PETSc: a SWOT analysis

David Keyes

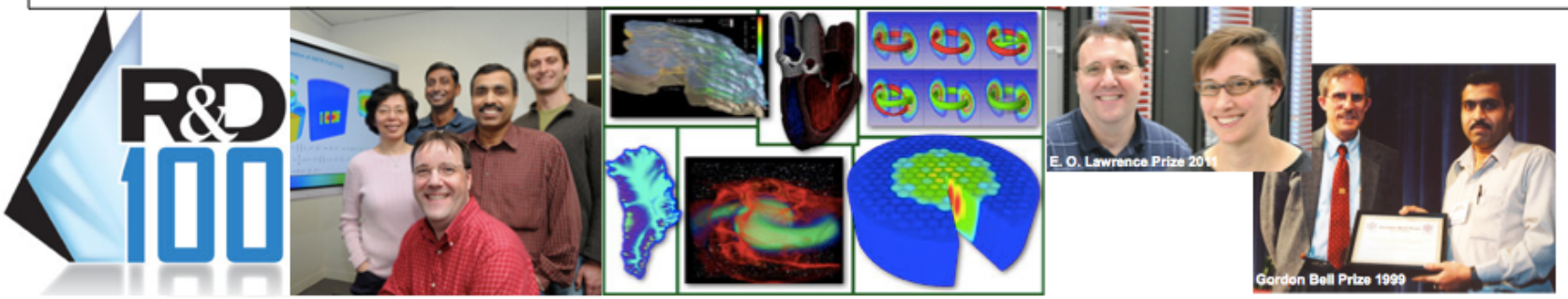
Extreme Computing Research Center

KAUST

NITRD agency success story: PETSc (1992-)

- **The Portable Extensible Toolkit for Scientific Computing (PETSc)**
  - **used in thousands of scientific and engineering codes**
  - **software structure has inspired countless other library developers**
- **Suite of distributed data structures and routines for the scalable solution of large systems of equations**
- **Has won R&D 100 award, been part of multiple Gordon Bell prizes, Best Paper prizes; its developers won DOE's E. O. Lawrence award in 2011**
- **Funded by Argonne National Laboratory, DOE, and NSF**

Acoustics, Aerodynamics, Air Pollution, Arterial Flow, Bone Fractures, Brain Surgery, Cancer Surgery, Cancer Hyperthermia, Carbon Sequestration, Cardiology, Cell Function, Combustion, Concrete, Corrosion, Data Mining, Dentistry, Earth Quakes, Economics, Fission, Fluid Dynamics, Fusion, Glaciers, Ground Water Flow, Hydrology, Linguistics, Mantle Convection, Magnetic Films, Material Science, Medical Imaging, Ocean Dynamics, Oil Recovery, PageRank, Polymer Injection Molding, Polymeric Membranes, Quantum Computing, Seismology, Semiconductors, Rockets, Relativity, …



https://en.wikipedia.org/wiki/SWOT_analysis

Sample questions

- Strengths
  - What does PETSc do better than most?
- Weaknesses
  - What could PETSc improve?
- Opportunities
  - What trends favor PETSc's future?
- Threats
  - What obstacles does PETSc face?

Strengths: PETSc Praised

- What does PETSc do better than most?
- What are PETSc's advantages?
- What resources can PETSc draw upon?

Weaknesses: PETSc Panned

- What could PETSc improve?
- What are PETSc's disadvantages?
- What resources does PETSc lack?

Opportunities: PETSc's Prospects

- What trends favor PETSc's future?
- What changes in technology?
- What changes in policy?
- What changes in culture?

Threats: PETSc's Perils

- What obstacles does PETSc face?
- What changes in technology?
- What changes in policy?
- What changes in culture?
- What are "competitors" doing?

PETSc's attributes

- Portability – across HW/SW platforms
- Extensibility – contexts, shells, custom registrations
- Composability – nested calls, plug-n-play
- Configurability – buildtime, runtime
- Callability – many language bindings, library style
- Compartmentability – subcommunicators
- Comprehensibility – API to external functionality
- Profilability – rich instrumentation
- Testability/reproducibility – repository, nightly regressions
- Availability – open source
- Reliability – debugged by the global community ☺
- Stability – 20+ year history
- Inclusivity – open to outside contributions
- Adoptability – open to new applications

PETSc set the culture of scientific software engineering

- Today, many frameworks aspire to and, to some extent, achieve these attributes
- PETSc has been enormously influential in creating this open ecosystem of quality software components

TOPS2 Midterm Review (2009)

"PETSc [59] includes a large suite of parallel linear and nonlinear equation solvers that are callable from C, C++, Fortran77, Fortran90, and Python. It provides distributed data structures with many mechanisms needed within parallel application codes, such as matrix and vector assembly routines that allow the overlap of communication and computation. It includes support for parallel hierarchical distributed arrays and PETSc3.0, developed under the current TOPS project, has its own mesh management library. PETSc provides access through its standard TOPS interface to the algebraic multigrid libraries BoomerAMG from hyper and ML from Trilinos and the smoothed aggregation AMG solver Prometheus [62] is now maintained as a PETSc package."

Keys to PETSc's success

- Tight integration with MPI's performance oriented features
- Control over granularity and scheduling of memory allocations and communications
- Opportunity to overlap useful computation with communication latency and synchronization latency (begin/end, get/restore)

More keys to PETSc's success

- Large variety of methods
- Large variety of data structures
- Variety of flavors (e.g., set versus add, matrix explicit versus matrix-free)
- Extensive services (e.g., monitors, viewers)
- Comprehensive instrumentation for correctness debugging and performance debugging

Still more keys to PETSc's success

- Rich set of test and tutorial codes
- Conservative defaults for parameters
  - robustness, for novices
- Multilayered access to parameters
  - progressive performance, for experts

Icing on the user's cake

- Quality, self-generating documentation
- Regularly available workshops and tutorials
- Conscientious user community responsiveness with e-mail tracking and blogging

Icing on the developer's cake

- Direct participation in algorithmic research
- Direct participation in stakeholder applications
- Adoption of PETSc as a software stack offering by commercial vendors (e.g., Cray) and as a driver for evaluating new processors (e.g., Intel)
- Snowball effects
  - feature combination ("features attract features")
  - community buy-in ("users attract users")
  - accomplishment ("citations attract citations", "prizes attract prizes")

PETSc checks the boxes

- Users of scalable solvers seek:
  - ☑ Interface standardization
  - ☑ Solver interoperability
  - ☑ Vertical integration
  - ☑ Architecture-adaptive performance
  - ☑ Application-adaptive convergence

Weaknesses: PETSc Panned

- What could PETSc improve?
- What are PETSc's disadvantages?
- What resources does PETSc lack?

Areas to improve today?

- Learning curve
- Naming conventions
- Built-in graphics
- Computational steering
- “Branding”
- Archiving of success stories
  - for more stable support
  - for more bootstrapped successes

Opportunities: PETSc's Prospects

- What trends favor PETSc's future?
- What changes in technology?
- What changes in policy?
- What changes in culture?

Trends favorable to PETSc

- More demanding applications
  - multi-physics, multi-scale, multi-dimensions, multi-models, etc.
- More sophisticated algorithms
  - polyalgorithms, multi-levels, multi-precisions, discretization adaptivity, convergence adaptivity, etc.
- More complex architectures
  - massively distributed, manycore, deeply hierarchical memory, hierarchical coherence domains
- PETSc functionality can expand user aspiration

More trends favorable to PETSc

- Still higher stacking for post-forward problems
  - Sensitivity
  - Validation and verification
  - Uncertainty quantification
  - Optimization
  - Inversion
  - Data Assimilation
  - Analytics
- Combined post-forward problems
  - Optimization under uncertainty
  - Design of experiments
- PETSc functionality can expand user aspiration

Still more trends favorable to PETSc

- With increasing hardware complexity and shrinking government and industrial investment, fewer simulation communities will “roll their own” below the level of the modeling
  - will demand infrastructure that allows higher productivity
- PETSc users exponentially beget future generations of PETSc users

Threats: PETSc's Perils

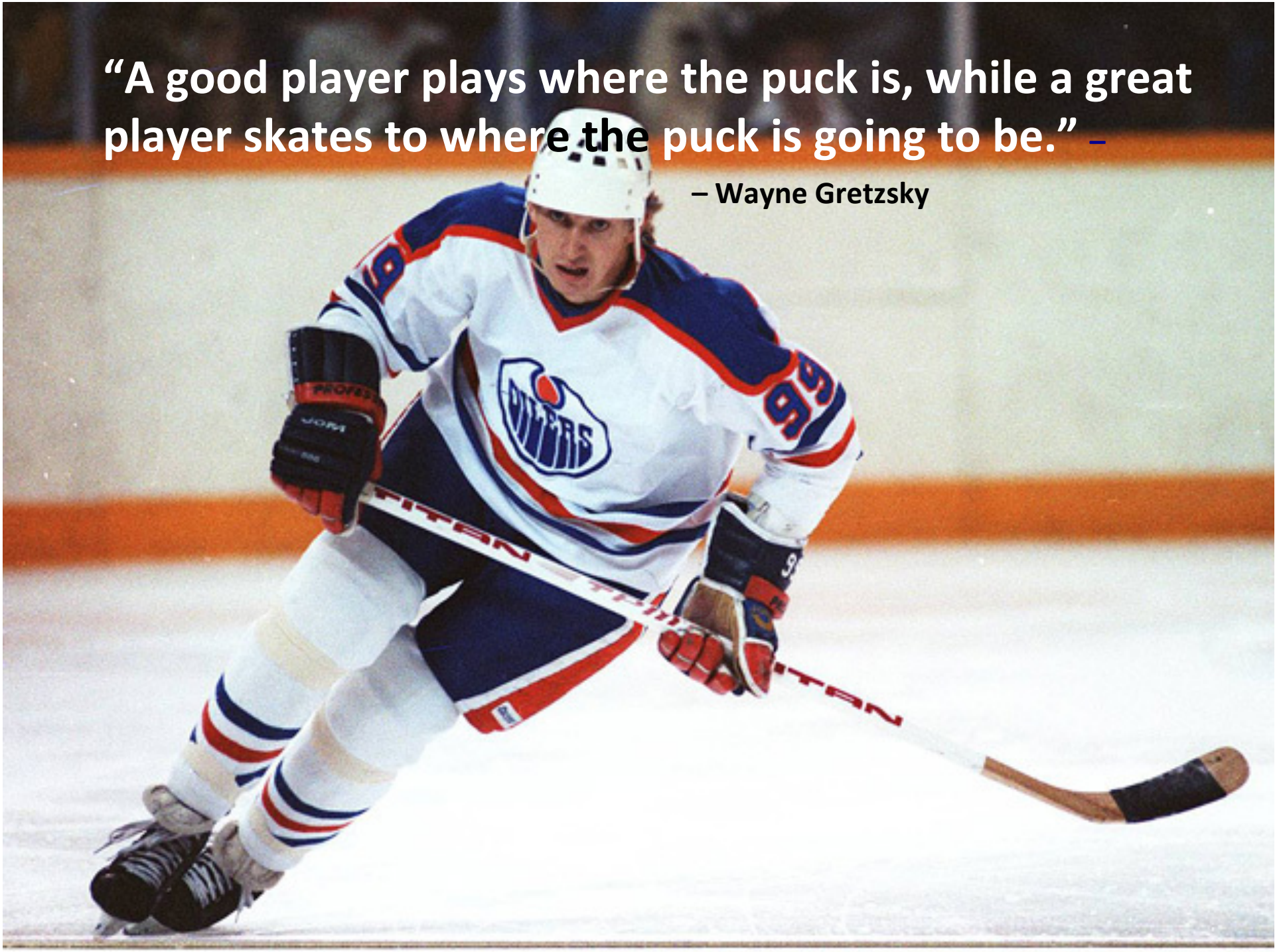
- What obstacles does PETSc face?
- What changes in technology?
- What changes in policy?
- What changes in culture?
- What are "competitors" doing?

Threats to PETSc

- Leaders are targets
  - common to see unfair comparisons against PETSc
- Some needed or wanted improvements will disrupt backward compatibility
- Difficult to attract support simply to maintain research infrastructure
  - increased overhead in raising support distracts from doing the work and inhibits long-term planning and hiring
- Increasing number of external dependencies increase burden on integrators closest to users

Biggest threat to PETSc

- Coming “discontinuity” in hardware and programming models


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


Energy-aware
generation
BSP
generation

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

The International Exascale Software Roadmap, J. Dongarra, P. Beckman, et al., *International Journal of High Performance Computer Applications* **25**(1), 2011, ISSN 1094-3420.

Uptake from IESP meetings

- While obtaining the next 2 orders of performance, we need 1-2 orders more Flop/s per Watt
  - target: 50 Gigaflop/s/W, today less than 5 Gigaflop/s/W
- Draconian reduction required in power per flop and per byte will make computing and moving data less reliable
  - circuit elements will be smaller and subject to greater physical noise per signal, with less space and time redundancy for resilience in the hardware
  - more errors must be caught and corrected in software
- Power may be cycled off and on or clocks slowed and speeded
  - based on compute schedules (user-specified or software adaptive) and dynamic thermal monitoring
  - makes per-node performance rate unreliable

Some exascale architecture themes

- Clock rates cease to increase while arithmetic capability continues to increase dramatically w/ concurrency consistent with Moore's Law
- Memory storage capacity diverges exponentially below arithmetic capacity
- 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 better algorithms arrive to span the architectural gap

Main challenge going forward for BSP

- Almost all "good" algorithms in linear algebra, differential equations, integral equations, signal analysis, etc., require frequent synchronizing global communication
  - inner products, norms, and fresh global 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 3 million on the most powerful system today

Bad news/good news (1)

- One will have to explicitly control more of the data motion
  - carries the highest energy cost in the exascale computational environment
- One finally will get the privilege of controlling the *vertical* data motion
  - *horizontal* data motion under control of users already
  - but vertical replication into caches and registers was (until recently with GPUs) mainly scheduled and laid out by hardware and runtime systems, mostly invisibly to users

Bad news/good news (2)

- "Optimal" formulations and algorithms may lead to poorly proportioned computations for exascale hardware resource balances
  - today's "optimal" methods presume flops are expensive and memory and memory bandwidth are cheap
- Architecture may lure scientific and engineering users into more arithmetically intensive formulations than (mainly) PDEs
  - tomorrow's optimal methods will (by definition) evolve to conserve whatever is expensive

Bad news/good news (3)

- Fully hardware-reliable executions may be regarded as too costly/synchronization-vulnerable
- 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
- Examples already in direct and iterative linear algebra
- Anticipated by Von Neumann, 1956 ("Synthesis of reliable organisms from unreliable components")

Bad news/good news (4)

- Default 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 will have to 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, but also the last values, and forward just the deltas
- Equidistributing errors properly to minimize resource use will lead to innovative error analyses in numerical analysis

Bad news/good news (5)

- 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.

What will first "general purpose" exaflop/s machines look like?

- *Hardware*: many potentially exciting paths beyond today's CMOS silicon-etched logic, but not commercially at scale within the decade
- *Software*: many ideas for general-purpose and domain-specific programming models beyond "MPI + X", but not penetrating the mainstream CS&E workforce for the next few years
  - "X" is CUDA, OpenMP, OpenACC, OpenCL, etc., or MPI, *itself*

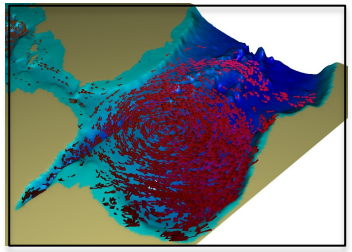
The gap

- Algorithms must adapt to span the gulf between demanding applications and austere architectures
  - full employment program for computational scientists and engineers
  - see, e.g., recent postdoc announcements from
    - Berkeley (8) for Cori Project (Cray & Intel MIC)
    - Oak Ridge (8) for CORAL Project (IBM & NVIDIA NVLink)
    - IBM (10) for Data-Centric Systems initiative

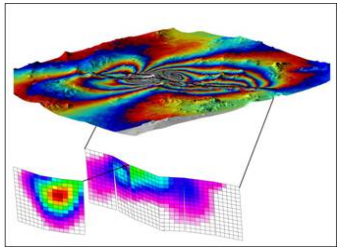
  for porting applications to emerging hybrid architectures

Philosophy of software investment

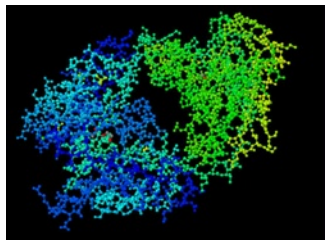
I. Hoteit



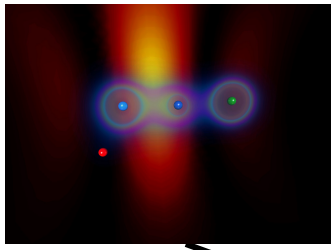
M. Mai



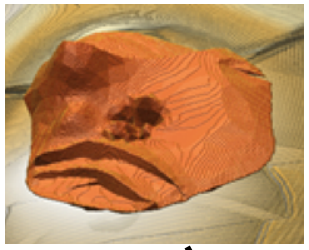
V. Bajic



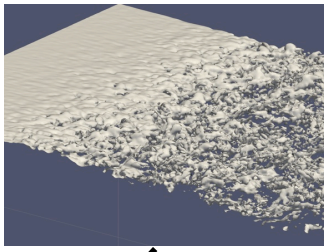
A. Fratalocchi



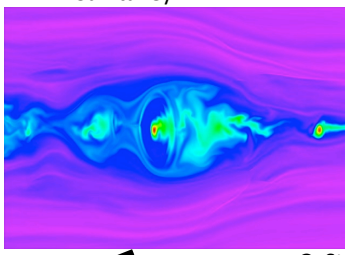
G. Schuster



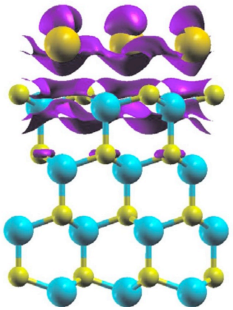
F. Bisetti



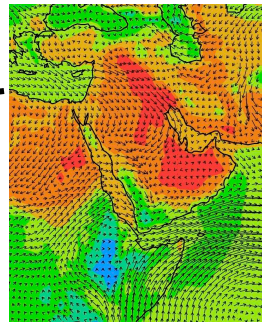
R. Samtaney

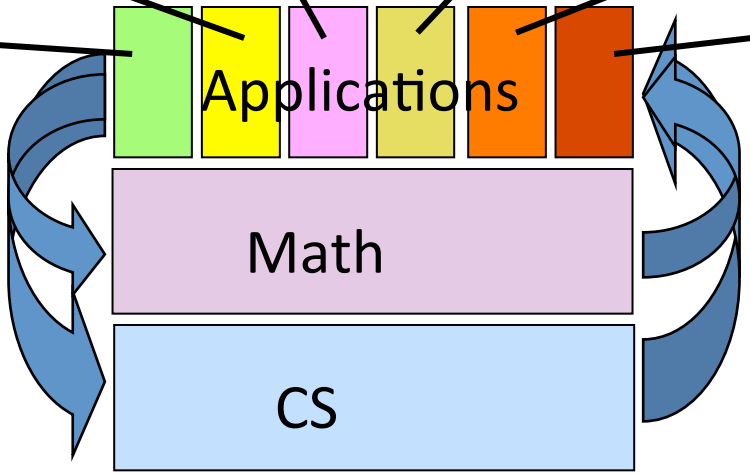


U. Schwingenschloegl



G. Stenchikov





Applications drive

Math & CS enable

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

Production-related

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

High-end computers come with little of this stuff. Most has to be contributed by the user community

Optimal hierarchical algorithms

- At large scale, one must start with algorithms with optimal asymptotic scaling, $O(N \log^p N)$
- 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)

"With great computational power comes great algorithmic responsibility." – Longfei Gao

Algorithmic agenda

- New formulations with
  - greater arithmetic intensity (flops per byte moved into and out of registers and upper cache)
    - including assured accuracy with (adaptively) less floating-point precision
  - reduced synchronization and communication
    - less frequent *and/or* less global
  - greater 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.

Last segment flashes sample "points of light" that accomplish one or more of these agendas

- DAG-based data flow for dense symmetric linear algebra
- GPU implementations of dense symmetric linear algebra
- Fast Multipole for Poisson solves
- Algebraic Fast Multipole for variable coefficient problems
- Nonlinear preconditioning for Newton's method
- New programming paradigms for PDE codes

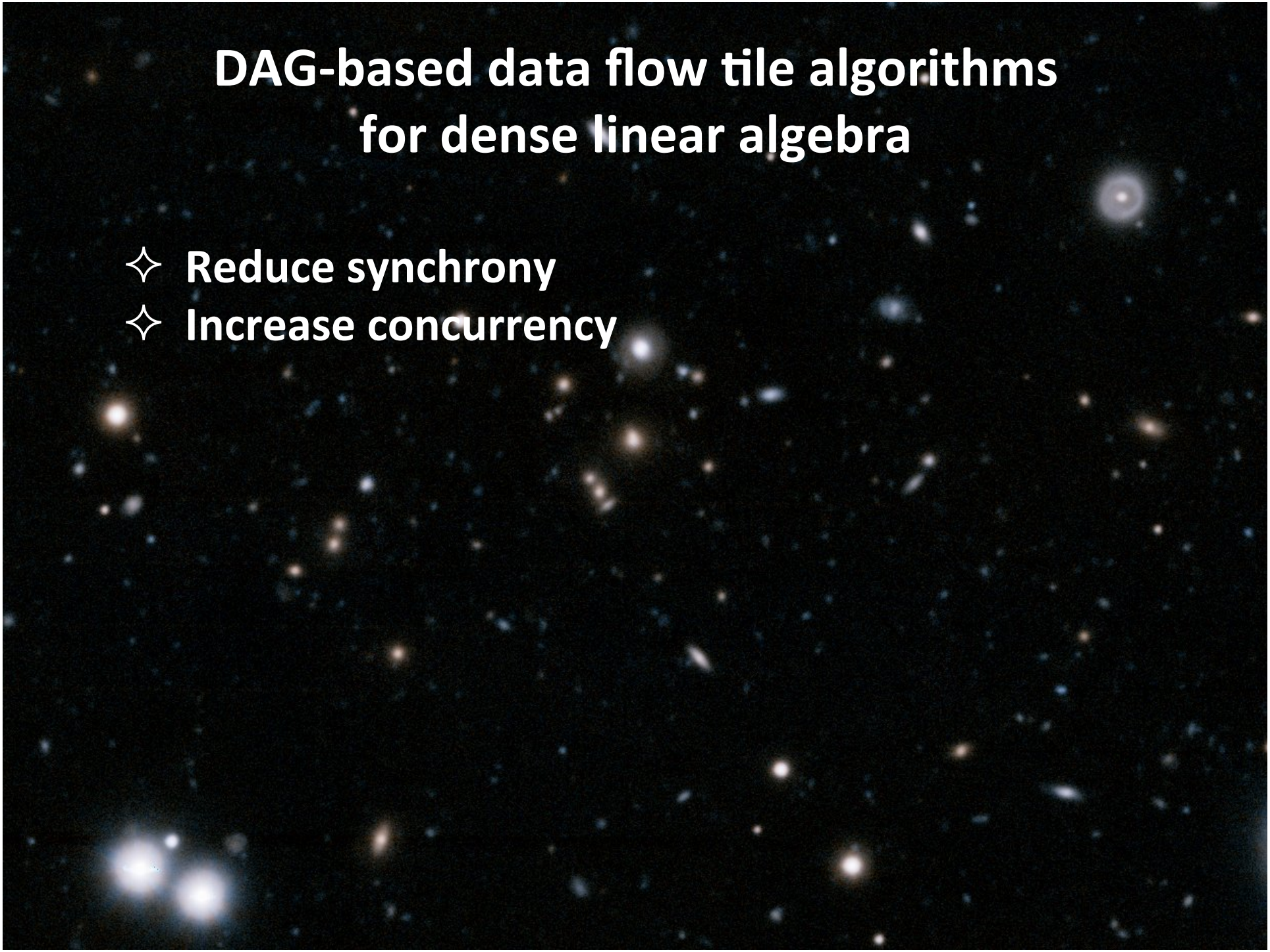
NVIDIA GPU RESEARCH CENTER

intel Intel Parallel Computing Centers

CRAY THE SUPERCOMPUTER COMPANY CENTER OF EXCELLENCE

Motivation

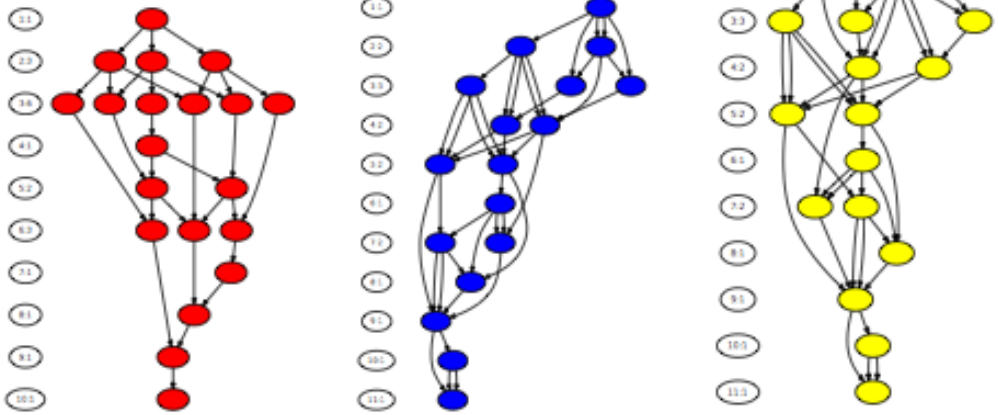
- Dominant use of wall-clock time at supercomputer centers goes to:
  - Poisson and other elliptic solves in molecular dynamics, DFT, CFD, E&M, porous media, etc.
  - Linear algebra on dense Hermitian matrices in Schroedinger, covariance, reduced Hessians, etc.
  - I/O ☹
- These first two are the major thrusts of the ECRC at KAUST
  - And also initiatives like FastMATH for the US DOE, ExaSolvers, ExaDune, etc., for the German DFG, etc.


DAG-based data flow tile algorithms
for dense linear algebra
✧ Reduce synchrony
✧ Increase concurrency

Reducing over-ordering and synchronization through dataflow: generalized eigensolver

$$Ax = \lambda Bx$$

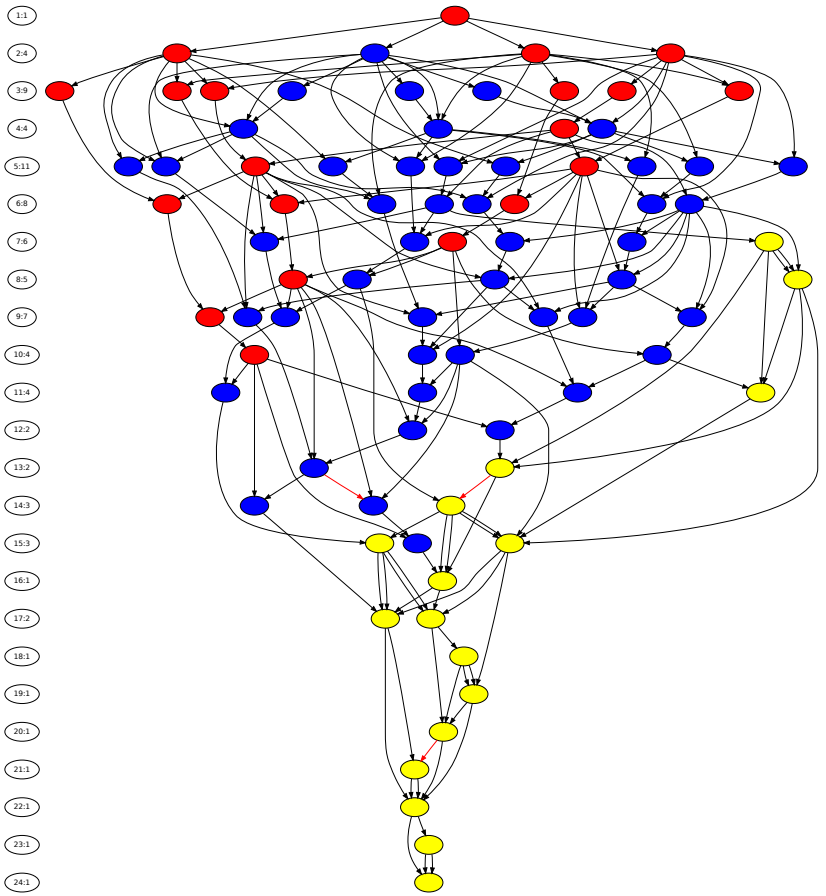
Operation	Explanation	LAPACK routine name
1. $B = L \times L^T$	Cholesky factorization	POTRF
2. $C = L^{-1} \times A \times L^{-T}$ or HEGST	application of triangular factors	SYGST
3. $T = Q^T \times C \times Q$	tridiagonal reduction	SYEVD or HEEVD
4. $Tx = \lambda x$	QR iteration	STERF



c/o H. Ltaief (KAUST)

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

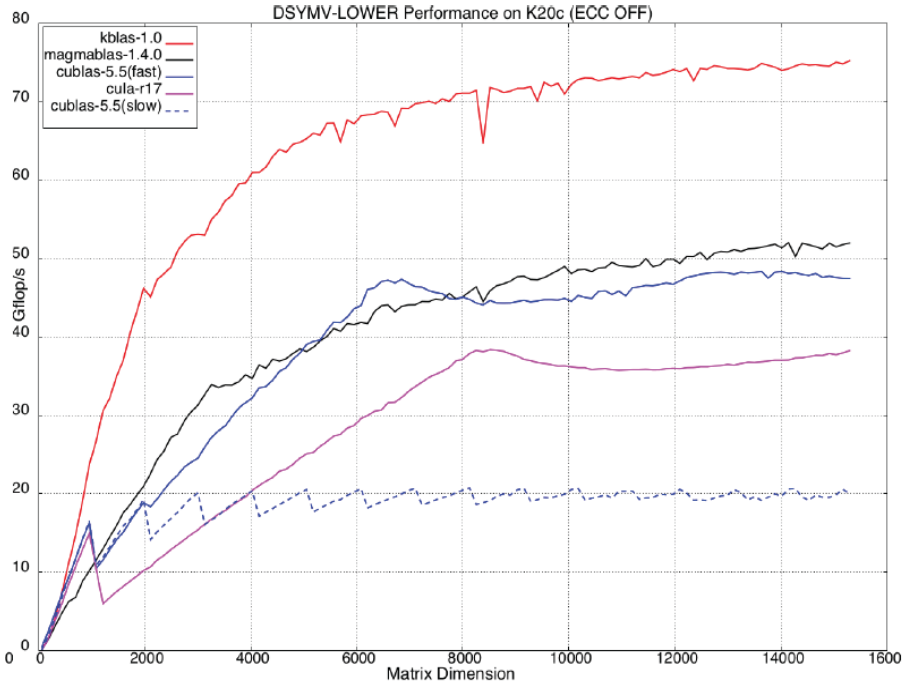


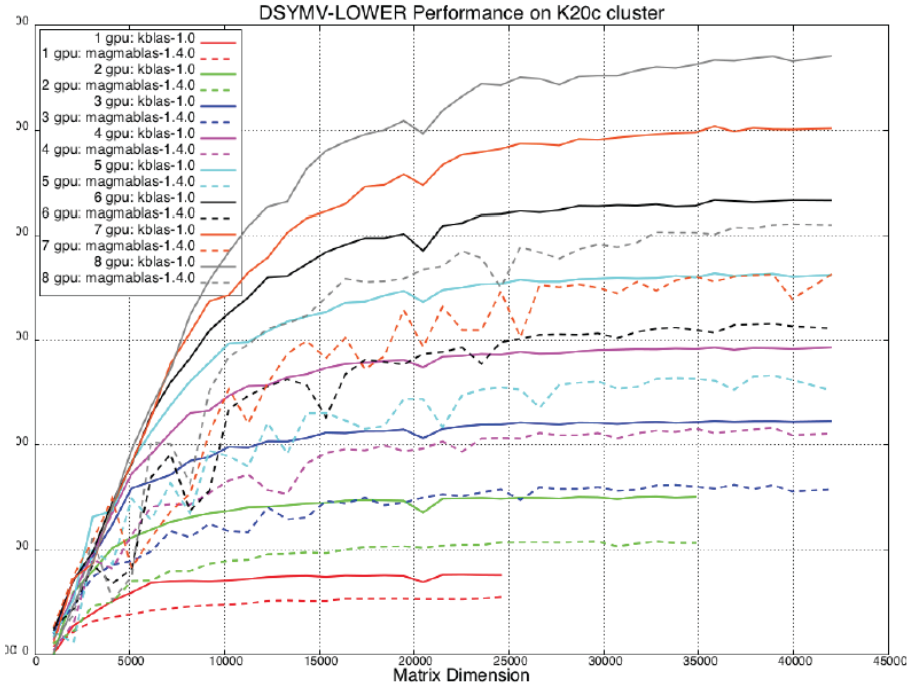
c/o H. Ltaief (KAUST)

GPU implementations of dense linear algebra

- Increase SIMD-style thread concurrency
  - overcome memory bandwidth limitations of the matrix-vector multiply, $y = \alpha A x + \beta y$
  - coalesced memory accesses
  - double buffering
  - polyalgorithmic approach based on block size

New linear algebra software, KAUST's GPU BLAS, now in NVIDIA's CUBLAS



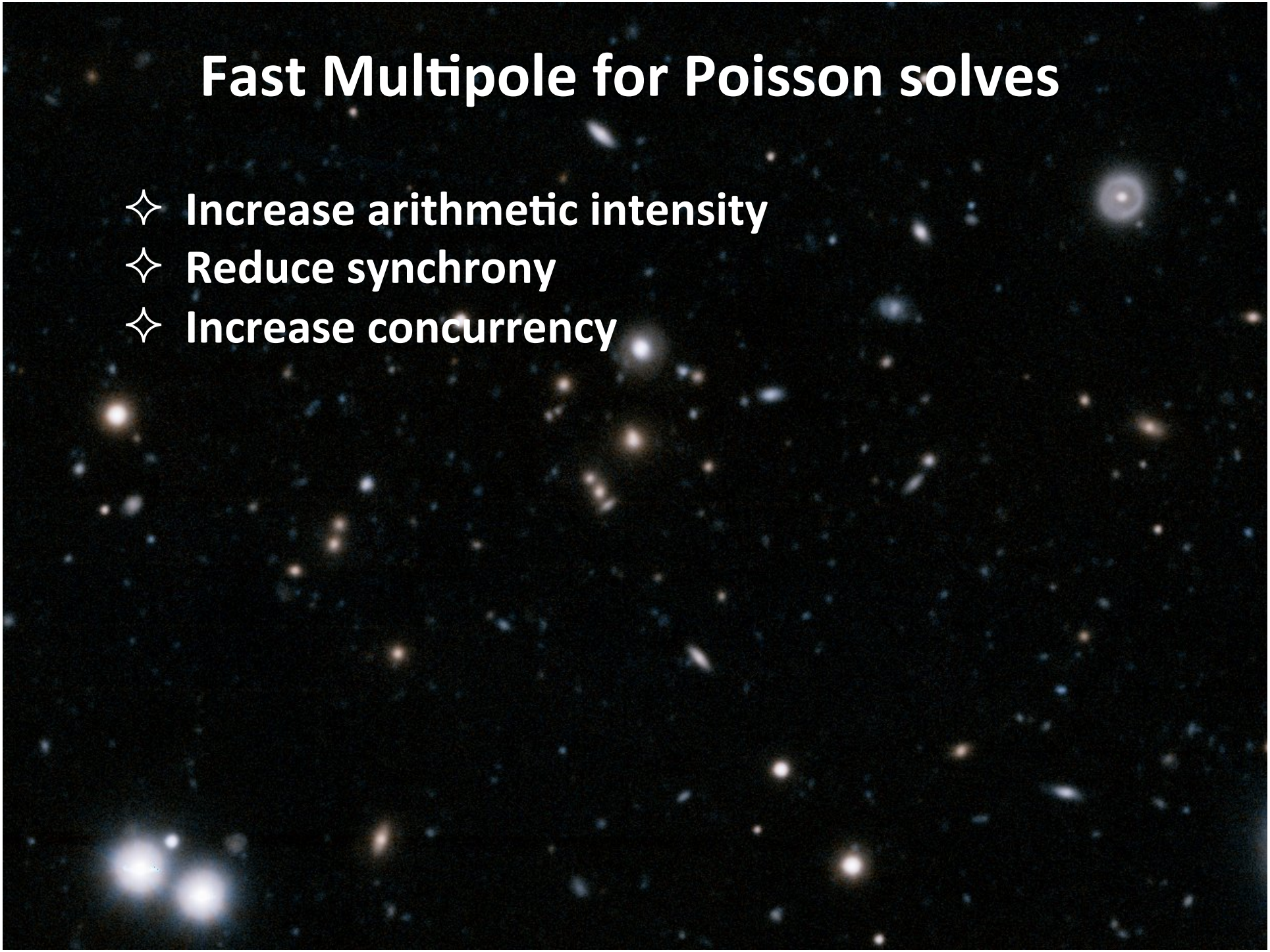


- Highly optimized GEMV/SYMV kernels
- NVIDIA has adopted for its CUBLAS 6.0 library

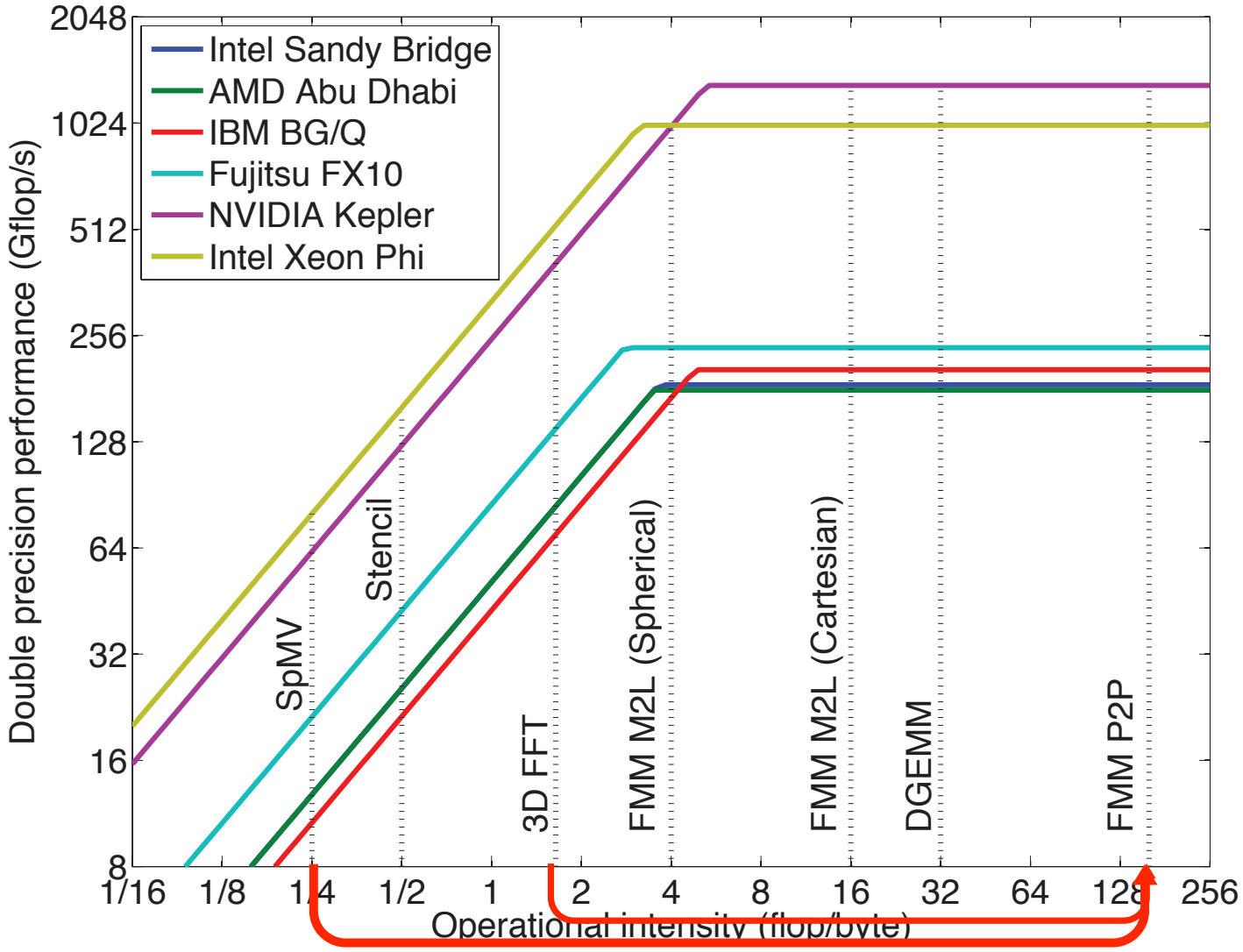
c/o A. Abdelfattah (UTenn ICL, KAUST)

Fast Multipole for Poisson solves

- ✧ Increase arithmetic intensity
- ✧ Reduce synchrony
- ✧ Increase concurrency



Arithmetic intensity of numerical kernels

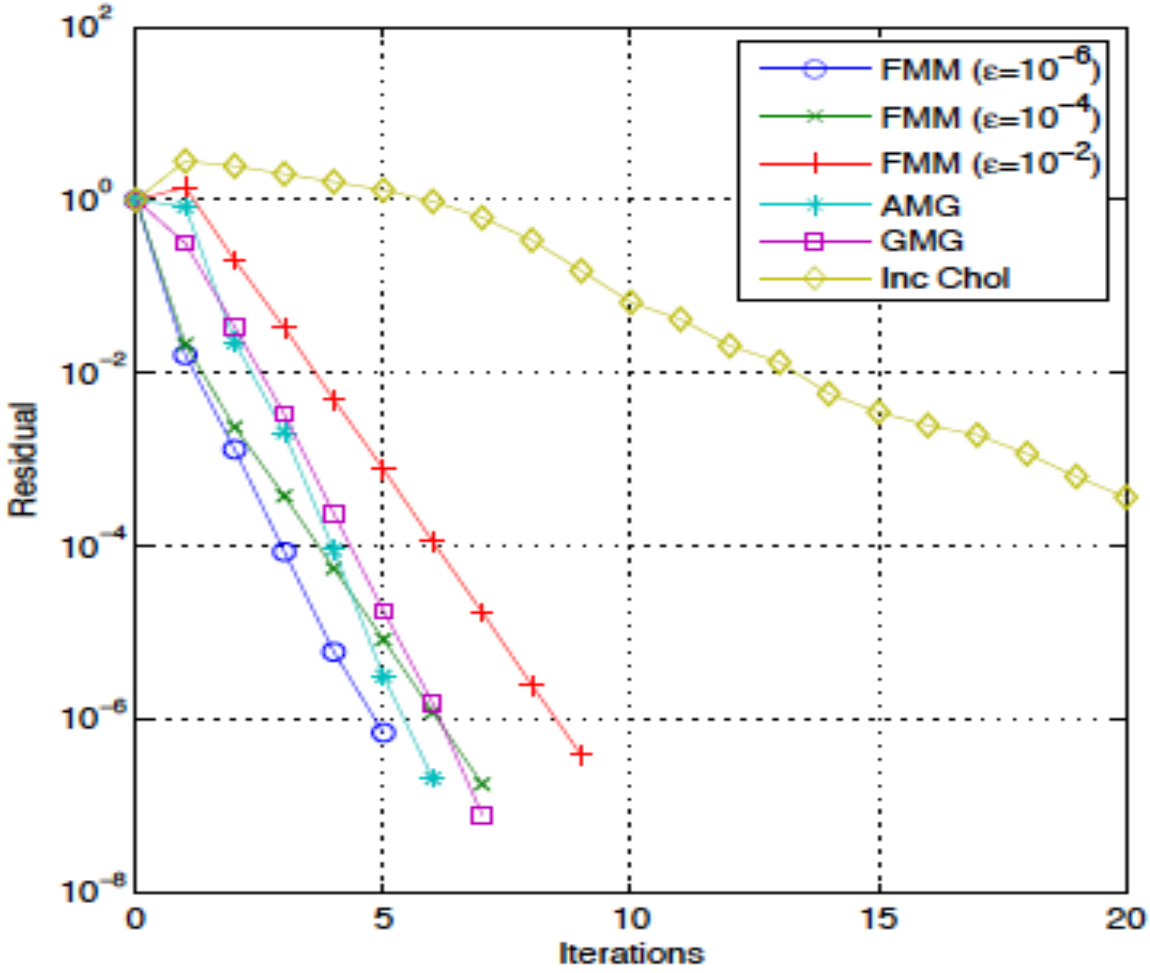


c/o R. Yokota (TiTech, KAUST)

FMM as preconditioner

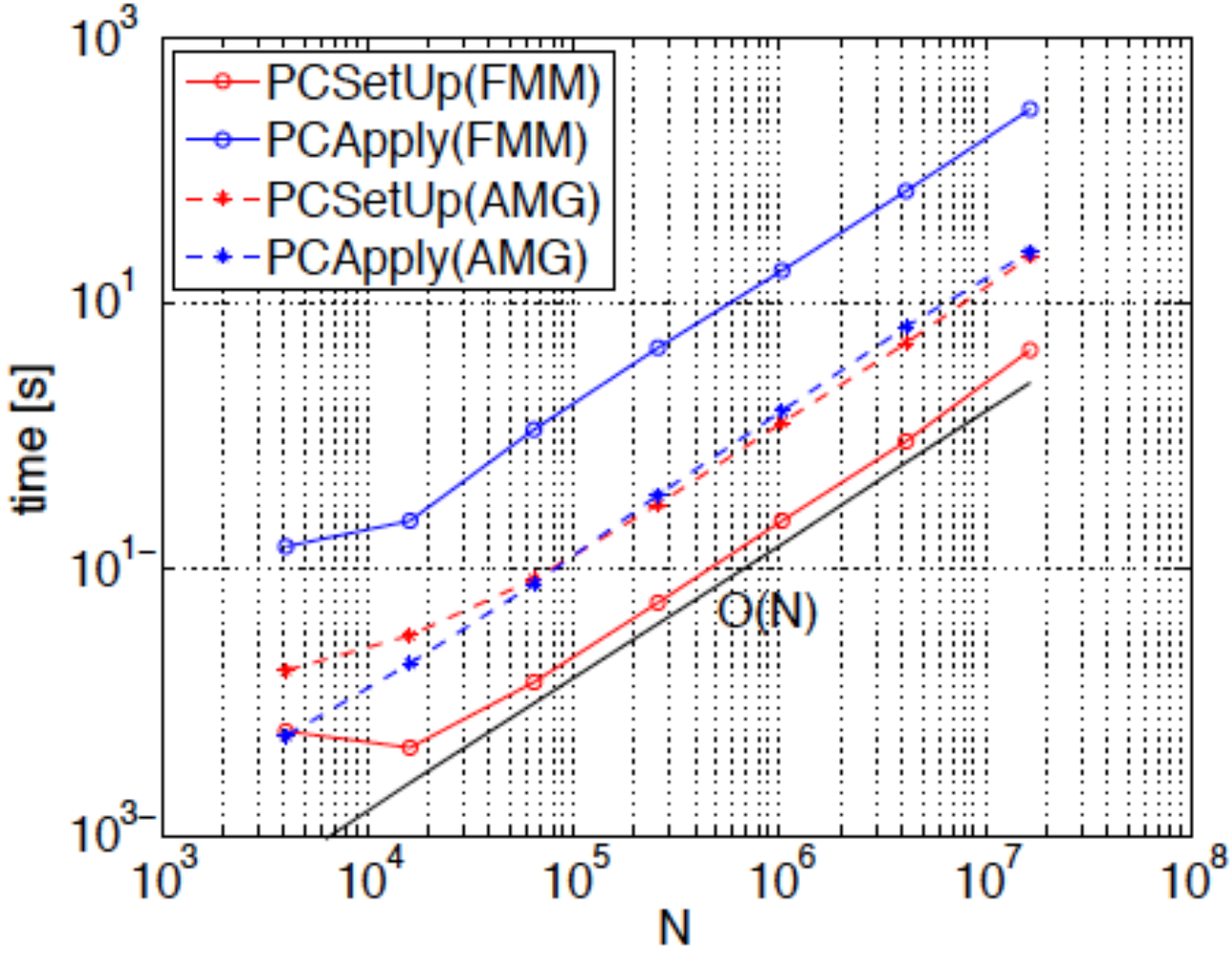
- FMM is a solver for free-space problems for which one has a Green's function
- For finite boundaries, FMM combines with BEM
- FMM and BEM have controllable truncation accuracies; can precondition other, different discretizations of the same PDE
- Can be regarded as a preconditioner for "nearby" problems,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

FMM/BEM preconditioning of FEM-discretized Poisson accelerated by CG



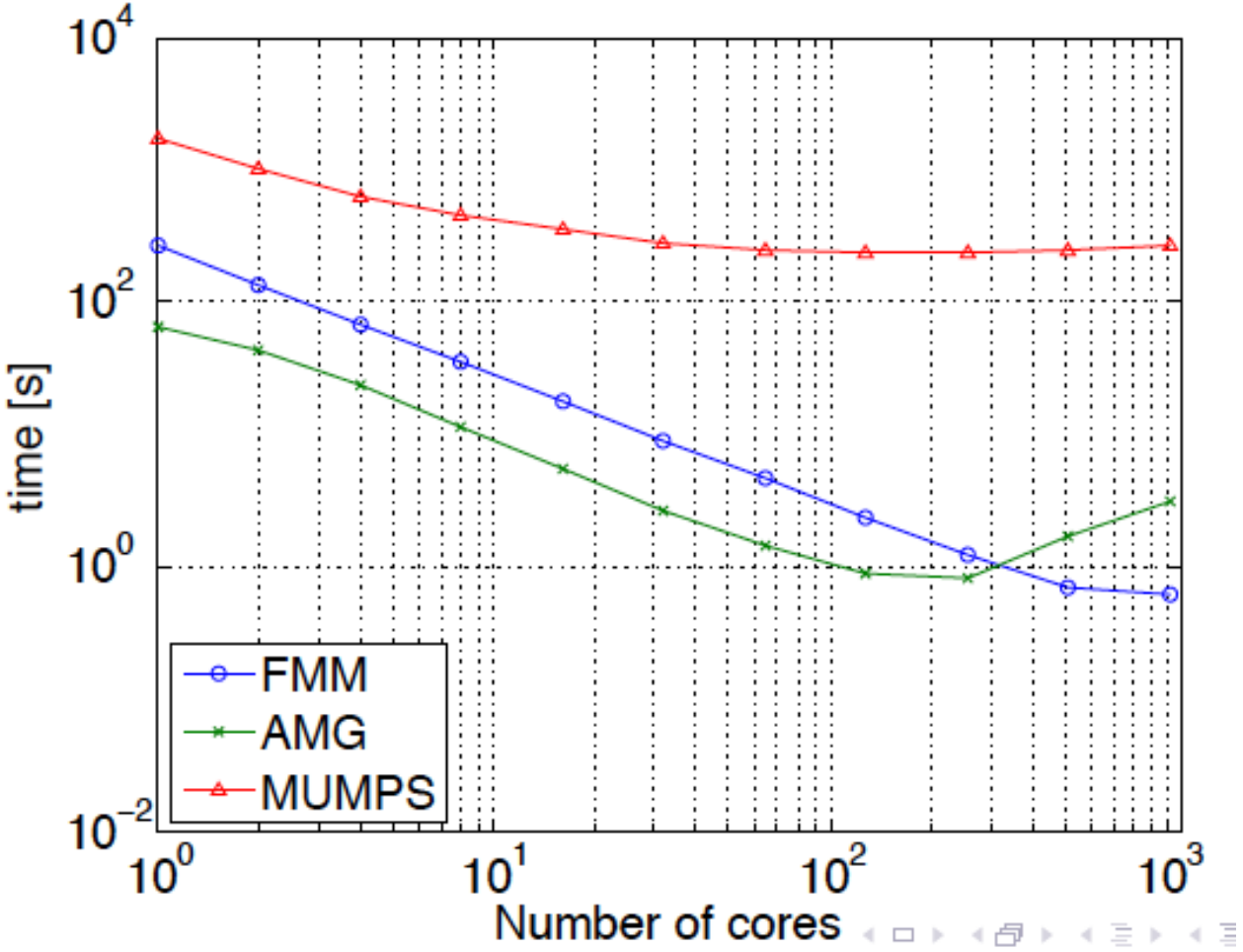
c/o H. Ibeid (KAUST)

FMM/BEM preconditioning of FEM-discretized Poisson: serial scaling



c/o H. Ibeid (KAUST)

FMM vs AMG preconditioning: strong scaling on Stampede*



16M dofs FEM Poisson problem, Dirichlet BCs via BEM (cost included)

c/o H. Ibeid (KAUST)

Algebraic Fast Multipole for variable coefficient problems

- ✧ **All the benefits of Fast Multipole**

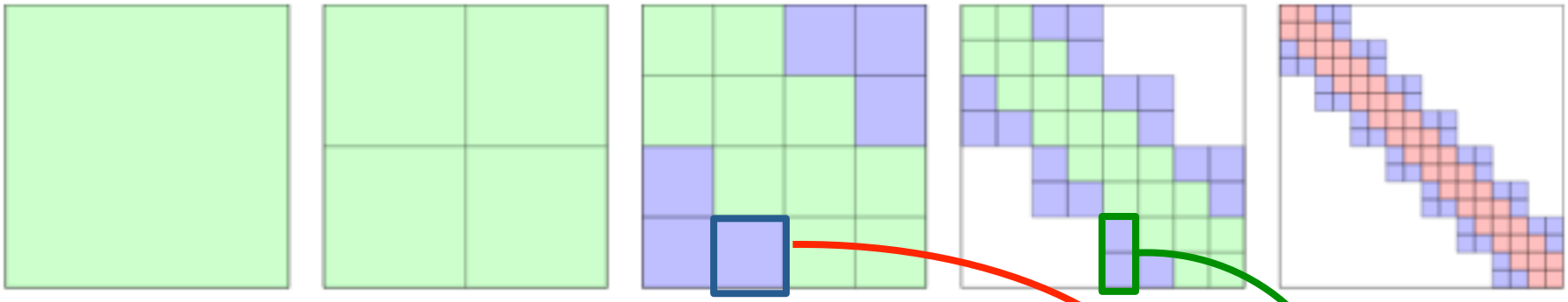
plus

- ✧ **Make Fast Multipole less fragile**

Part of the Intel-sponsored “HiCMA” project
Hierarchical Computations on Many Core Architectures

Is there an algebraic FMM?

- Consider the H^2 hierarchical matrix method of Hackbusch, *et al.*

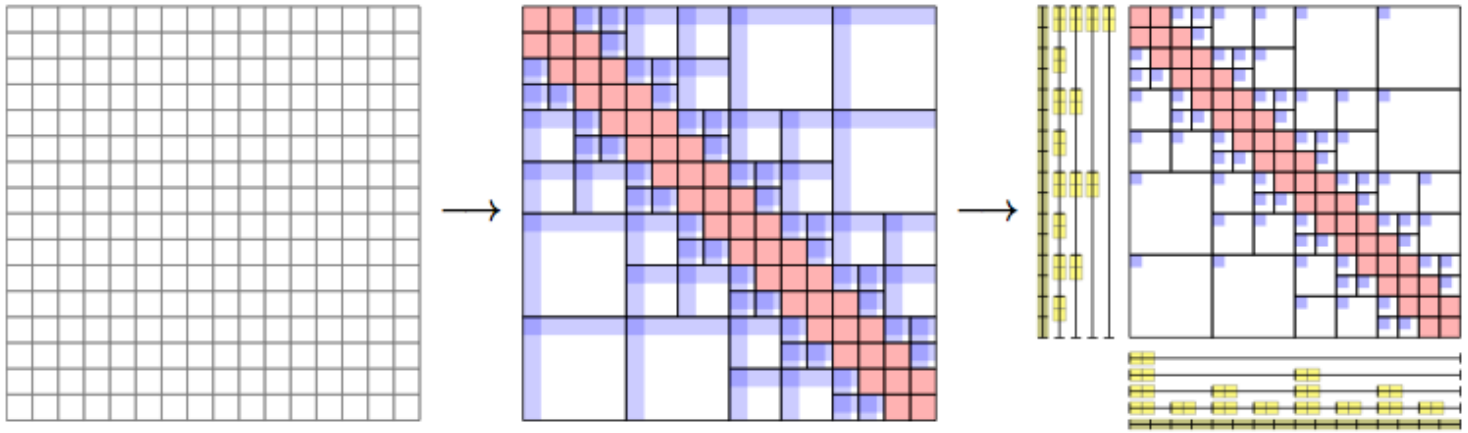


- Off diagonal blocks $A_{ij} \cong U_i S_{ij} V_j$ can have low rank, based on an "admissibility condition"
- Bases can be hierarchically nested
  - U_i for columns, V_j for rows

c/o G. Turkiyyah (KAUST)

Is there an algebraic FMM?

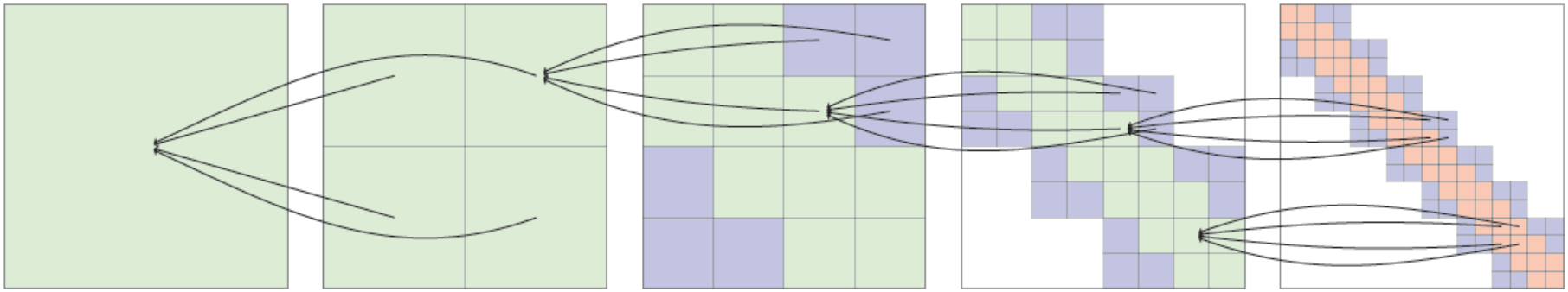
- One needs to store the unreducible diagonal blocks, A_{ii}
- For the entire rest of the matrix, first the S_{ij} , the U_i and V_j at the finest level
- Then the E_{ij} (column basis conversion) and F_{ij} (row basis conversion) blocks at each level
- Two stage compression procedure: SVD each block, then convert to common bases



c/o G. Turkiyyah (KAUST)

“Algebraic Fast Multipole” (AFM)

- Can we cast general matrix operations (add, multiply, invert, etc.) in terms of the fast multipole recursive “tree-based” data structure?

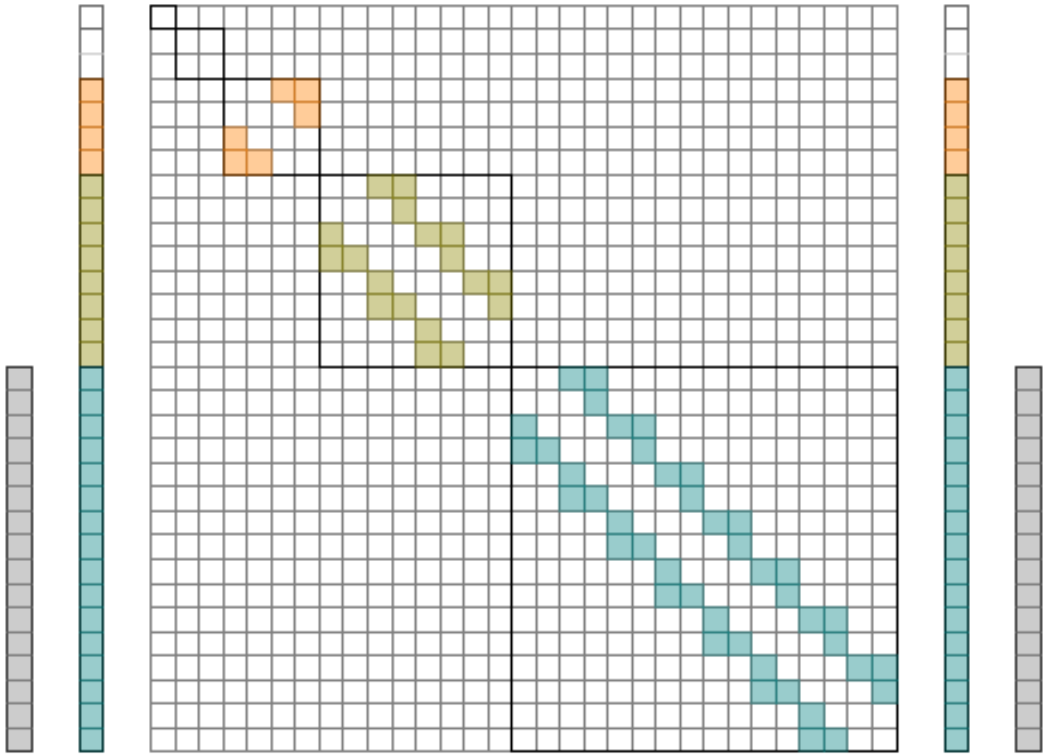


- Yes, after compressing the matrix in H^2 form
  - presumes hierarchical low rank structure
  - may offer breakthrough in application performance
  - See *Supercomput. Front. Innov*. 1:62-83 (2014)

c/o G. Turkiyyah (KAUST)

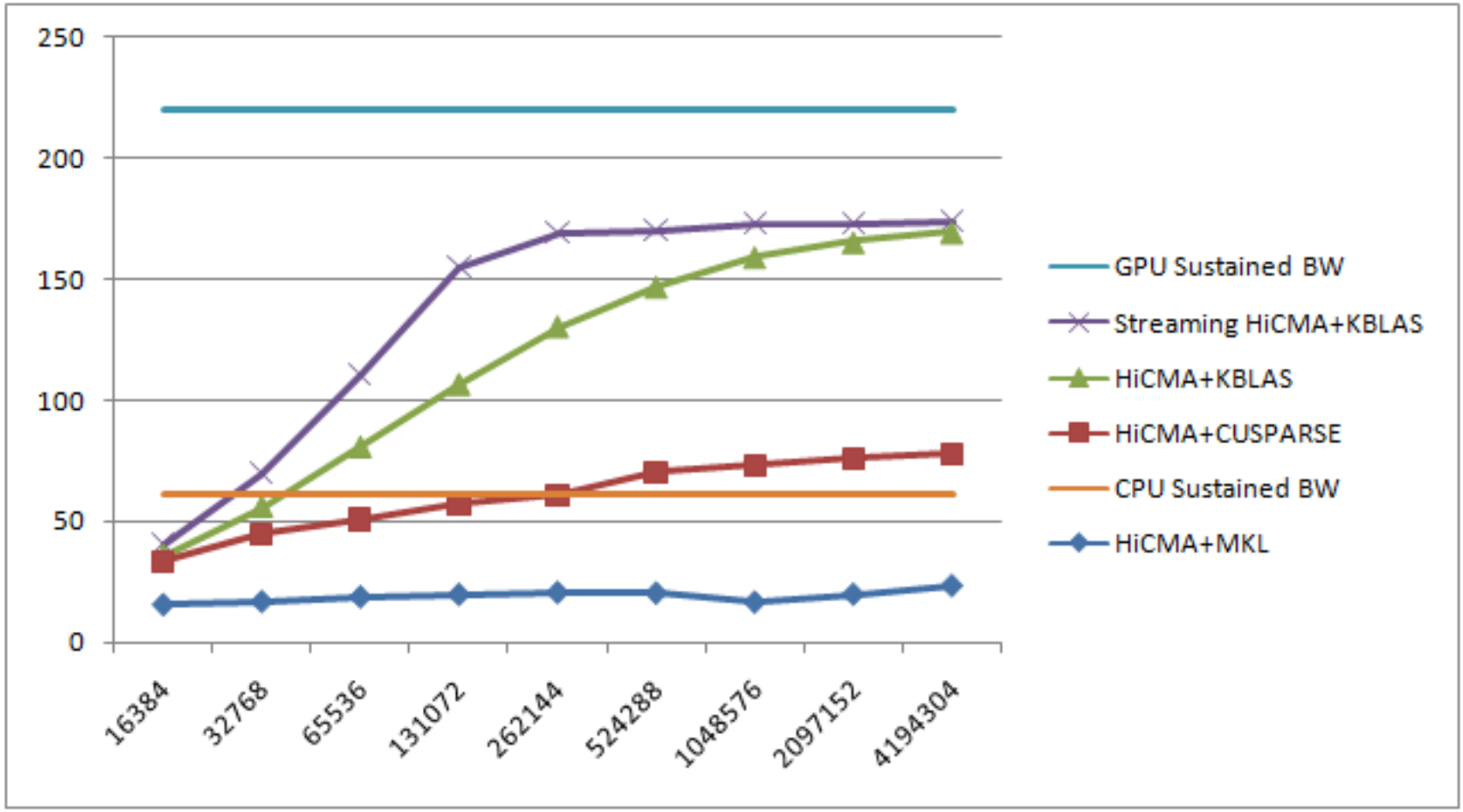
Fast matrix-vector multiply, $y = Ax$

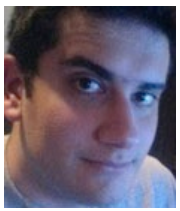
$$y = \left(\sum_{(i,j)\in D} A_{ij} \right) x + \left(\sum_{(i,j)\in L} U_i S_{ij} V_j^t \right) x = \underbrace{\sum_{(i,j)\in D} A_{ij} x_j}_{\text{Dense mat-vecs operations}} + \underbrace{\sum_{i\in I} U_i \underbrace{\sum_{(i,j)\in L} S_{ij} \underbrace{V_j^t x}_{\text{Upsweep}}}_{\text{Coupling phase}}}_{\text{Downsweep}}$$



c/o W. Bukharam (KAUST)

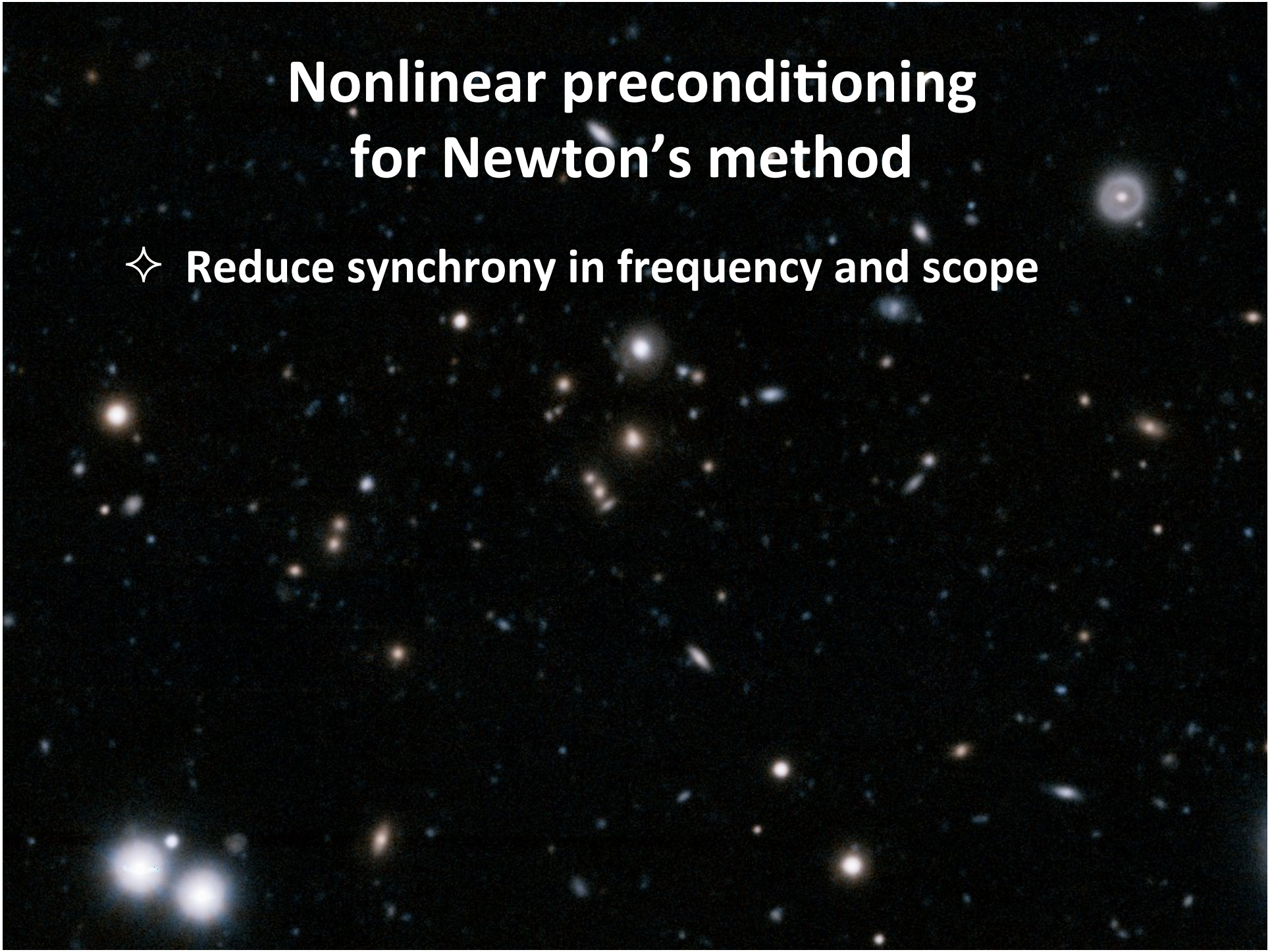
Fast matrix-vector multiply, $y = Ax$





achieved bandwidth (GB/s) versus matrix dimension for rank k=8 and leaf size n=32, integral equation kernel

c/o W. Bukharam (KAUST)


Nonlinear preconditioning
for Newton's method
✧ Reduce synchrony in frequency and scope

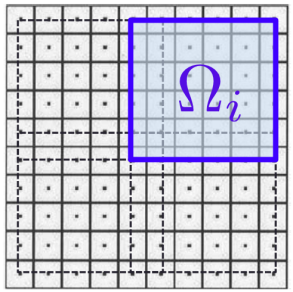
Implemented in PETSc, as “ASPIN”

Key idea

Finding the solution u^* by solving an equivalent nonlinear system

$$\mathcal{F}(u^*) = 0 \Leftrightarrow F(u^*) = 0$$

using Inexact Newton with Backtracking



How to construct the equivalent nonlinear system?

$$F_{\Omega_i}(u - T_{\Omega_i}(u)) = 0, \qquad i = 1, \ldots, N$$

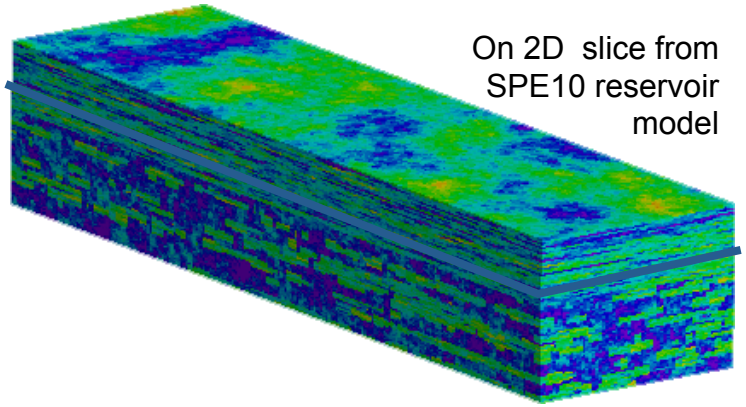
$$\mathcal{F}(u) = \sum_{i=1}^{N} T_{\Omega_i}(u) \qquad \bigcup_{i=1}^{N} \Omega_i = \Omega$$

Assumption

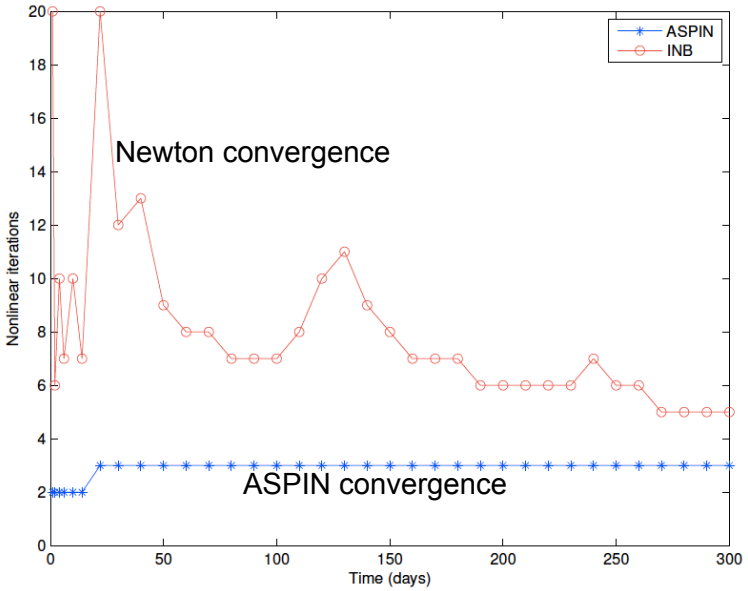
$F'(u)$ is continuous in a neighborhood D of the exact solution u^*, and the matrix $F'(u^*)$ is nonsingular.

Theorem

(Cai and Keyes, 2002). $F(u)$ and $\mathcal{F}(u)$ are equivalent in the sense that they have the same solution in a neighborhood of u^* in D.



On 2D slice from SPE10 reservoir model



c/o L. Liu (KAUST)

Other examples being developed at the Extreme Computing Research Center at KAUST

- ACR, a new spin on 45-year-old cyclic reduction that recursively uses $\mathcal{H}$ matrices on Schur complements to reduce $O(N^2)$ complexity to $O(N \log^2 N)$
- BDDC, a preconditioner well suited for high-contrast elliptic problems that trades lots of local flops for low iteration count, now in PETSc
- QDWH-SVD, a 2-year-old SVD algorithm that performs more flops but generates essentially arbitrary amounts of dynamically schedulable concurrency, and beats state-of-the-art on GPUs
- MWD, a multicore wavefront diamond-tiling stencil evaluation library that reduces memory bandwidth pressure on multicore processors

How will PDE computations 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
  - e.g., ADLB, Charm++, Quark, StarPU, OmpSs, Parallex, Argo
- Noncritical parts will be made available for NUMA-aware work-stealing in economically sized chunks

Adaptation to 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; 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 mining

Trends according to Pete Beckman, Argonne

Trending Up	Trending Down
Asynchrony, Latency Hiding	Block synchronous
Over Decomp & Load Balancing	Static partitioning per core
Massive Parallelism	Countable parallelism
Reduced RAM per Flop	Whole-socket shared memory
Software-managed memory	Simple NUMA
Expensive Data Movement	Expensive flops
Fault / Resilience Strategies	Pure checkpoint/restart
Low BW to Storage, in-situ analysis	Save all

c/o P. Beckman (Argonne)

Algorithmic trends

Trending Up	Trending Down
User-controlled data replication	System-controlled data replication
User-controlled error handling	System-controlled error handling
Adaptive variable precision	Default high precision
Computing with "deltas"	Computing directly with QoI
High order discretizations	Low order discretizations
Exploitation of low rank	Default full rank

An algorithmic theme: defeat the "curses" of dimensionality and multiple scales with the "blessings" of continuity and low rank

Shaheen II

KAUST Supercomputing Lab

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Shaheen II Specs

- 36 cabinets of Cray XC40 with Intel Haswell 2.3 Ghz with 16 cores
- 128 GB of RAM per node
- Number of nodes: 6192
- Number of cores: 198144
- Peak Performance: 7.3 PFlops/s
- LINPACK : 5.6 PFlops/s
- 2.8 MW at peak
- 17.4 PB of Parallel File System
- I/O throughput: over 500 GB/s
- Burst Buffer capacity: 1.5 TB
- Burst Buffer throughput: over 1.2 TB/s

If ranked on Nov'14 lists:
#7 on HPL
#5 on HPCG

~2 GF/s/W

“Quality software renormalizes the difficulty of doing computation.”

– Peter Lax


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