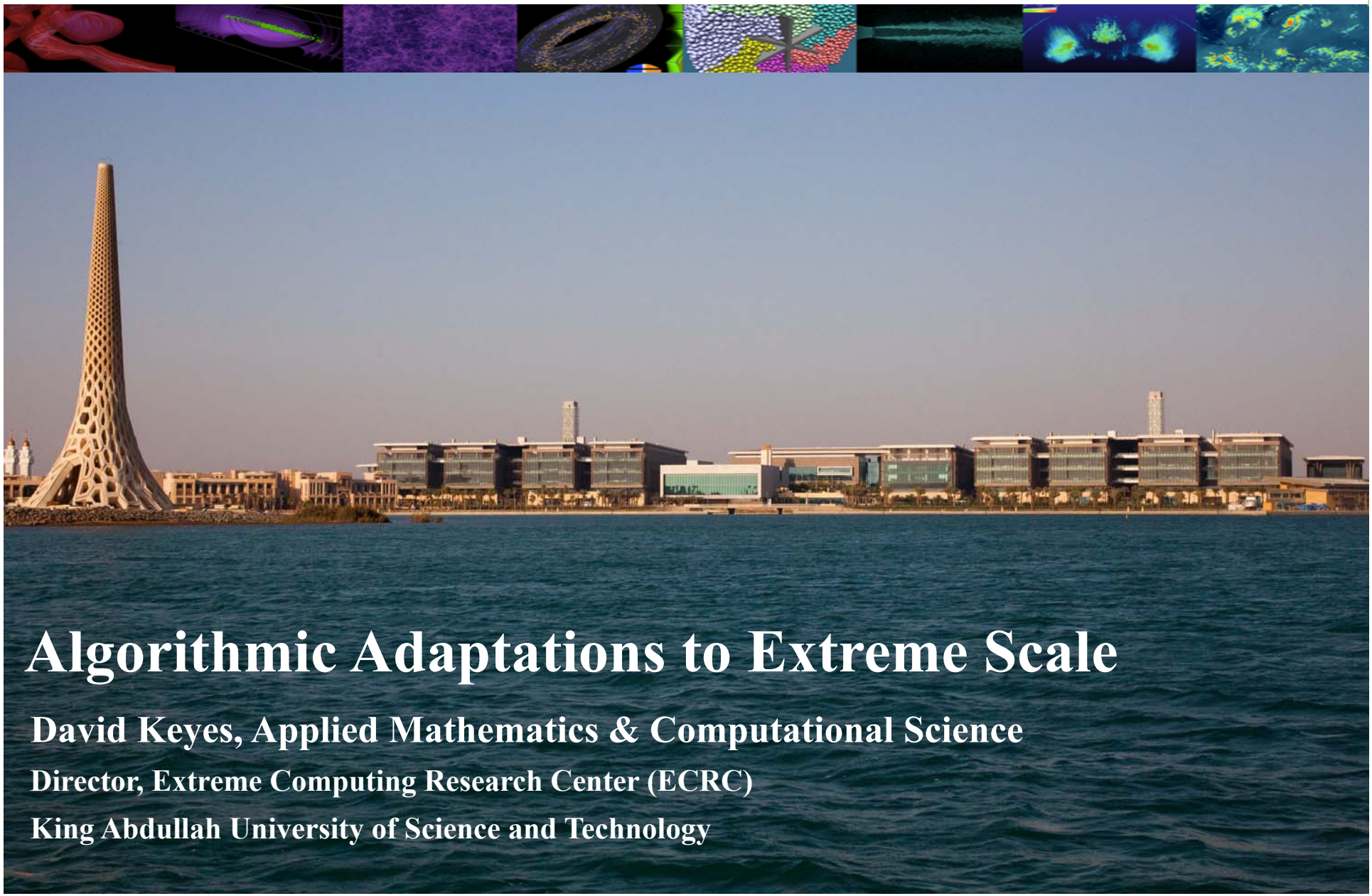
ATPESC 2015

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING



Algorithmic Adaptations to Extreme Scale

David Keyes, Applied Mathematics & Computational Science

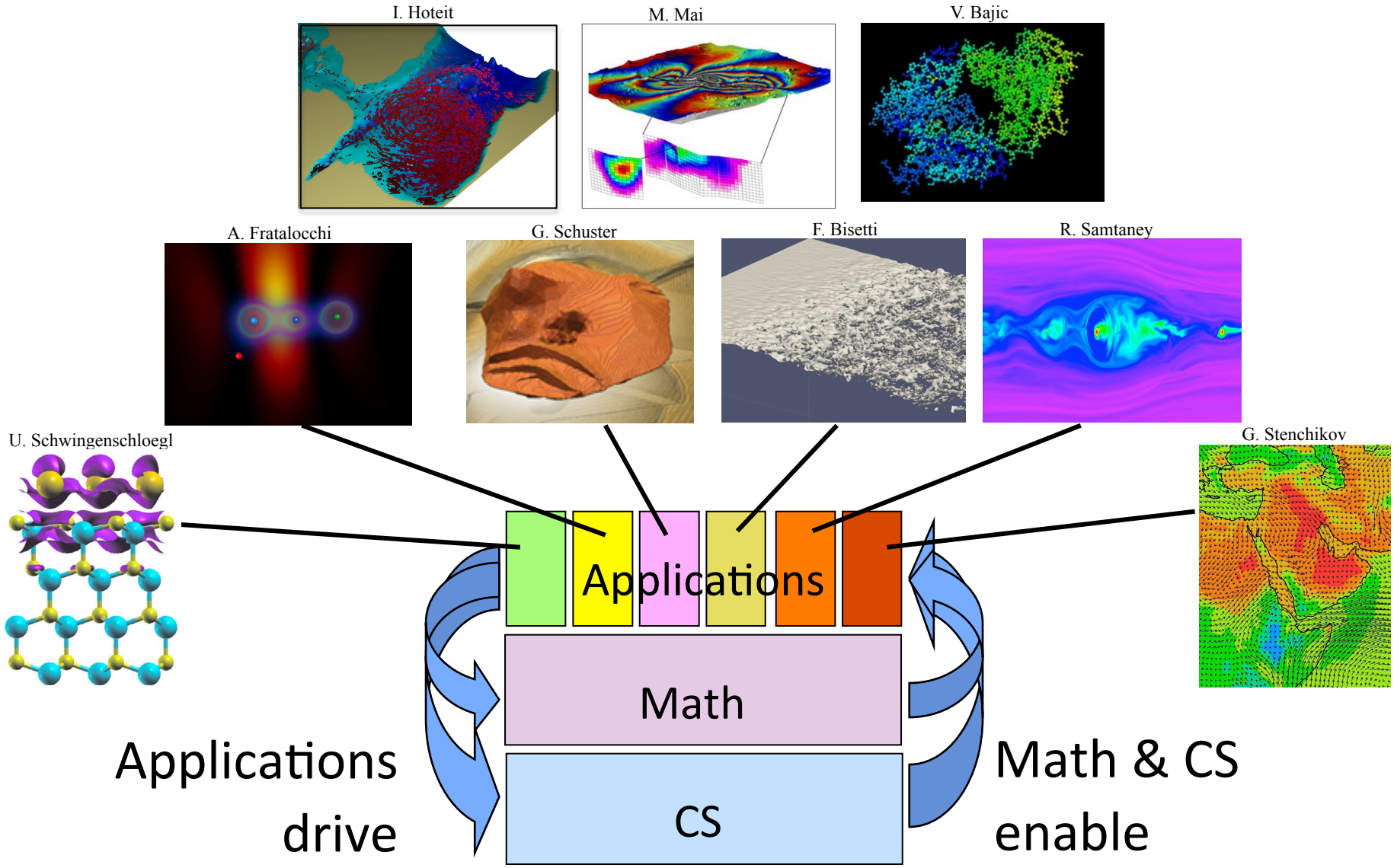
Director, Extreme Computing Research Center (ECRC)

King Abdullah University of Science and Technology

Tie-ins to other ATPESC'15 presentations

- **Numerous!**
  - **architecture, applications, algorithms, programming models & systems software form an *interconnected ecosystem***
  - **algorithms/software span diverging requirements in architecture (more uniformity) & application (more irregularity)**
- **Architecture presentations today:**
  - **Beckman, Balaji**
- **Programming models tonight through Thursday:**
  - **Intel, NVIDIA, MPI, OpenMP, Open ACC, OCCA, Chapel, Charm++, UPC++, ADLB**
- **Algorithms Friday and Saturday:**
  - **Demmel, Dongarra, FASTMath team**

Philosophy of software investment




جامعة الملك عبد الله
للعلوم والتقنية
King Abdullah
University of Science...
Discovery
Square
Economic Development
and Research Park

Society for Industrial and Applied Mathematics

SIAM NEWS >

CSE 2009: The World's First CSE University

June 15, 2009

The King Abdullah University of Science and Technology, scheduled to welcome its first class of students in September, sponsored a reception in Miami on March 2, the first day of the SIAM Conference on Computational Science and Engineering. David Keyes and Omar Ghattas, involved in different ways in the new venture, hosted the reception and made informal presentations to the assembled crowd.

Most readers will know something of KAUST, which for the record is a graduate-only (master's and doctoral) university being constructed in Saudi Arabia, on the eastern edge of the Red Sea, not far from Jeddah. Keyes, the inaugural chair of KAUST's Mathematical and Computer Sciences and Engineering Division, offered examples of research areas of particular interest to Saudi Arabia and the region that will be emphasized; among them are geophysics, seismology, reservoir modeling, CO2 sequestration, photovoltaics, stress-tolerant agriculture, desalination, catalysis, and materials, along with the applied mathematics and computer science required to support them.

Sizeable recruitment ads for KAUST have appeared in many recent issues of *SIAM News*, often side by side with ads placed by partners of the new university, such as the

Shaheen II

KAUST Supercomputing Lab

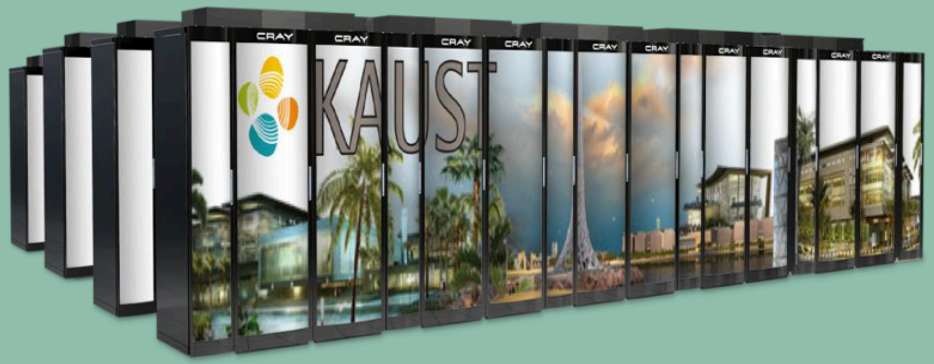
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

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

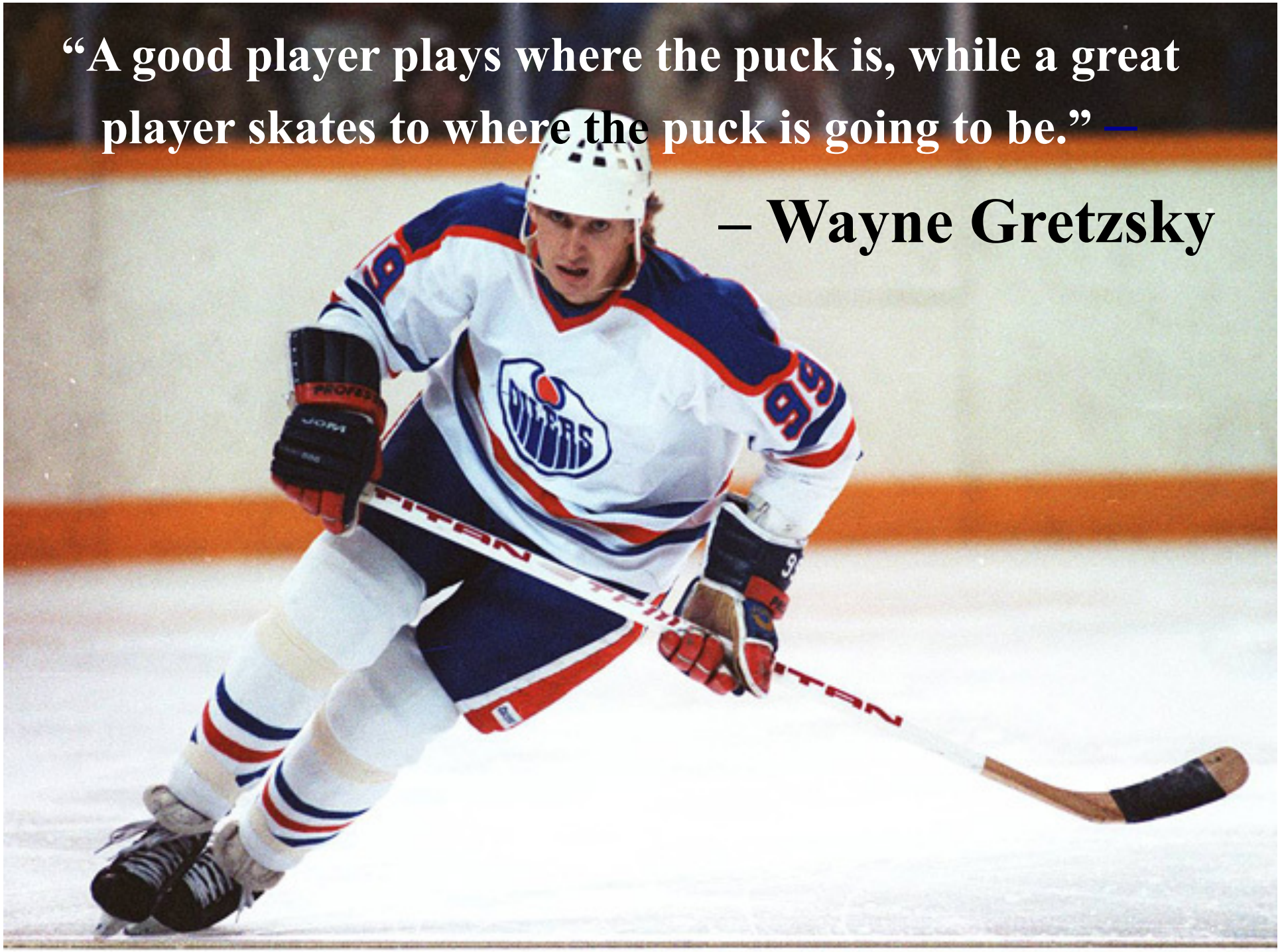
Ranked in Jul'15 lists:
#7 on HPL
#7 on HPCG

~2 GF/s/W



Shaheen I → Shaheen II

IBM Blue Gene/P	Cray XC40
June 2009	**May 2015**
Speed: **.222 Petaflop/s (peak)** Ranking: **#14 HPL R_{max} then (#335 now)**	Speed: **7.3 Petaflop/s (peak, ↑ ~33X)** Ranking**: #7 HPL R_{max} now**
Power: **0.5 MW (0.44 GF/s/W)** Cooling: **air**	Power: **2.8 MW (~2 GF/s/Watt, ↑ ~5X)** Cooling: **water**
Memory: **65 TeraBytes** Amdahl-Case Ratio: **0.29 B/F/s**	Memory: **793 TeraBytes (↑ ~12X)** Amdahl-Case Ratio: **0.11 B/F/s (↓ ~3X)**
I/O bandwidth: **25 GB/s** Storage: **2.7 PetaBytes**	I/O bandwidth: **500 GB/s (↑ ~20X)** Storage: **17.6 PetaBytes (↑ ~6.5X)**
Nodes: **16,384** Cores: **65,536 at 0.85 Ghz**	Nodes: **6,192** Cores: **198,144 at 2.3 Ghz**
Burst buffer: **none**	Burst buffer: **1.5 Petabytes, 1.2 TB/s bandwidth**


"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 their sweet spots.

Examples being developed at the Extreme Computing Research Center (ECRC)

- **ACR(ε), 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($N \log^2 N$)**
- **FMM(ε), a 30-year-old O(N) solver for potential problems with good asymptotic complexity but a bad constant when used at high accuracy, used in low accuracy as a FEM preconditioner**
- **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**
- **BDDC, a preconditioner well suited for high-contrast elliptic problems that trades lots of local flops for low iteration count, now in PETSc**
- **MSPIN, a new nonlinear preconditioner that replaces most of the global synchronizations of Newton iteration with local problems**

Background of this talk:

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

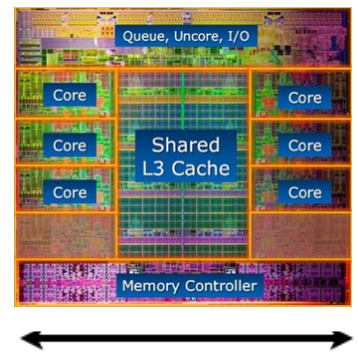
- **While obtaining the next 2 orders of performance, we need 1-2 orders more Flop/s per Watt**
  - **target: 50 Gigaflop/s/W, today less than 5 Gigaflop/s/W**
- **Draconian reduction required in power per flop and per byte will make computing and moving data less reliable**
  - **circuit elements will 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**

Why exa- is different

Which steps of FMADD take more energy?

64-bit floating-point fused multiply add **or** moving four 64-bit operands 20 mm across the die

```
      934,569.299814557                    input
x          52.8274194891359 04             input
-----------------------------
= 49,370,884.442971624253823
+           4.20349729193958               input
-----------------------------
= 49,370,888.64646892                      output
```



(Intel Sandy Bridge, 2.27B transistors)

Going across the die will require an order of magnitude more!

DARPA study predicts that by 2019:

- **Double precision FMADD flop: 11pJ**
- **cross-die per word access (1.2pJ/mm): 24pJ (= 96pJ overall)**

after DARPA report of P. Kogge (ND) *et al.* and T. Schulthess (ETH)

Today's power costs per operation

Operation	approximate energy cost
DP FMADD flop	100 pJ
DP DRAM read-to-register	4800 pJ
DP word transmit-to-neighbor	7500 pJ
DP word transmit-across-system	9000 pJ

Remember that 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_LC	1
Line current density I/A	κ

Robert Dennard, IBM
(inventor of DRAM, 1966)

Eventually processing is limited by transmission, as known for > 4 decades

Some exascale architecture trends

- 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 architecture-applications gap

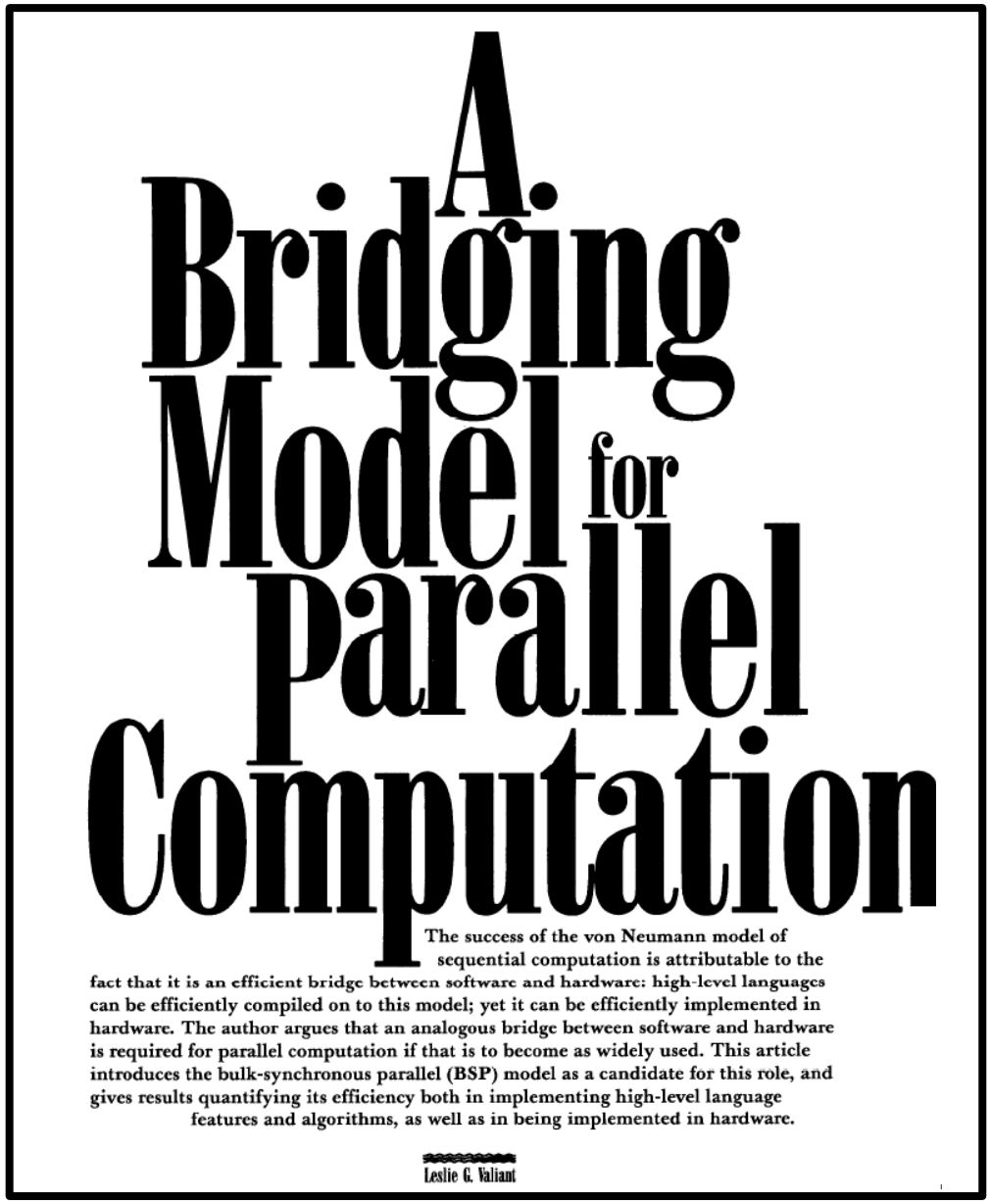
Node-based "weak scaling" is routine; thread-based "strong scaling" is the game

- **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**
- **The real challenge is usefully expanding the number of cores 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 battle is fought on individual shared-memory nodes**


Energy-aware
generation
BSP
generation

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.

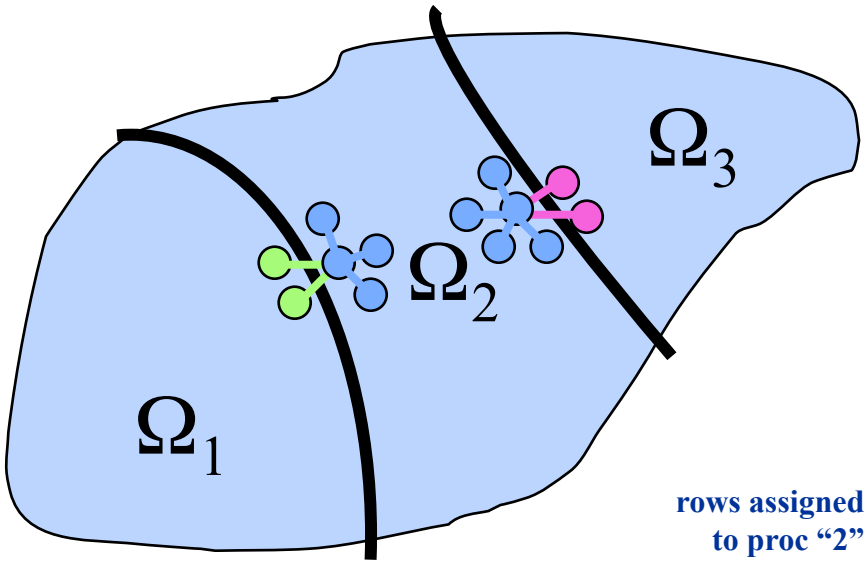
Leslie G. Valiant

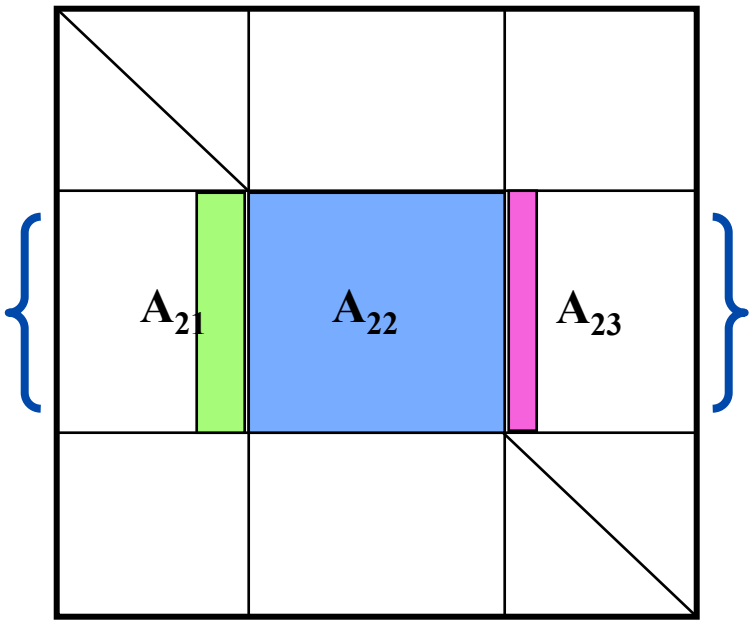
Comm. of the ACM, 1990

How are most 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 at its boundaries with other processors that own portions 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

BSP parallelism w/ domain decomposition





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

BSP has an impressive legacy

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

Gordon Bell Prize: Peak Performance 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

Extrapolating exponentials eventually fails

- **Scientific computing at a crossroads w.r.t. extreme scale**
- **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)**

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**
  - **not a "distraction," but an intellectual stimulus**
  - **potentially big gains in adapting to new hardware environment**
  - **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., 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**

Conclusions, up front

- **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 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"**

Warning: not all accept this 4-fold agenda

- **Non-controversial:**
  - **reduced synchrony (in frequency and/or span)**
  - **greater arithmetic intensity**
- **Semi-controversial, when it comes to serving real applications:**
  - **greater SIMD-style shared-memory concurrency**
- **Controversial:**
  - **built-in resilience ("algorithm-based fault tolerance" or ABFT) to arithmetic/memory faults or lost/delayed messages**

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***

Philosophy

- **Algorithms must adapt to span the gulf between aggressive 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

Required software

Model-related

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

Development-related

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

High-end computers come with little of this 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

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

Recap of 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.**

Some algorithmic "points of light"

Remaining 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
- Multicore implementations of sparse 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

Comment

- **Dominant consumers in applications that tie up major supercomputer centers are:**
  - **Linear algebra on dense symmetric/Hermitian matrices**
    - **generalized eigenproblems (Schroedinger) in chemistry/ materials**
    - **reduced Hessians in optimization**
    - **covariance matrices in statistics**
  - **Poisson solves**
    - **highest order operator in many PDEs in fluid and solid mechanics, E&M, DFT, MD, etc.**
- **These are two of the major thrusts of the ECRC at KAUST**

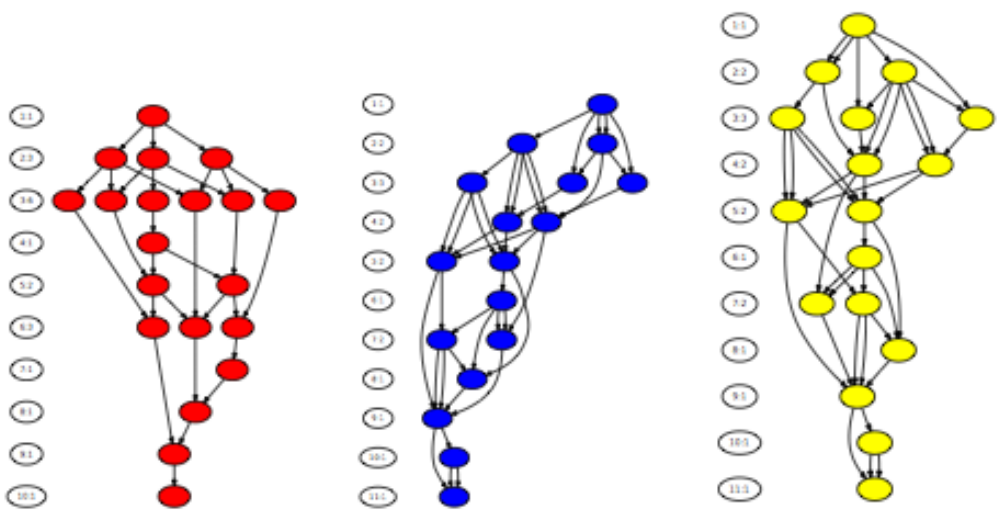
DAG-based data flow tile algorithms for dense linear algebra

- Reduce synchrony
- Increase concurrency

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

$$Ax = \lambda Bx$$

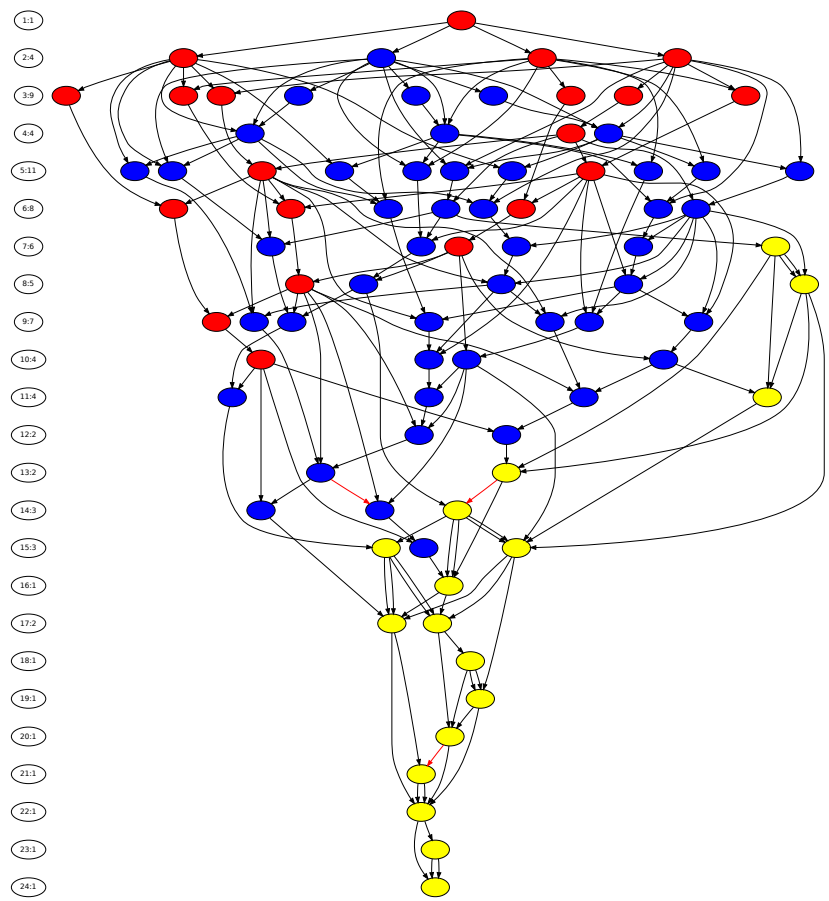
Operation	Explanation	LAPACK routine name
1 $B = L \times L^T$	Cholesky factorization	POTRF
2 $C = L^{-1} \times A \times L^{-T}$ or HEGST	application of triangular factors	SYGST
3 $T = Q^T \times C \times Q$	tridiagonal reduction	SYEVD or HEEVD
4 $Tx = \lambda x$	QR iteration	STERF



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

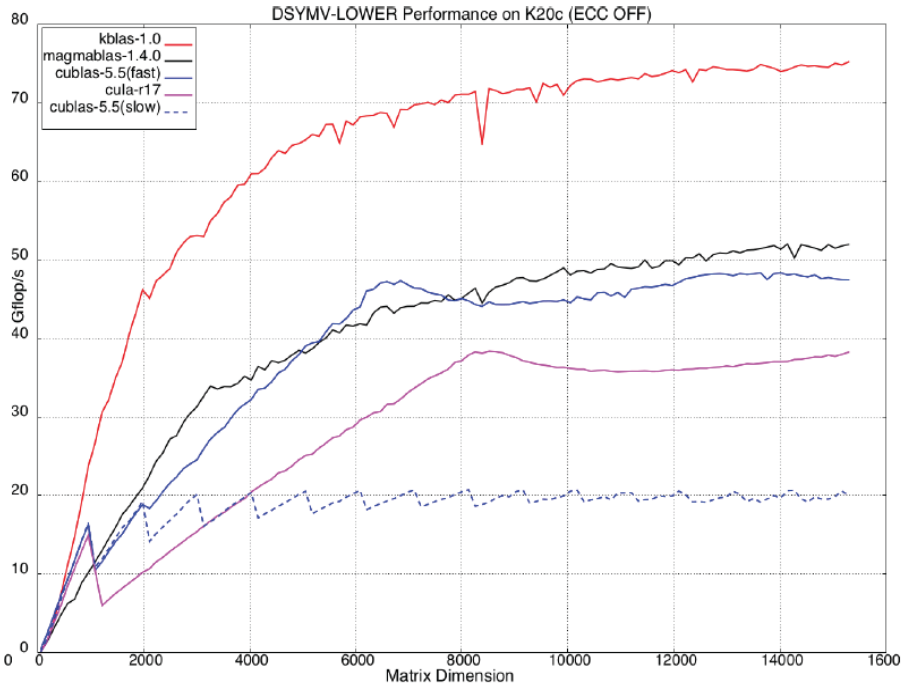


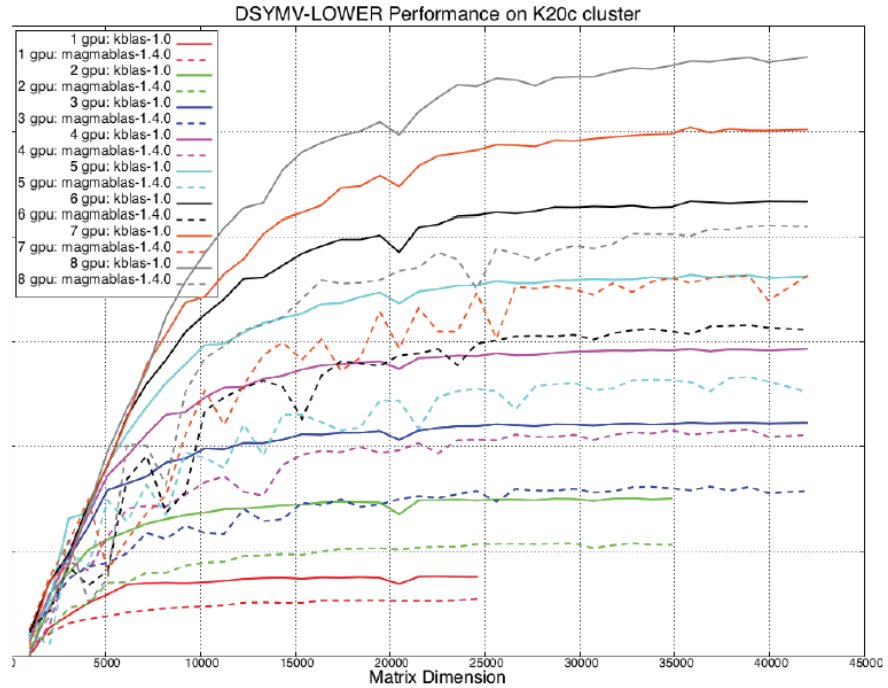
c/o H. Ltaief (KAUST)

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 and beyond**

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

“KBLAS inside”

NVIDIA DEVELOPER ZONE — CUDA TOOLKIT DOCUMENTATION — Search

C. Acknowledgements

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

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

Notices

Notice

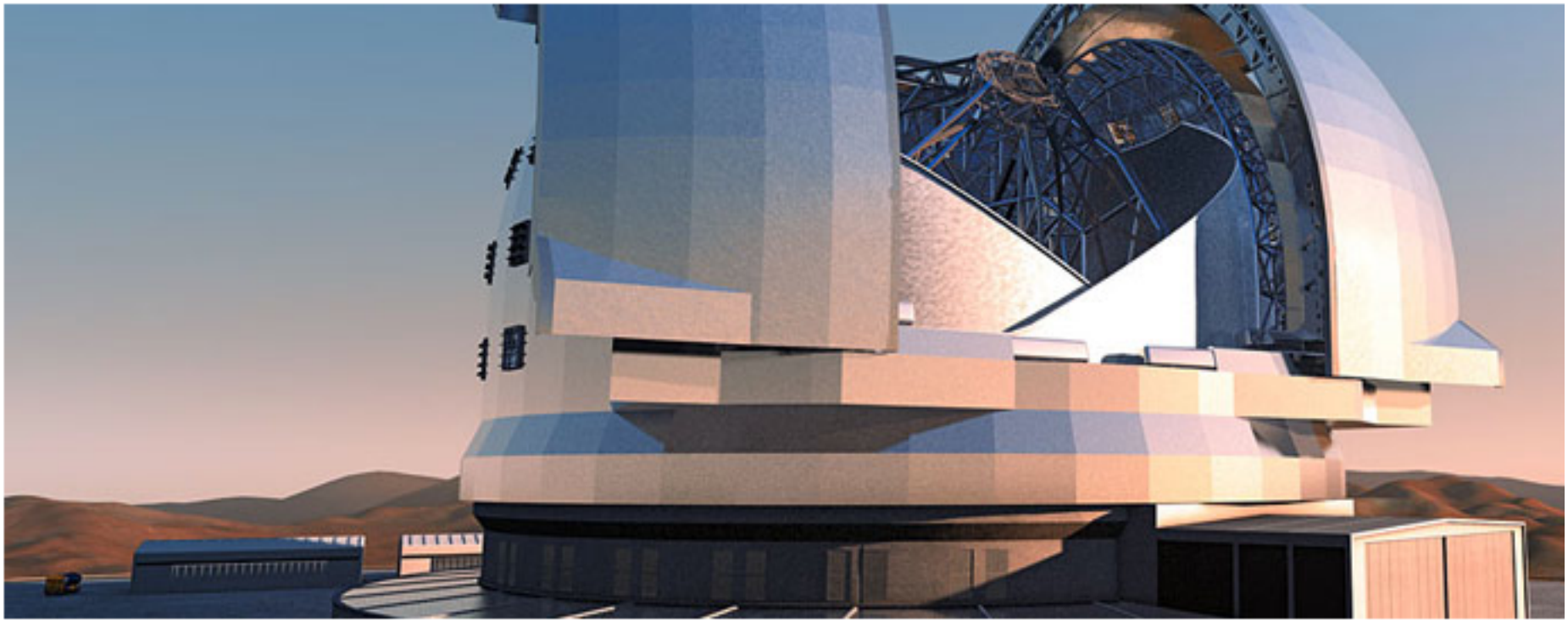
Trademarks

Copyright

Applied in European telescope (ELT) (13X speedup - paper in SC'14)

The European Extremely Large Telescope

The world's biggest eye on the sky



© ESO, https://www.eso.org/sci/facilities/develop/ao/ao_modes.html

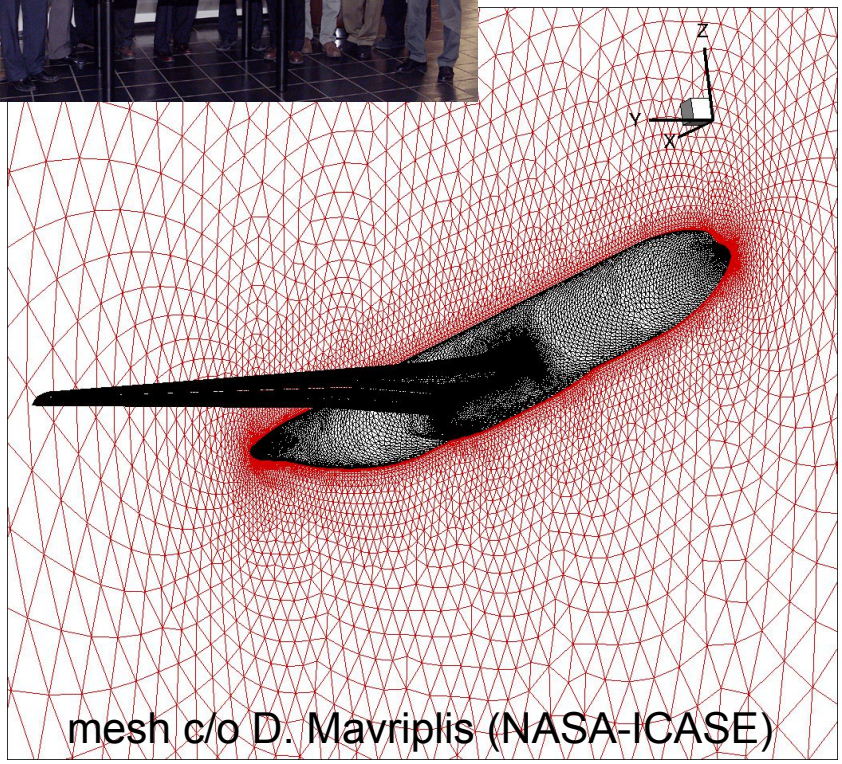
c/o A. Charara (KAUST)

Multicore implementations of sparse linear algebra

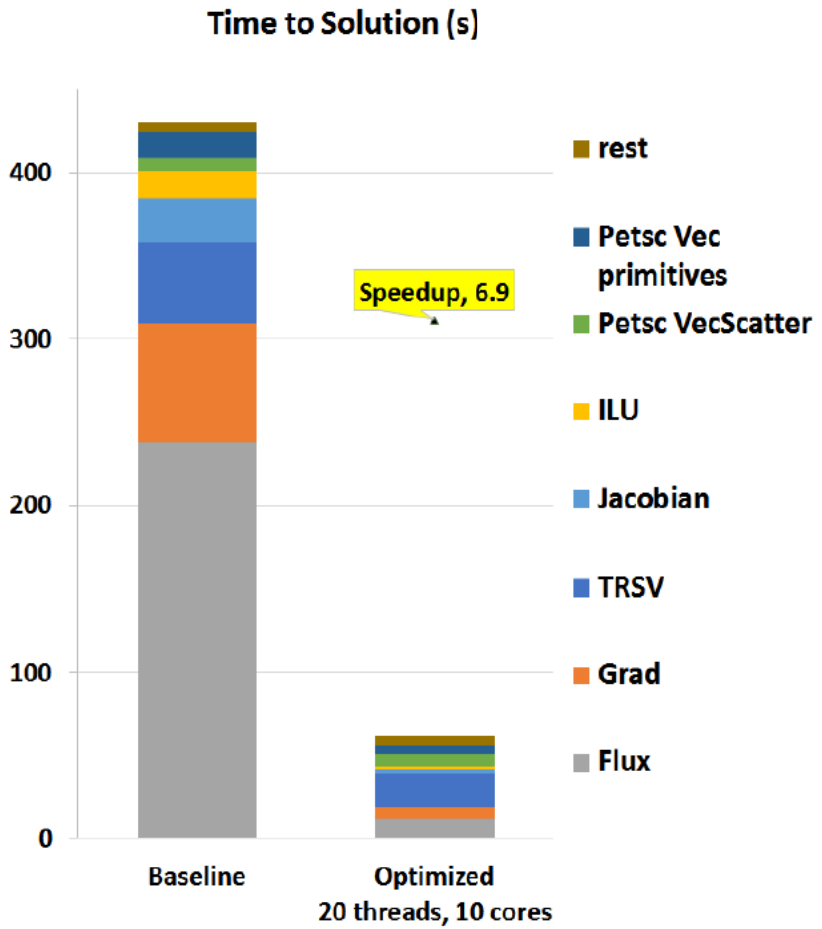
- Increase arithmetic intensity
- Increase concurrency

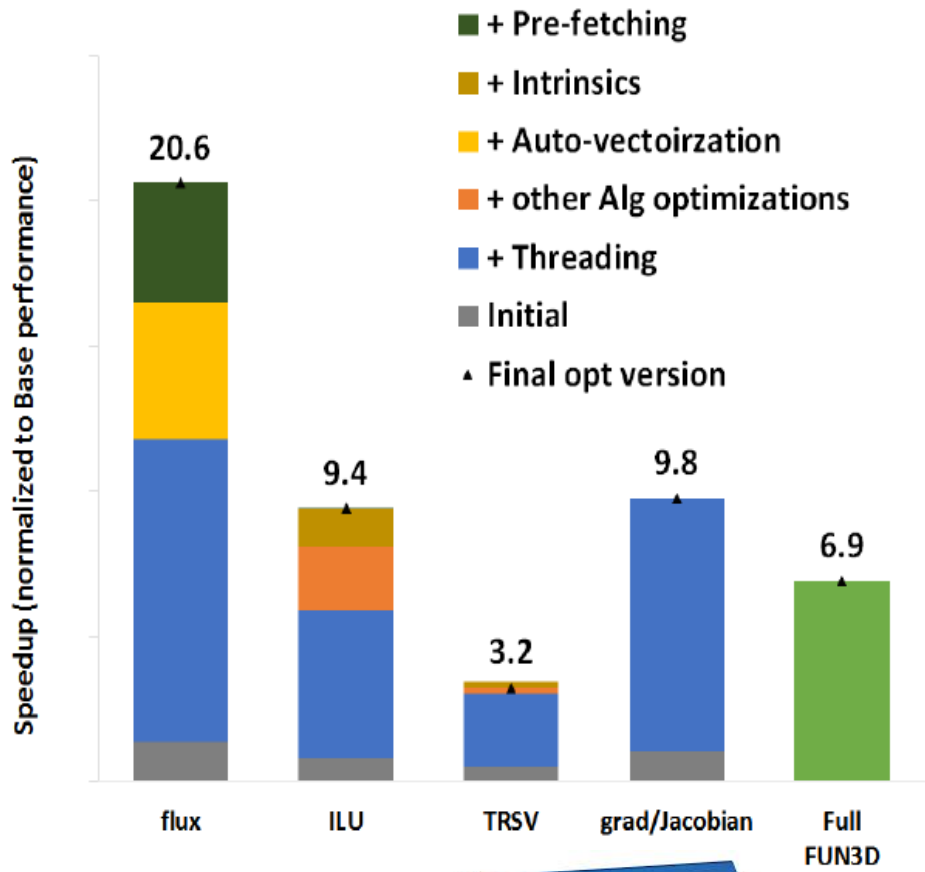
Newton-Krylov-Schwarz based CFD

- **Our PETSc-FUN3D won an ACM Gordon Bell Prize in 1999 for distributed memory scaling of a fully implicit unstructured grid NASA external aircraft flow**
- **With Intel-Bangalore, we ported this implicit unstructured grid code to Intel multi-core chips for strong shared-memory scaling within a single compute node (and extension to many-core is ongoing)**



PETSc-FUN3D on Intel "Ivy Town"



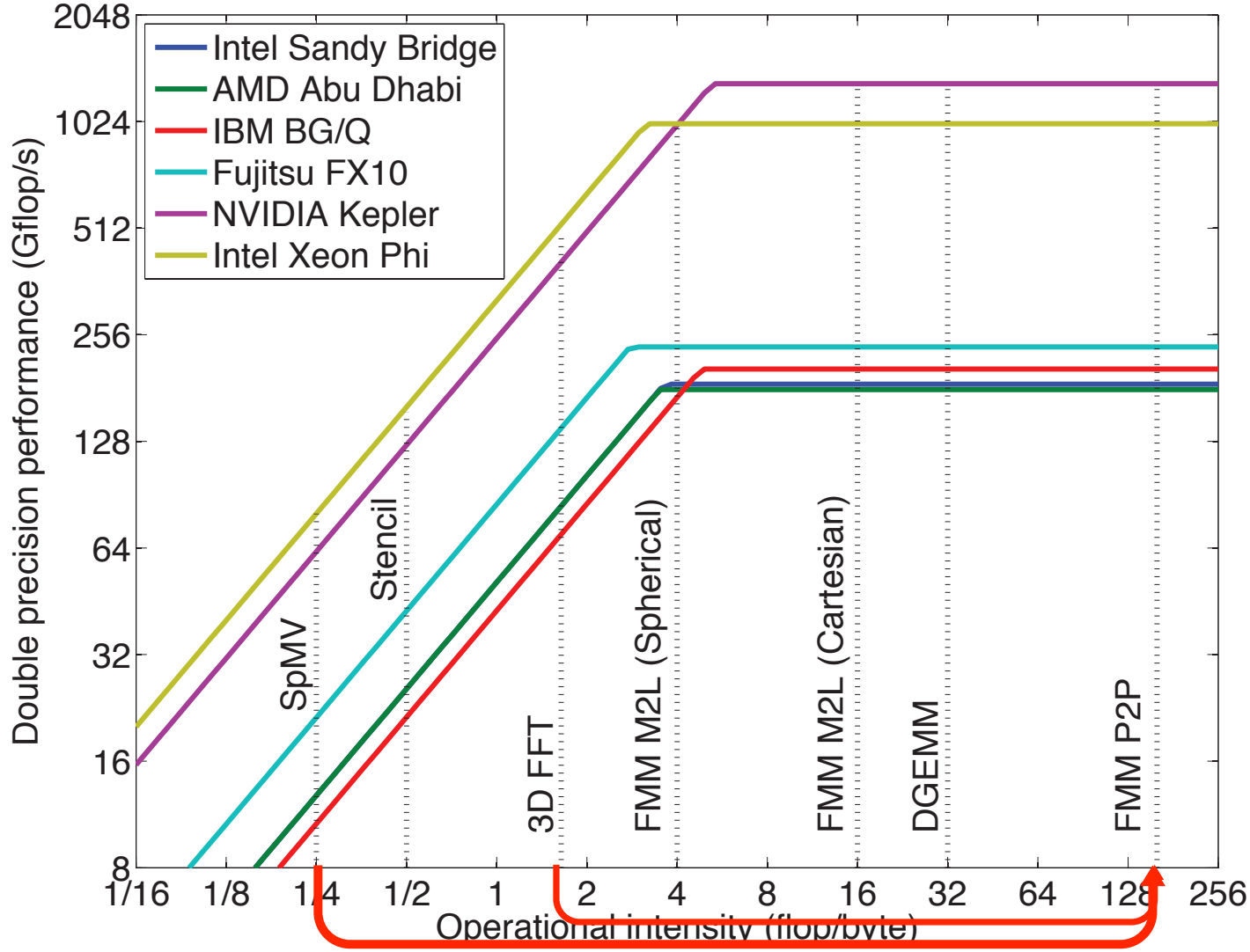


c/o A. Deshpande (Intel), IPDPS'15

Fast Multipole for Poisson solves

- Increase arithmetic intensity
- Reduce synchrony
- Increase concurrency

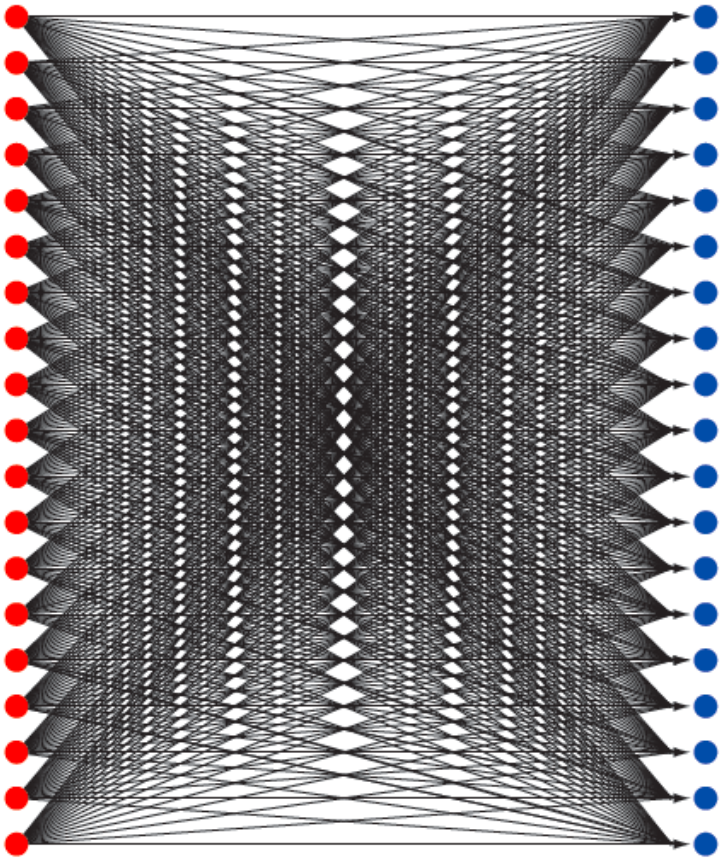
Arithmetic intensity of numerical kernels



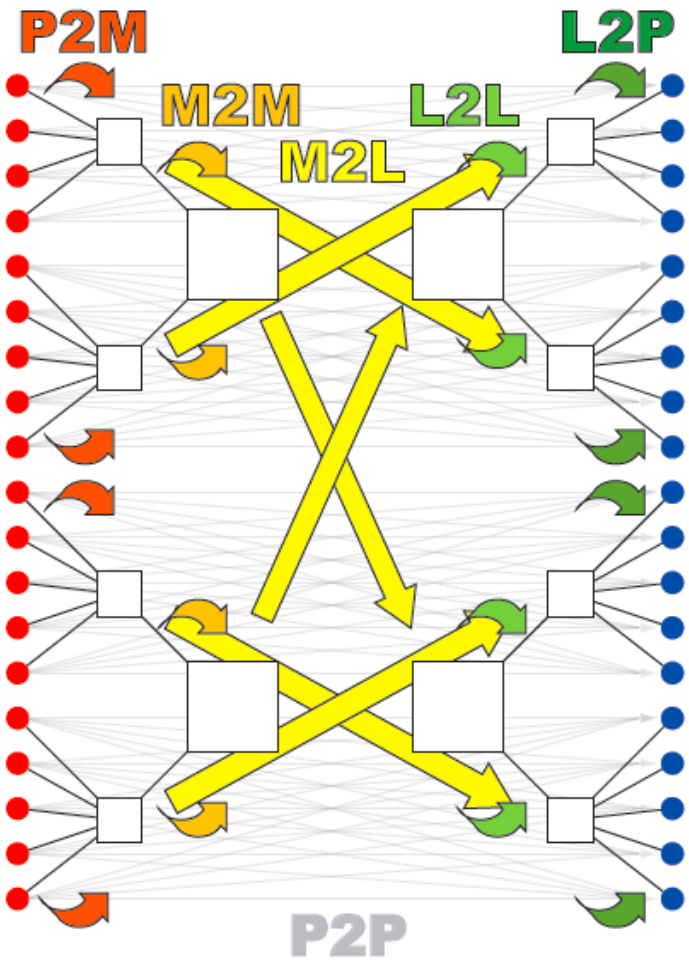
c/o R. Yokota (TiTech, KAUST)

two orders of magnitude variation

Hierarchical interactions of Fast Multipole



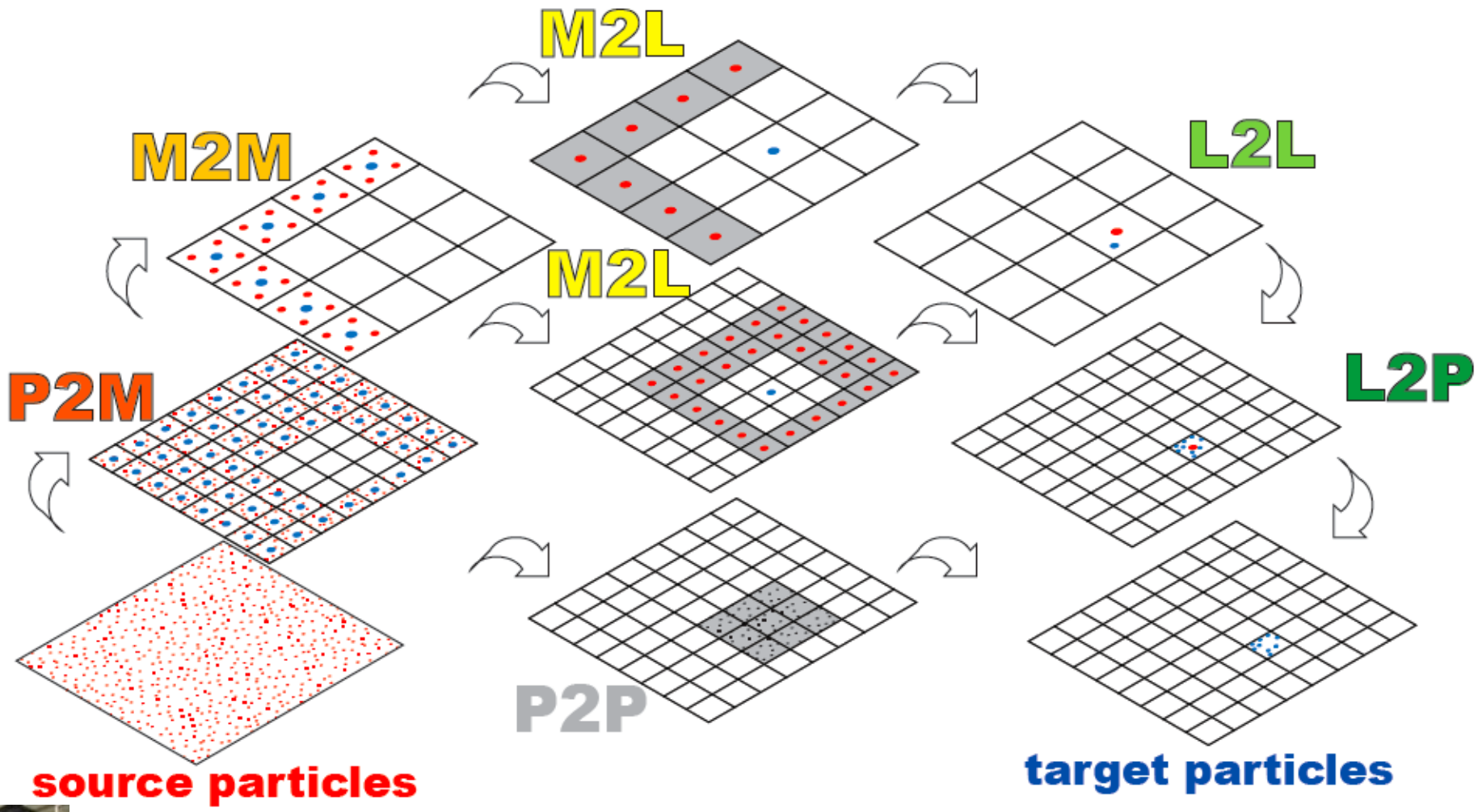
(a) Direct method



(b) Fast Multipole Method

c/o R. Yokota (TiTech, KAUST)

Geometrical structure of Fast Multipole



c/o R. Yokota (TiTech, KAUST)

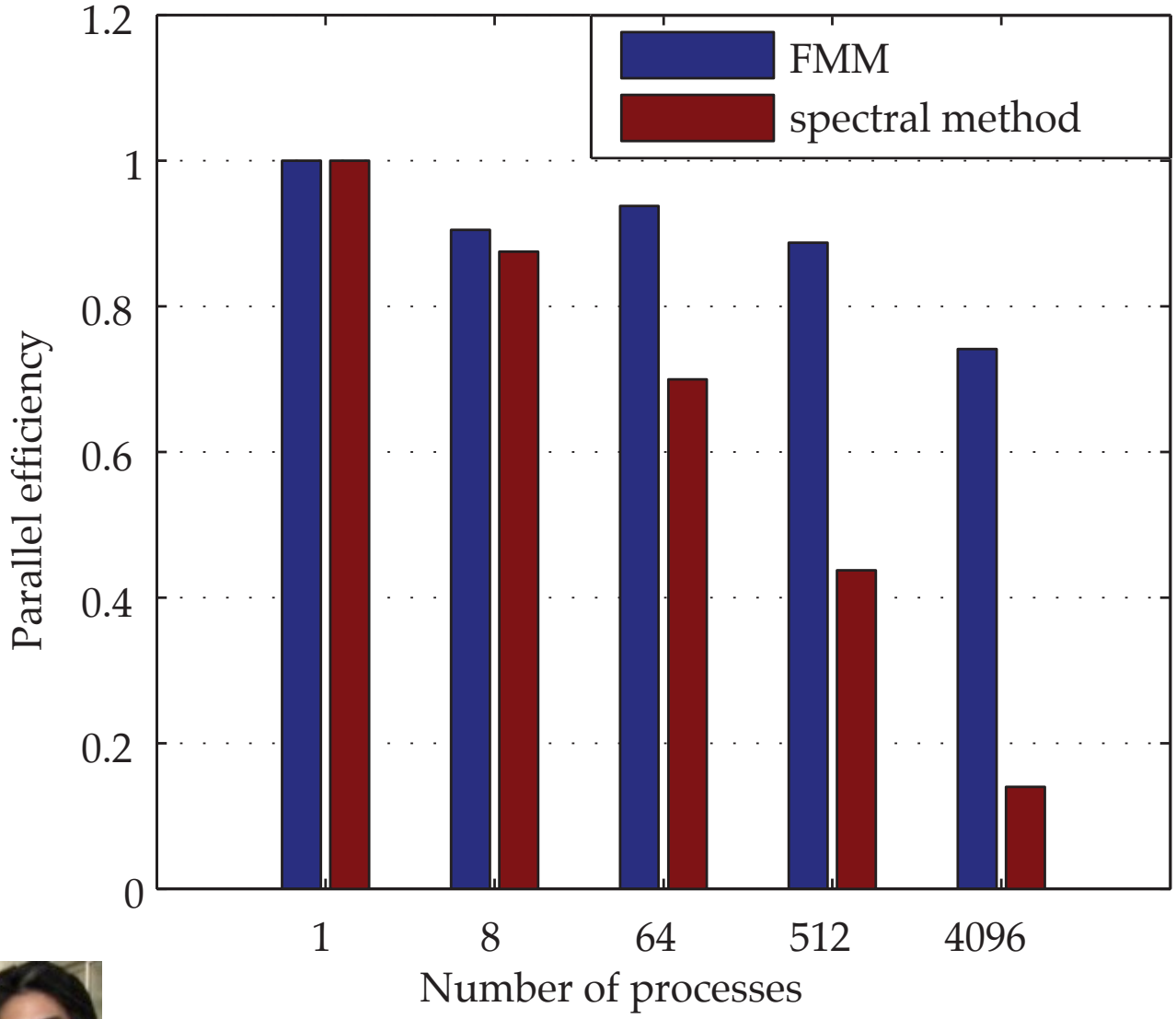
Synchronization reduction – FMM

- **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**

Salient 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)**

FMM vs. FFT in weak scaling



Weak scaling of a vortex-formulation 3D Navier-Stokes code simulating decaying isotropic turbulence, referenced to the pseudo-spectral method, which uses FFT.

FFT: 14% parallel efficiency at 4096 processes, no GPU use.

FMM: 74% going from one to 4096 processes at one GPU per MPI process, 3 GPUs per node.

Largest problem corresponds to a 4096^3 mesh, i.e., almost 69 billion points (about 17 million points per process).

Run on the TSUBAME 2.0 system of the Tokyo Institute of Technology.

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

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

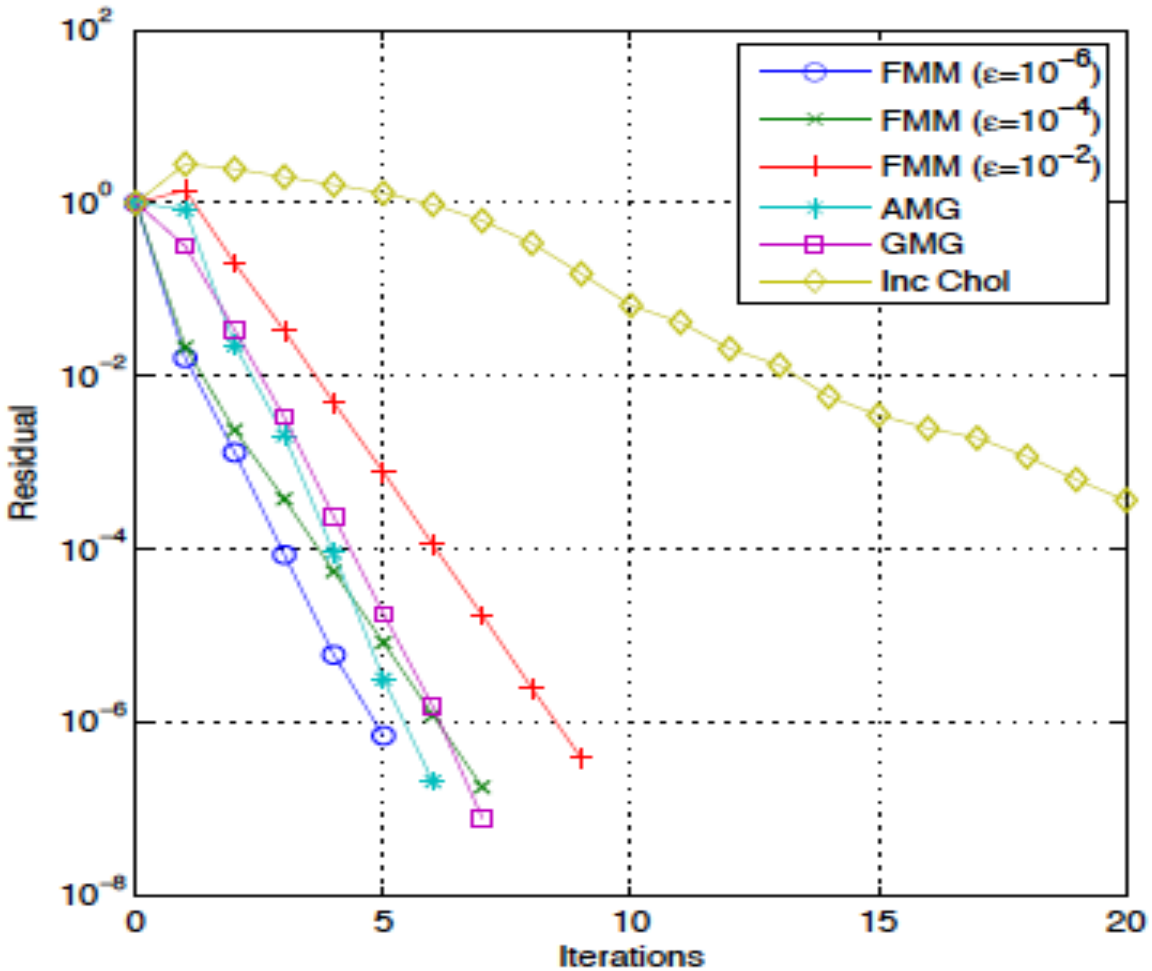
$$N_\Omega \left\{ \begin{bmatrix} \vdots \\ u_i \\ \vdots \end{bmatrix} \right. = \overbrace{\overbrace{\begin{bmatrix} \ddots & & \\ & G_{ij} & \\ & & \ddots \end{bmatrix}}^{N_\Gamma} \begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{bmatrix} - \overbrace{\begin{bmatrix} \ddots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

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.**

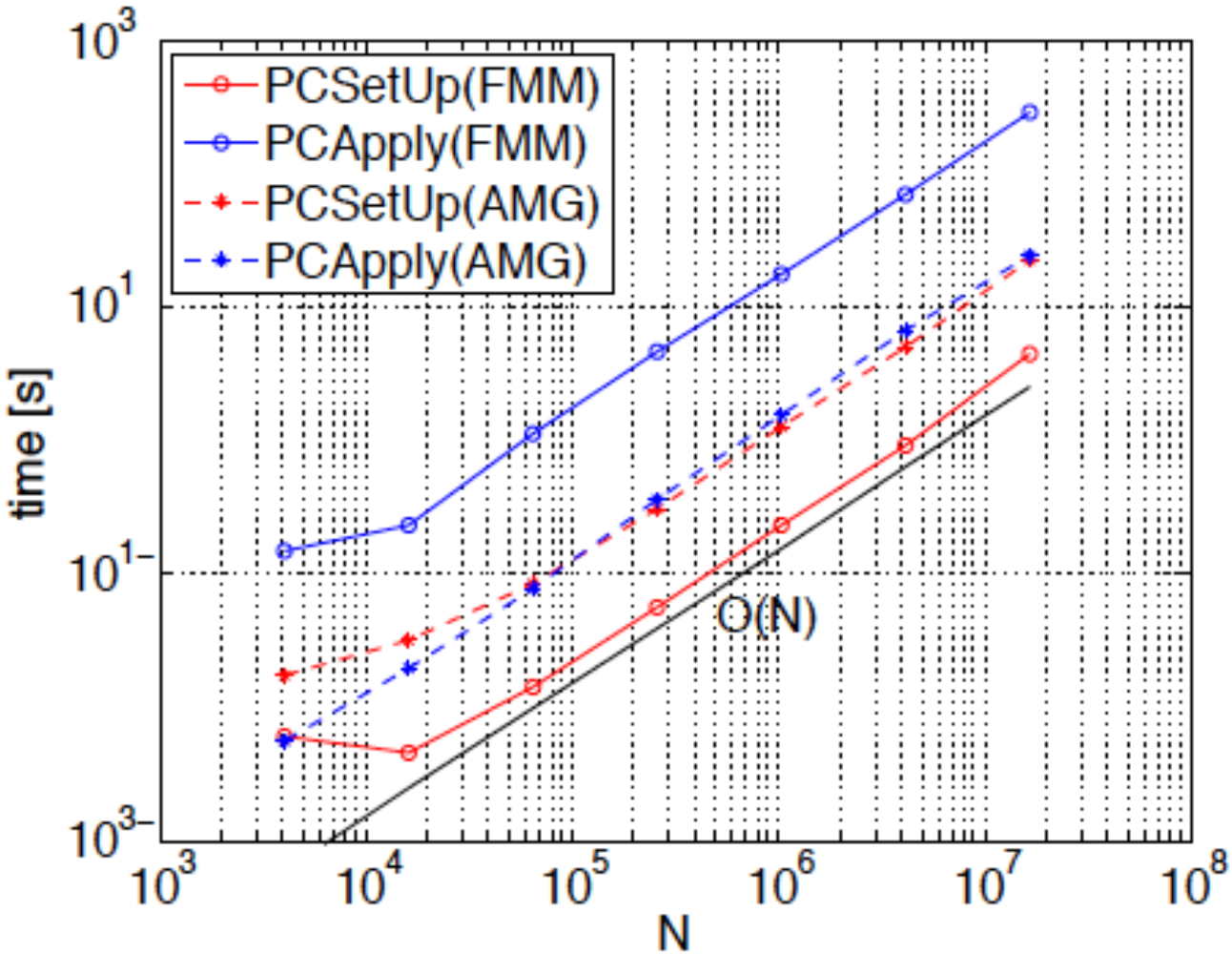
c/o H. Ibeid (KAUST)

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



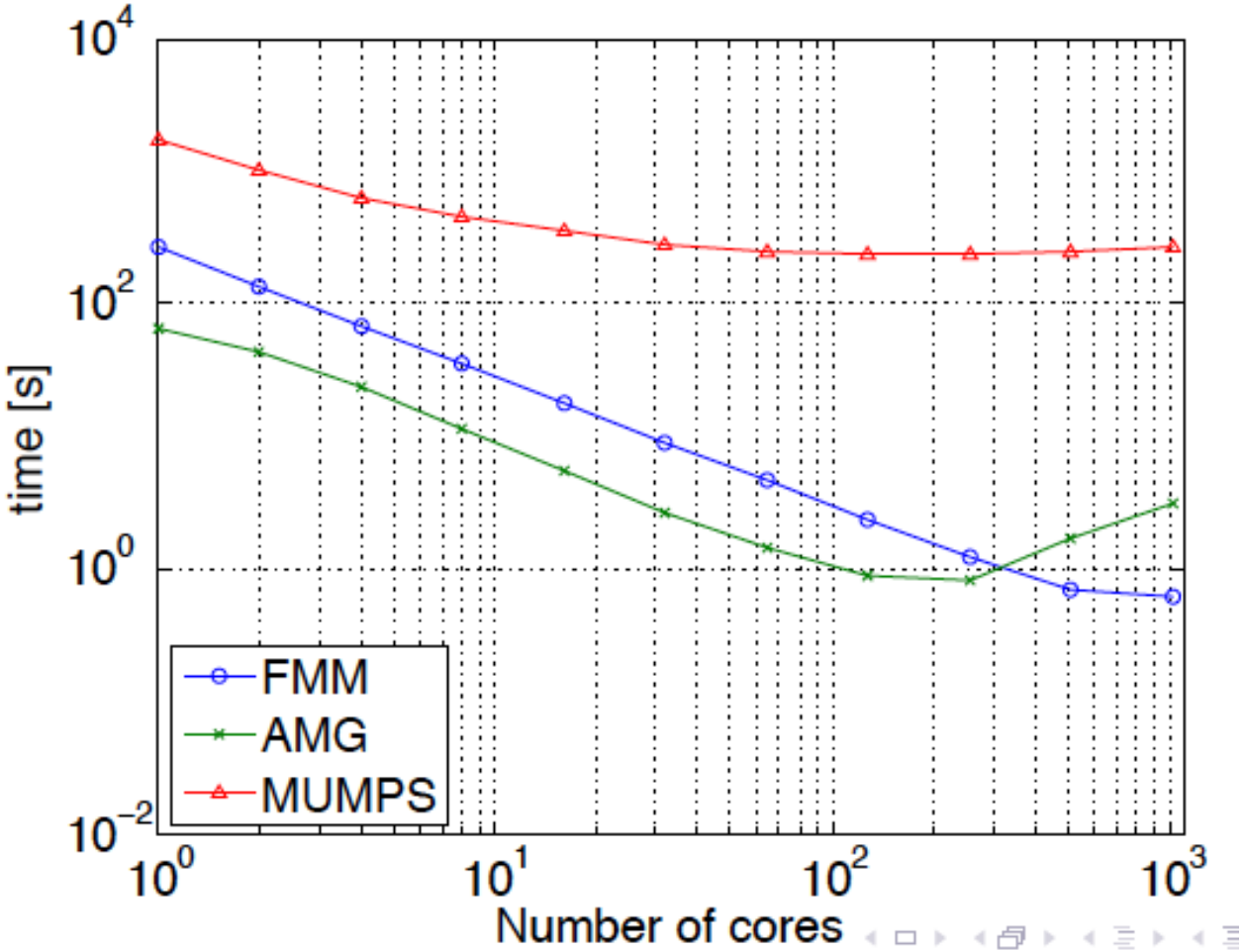
c/o H. Ibeid (KAUST)

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

Key tool: hierarchical matrices

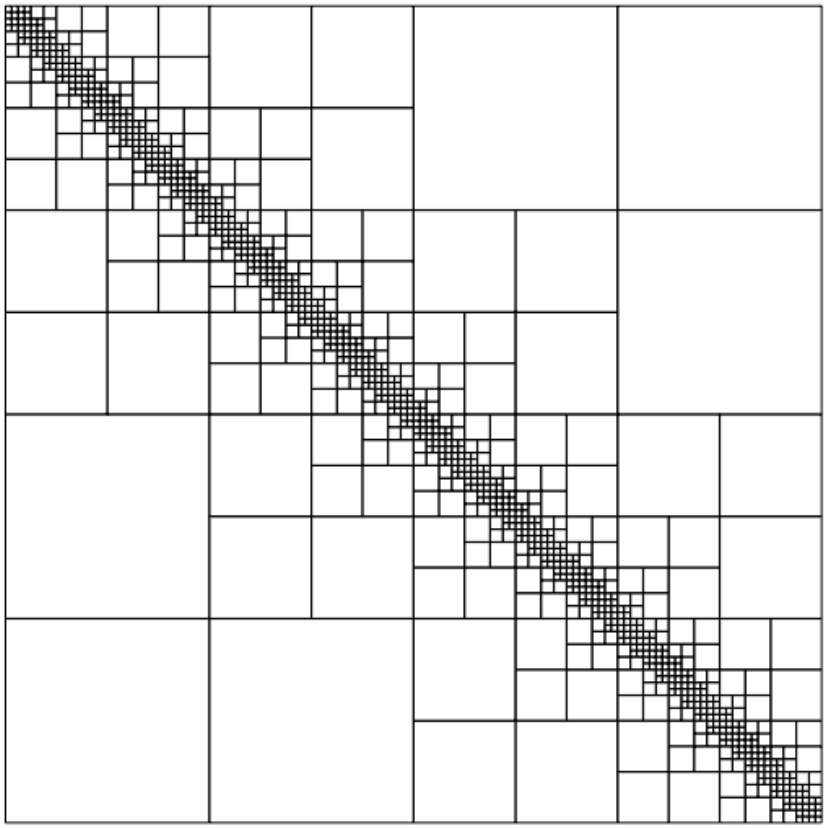
- [Hackbusch, 1999] **: off-diagonal blocks of typical differential and integral operators have low effective rank**
- **By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n**
  - **polynomial in k**
  - **lin-log in n**
  - **constants carry the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

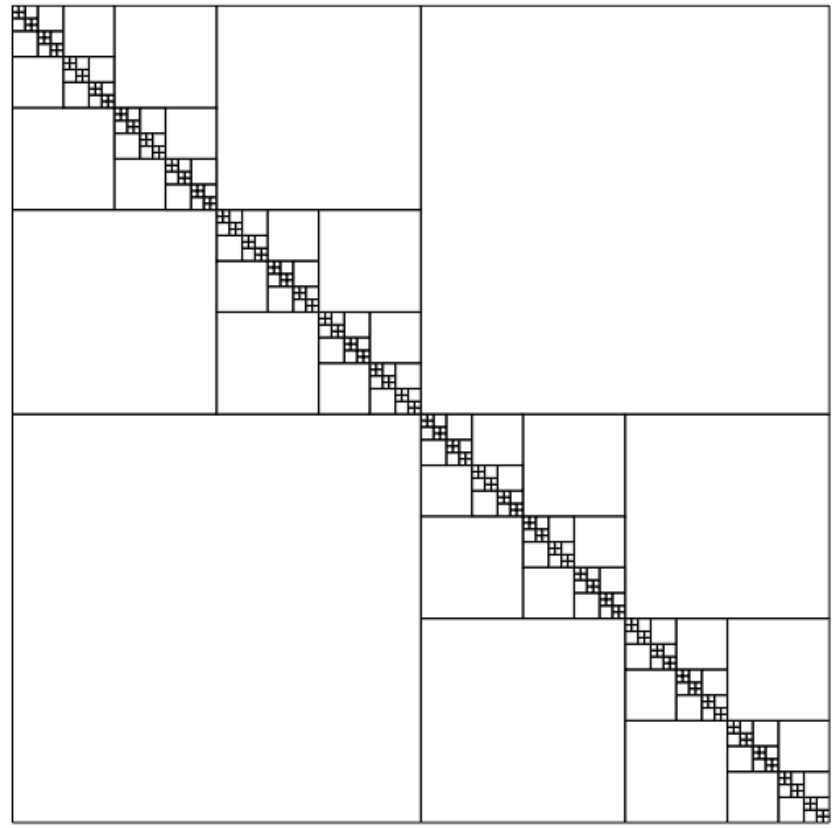
$$A = \left[\begin{array}{ccc|cccc} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ \hline & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{array}\right] \longleftrightarrow \boxed{= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix}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

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

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



strong admissibility

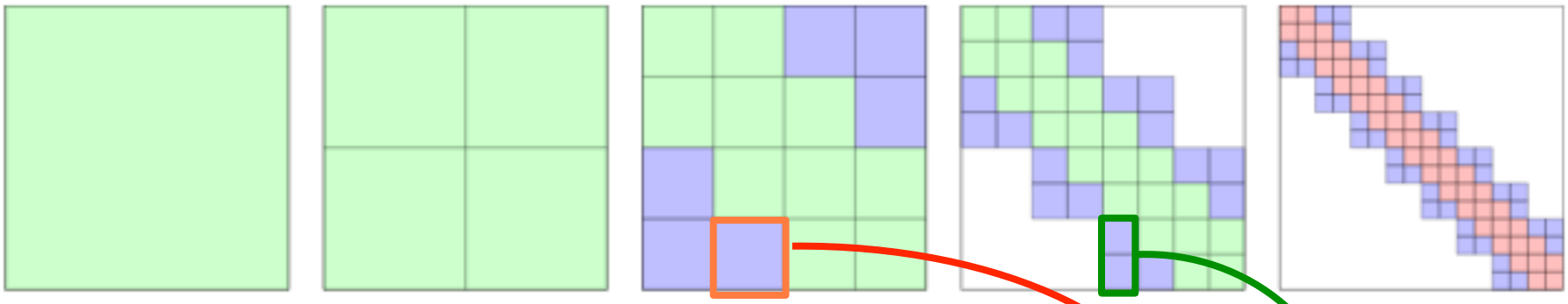


weak admissibility

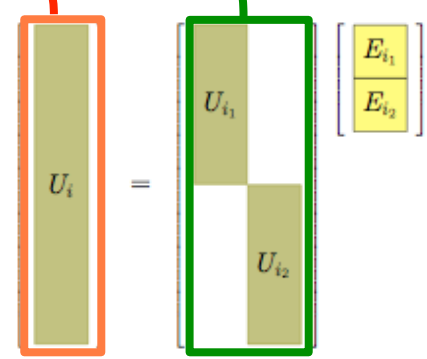
After [Hackbusch, et al., 2003]

Is there an "algebraic FMM"?

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

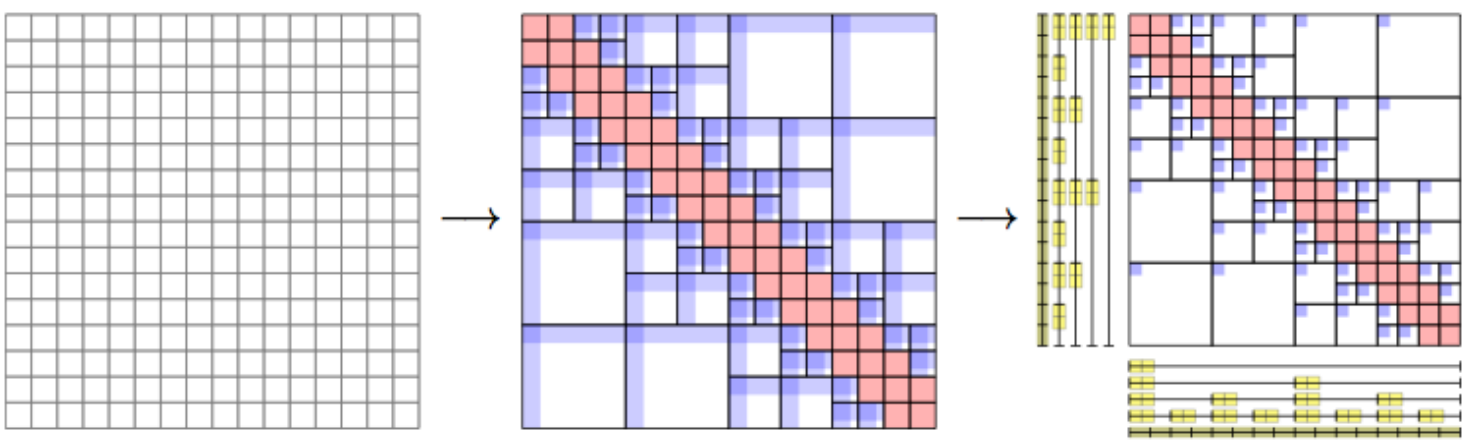


- **Off diagonal blocks $A_{ij} \cong U_i S_{ij} V_j$ can have low rank, based on an "admissibility condition"**
- **Bases can be hierarchically nested**
  - **U_i for columns, V_j for rows**



An "algebraic FMM"

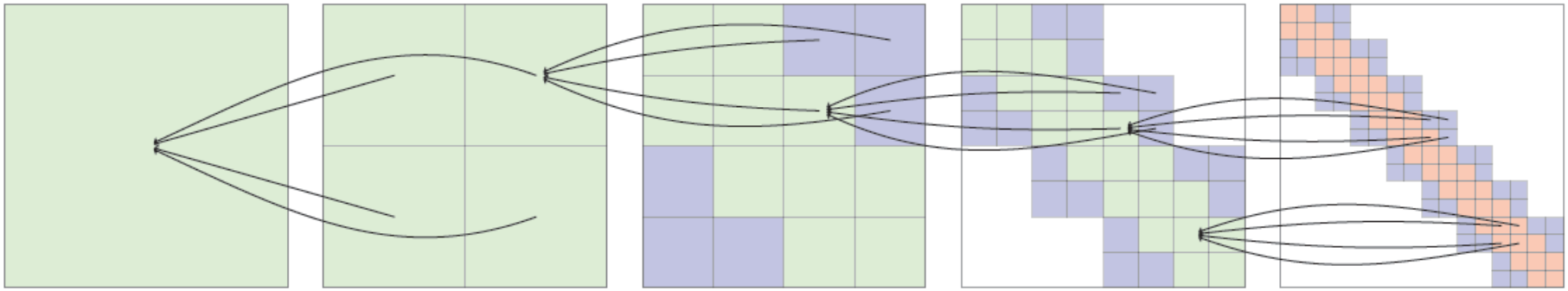
- One needs to store the unreducible diagonal blocks, A_{ii}
- For the entire rest of the matrix, first the S_{ij} , the U_i and V_j at the finest level
- Then the E_{ij} (column basis conversion) and F_{ij} (row basis conversion) blocks at each level
- Two stage compression procedure: SVD each block,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

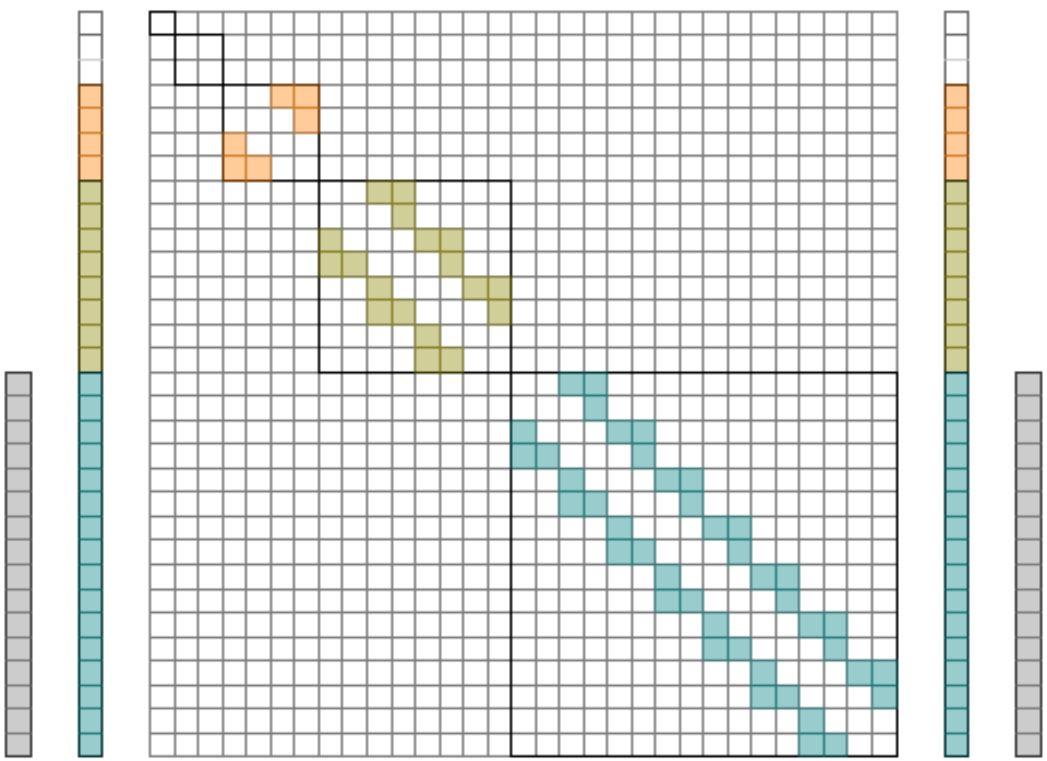


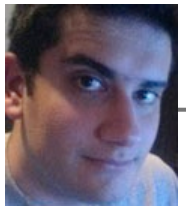
- **Yes, after compressing the matrix in H^2 form**
  - **presumes hierarchical low rank structure**
  - **may offer breakthrough in application performance**
  - **See *Supercomput. Front. Innov*. 1:62-83 (2014)**

c/o G. Turkiyyah (KAUST)

Fast matrix-vector multiply, $y = Ax$

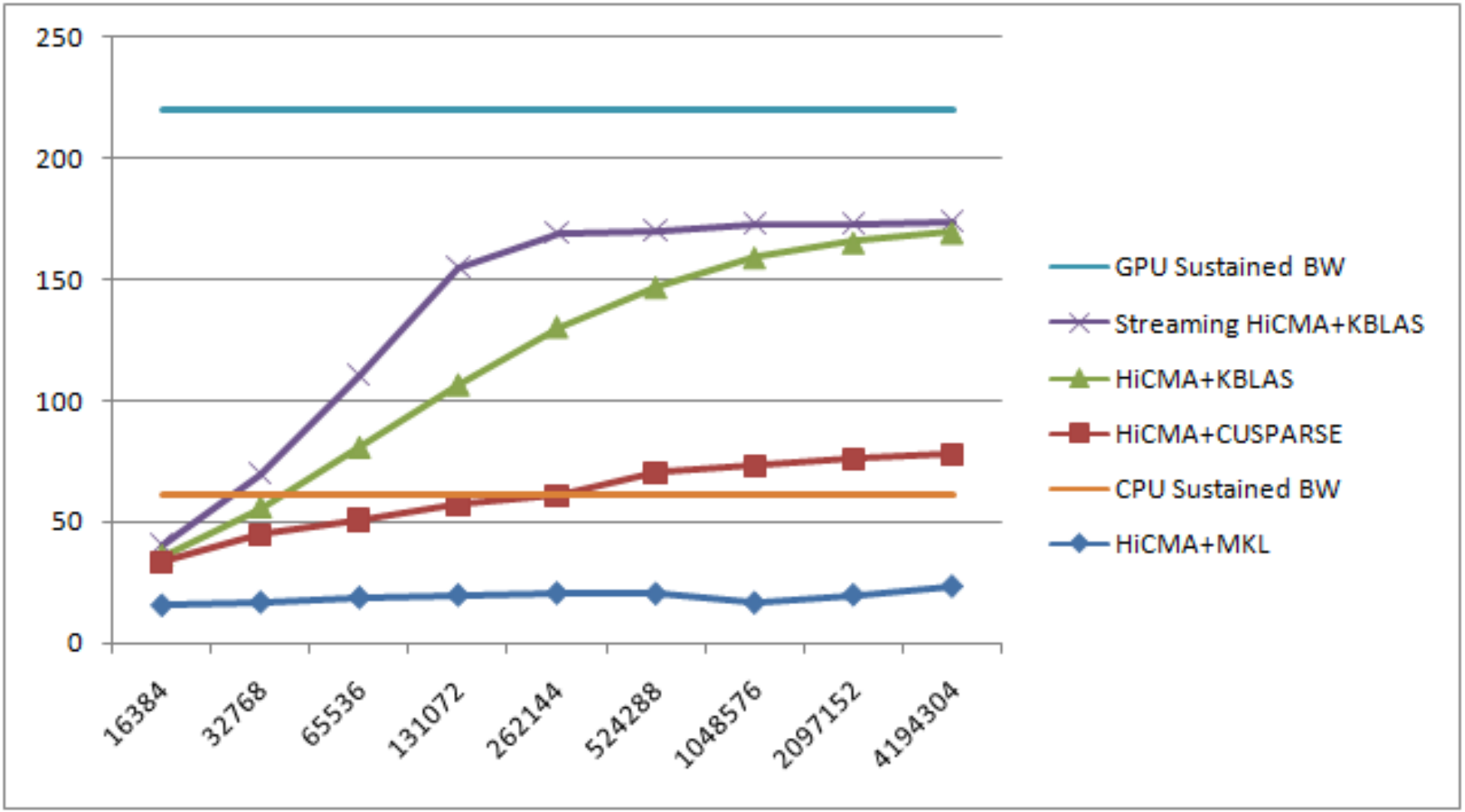
$$y = \left(\sum_{(i,j)\in D} A_{ij} \right) x + \left(\sum_{(i,j)\in L} U_i S_{ij} V_j^t \right) x = \underbrace{\sum_{(i,j)\in D} A_{ij} x_j}_{\text{Dense mat-vecs operations}} + \underbrace{\sum_{i\in I} U_i \underbrace{\sum_{(i,j)\in L} S_{ij} \underbrace{V_j^t x}_{\text{Upsweep}}}_{\text{Coupling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

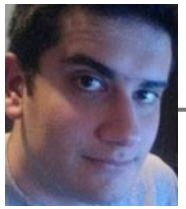




c/o W. Bukharam (KAUST)

Fast matrix-vector multiply, $y = Ax$





achieved bandwidth for rank k=8 and leafsize n=32, integral equation kernel

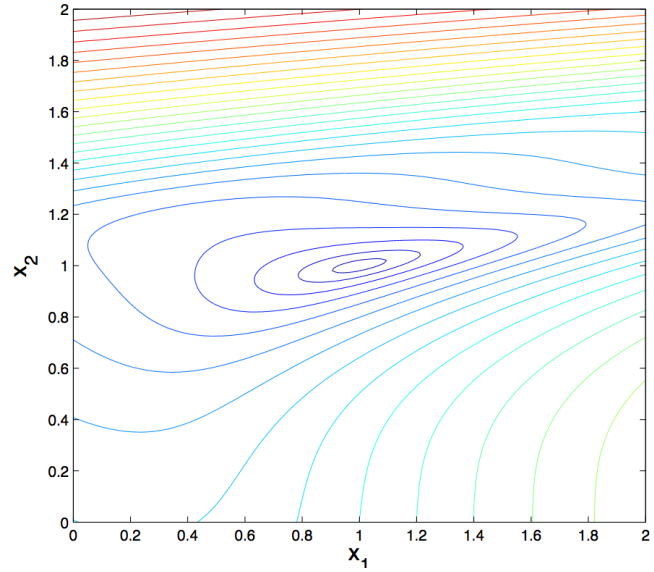
c/o W. Bukharam (KAUST)

Nonlinear preconditioning for Newton's method

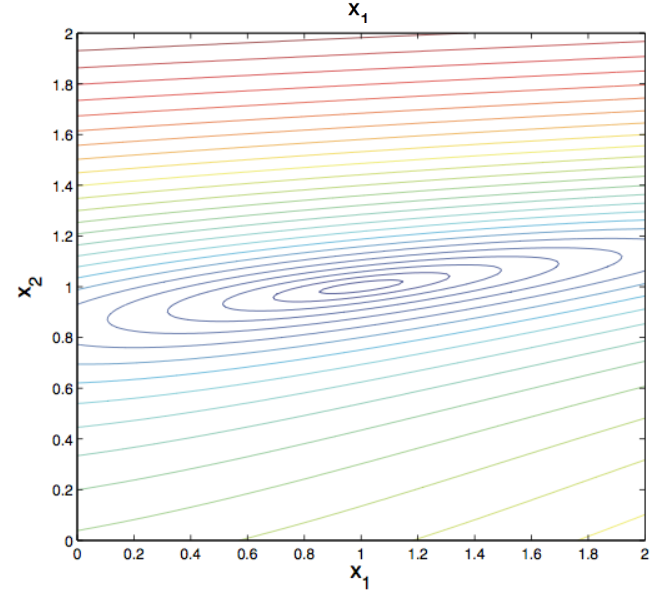
- Reduce synchrony in frequency and scope

Ill-conditioning of nonlinear and linear types

- **A nonlinear system** $F(u) = 0$ **may be "stiff," in the sense that the iso-contours of the merit function, e.g.,** $f(u) = ||F(u)||^2$ **, are far from hyperellipsoidal, giving a small local convergence domain**



- **This may be combined with linear ill-conditioning, in the sense that the hyperellipsoids are locally badly stretched**



Reduction of domain of synchronization in nonlinearly preconditioned Newton

- **Newton method for a global nonlinear system, $F(u)=0$,**
  - **computes a global distributed Jacobian matrix and synchronizes globally in both the Newton step and in solving the global linear system for the Newton**
- **Nonlinearly preconditioned Newton replaces this with a set of local problems on subsets of the global nonlinear system**
  - **each local problem has only local synchronization**
  - **each of the linear systems for local Newton updates has only *local* synchronization**
  - **there is still global synchronization in a number of steps, hopefully *many fewer* than required in the original Newton method**

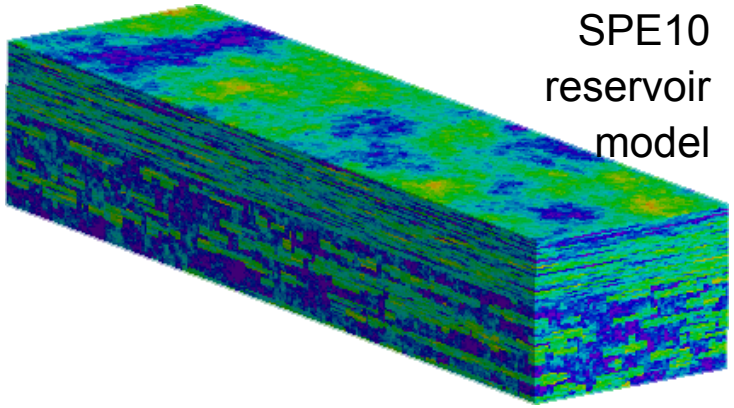
Implemented in PETSc, as "ASPIN"

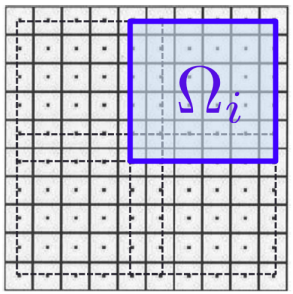
Key idea

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?

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

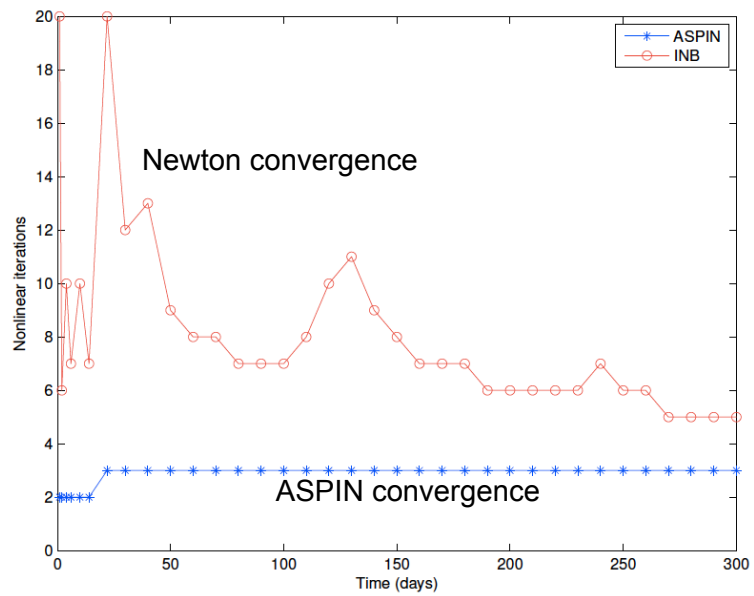
$$\mathcal{F}(u) = \sum_{i=1}^{N} T_{\Omega_i}(u) \qquad \bigcup_{i=1}^{N} \Omega_i = \Omega$$

Assumption

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

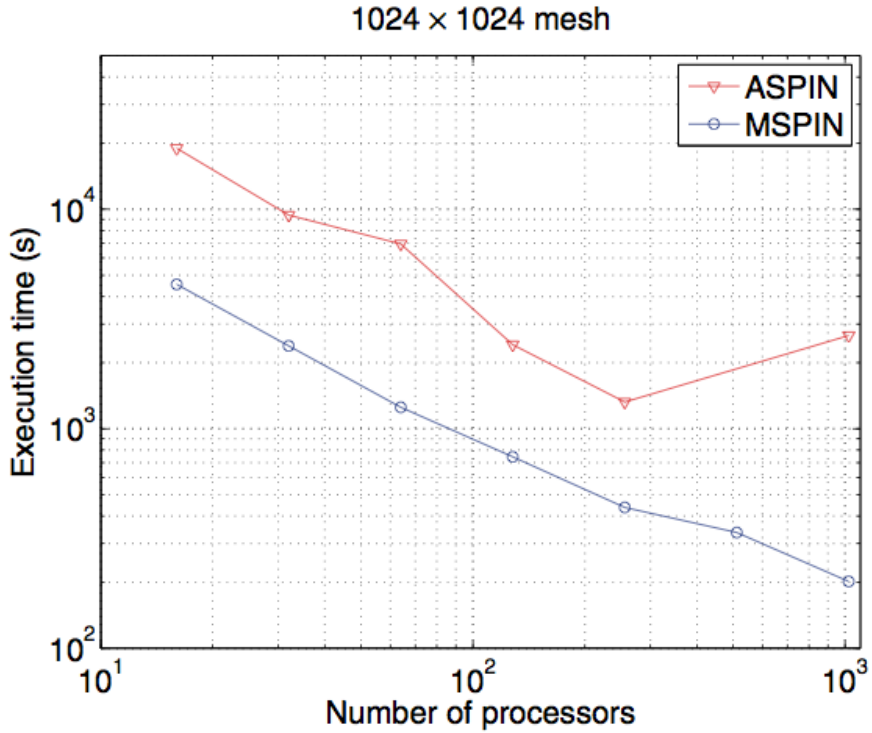
Theorem

(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.



c/o L. Liu (KAUST)

MSPIN: multiplicative by field
3-field PDE example



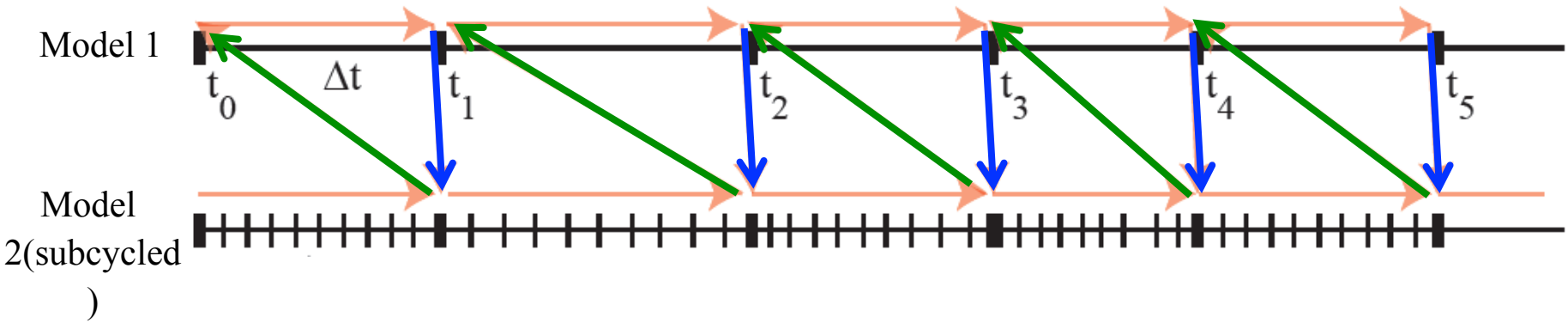
Linear subsystems solved with hypre's BoomerAMG

FIG. 5. *Strong scaling for the driven cavity flow problem on a* 1024×1024 *mesh at Reynolds number* 1000. *The initial guess is still zero for* u, v, ω. $\epsilon_{global-linear-rtol} = 10^{-3}$, $\epsilon_{global-nonlinear-rtol} = 10^{-8}$, $\epsilon_{sub-rtol} = 10^{-3}$, *and* $\epsilon_{Jac-rtol} = 10^{-3}$. $\epsilon_{sub-rtol}$ *denotes the relative tolerance for the subproblems (which are linear in this example), and we specify* $\epsilon_{Jac-rtol}$ *as the relative tolerance for the linear problems in* (2.13) *and* (2.29). *The finite difference step size for the matrix-free Jacobian applications is* 10^{-8}. *Execution time for ASPIN using* 512 *processors is not shown since it fails to converge on this mesh and this Reynolds number from a zero initial guess.*

New programming paradigm for PDE codes

- Reduce synchrony

Multiphysics w/ legacy codes: *an endangered species?*



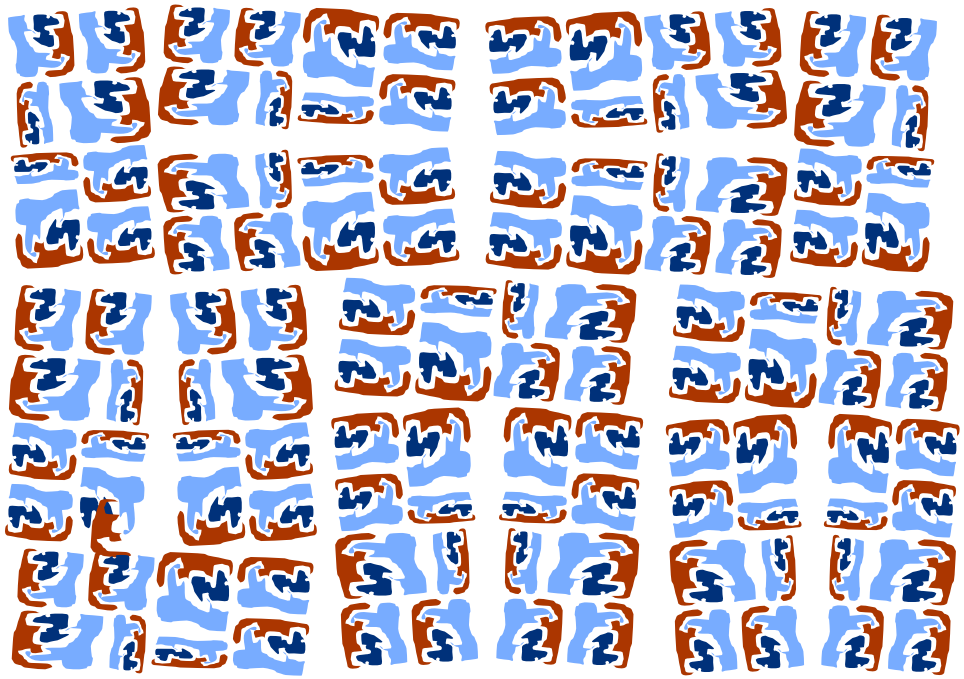
- **Many multiphysics codes operate like this, where the models may occupy the same domain in the bulk (e.g., reactive transport) or communicate at interfaces (e.g., ocean-atmosphere)***
- **The data transfer cost represented by the blue and green arrows may be much higher than the computation cost of the models, even apart from first-order operator splitting error and possible instability**

*see "Multiphysics simulations: challenges and opportunities" (IJHPCA)

Many codes have the algebraic and software structure of multiphysics

- **Exascale is motivated by these:**
  - **uncertainty quantification, inverse problems, optimization, immersive visualization and steering**
- **These may carry auxiliary data structures to/from which blackbox model data is passed and they act like just another "physics" to the hardware**
  - **pdfs, Lagrange multipliers, etc.**
- **Today's separately designed blackbox algorithms for these may not live well on exascale hardware: co-design may be required due to data motion**

Multiphysics layouts must invade blackboxes



- Each application must first be ported to extreme scale (distributed, hierarchical memory)
- Then applications may need to be interlaced at the data structure level to minimize copying and allow work stealing at synchronization points

THE INTERNATIONAL JOURNAL of
HIGH PERFORMANCE COMPUTING APPLICATIONS

Multiphysics simulations: Challenges and opportunities

The International Journal of High Performance Computing Applications
27(1) 4–82

DOI: 10.1177/1094342012468181
hpc.sagepub.com
SAGE

David E Keyes[1,2], Lois C McInnes[3], Carol Woodward[4], William Gropp[5], Eric Myra[6], Michael Pernice[7], John Bell[8], Jed Brown[3], Alain Clo[1], Jeffrey Connors[4], Emil Constantinescu[3], Don Estep[9], Kate Evans[10], Charbel Farhat[11], Ammar Hakim[12], Glenn Hammond[13], Glen Hansen[14], Judith Hill[10], Tobin Isaac[15], Xiangmin Jiao[16], Kirk Jordan[17], Dinesh Kaushik[3], Efthimios Kaxiras[18], Alice Koniges[8], Kihwan Lee[19], Aaron Lott[4], Qiming Lu[20], John Magerlein[17], Reed Maxwell[21], Michael McCourt[22], Miriam Mehl[23], Roger Pawlowski[14], Amanda P Randles[18], Daniel Reynolds[24], Beatrice Rivière[25], Ulrich Rüde[26], Tim Scheibe[13], John Shadid[14], Brendan Sheehan[9], Mark Shephard[27], Andrew Siegel[3], Barry Smith[3], Xianzhu Tang[28], Cian Wilson[2] and Barbara Wohlmuth[23]

Abstract

We consider multiphysics applications from algorithmic and architectural perspectives, where "algorithmic" includes both mathematical analysis and computational complexity, and "architectural" includes both software and hardware environments. Many diverse multiphysics applications can be reduced, en route to their computational simulation, to a common algebraic coupling paradigm. Mathematical analysis of multiphysics coupling in this form is not always practical for realistic applications, but model problems representative of applications discussed herein can provide insight. A variety of software frameworks for multiphysics applications have been constructed and refined within disciplinary communities and executed on leading-edge computer systems. We examine several of these, expose some commonalities among them, and attempt to extrapolate best practices to future systems. From our study, we summarize challenges and forecast opportunities.

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**

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, too**

Other galaxies

Other hopeful algorithmic directions

- **74 two-page whitepapers contributed by the international community to the Exascale Mathematics Working Group (EMWG) at**

  https://collab.mcs.anl.gov/display/examath/Submitted+Papers

- **20-21 August 2013 in Washington, DC**
  - **Randomized algorithms**
  - **On-the-fly data compression**
  - **Algorithmic-based fault tolerance**
  - **Adaptive precision algorithms**
  - **Concurrency from dimensions beyond space (time, phase space, stochastic parameters)**
  - **etc.**

Trends according to Pete Beckman

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)

More 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


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