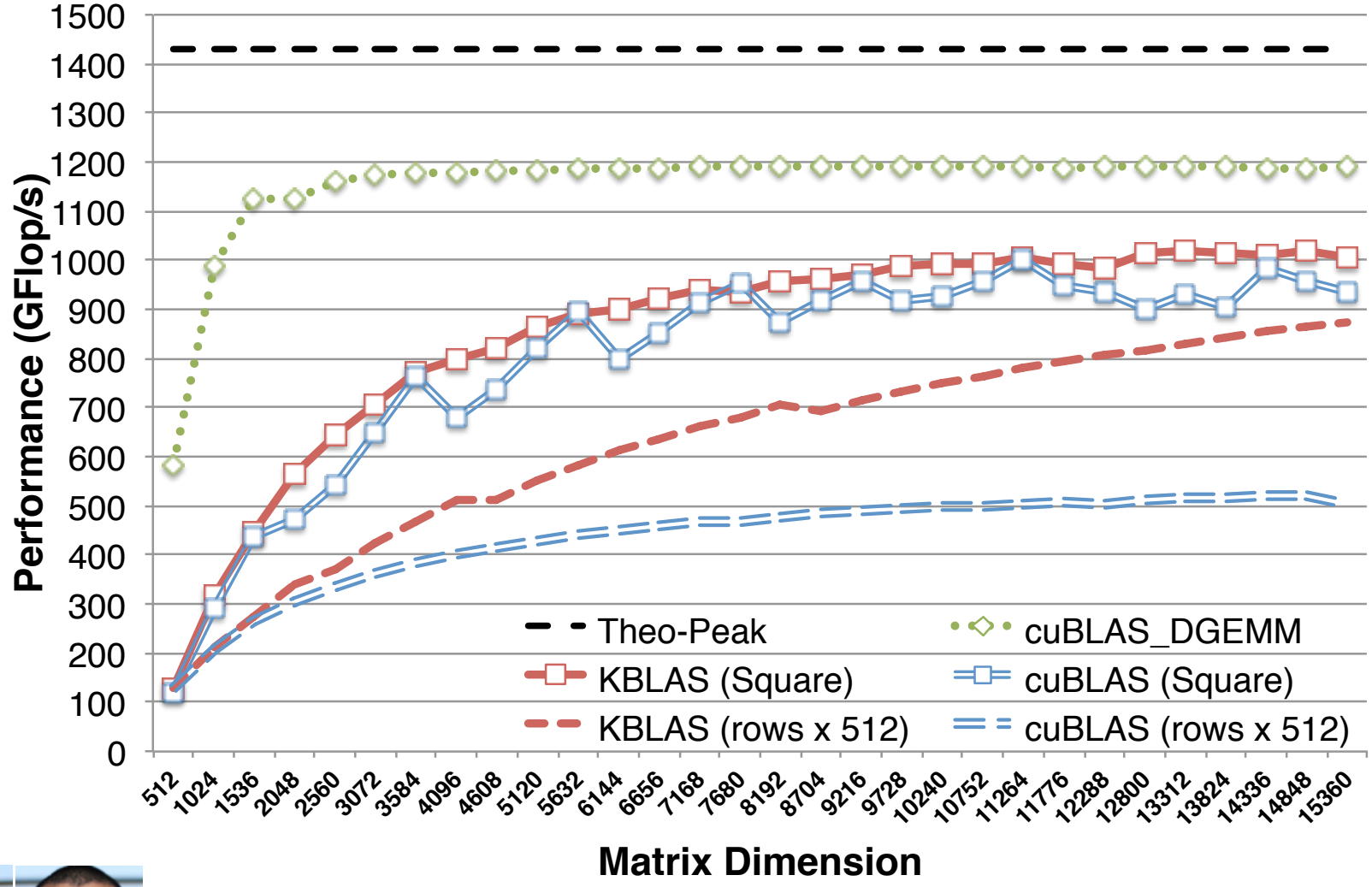
KBLAS DTRMM



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

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

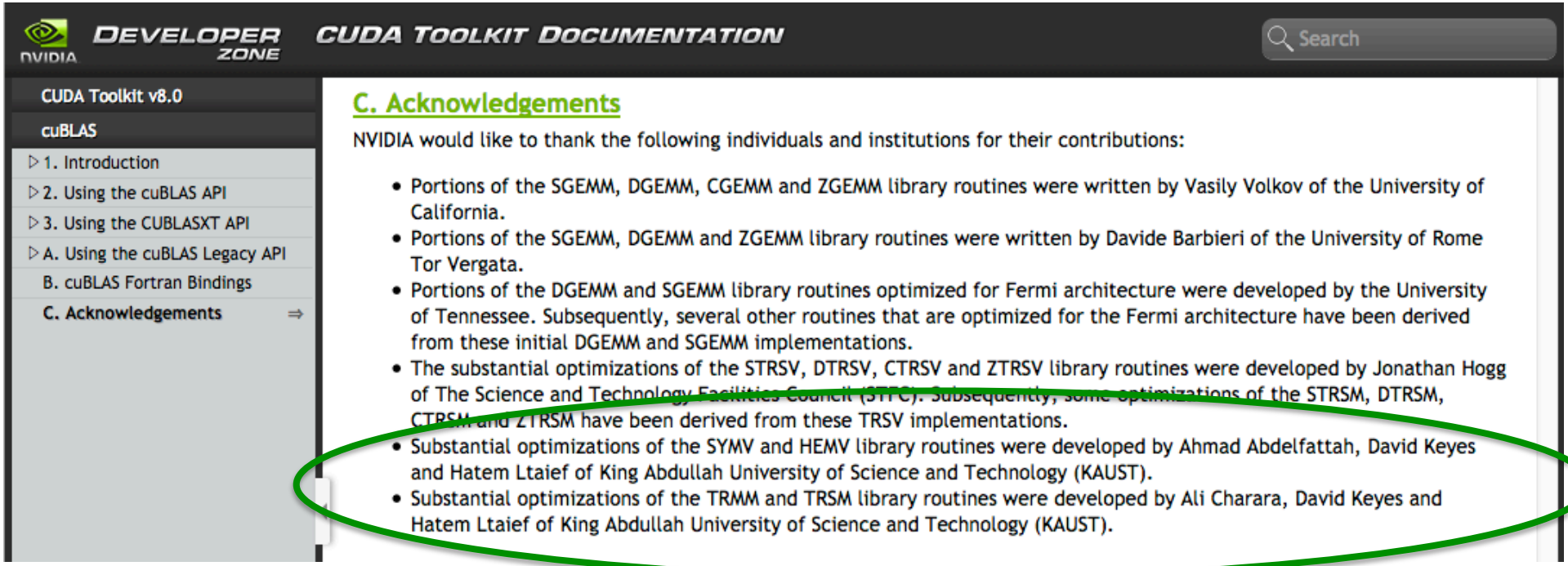
KBLAS DTRSM

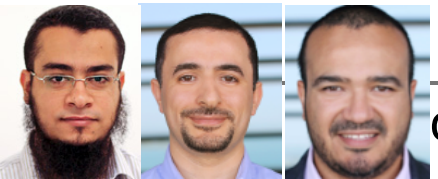


c/o A. Charara & H. Ltaief (KAUST)

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

KBLAS now in CUDA 8.0



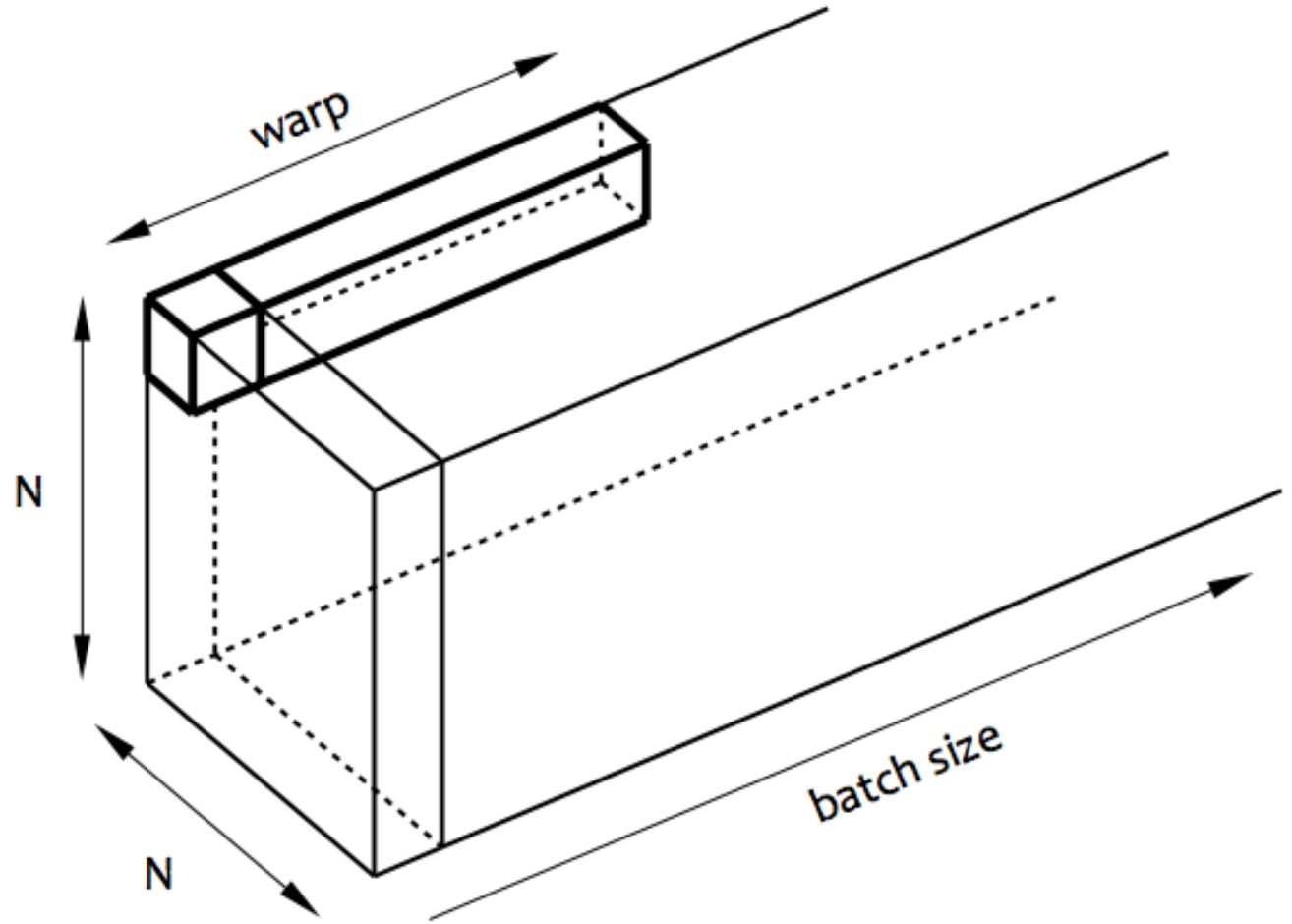


c/o A. Abdelfattah, A. Charara & H. Ltaief (KAUST)

Extending KBLAS to batched execution

- **Batched BLAS workshop:**
  - http://bit.ly/Batch-BLAS-2017
- **Problem:**
  - 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

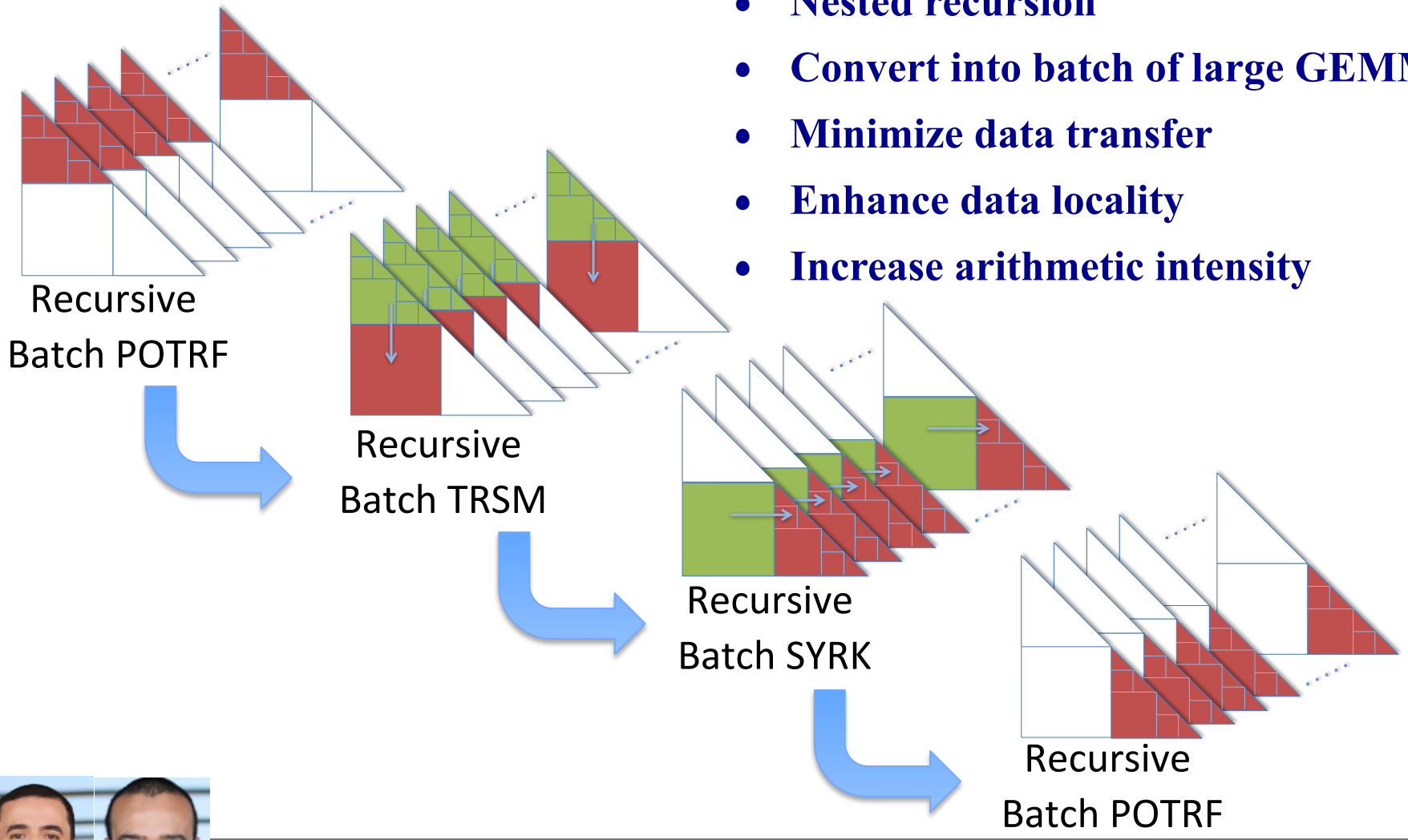
Batched operations



c/o Jacob Kurzak (ICL, U Tennessee)

KBLAS
Example: Batched POTRF

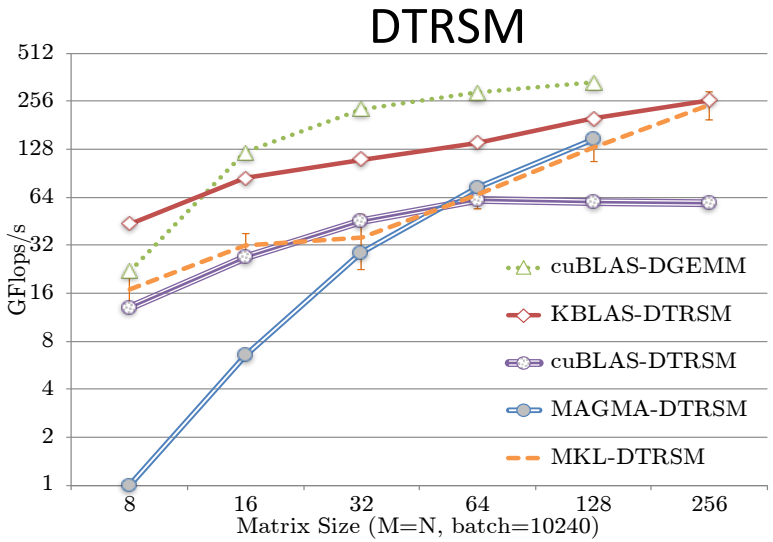
- Nested recursion
- Convert into batch of large GEMMs
- Minimize data transfer
- Enhance data locality
- Increase arithmetic intensity

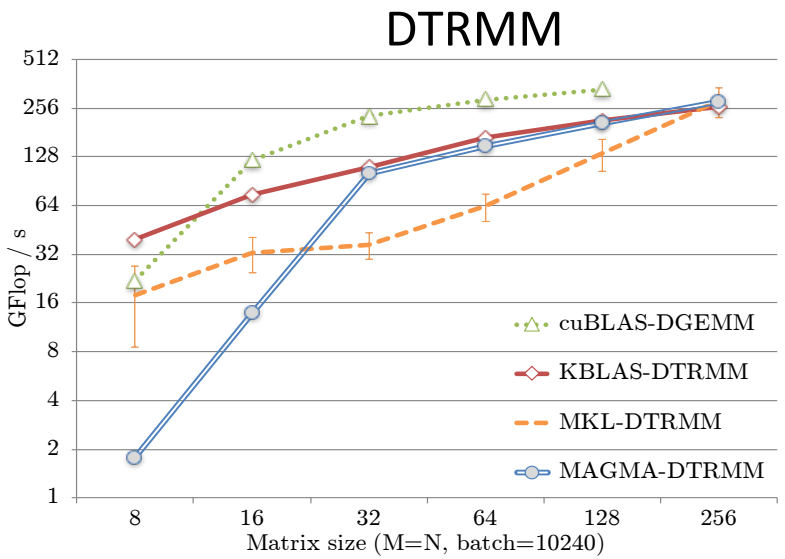


c/o A. Charara & H. Ltaief (KAUST)

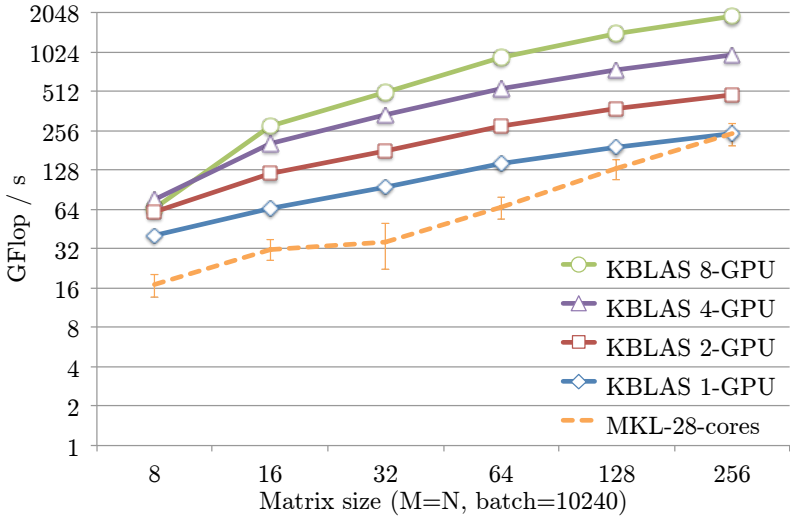
Batched KBLAS performance comparisons

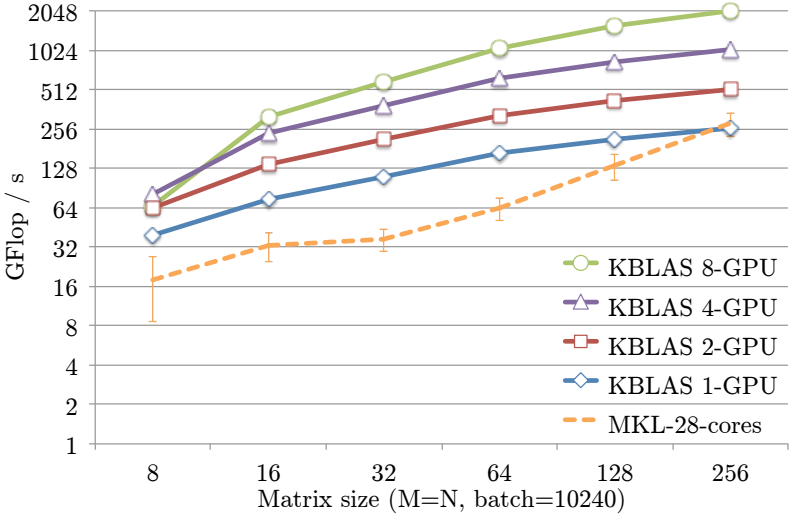
Single K40 (MKL on 28-core Broadwell)





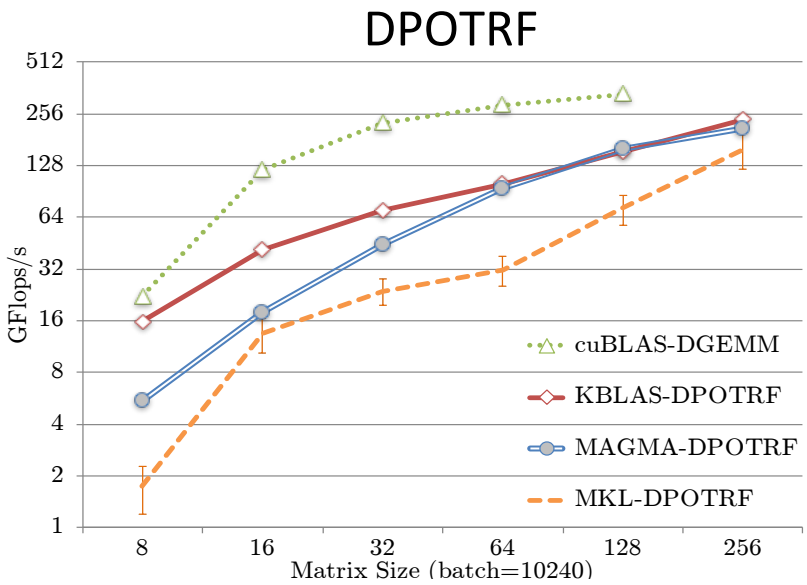
Multiple K40s (MKL on 28-core Broadwell)

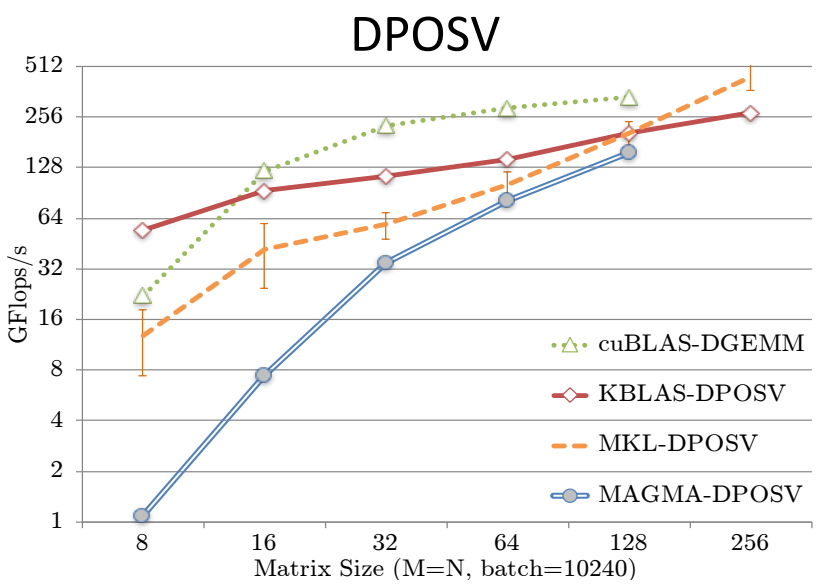




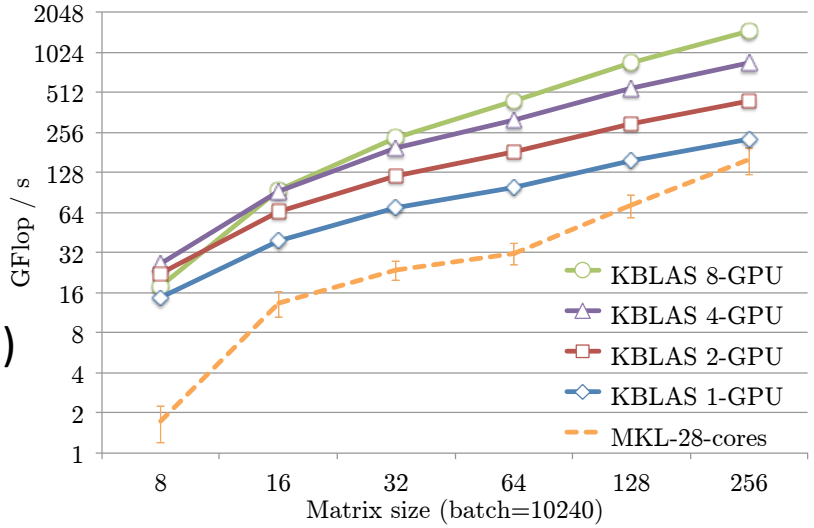
Batched KBLAS performance comparisons

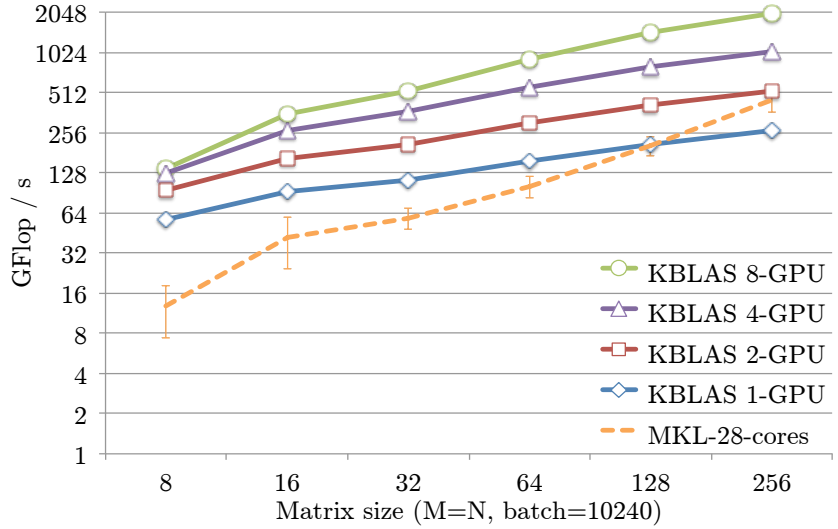
Single K40 (MKL on 28-core Broadwell)



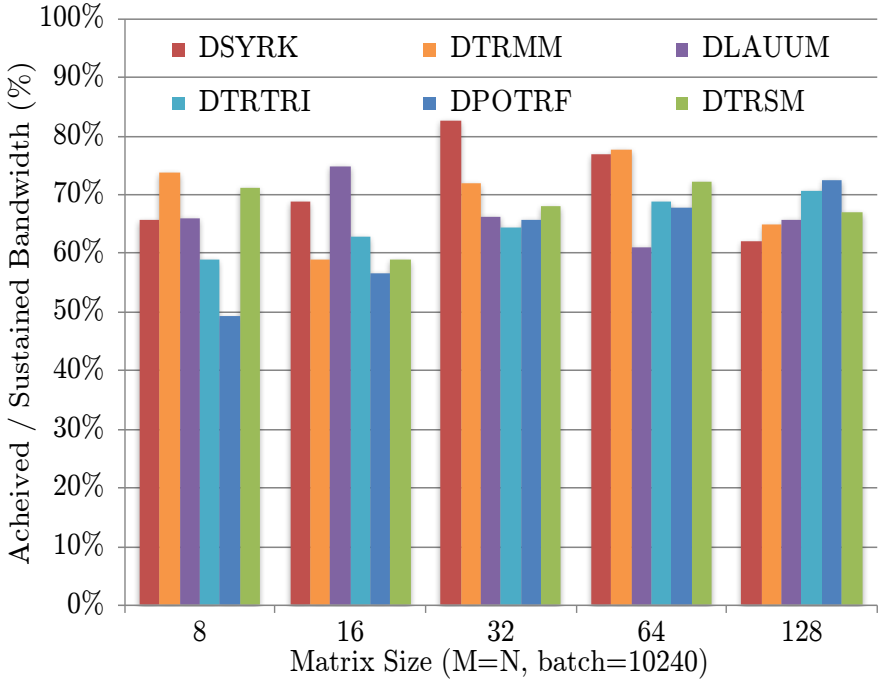


Multiple K40s (MKL on 28-core Broadwell)

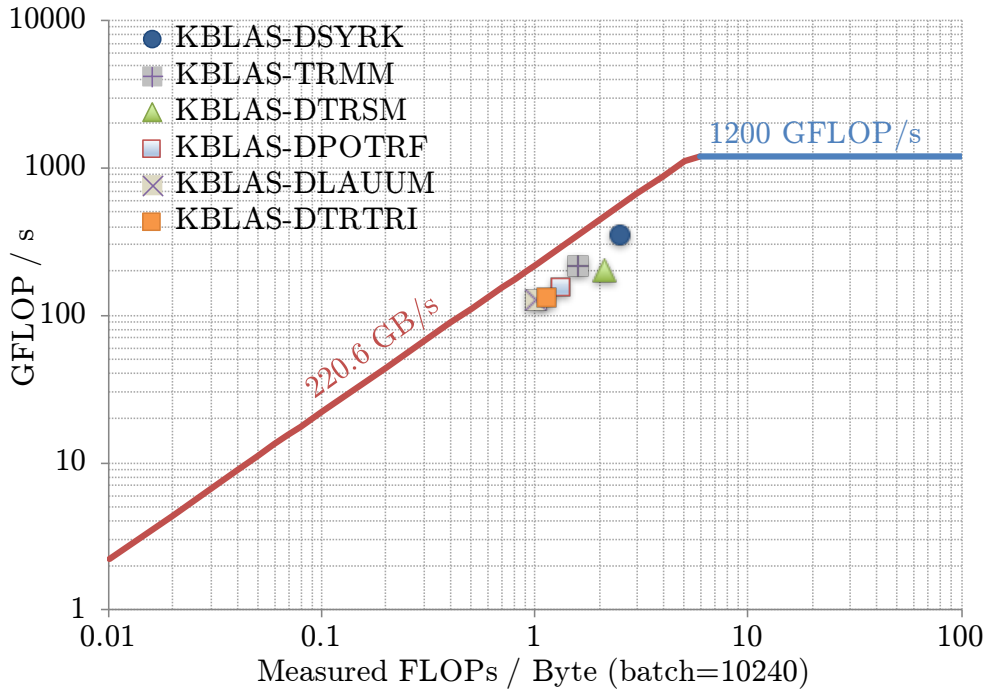




Batched KBLAS performance



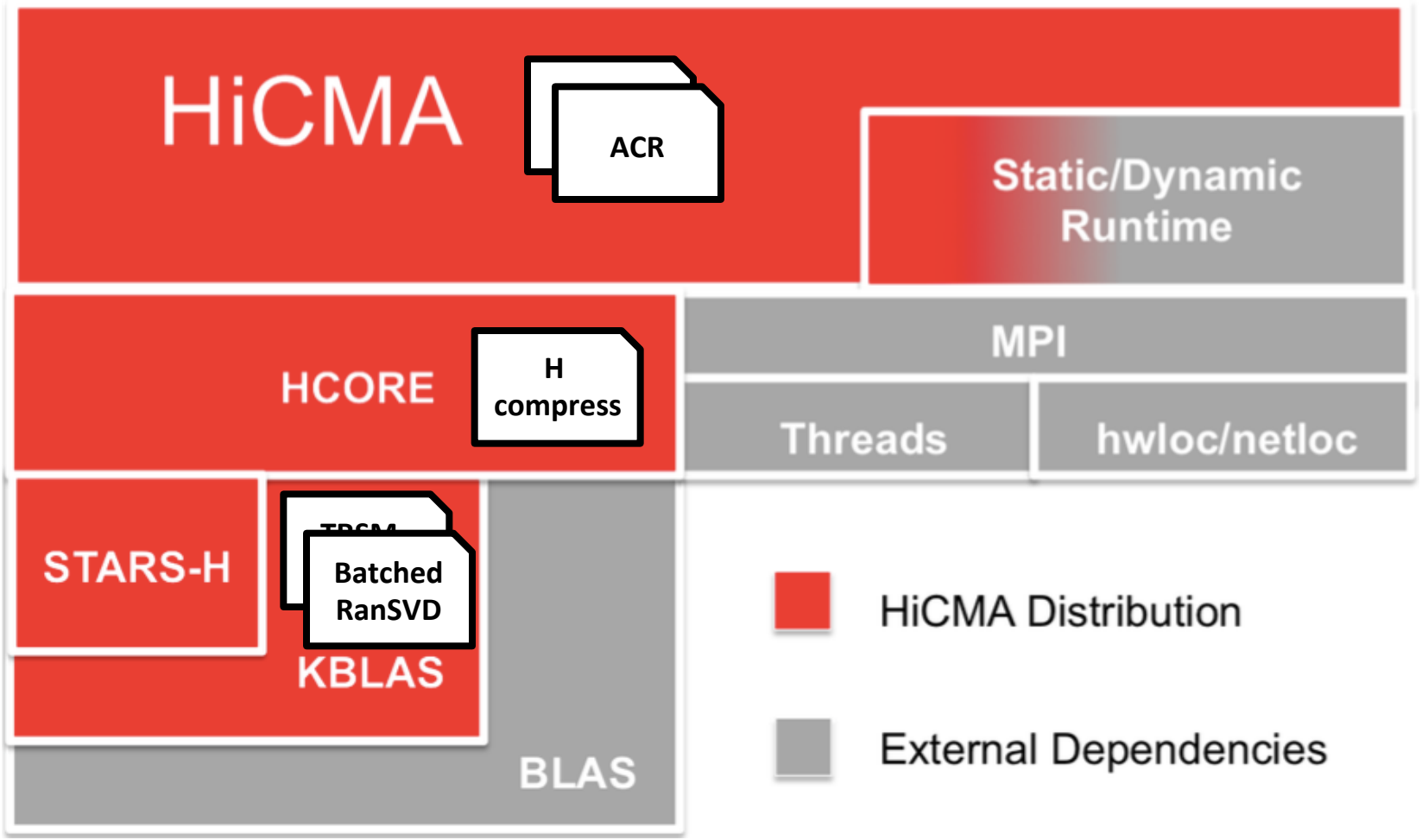
Ratio of achieved to sustained bandwidth of various KBLAS batched operations in double precision on a K40 GPU with 10240 batch size.



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.

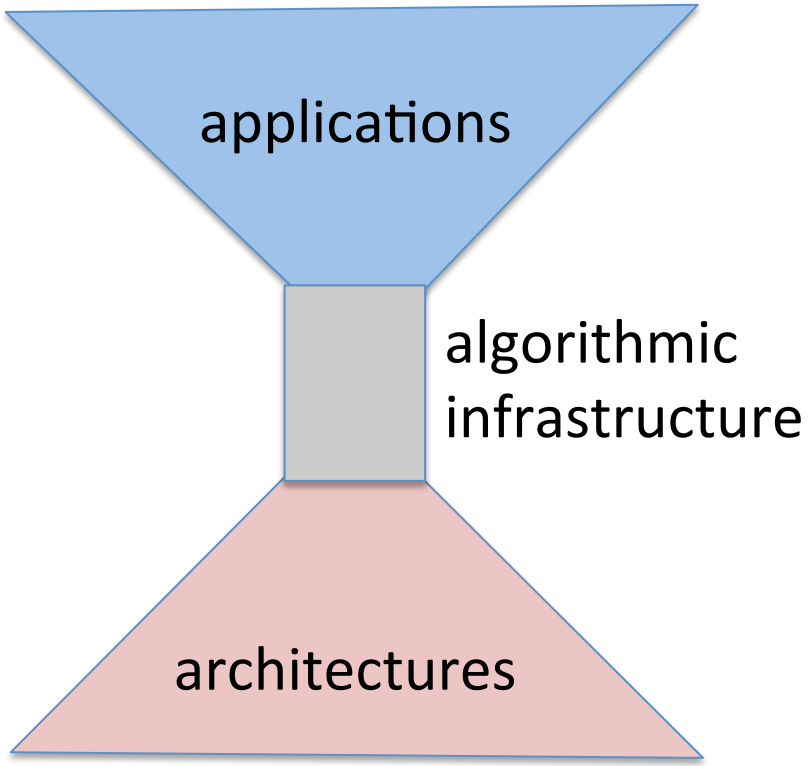
c/o A. Charara & H. Ltaief (KAUST)

Hierarchical Computations on Manycore Architectures: HiCMA*

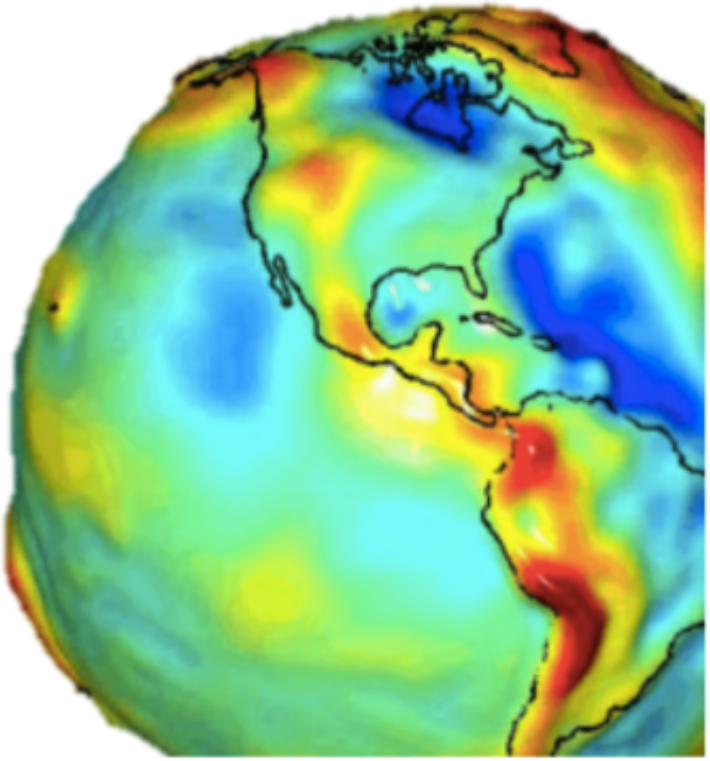


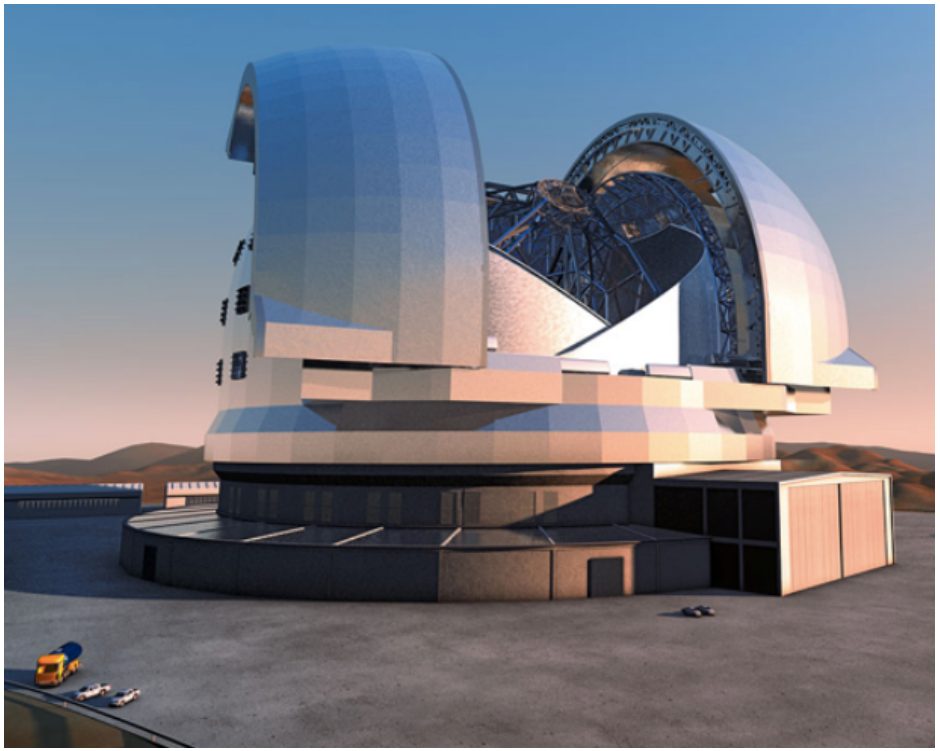
* “Hikmah” is the Arabic word for wisdom

Hourglass model for algorithms



Clients: statisticians and astronomers





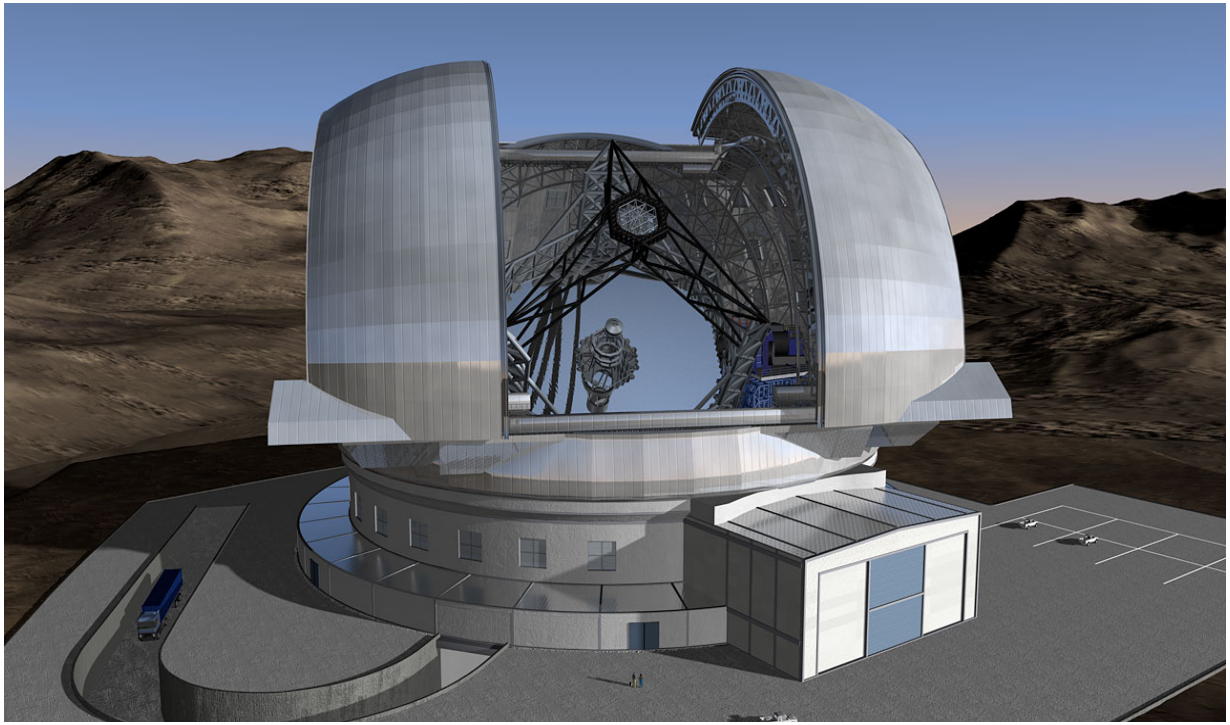
Large co-variance matrices are everywhere, but many statisticians work in MATLAB or R and can't scale their science

log-likelihood function

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

European Extremely Large Telescope

"The world's biggest eye on the sky" (40m diameter)
To be deployed in the Chilean mountains by 2024



- **Multi-objective Adaptive optics: a real time application being pursued with Observatoire de Paris**
- **De-convolve aberrations from atmospheric turbulence by dynamically controlling up to 100,000 small mirrors – a dense Cholesky inversion**

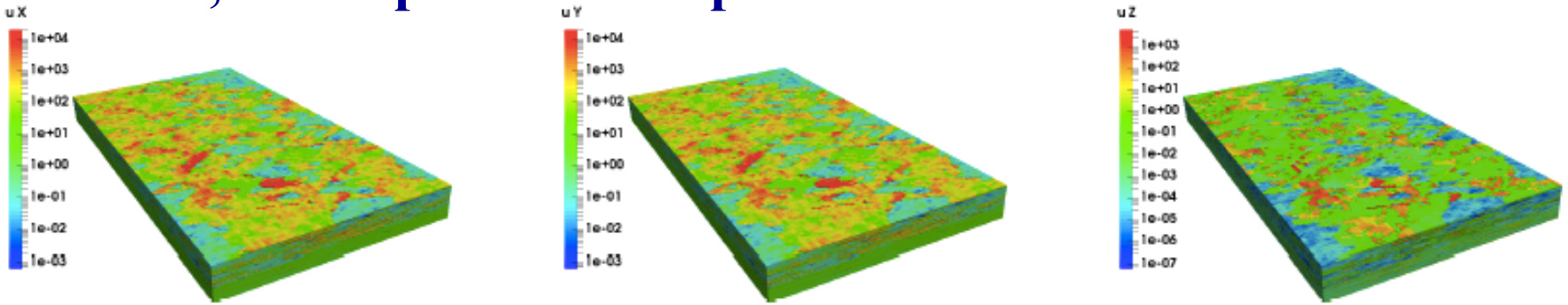
Credit: ESO (http://www.eso.org/public/teles-instr/e-elt/)

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

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	1.7?/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)

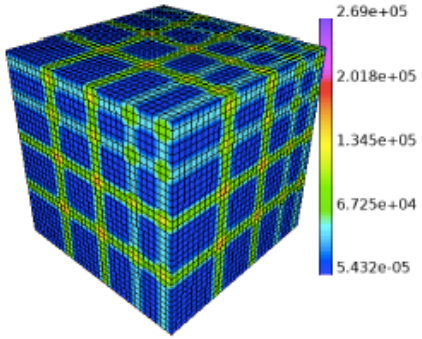
N	$\lambda = 10$	$\lambda = 5$	$\lambda = 2.5$	$\lambda = 1.5$
1024	16.1/24	7.51/16	3.49/10	1.60/6
2048	16.7/25	7.45/16	3.53/10	1.61/6
4096	15.5/24	7.53/16	3.57/10	1.58/6
8192	15.9/24	7.77/17	3.53/10	1.59/6

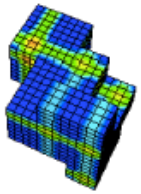
c/o S. Zampini (KAUST)

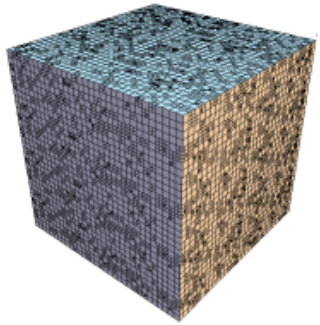
BDDC: a very robust preconditioner

- **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)

	-	λ=10	λ=5	λ=2.5
κ	150.2	7.5	4.6	2.2
it	54	15	12	8
C/Γ	0.01	0.05	0.06	0.09

p=2 (1.2M dofs)

	-	λ=10	λ=5	λ=2.5
κ	413.3	5.9	4.3	2.3
it	113	15	12	9
C/Γ	0.01	0.02	0.02	0.04

Hexahedral non-conforming mesh

p=1 (330K dofs)

	-	λ=10	λ=5	λ=2.5
κ	203.4	5.8	3.2	2.0
it	62	13	10	7
C/Γ	0.02	0.05	0.06	0.09

p=2 (3.5M dofs)

	-	λ=10	λ=5	λ=2.5
κ	330.8	5.1	3.4	2.0
it	97	14	11	8
C/Γ	0.01	0.01	0.02	0.04

c/o S. Zampini (KAUST)

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

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

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

$$\mathsf{A} = \begin{bmatrix} \mathsf{A}_{II} & \mathsf{A}_{I\Gamma} \\ \mathsf{A}_{I\Gamma}^T & \mathsf{A}_{\Gamma\Gamma} \end{bmatrix} = R^T \mathsf{A}^* R, \quad \mathsf{A}^* = \begin{bmatrix} \mathsf{A}^{(1)} & & \\ & \ddots & \\ & & \mathsf{A}^{(N)} \end{bmatrix},$$

with

$$\mathsf{A}^{(i)} = \begin{bmatrix} \mathsf{A}_{II}^{(i)} & \mathsf{A}_{I\Gamma}^{(i)} \\ \mathsf{A}_{I\Gamma}^{(i)T} & \mathsf{A}_{\Gamma\Gamma}^{(i)} \end{bmatrix}$$

the matrix of the FEM problem on Ω_i.

c/o S. Zampini (KAUST)

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

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

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

c/o S. Zampini (KAUST)

Global Schur complement is subassembled

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

Block Cholesky

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

- **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**

c/o S. Zampini (KAUST)

BDDC with low rank Schur approximations

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

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

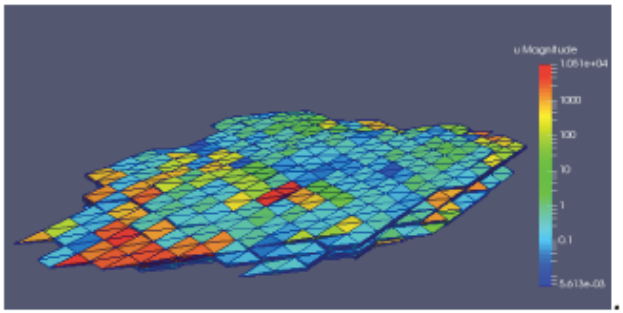
At full accuracy

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

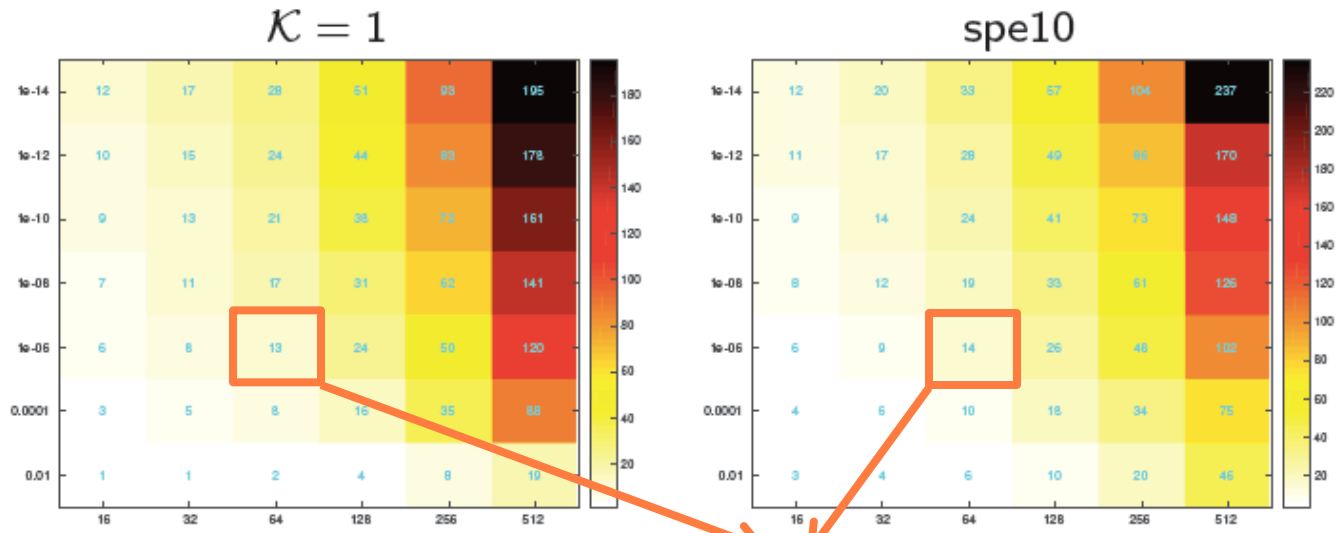
c/o S. Zampini (KAUST)

BDDC with low rank Schur approximations

Darcy problem, SPE10 benchmark. One representative subdomain



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

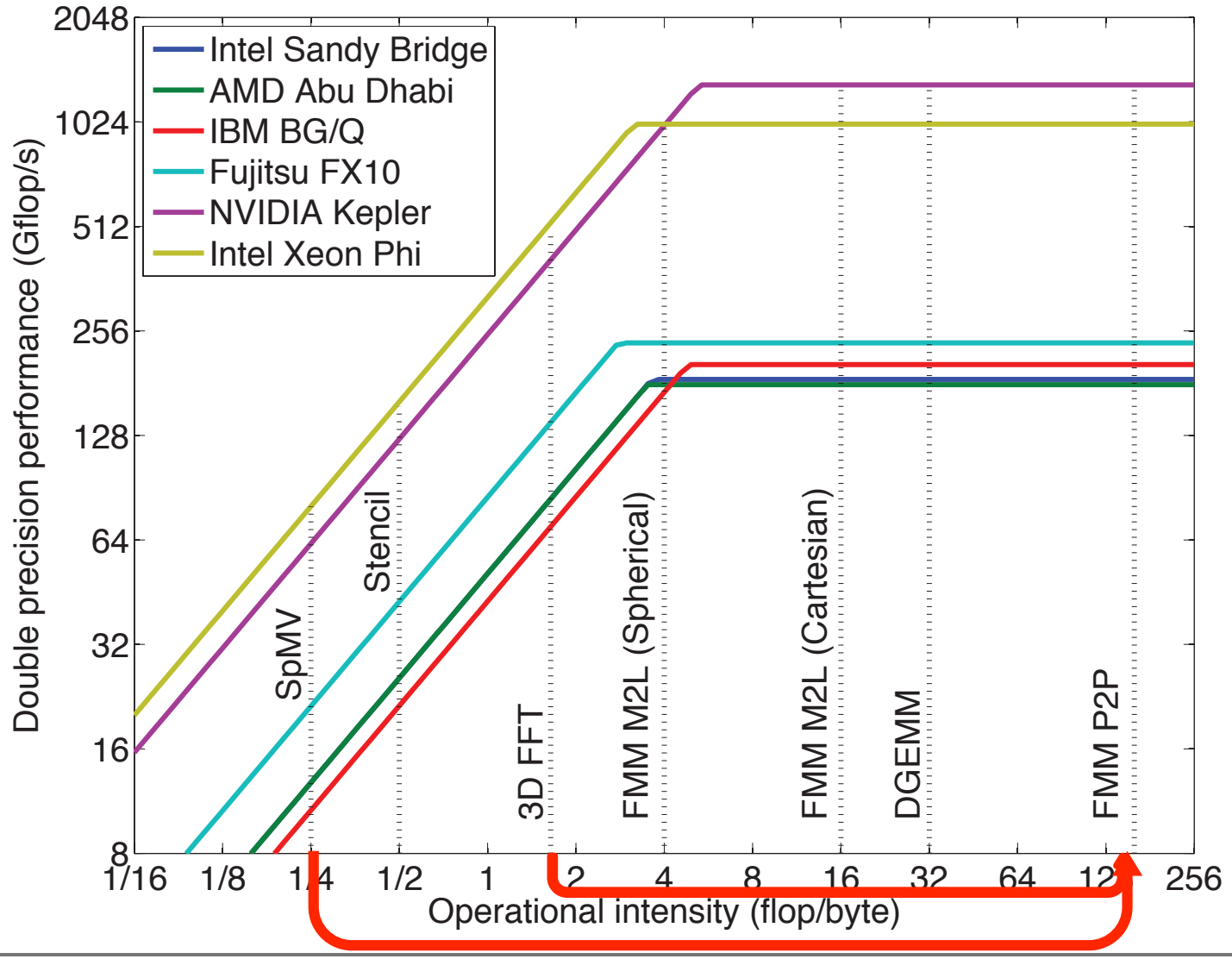


c/o S. Zampini (KAUST)

Fast Multipole for Poisson solves

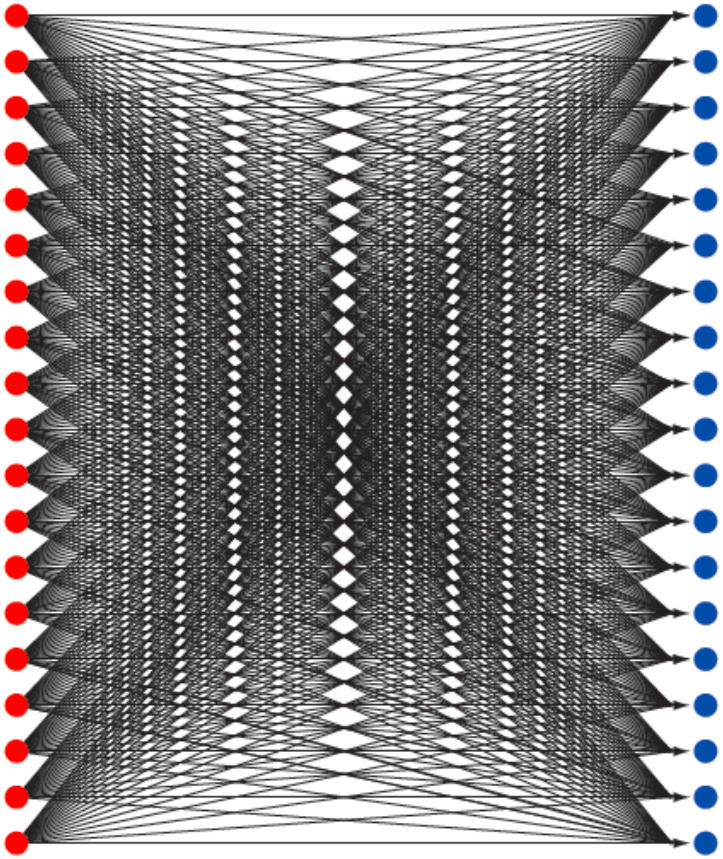
- Increase arithmetic intensity
- Reduce synchrony
- Increase concurrency

Arithmetic intensity of numerical kernels

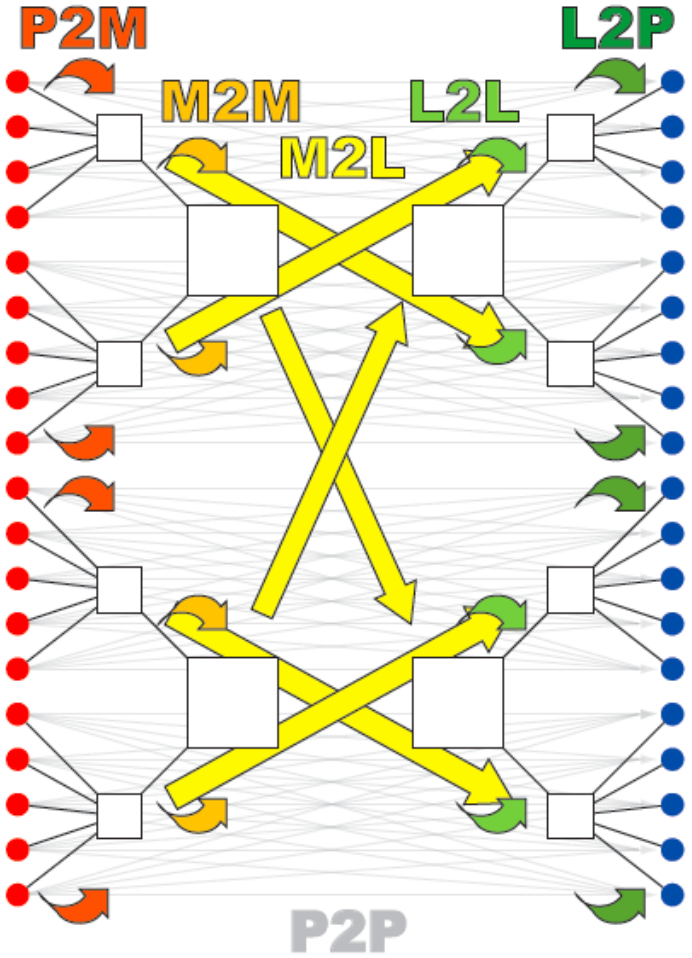


c/o R. Yokota (TiTech, KAUST)

Hierarchical interactions of Fast Multipole



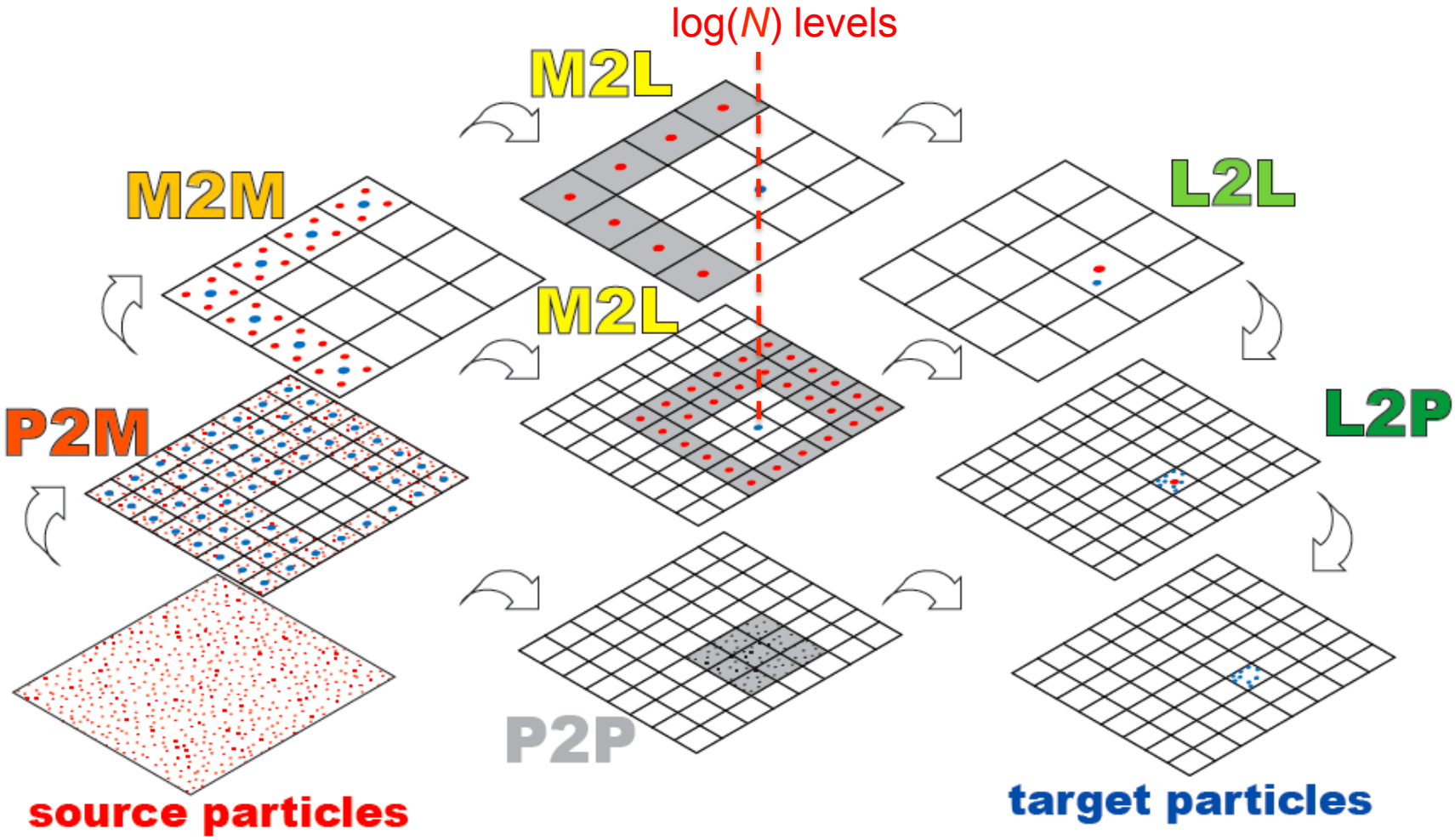
(a) Direct method



(b) Fast Multipole Method

c/o R. Yokota (TiTech, KAUST)

Geometrical structure of Fast Multipole



c/o R. Yokota (TiTech, KAUST)

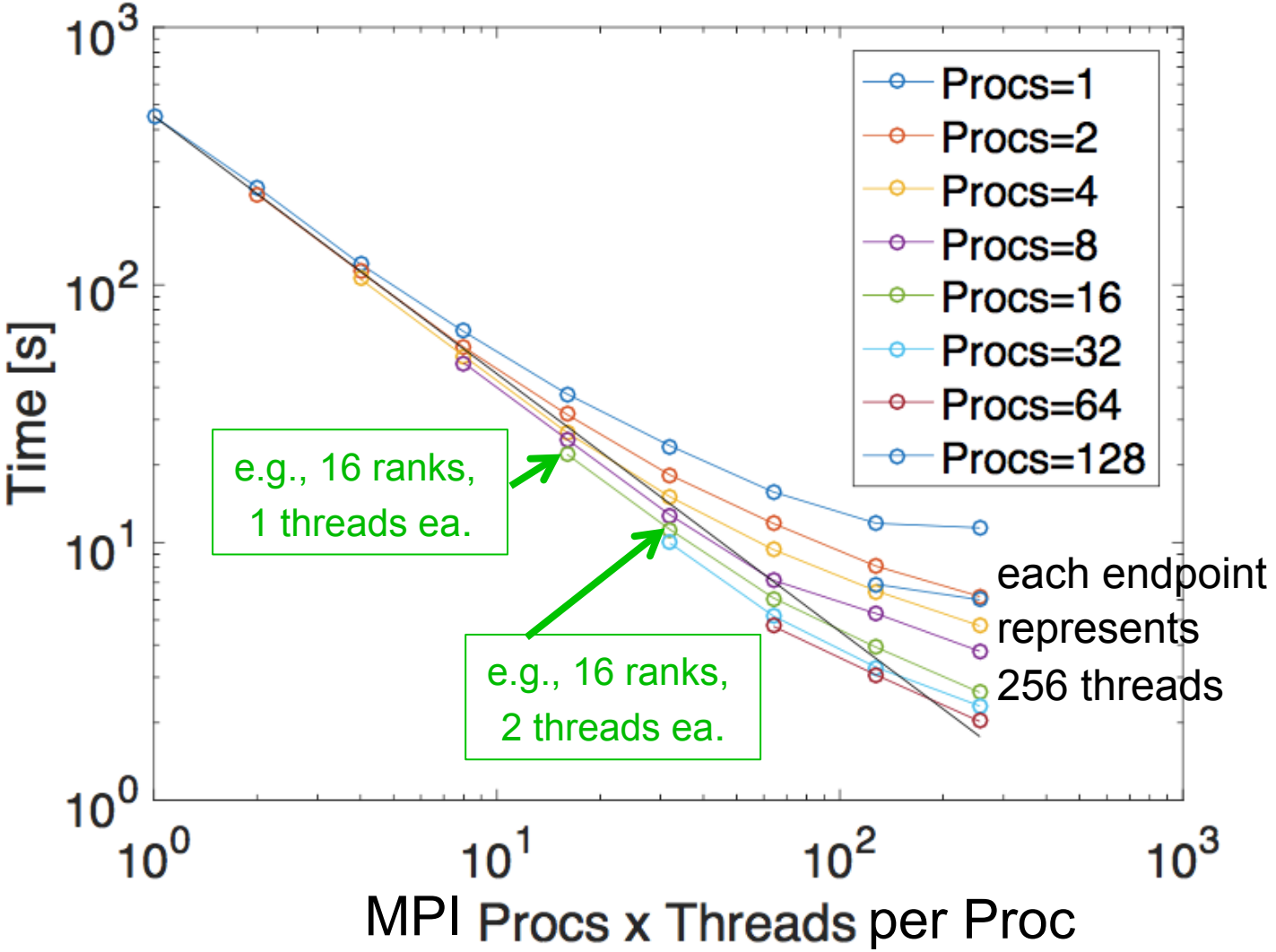
Synchronization reduction – FMM

- **Within an FMM application, data pipelines of different types and different levels can be executed asynchronously**
  - **FMM simply adds up (hierarchically transformed) contributions**
  - **e.g., P2P and P2M -> *M2M* -> M2L -> *L2L* -> L2P**
- **Geographically distinct targets can be updated asynchronously**

Features of FMM

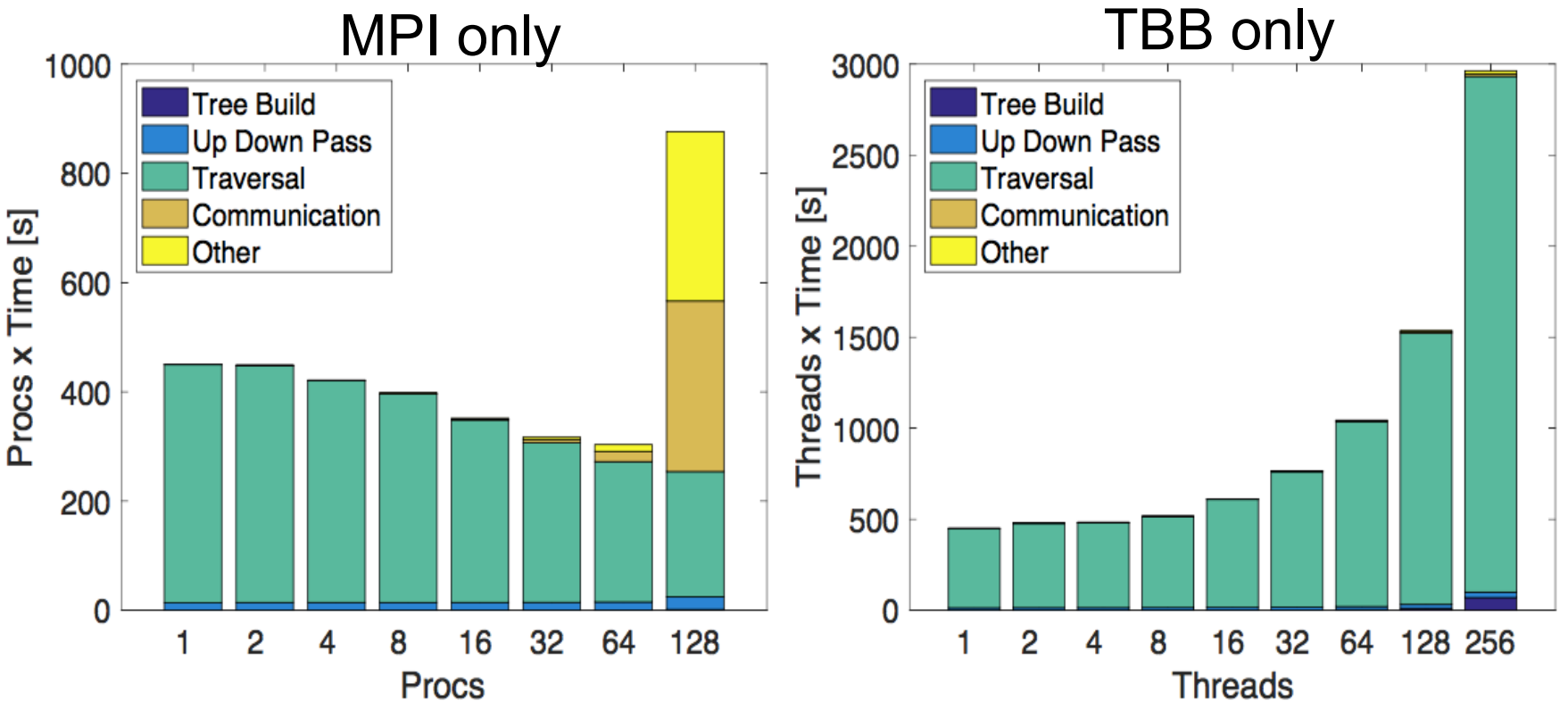
- **High arithmetic intensity**
- **No all-to-all communication**
- $O(\log P)$ **messages**
  - **with high concurrency and asynchrony among themselves**
- **Up to** $O(N)$ **arithmetic concurrency**
- **Tunable granularity in the sense of "*h-p*"**
  - **based on analytic "admissibility condition"**
- **Inside 8 Gordon Bell Prizes, 1997-2012**
- **Many effective implementations on GPUs**
- ***But fragile* (based on analytical forms of operators)**

ExaFMM on KNL in all-to-all cluster mode



c/o R. Yokota (TiTech, KAUST)

ExaFMM on KNL



(a) Process scalability using 1 thread (flat model) (b) Thread scalability using 1 process (flat model)

Figure 5: Breakdown of the calculation time for TBB thread & MPI process scalability.

c/o R. Yokota (TiTech, KAUST)

FMM as preconditioner

- FMM is a solver for free-space problems for which one has a Green's function
- For finite boundaries, FMM combines with BEM
- FMM and BEM have controllable truncation accuracies; can precondition other, different discretizations of the same PDE
- Can be regarded as a preconditioner for "nearby" problems, e.g., ∇^2 for $\nabla \cdot (1+\varepsilon(\vec{x}))\nabla$

FMM's role in solving PDEs

$$u = \int_\Gamma \frac{\partial u}{\partial n} G d\Gamma - \int_\Gamma u \frac{\partial G}{\partial n} d\Gamma + \int_\Omega f G d\Omega \quad \text{in } \Omega$$

BEM FMM

$$N_\Omega \left\{ \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} \right. = \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{bmatrix} - \overbrace{\begin{bmatrix} \ddots & & \\ & \frac{\partial G_{ij}}{\partial n} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ u_j \\ \vdots \end{bmatrix} + \overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Omega} \begin{bmatrix} \vdots \\ f_j \\ \vdots \end{bmatrix}$$

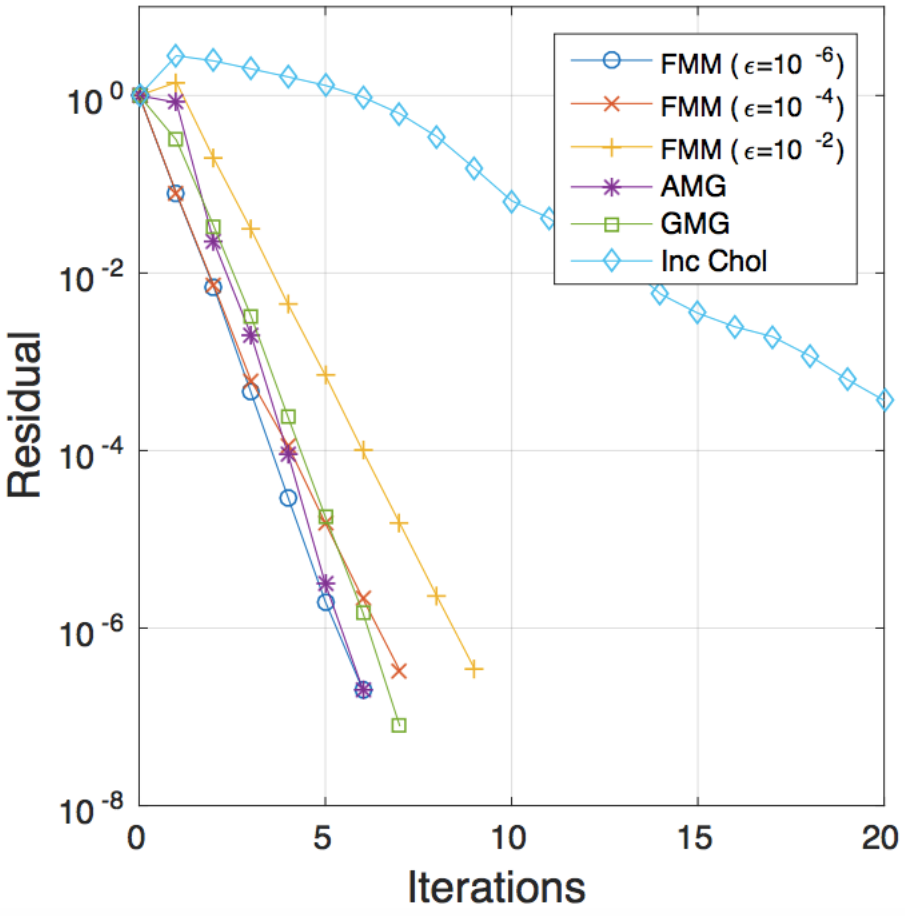
The preconditioner is reduced to a matvec, like the forward operator itself – the same philosophy of the sparse approximate inverse (SPAI), but cheaper.

***More concurrency, more intensity, less synchrony* than ILU, MG, DD, etc.**

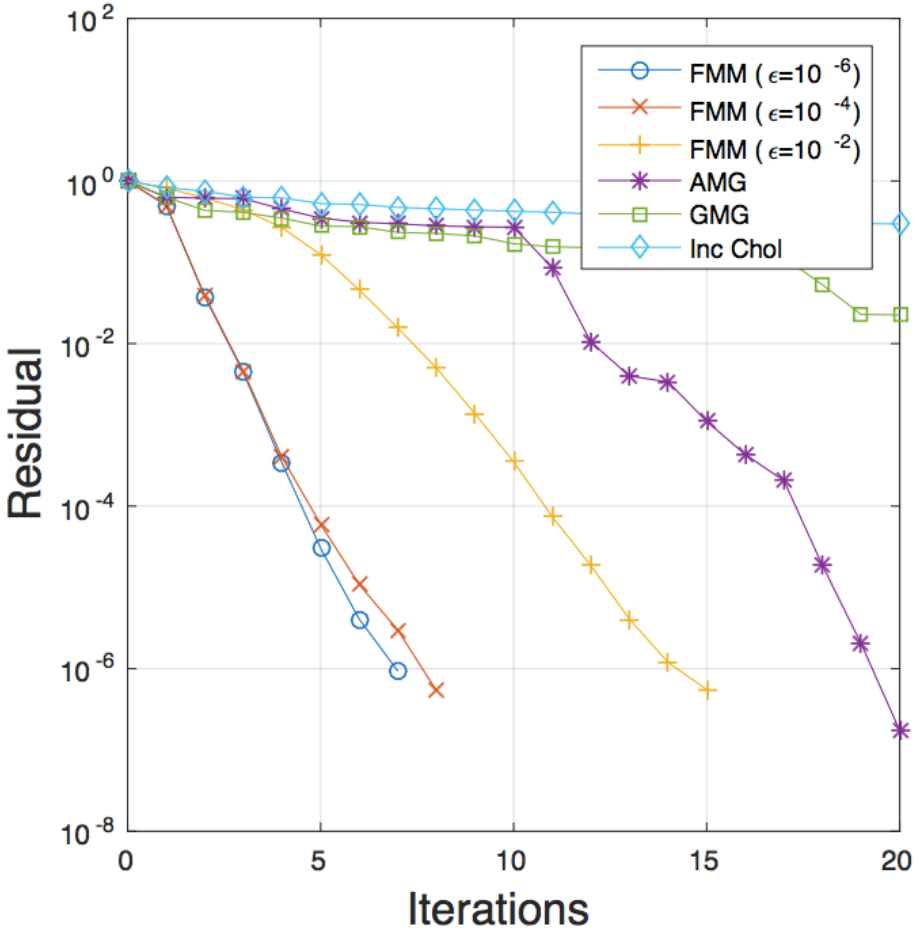
c/o H. Ibeid (UIUC, KAUST'16)

FMM/BEM preconditioning of FEM

Poisson



Wave Helmholtz



c/o H. Ibeid (UIUC, KAUST'16)

Other galaxies?

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 artifactual 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)**
  - **greater arithmetic intensity**
  - **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:

CENTER OF EXCELLENCE


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