4-5-2021

Lecture 01: Scalable Solvers: Universals and Innovations

David Keyes
King Abdullah University of Science and Technology, david.keyes@kaust.edu.sa

University of Arkansas Department of Mathematical Sciences

46th Spring Lecture Series

David Keyes

Extreme Computing Research Center

King Abdullah University of Science and Technology

5-9 April 2021

Lecture 1

Scalable Solvers: Universals and Innovations

My goals for the series

- **Recruit researchers to a "renaissance" in scalable solvers**
  - **set the stage of opportunity**
  - **introduce global leaders in core techniques as guest lecturers**
  - **introduce several "universals" that govern scalable computing into the indefinite future**
  - **show some current algorithmic developments that relate to these universals**
- **Provide motivation to grow the computational mathematics community at our host institution**
- **Feature the research of young colleagues in KAUST's Extreme Computing Research Center**
- **Encourage gender diversity in the math sciences**
- **Celebrate the elegance and power of the math sciences**

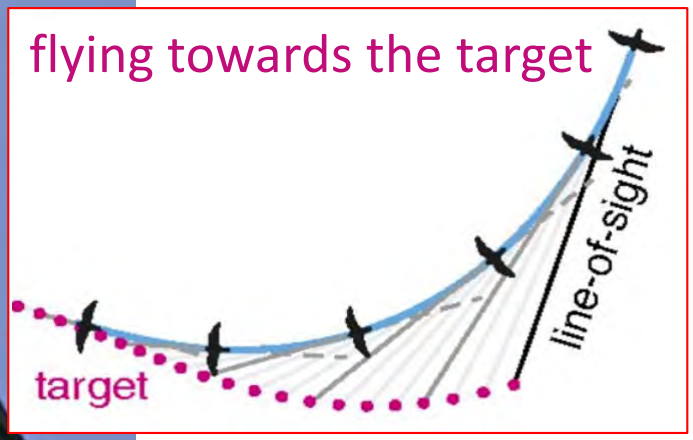
Structure of the series

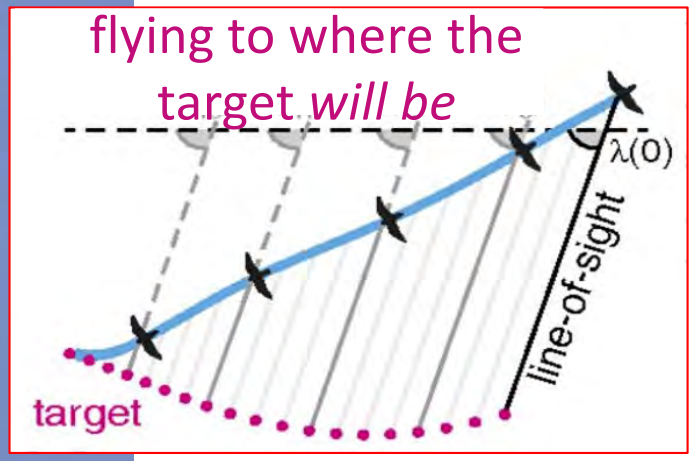
- **Traditional SLS structure**
  - **five principal lectures**
  - **ten guest lectures**
- **Plus a public outreach lecture**
  - ***Harnessing the power of mathematics for HPC***
- **Plus a panel on Women in STEM**
- **A venerable tradition going back to 1977**
  - **An honor for us to be associated with the influential mathematical scientists – pure, applied, statistical, and computational – that have graced this series so far**
  - **Thanks to Professor Tulin Kaman for her vision, initiative, persistence, and logistics**

SLS weeklong schedule

	Mon, April 5th	Tue, April 6th	Wed, April 7th	Thu, April 8th	Fri, April 9th
9:00am	**David Keyes** Universals of Extreme Scale Computing	**David Keyes** Tile Low Rank Methods & Applications	**David Keyes** Hierarchically Low Rank Methods & Applications	**David Keyes** Spatial Statistics with HRL, TLR & Mixed Precision	**David Keyes** Convergence of Big Data and Extreme Computing
10:15am	**Ulrike Yang** Algebraic Multigrid Methods	**Laura Grigori** Multilevel and Randomized Methods	**Xiao-Chuan Cai** Nonlinear Preconditioning Methods	**Zhaojun Bai** Eigen Decomposition Methods	**Rio Yokota** Low-rank & Kroenecker Methods
11:30am	**Edmond Chow** Preconditioned Iterative Methods	**Jack Dongarra** Dense Linear Methods	**Carol Woodward** Time Integration Methods	**Sherry Li** Low-rank Factorization Methods	**Ilse Ipsen** Randomized Least Squares Methods
6:00pm	**Ann Almgren** Mathematics for High Performance		**AWM Panel** Women in STEM		

A falcon flies to where the prey *will be* …





… rather than where it is

C. H. Brighton, et al., PNAS (2017)

Let's fly with the falcons…
to where computer architectures will be

Some "universals" of exascale computing

Architectural imperatives

- Reside "high" on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized "back-ends" while presenting high-level APIs to general users
- Exploit data sparsity to meet "curse of dimensionality" with "blessing of low rank"
- Process "on the fly" rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Timely appearance in *CACM*

82 COMMUNICATIONS OF THE ACM | APRIL 2021 | VOL. 64 | NO. 4

Figure 1. Fifteen "universals" of exascale computing.

Architectural Imperatives

- Reside "high" on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Strategies in Practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (with more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Strategies in Progress

- Employ dynamic scheduling capabilities, for example, , dynamic runtime systems based DAGs
- Code to specialized "back-ends" while presenting high-level APIs to general users
- Exploit data sparsity to meet "curse of dimensionality" with "blessing of low rank"
- Process "on the fly" rather than storing all at once (especially large dense matrices)
- Co-design algorithms with hardware, including computing in the network or in memory

Timely US interagency topic

Software in the Era of Extreme Heterogeneity

Joint virtual workshop of the NITRD HEC & SPSQ Interagency Working Groups

September 22 – 24, 2020

The Networking and Information Technology Research and Development Program

The HEC IWG also coordinates the operation and modernization of HCC infrastructure and applications to accelerate scientific discoveries and technological innovations. HCIA investments provide researchers from academia, government, and industry tens of billions of computing hours annually on the Nation's most powerful computing platforms and provide the ecosystem and expertise the United States needs in areas such as materials discovery and design, earth and space science, detection and treatment of diseases, and other applications of national interest.

The SPSQ Interagency Working Group (IWG) was formed in 1991 to coordinate Federal SPSQ R&D across 19 participating agencies to achieve orders-of-magnitude reduction in software defects and the time and cost of developing and sustaining software. The IWG gathers information from software experts to ensure that government investment in SPSQ R&D results in the software development technology that is essential to U.S. innovation, emerging technology leadership, security, and prosperity.

https://nitrd.gov

Timely global topic

INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP 1.0

Key concepts:

- "co-design" of architectures and applications
- coordination of enabling software development

"Poster child" example:

- Quantum Chromodynamics (QCD), the application that led to IBM's Blue Gene/L

SPONSORS

https://www.exascale.org

THE INTERNATIONAL JOURNAL of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
000(00) 1–58

DOI: 10.1177/1094342010391989
hpc.sagepub.com
SAGE

The International Exascale Software Project roadmap

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, Zhong Jin, 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 S. Mueller, Wolfgang E. 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 and Kathy Yelick

Abstract

Over the last 20 years, the open-source community has provided more and more software on which the world's high-performance computing systems depend for performance and productivity. The community has invested millions of dollars and years of effort to build key components. However, although the investments in these separate software elements have been tremendously valuable, a great deal of productivity has also been lost because of the lack of planning, coordination, and key integration of technologies necessary to make them work together smoothly and efficiently, both within individual petascale systems and between different systems. It seems clear that this completely uncoordinated development model will not provide the software needed to support the unprecedented parallelism required for peta/exascale computation on millions of cores, or the flexibility required to exploit new hardware models and features, such as transactional memory, speculative execution, and graphics processing units. This report describes the work of the community to prepare for the challenges of exascale computing, ultimately combing their efforts in a coordinated International Exascale Software Project.

Keywords

exascale computing, high-performance computing, software stack

Table of Contents

University of Tennessee at Knoxville, USA

Corresponding author:
Jack Dongarra, University of Tennessee, at Knoxville, 1122 Volunteer Boulevard, Suite 203, Knoxville, TN 37996-3450, USA.
Email: dongarra@cs.utk.edu

Timely global topic (see lecture 5)

Table 1. A 3 × 3 matrix showing beneficial interactions in the six off-diagonal locations between large-scale simulation, data analytics, and machine learning.

	To large-scale simulation	To data analytics	To machine learning
Simulation provides:	—	Physics-based "regularization"	Data for training, augmenting real-world data
Analytics provides:	Steering in high-dimensional parameter space; in situ processing	—	Feature vectors for training
Learning provides:	Smart data compression; replacement of models with learned functions	Imputation of missing data; detection and classification	—

https://www.exascale.org/bdec

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2018, Vol. 32(4) 435–479

DOI: 10.1177/1094342018778123
journals.sagepub.com/home/hpc
SAGE

Big data and extreme-scale computing: Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry

M Asch, T Moore, R Badia, M Beck, P Beckman, T Bidot, F Bodin, F Cappello, A Choudhary, B de Supinski, E Deelman, J Dongarra, A Dubey, G Fox, H Fu, S Girona, W Gropp, M Heroux, Y Ishikawa, K Keahey, D Keyes, W Kramer, J-F Lavignon, Y Lu, S Matsuoka, B Mohr, D Reed, S Requena, J Saltz, T Schulthess, R Stevens, M Swany, A Szalay, W Tang, G Varoquaux, J-P Vilotte, R Wisniewski, Z Xu and I Zacharov

Abstract

Over the past four years, the Big Data and Exascale Computing (BDEC) project organized a series of five international workshops that aimed to explore the ways in which the new forms of data-centric discovery introduced by the ongoing revolution in high-end data analysis (HDA) might be integrated with the established, simulation-centric paradigm of the high-performance computing (HPC) community. Based on those meetings, we argue that the rapid proliferation of digital data generators, the unprecedented growth in the volume and diversity of the data they generate, and the intense evolution of the methods for analyzing and using that data are radically reshaping the landscape of scientific computing. The most critical problems involve the logistics of wide-area, multistage workflows that will move back and forth across the computing continuum, between the multitude of distributed sensors, instruments and other devices at the networks edge, and the centralized resources of commercial clouds and HPC centers. We suggest that the prospects for the future integration of technological infrastructures and research ecosystems need to be considered at three different levels. First, we discuss the convergence of research applications and workflows that establish a research paradigm that combines both HPC and HDA, where ongoing progress is already motivating efforts at the other two levels. Second, we offer an account of some of the problems involved with creating a converged infrastructure for peripheral environments, that is, a shared infrastructure that can be deployed throughout the network in a scalable manner to meet the highly diverse requirements for processing, communication, and buffering/storage of massive data workflows of many different scientific domains. Third, we focus on some opportunities for software ecosystem convergence in big, logically centralized facilities that execute large-scale simulations and models and/or perform large-scale data analytics. We close by offering some conclusions and recommendations for future investment and policy review.

Keywords

Big data, extreme-scale computing, future software, traditional HPC, high-end data analysis

1. Executive summary

Although the "big data" revolution first came to public prominence (circa 2010) in online enterprises like Google, Amazon, and Facebook, it is now widely recognized as the initial phase of a watershed transformation that modern society generally—and scientific and engineering research in particular—are in the process of undergoing. Responding to this disruptive wave of change, over the past 4 years,

Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA

Corresponding author:
J Dongarra, University of Tennessee, Knoxville, TN 37996, USA.
Email: dongarra@icl.utk.edu

Exascale software agenda

- **Emphasize heterogeneity and hierarchy**
  - Heterogeneity is the new normal
  - Hierarchy is the key to efficient representation and access of big data
- **Watch hardware opportunities**
  - Processors: CPU, vector, GPU, TPU, FPGA, neuromorphic, quantum, …
  - Memories: cache, HBM, DRAM, NVRAM, …
  - Channels: copper, optical fiber, direct optical
- **Think on two levels**
  - High-level: how to find thresholds that amortize overheads for changing devices (heterogeneity) or scales (hierarchy)
  - Low-level: how to express (vector extensions, CUDA, libraries for remote ops)
- **Gain hands-on experience and integration**
  - Ideally in a multidisciplinary team, so one's specialized efforts are part of something bigger that motivates and brings visibility and sponsorship

Exascale algorithmic opportunity

To "go big" and achieve the potential of emerging architectures for scientific applications, we need implementations of fast

- **linear and least squares solvers**
- **singular value and eigensolvers**
- **nonlinear solvers and optimizers**
- **integrators and sensitivity solvers**
- **stencil and tensor operators**

that

- **offer tunable accuracy-time-space tradeoffs**
- **exploit data sparsity**
- **exploit hierarchy of precisions**
- ***may* require more flops but complete earlier, thanks to more concurrency or less communication or synchronization**
- **are energy efficient**

Two computational universes exist side-by-side

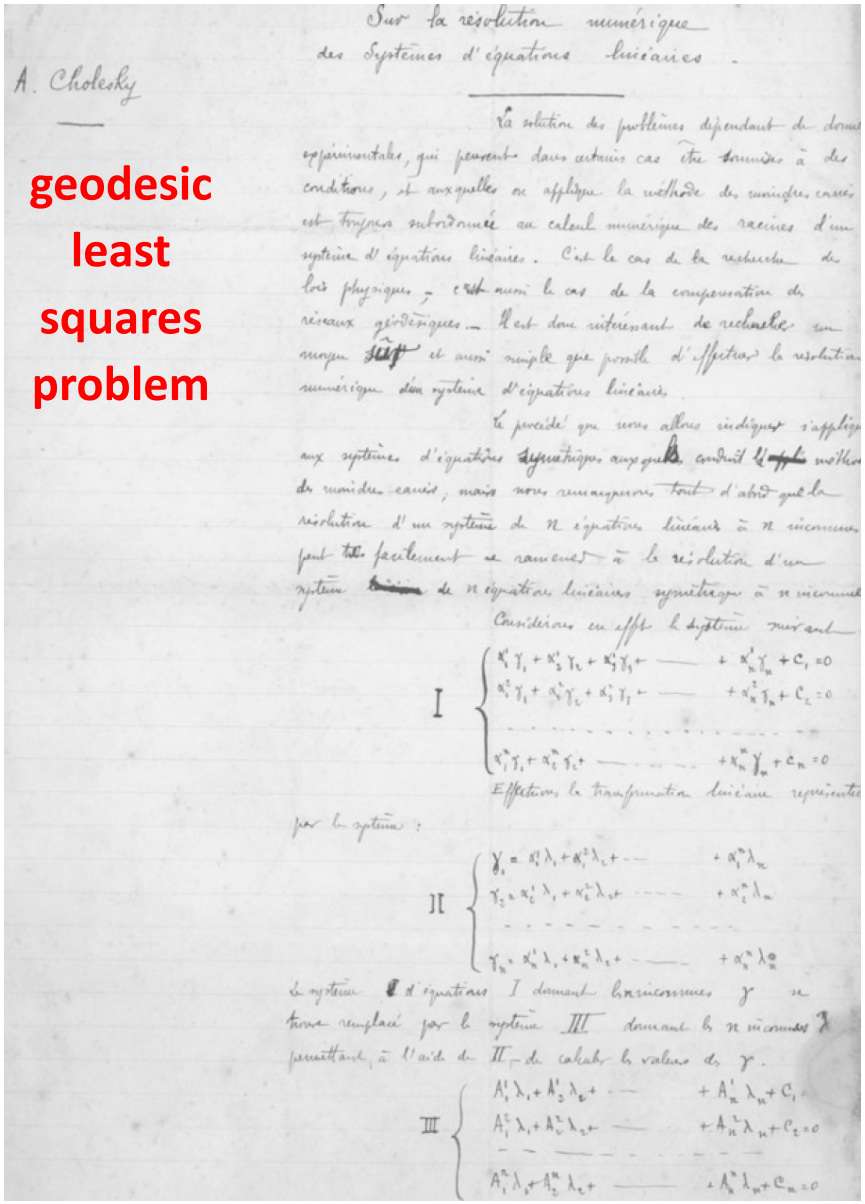
Flat

```
* Global indices *
do i {
    do j {
        for (i,j) in S do op
    }
}
```

Hierarchical

```
* Local indices *
for matrix blocks (k,l)
  do i {
      do j {
          for (i,j) in S_{k,l} do op
      }
  }
```

Algorithms were once flat (Cholesky, 1910)



A. Cholesky

Sur la résolution numérique des systèmes d'équations linéaires

geodesic least squares problem

$$A = LL^T \text{ or } A = R^T R \text{ or } A = LDL^T$$

for $j = 1:n$ across columns

  for $i = 1:j-1$ top to diag of right factor

    $r_{ij} = (a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}) / r_{ii}$

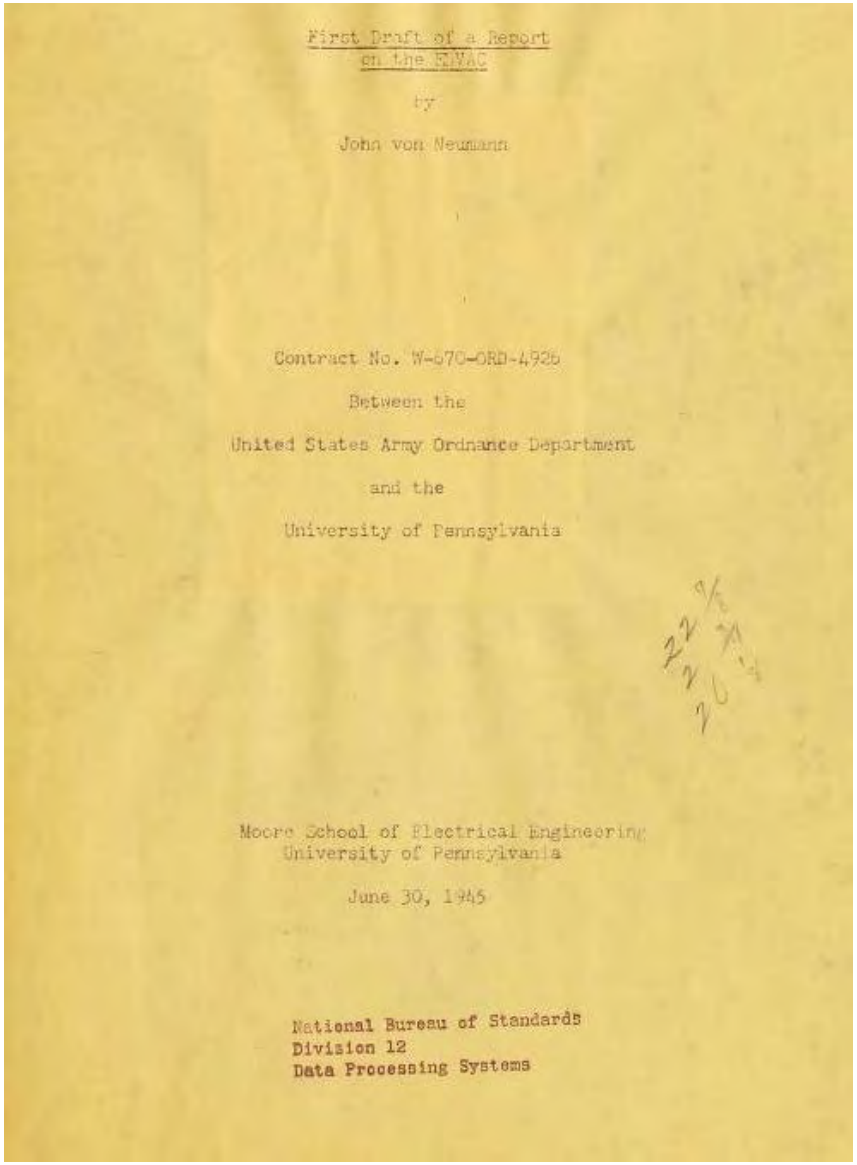
  end inner prod length "*i*"

  $r_{jj} = (a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2)^{1/2}$

end

classical global triangular loop, $O(n^3)$

Architectures were flat, as well (vN, 1945)



First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W-670-ORD-4926

Between the

United States Army Ordnance Department

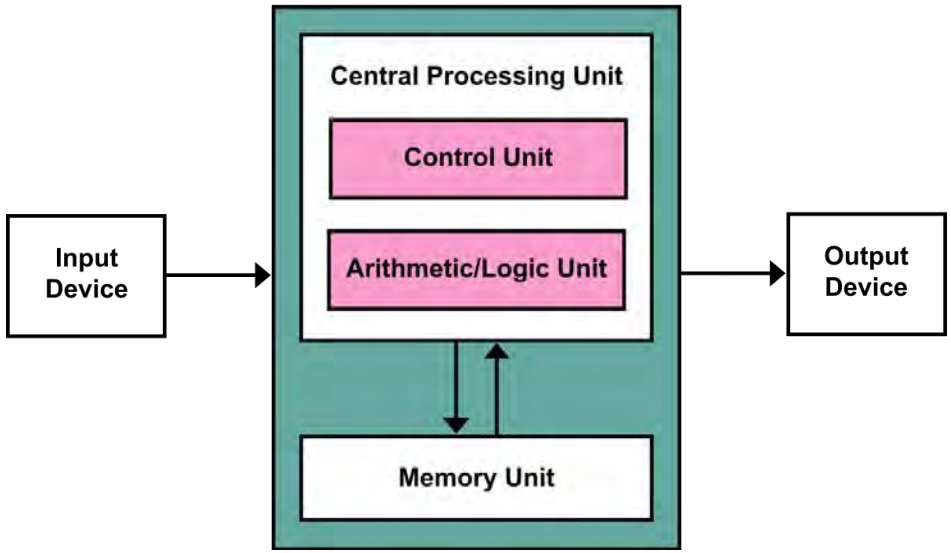
and the

University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

National Bureau of Standards
Division 12
Data Processing Systems



classical separation of ALU & memory

One hierarchy is not so bad…

As humans managing implementation complexity, we would prefer:

- **hierarchical algorithms on flat architectures**

or even (suboptimally)

- **flat algorithms on hierarchical architectures**

… but two independent hierarchies may not match

- **need to marshal irregular structures into uniform batches *and/or***
- **to feed dynamic runtime queues**
- **to best exploit hierarchical memory and heterogeneous accelerators**

Hierarchies may not perfectly match, but…

We go to exascale with the architectures we have, not with the architectures we want!

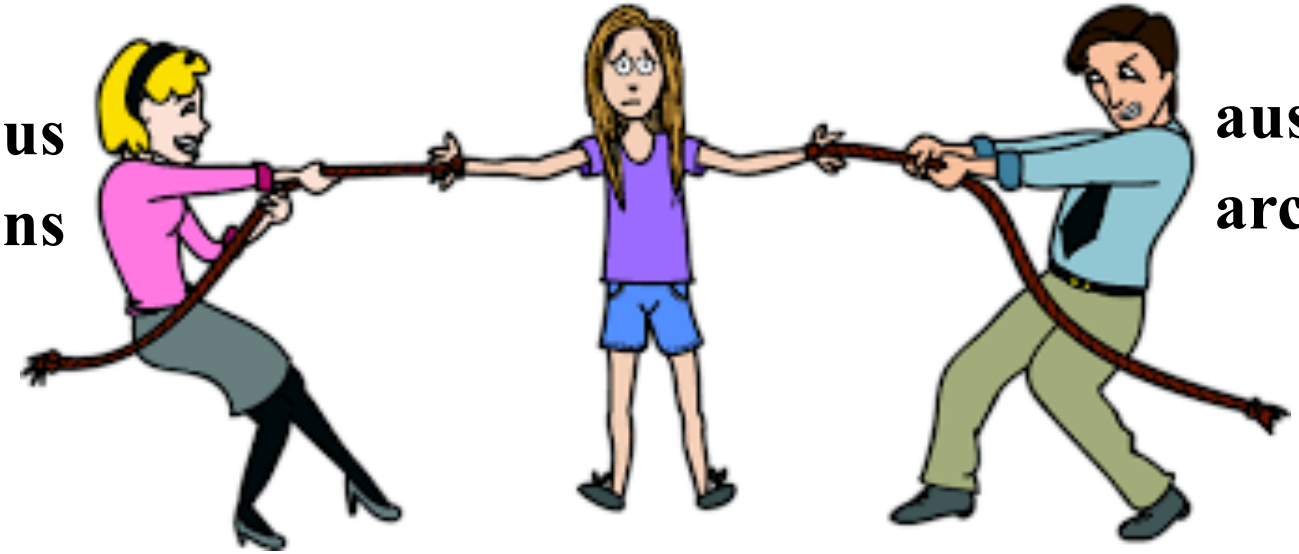
- First exascale Gordon Bell Prize (2018) awarded on the heterogeneous Summit system at ORNL (currently the #2 ranked system by HPL), with GPUs and Power9 cores
- A 4,000-node subset of Summit sustained 1.88 ExaOp/s of mixed precision on a genome-wide association studies (GWAS) application
- Majority of these operations are half-precision (16-bit floating point) NVIDIA tensor-core matrix-matrix multiplies, 64 FP FMADD operations per clock

Algorithmic philosophy

Algorithms must span a widening gulf …

adaptive algorithms

ambitious applications



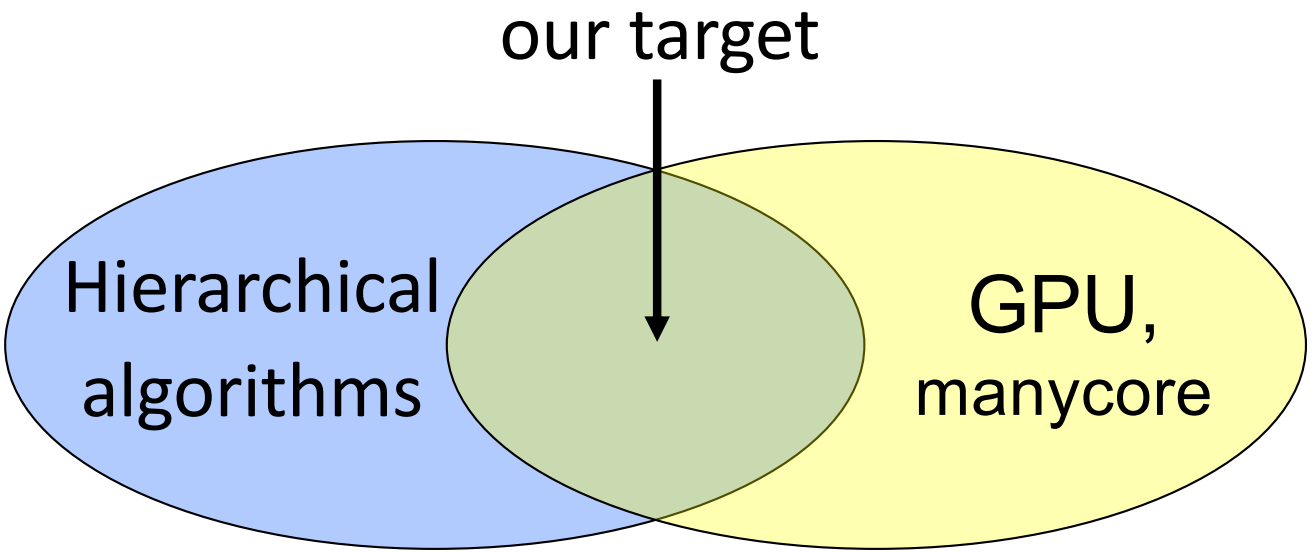
austere architectures

A full employment program for algorithm developers ☺

Hierarchical algorithms and extreme scale

Must address the *tension* between

- **highly uniform vector, matrix, and general SIMT operations – *prefer regularity and predictability***
- **hierarchical algorithms with tree-like data structures and scale recurrence – *possess irregularity and adaptability***



➔ Billions of

of investment worldwide in open source and commercial scientific software hangs in the balance until our algorithmic infrastructure evolves to span the architecture-applications gap

Required software

Model-related

- Geometric modelers
- Meshers
- Discretizers
- Partitioners
- Solvers / integrators
- Dynamic load balancers
- Discretization adaptors
- Data (de-)compressors
- Random no. generators
- Uncertainty quantifiers
- Graph & combinatorial operators
- Subgridscale physics machine learners

Development-related

- Build configurers
- Source-to-source translators
- Compilers
- Simulators
- Message passers
- Debuggers
- Profilers

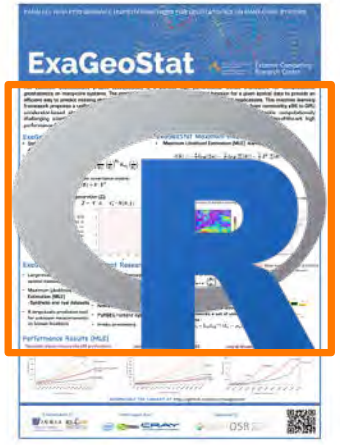
High-end computers come with little of this. Most is contributed by the user community.

Production-related

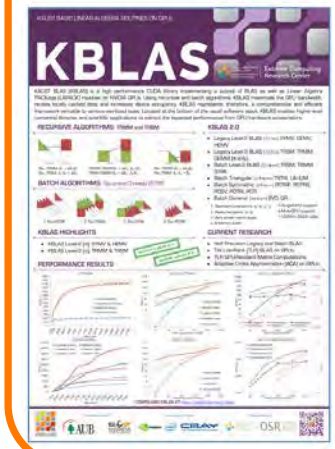
- Dynamic resource managers
- Dynamic performance optimizers
- Authenticators
- I/O optimizers
- Visualizers
- Workflow controllers
- Data miners
- Fault monitors & recoverers

Our modest contributions at

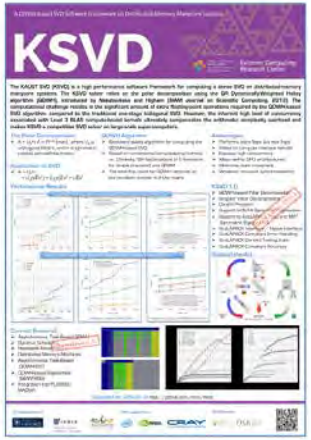
https://github.com/ecrc



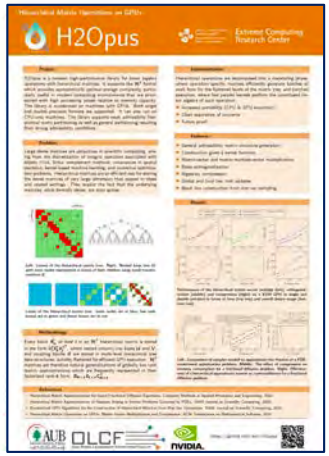
in NVIDIA cuBLAS

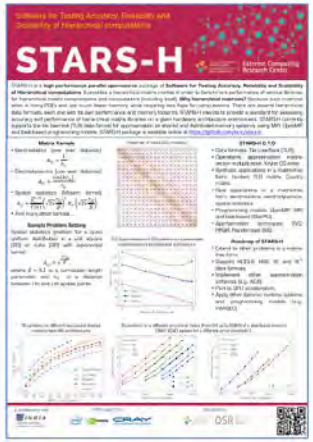


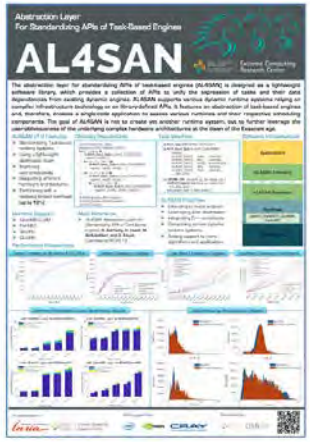
in Cray LibSci

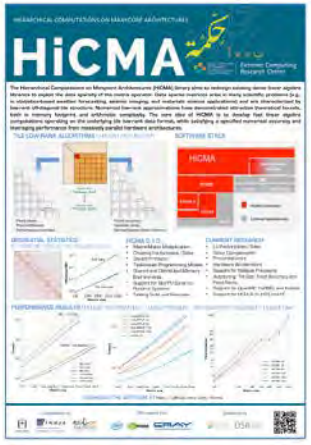


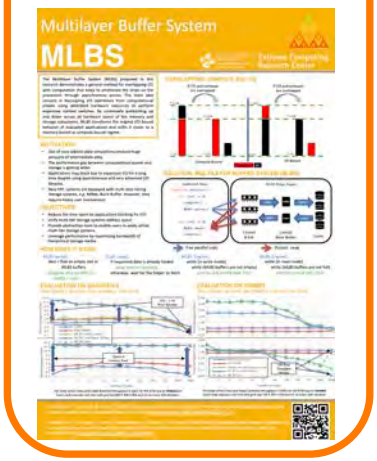
Aramco ExaWave







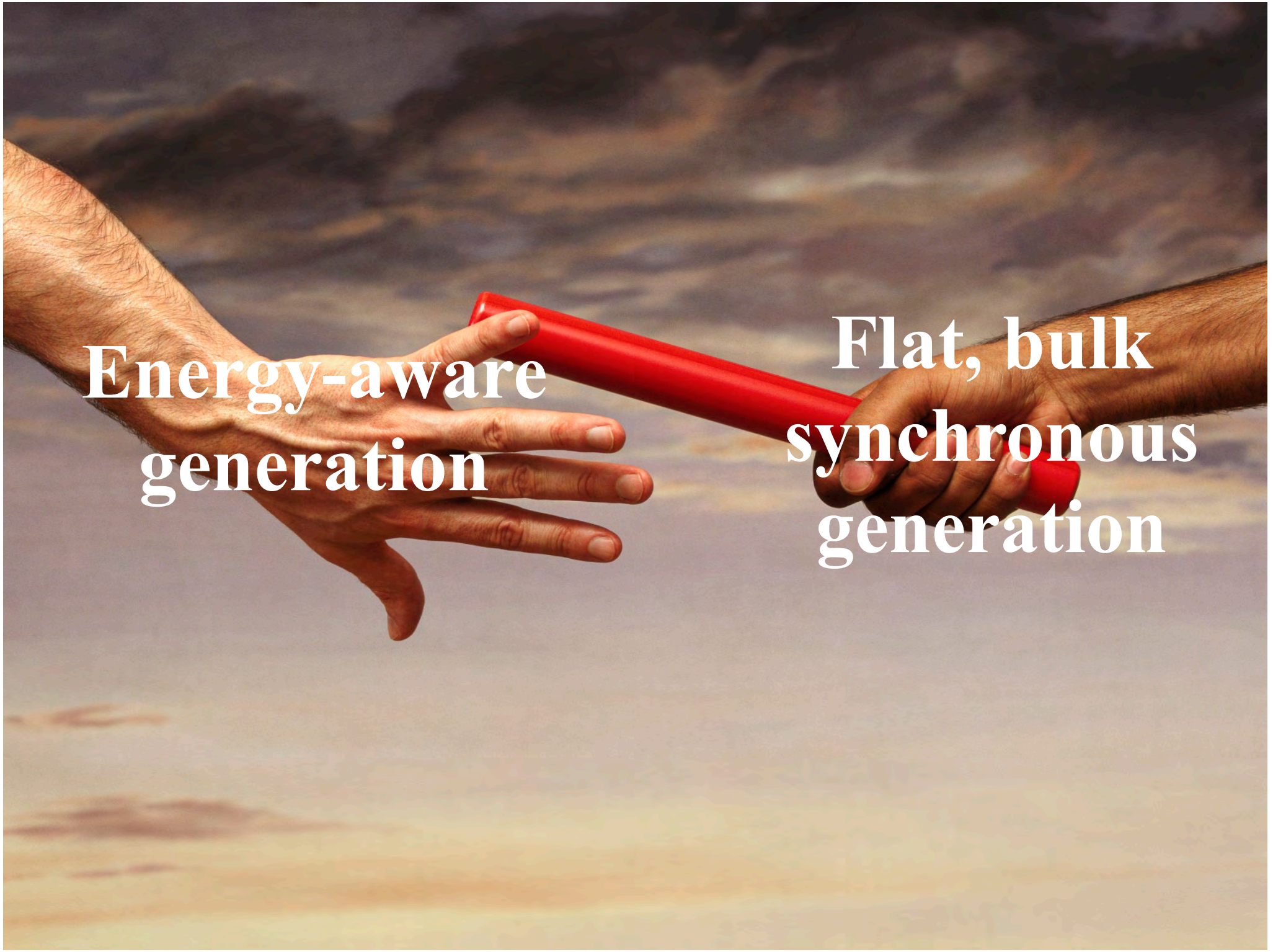




What will exascale algorithms look like?

- **Attempt to start with algorithms as close as possible to optimal asymptotic order,** $O(N \log^p N)$
- **Some such optimal (typically 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)**
  - **<What will you call your contribution?> (2020's)**

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

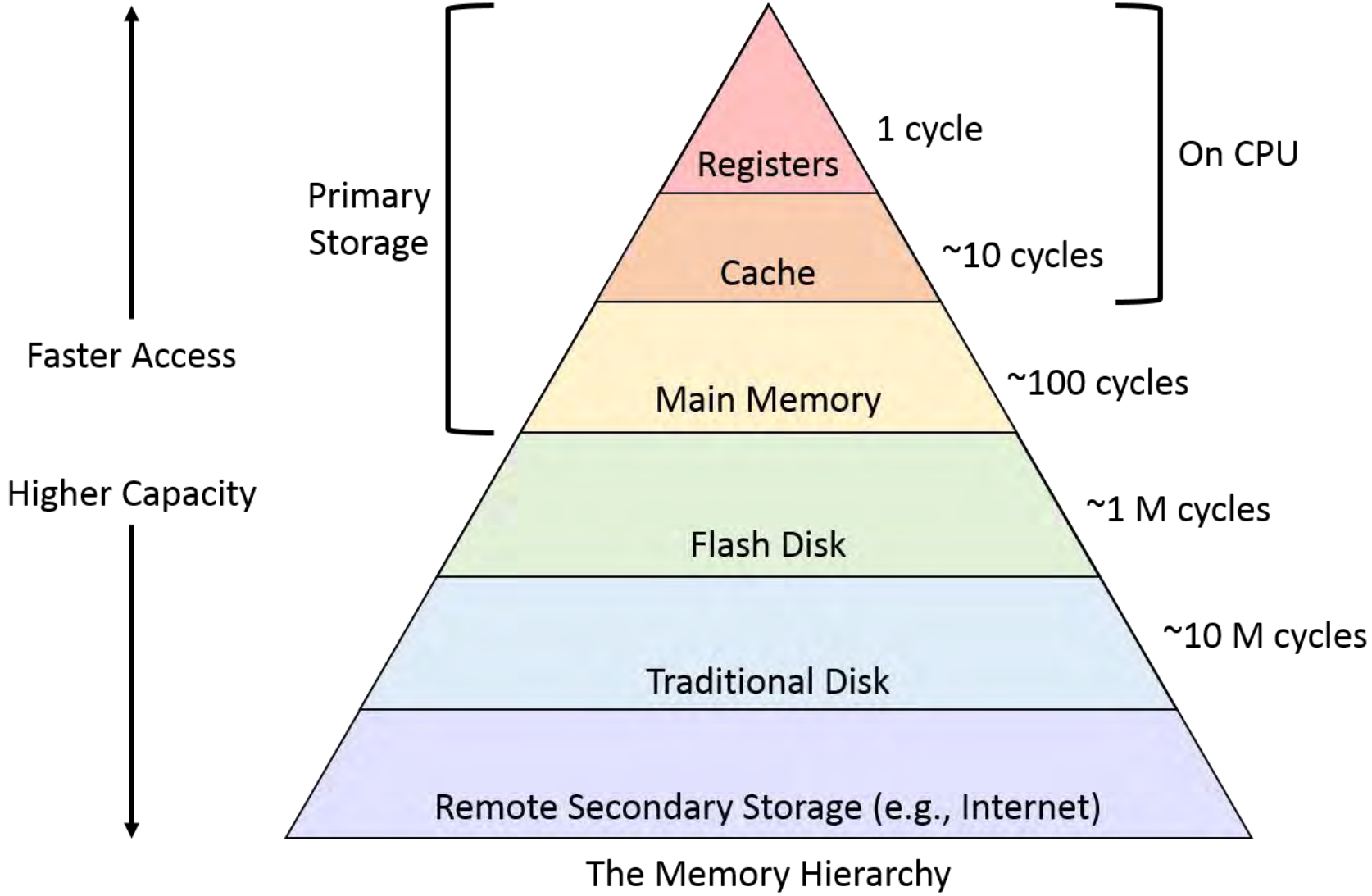

Energy-aware generation
Flat, bulk synchronous generation

Some “universals” of exascale computing

Architectural imperatives

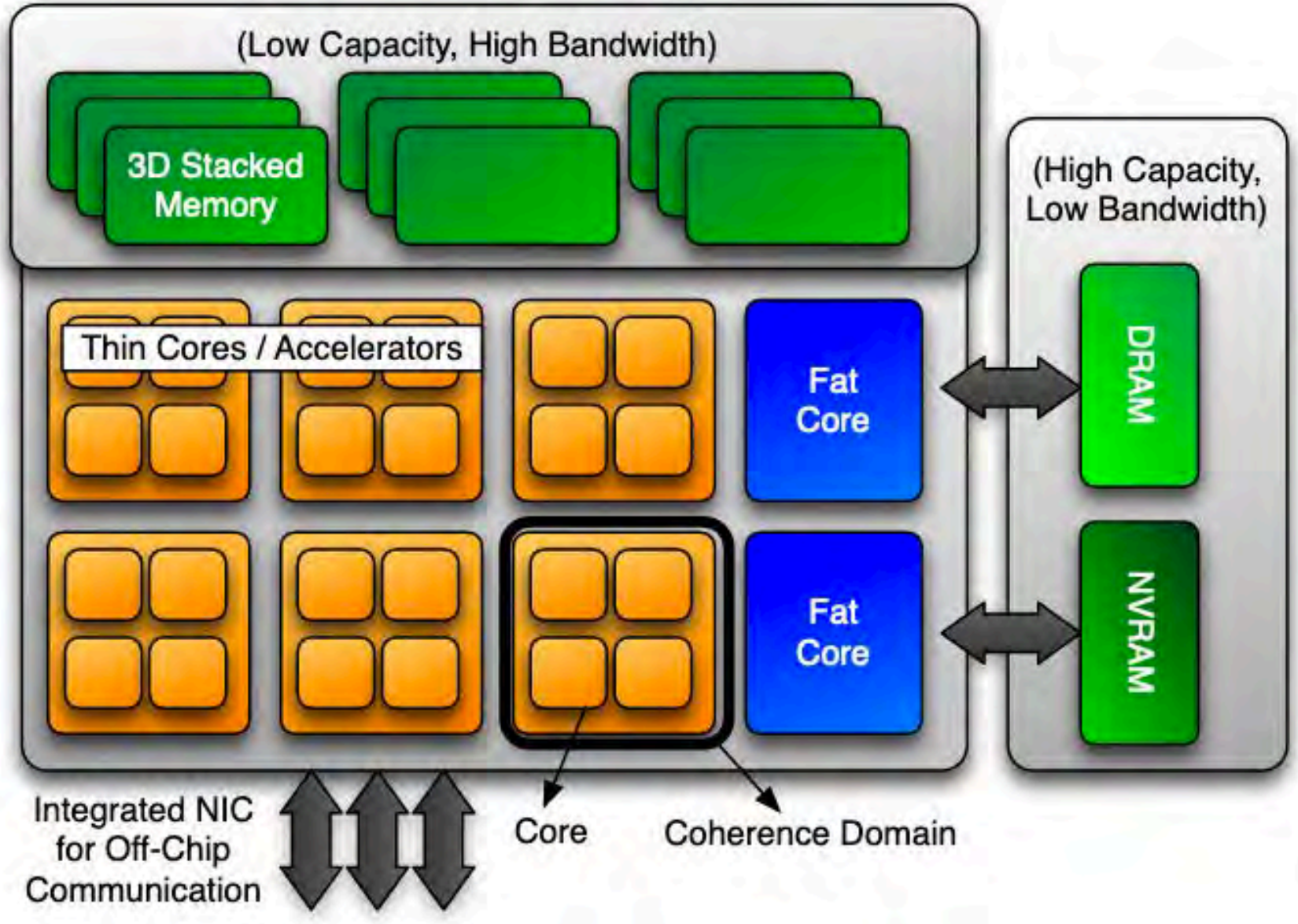
- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Classical memory hierarchy



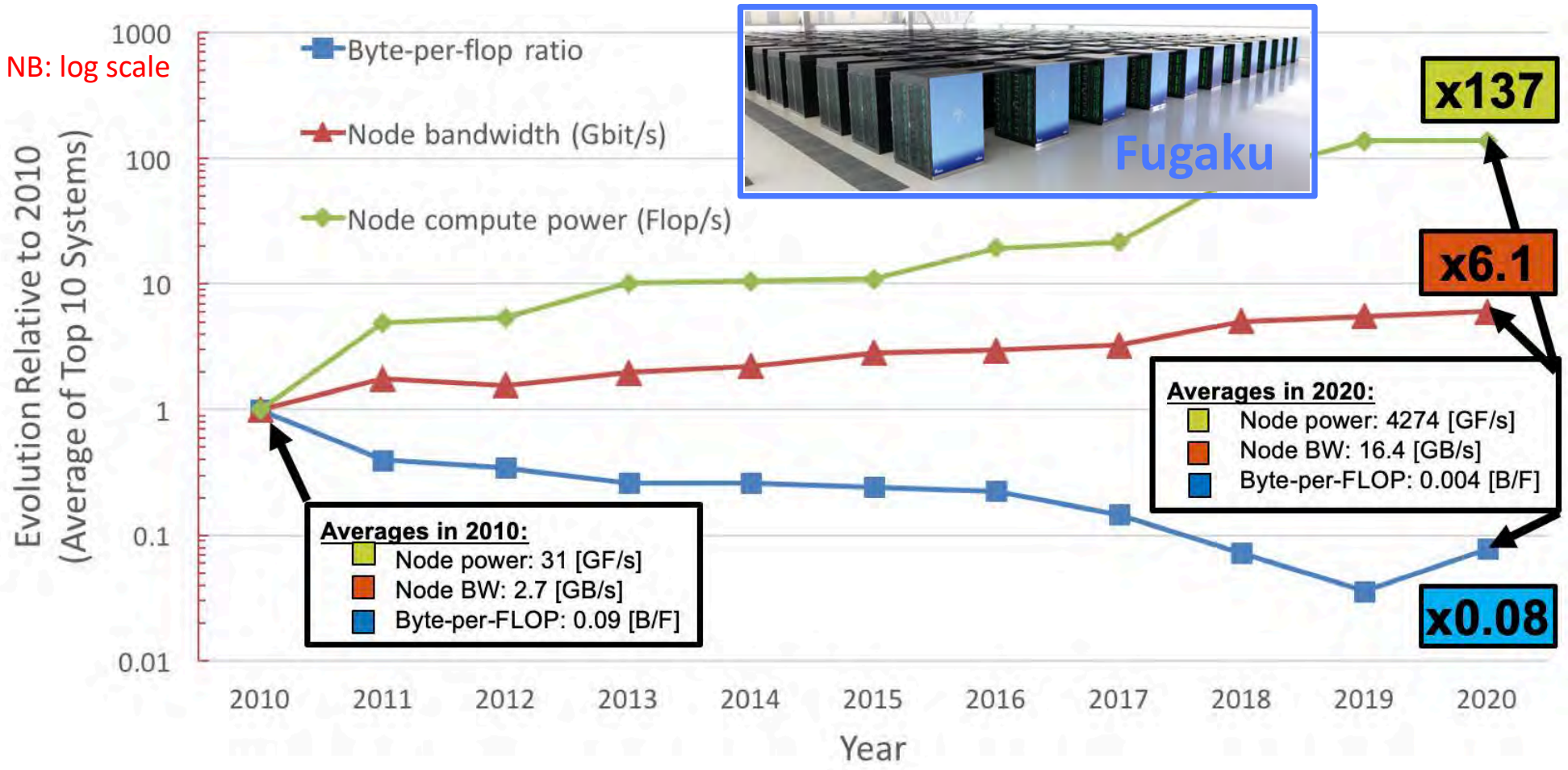
c/o K. Webb (2018)

Memory placement increasingly a user decision



c/o J. Ang et al (2014)

HPL Top 10 memory BW trends, 2010-2020



Keren Bergman's lab at Columbia has been tracking architectural trends in memory and networking interconnects for two decades. This slide is updated for Fugaku.

The last three #1 systems

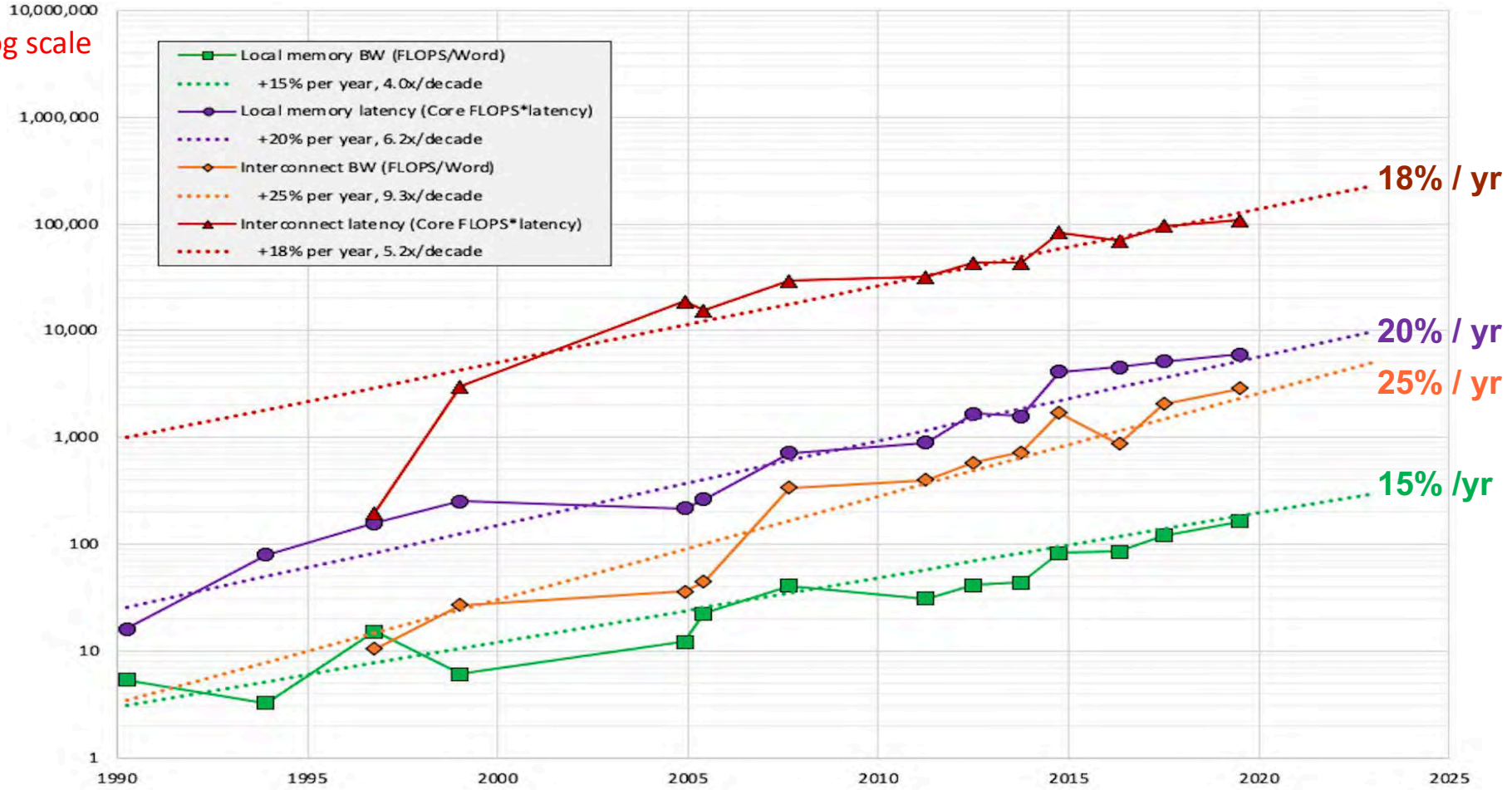
TaihuLight (Nov 2017) B/F = 0.004

Summit (June 2018) B/F = 0.0005

Fugaku (June 2020) B/F = 0.303

Single-node speeds/feeds ratios, 1990-2020

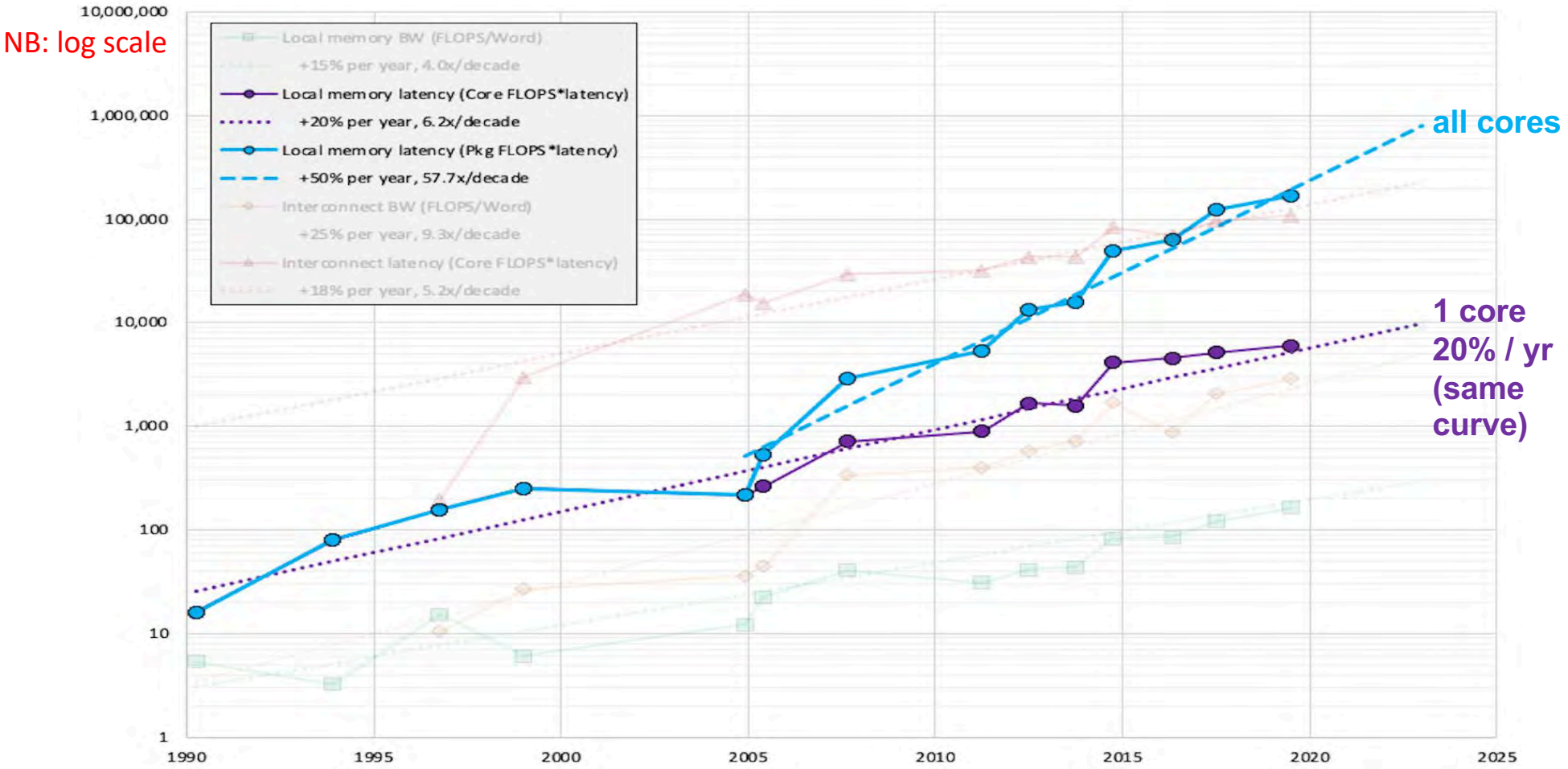
NB: log scale



John McCalpin, now at TACC, has been tracking architectural trends through the STREAM benchmark since 1990, when he noticed that code loops that gave 90% peak on Cray gave less than 10% on RISC

- BW based on node level GF/s divided by node level sustainable BW (memory or network)
- Latency based on GF/s for one core and latency for "load" to local memory or "get" from another node

On-node memory latency, 1990-2020



What happens if all cores stall on a local memory latency?

- **50% / yr increase reflects increase in # cores per socket package**
- **This worse-than-single-core scenario prevails, for example, if an OpenMP coordinating thread is in a serial section while the other cores are idle, and gets worse with flooding of cores per socket**

Why exa-… is hard

Moore's Law (1965) has not fully ended
but Dennard's MOSFET scaling (1972) has

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 decades

Dennard et al., *IEEE J. Solid-State Circuits* (1974)

Typical power costs per operation

Operation	approximate energy cost
DP FMADD flop	100 pJ
DP DRAM read-to-register	5,000 pJ
DP word transmit-to-neighbor	7,500 pJ
DP word transmit-across-system	10,000 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)

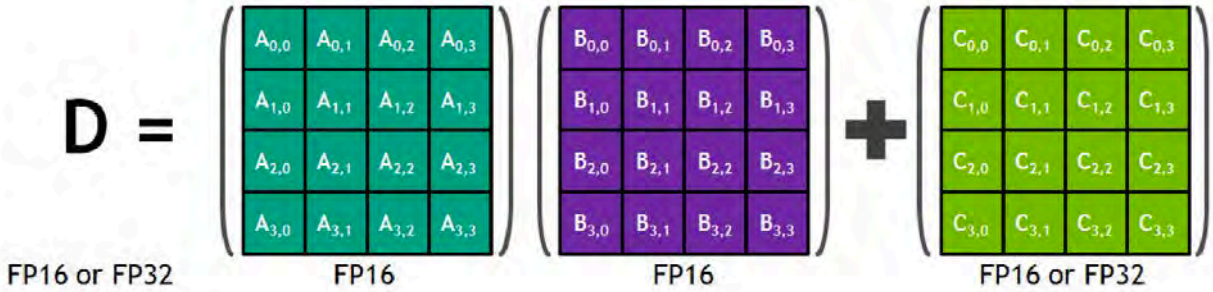
Some “universals” of exascale computing

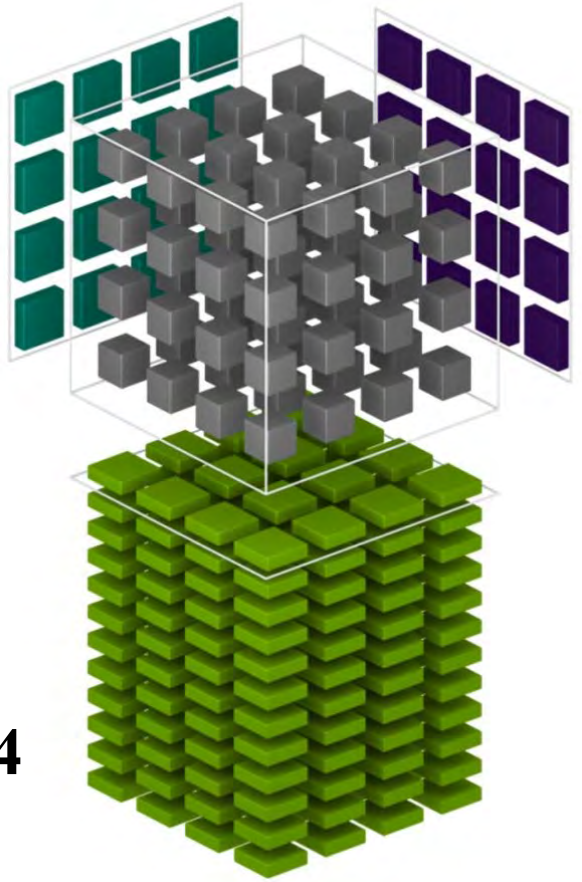
Architectural imperatives

- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Rely on SIMD/SIMT tasks

- **Many specialized operations are now hard-wired, e.g.,**
  - **traditional vector triadic operations**
  - **matrix-matrix operations used in DL**

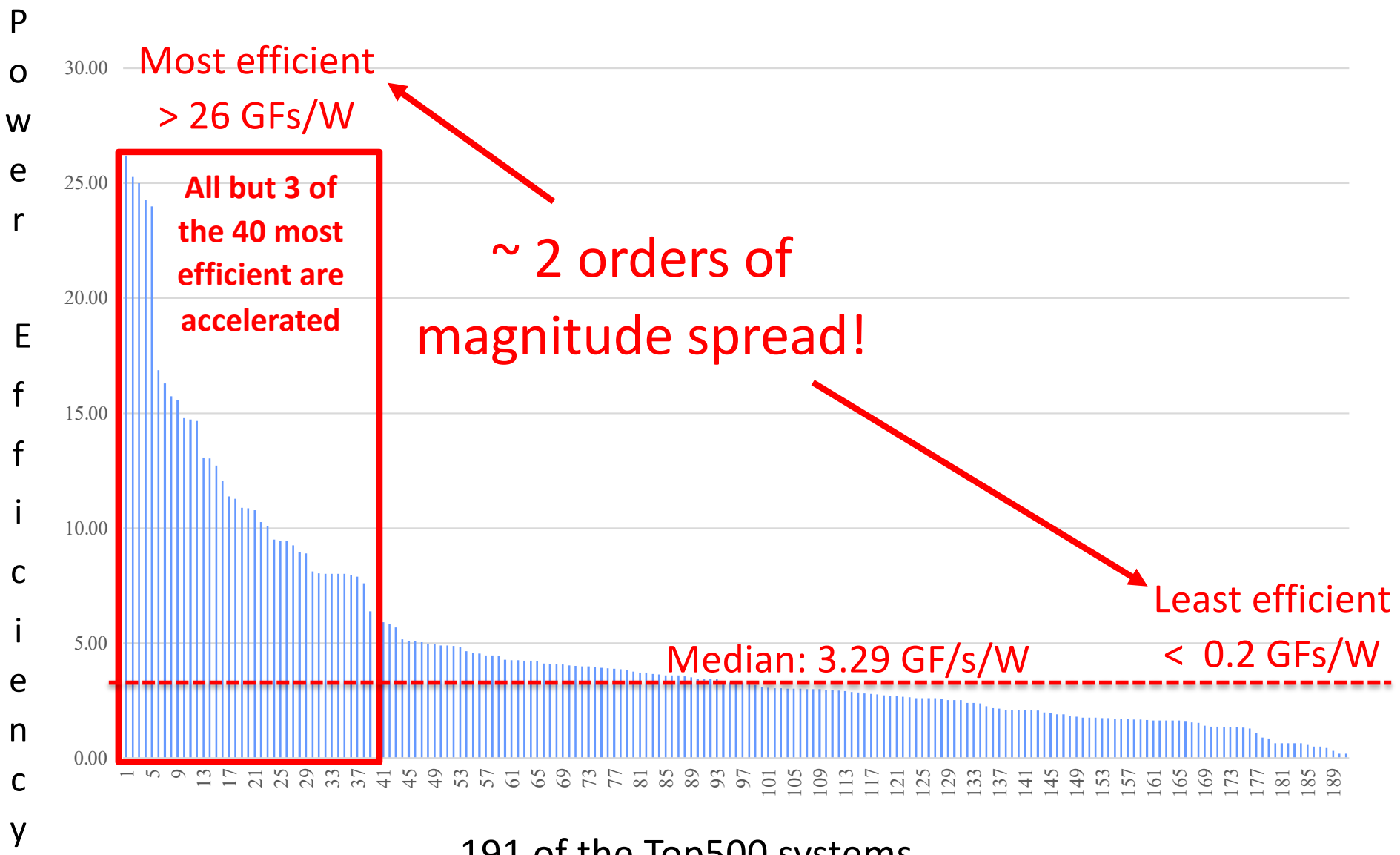




- **Such instructions cannot be ignored**
  - **4x4 matrix-matrix multiply-add does 64 FMADD instructions in one clock cycle**
  - **varieties of scales and precisions abound**
  - **more than an order of magnitude efficiency at stake**

Power efficiencies

(191 entries of Nov 2020 Top500 report efficiency rating)



Specialization includes precision choice

$$D = C + AB$$

Year of release	Device	Matrix dimensions	Input format	Output format
2016	Google TPU v2	$128 \times 128 \times 128$	bfloat16	fp32
2017	Google TPU v3	$128 \times 128 \times 128$	bfloat16	fp32
2017	NVIDIA V100	$4 \times 4 \times 4$	fp16	fp32
2018	NVIDIA T4	$4 \times 4 \times 4$	fp16	fp32
2019	ARMv8.6-A	$2 \times 4 \times 2$	bfloat16	fp32
2020	NVIDIA A100	$8 \times 8 \times 4$	bfloat16	fp32
		$8 \times 8 \times 4$	fp16	fp32
		$4 \times 2 \times 2$	fp64	fp64
		$4 \times 8 \times 4$	TensorFloat-32	fp64

Each halving of precision generally doubles execution rate

- **sometimes more than 2x from higher memory residency for given no. of elements**

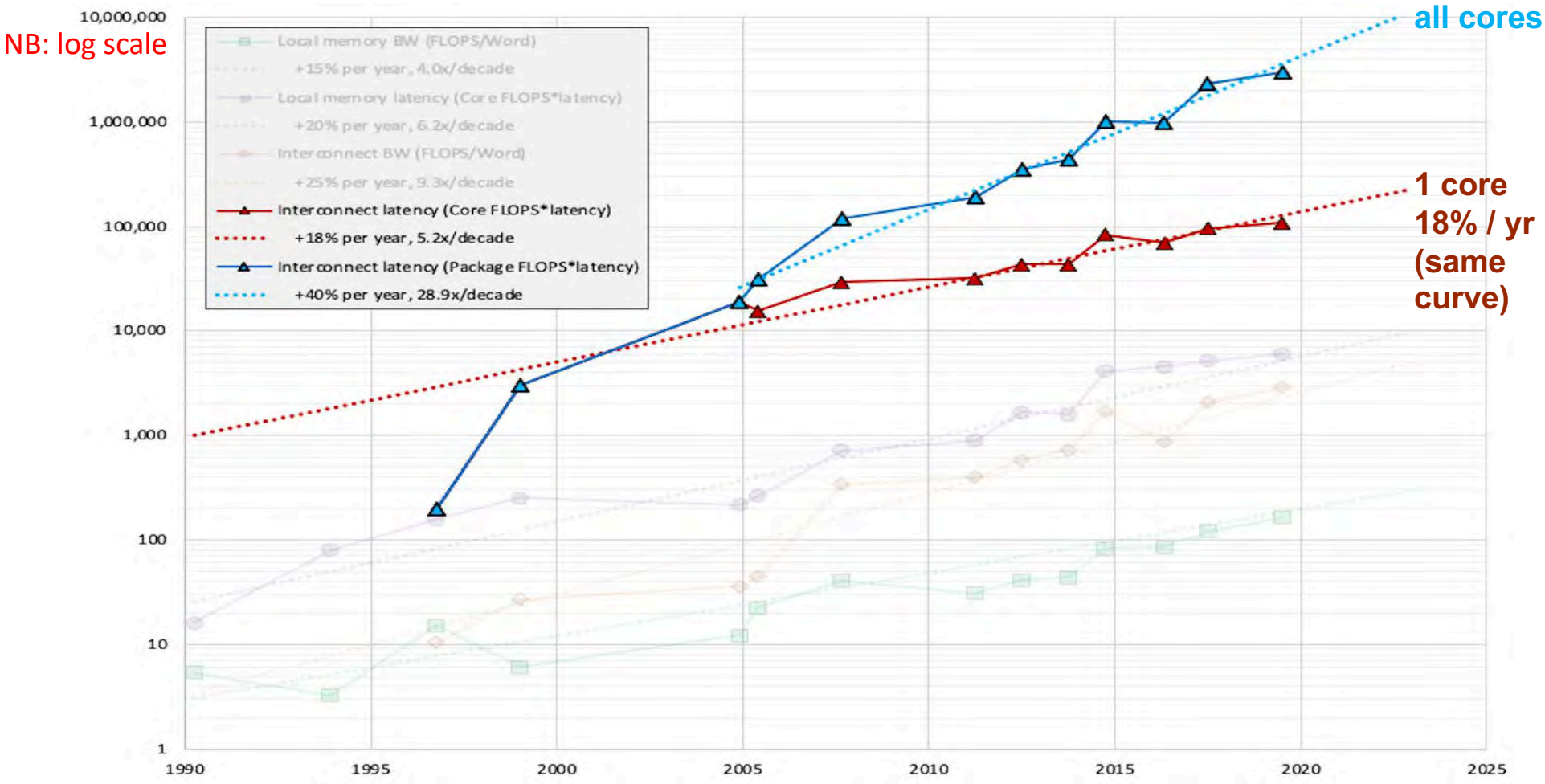
c/o Nick Higham (2021)

Some “universals” of exascale computing

Architectural imperatives

- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Off-node data latency, 1990-2020



What happens if all cores stall on a network latency?

- **Duration of the stalls will be something like $\log_2(P)$ network latencies**
- **This is common during MPI collective operations that synchronize all participating cores, e.g., inner products, norms, barriers**
- **"Bandwidth is limited by money, but latency is limited by physics"**

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.

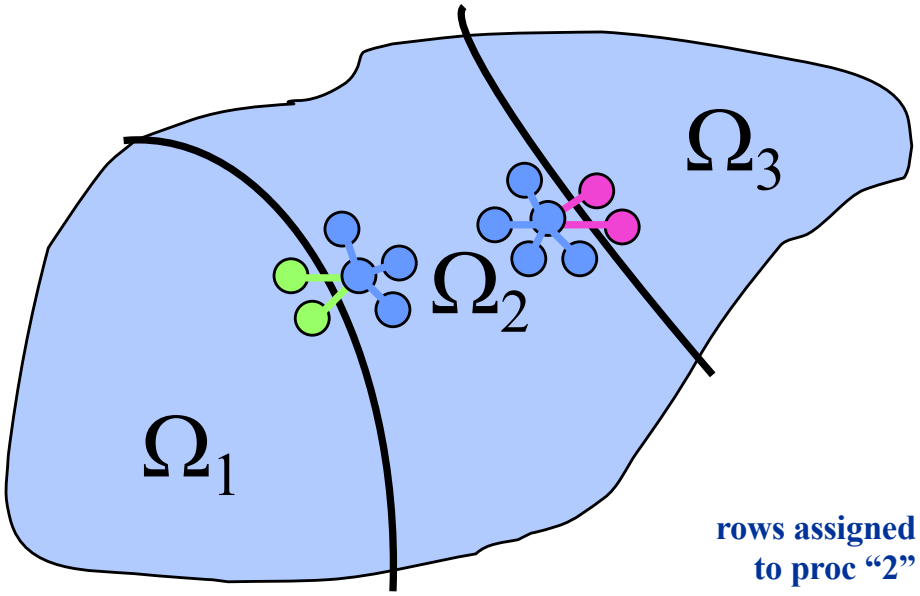
Leslie G. Valiant

Communications of the ACM, 1990

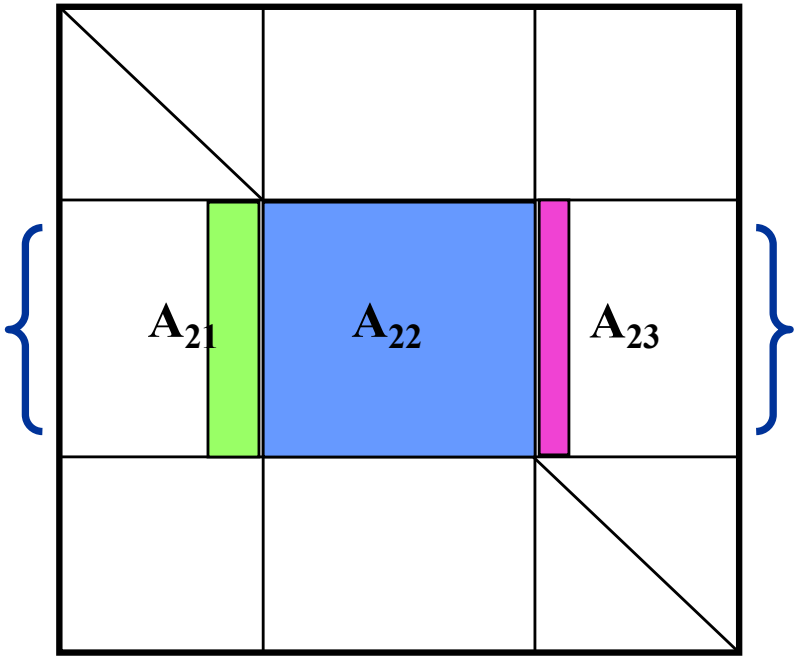
How are most simulations implemented at the petascale today?

- **Iterative methods based on data decomposition and message-passing**
  - **data structures (e.g., grid points, particles, agents) are distributed**
  - **each individual processor works on a subdomain of the original ("owner computes")**
  - **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



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

Extrapolating exponentials eventually fails

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)**

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 threads**
- **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 10.6 million on TaihuLight (currently #4 overall)**

Some “universals” of exascale computing

Architectural imperatives

- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Motivation to communicate less

2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Preliminary Performance Analysis of Multi-rail Fat-tree Networks

Noah Wolfe*, Misbah Mubarak†, Nikhil Jain‡, Jens Domke§, Abhinav Bhatele‡,
Christopher D. Carothers*, Robert B. Ross†

*Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York
†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois
‡Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California
§Institute of Computer Engineering, Technische Universität Dresden, Dresden, Germany

***Abstract*—Among the low-diameter, high-radix networks being deployed in next-generation HPC systems, dual-rail fat-tree networks are a promising approach. Adding additional injection connections (rails) to one or more network planes allows multi-rail fat-tree networks to alleviate communication bottlenecks. These multi-rail networks necessitate new design considerations, such as routing choices, job placements, and scalability of rails. We extend our fat-tree network model in the CODES parallel simulation framework to support multi-rail and multi-plane configurations in addition to different types of static routing, resulting in a powerful research vehicle for fat-tree network analysis. Our detailed packet-level simulations use communication traces from real applications to make performance predictions and to evaluate the impact of single- and multi-rail networks in conjunction with schemes for injection rail selection and intra-plane routing.**

jobs executing in parallel using different HPC job placement schemes that represent potential full system workloads. In order to mimic the large compute node performance, we map multiple MPI processes to a node. We use our detailed packet-level network model to evaluate the effectiveness of a dual-rail dual-plane network in improving performance by comparing with a corresponding single-rail single-plane network.

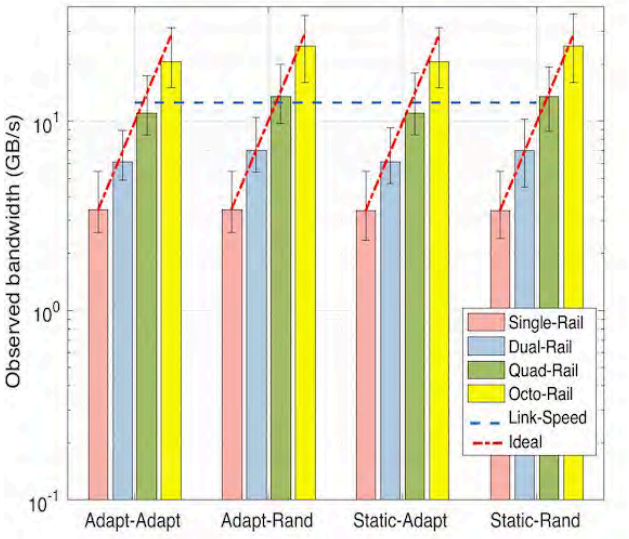
II. Background

In this section, we describe the functionality provided by the CODES and ROSS simulation frameworks and the communication patterns of the applications used for the evaluation.

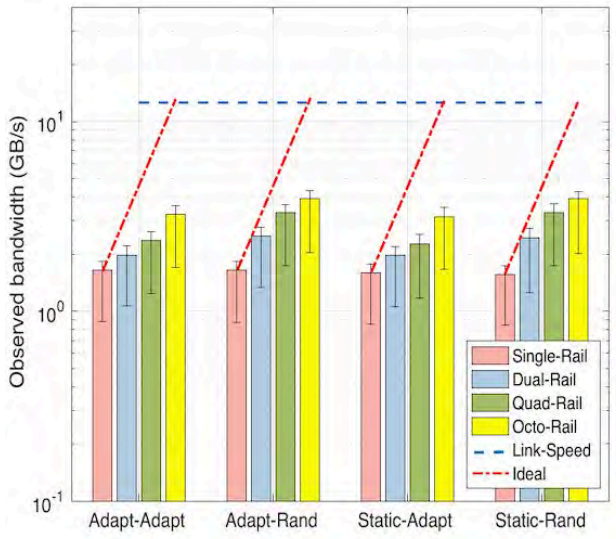
A. ROSS Discrete-event Simulator

How 3 solvers exploit more bandwidth

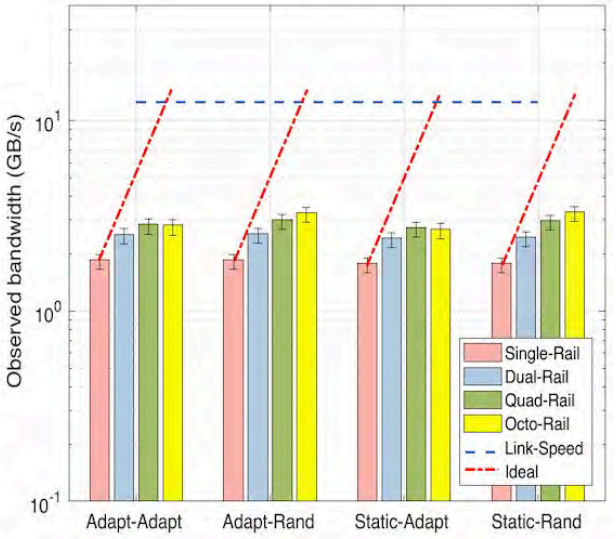
Geometric MG (LBNL)
many-to-many msgs, 5% of runtime



Algebraic MG (LLNL)
many small msgs, 40% of runtime



Spectral (ANL)
large msgs, 68.5% runtime



- **Improvements resulting from additional rails in a fat-tree network depend on the application's communication pattern**
- **For some apps, reduction in communication, not more bandwidth is the only alternative for runtime improvements**
- **Applications sending large numbers of small packets with fewer synchronization points (left) can see major improvements**
- **Applications transferring small numbers of larger packets with frequent synchronization (right) see diminished improvement**

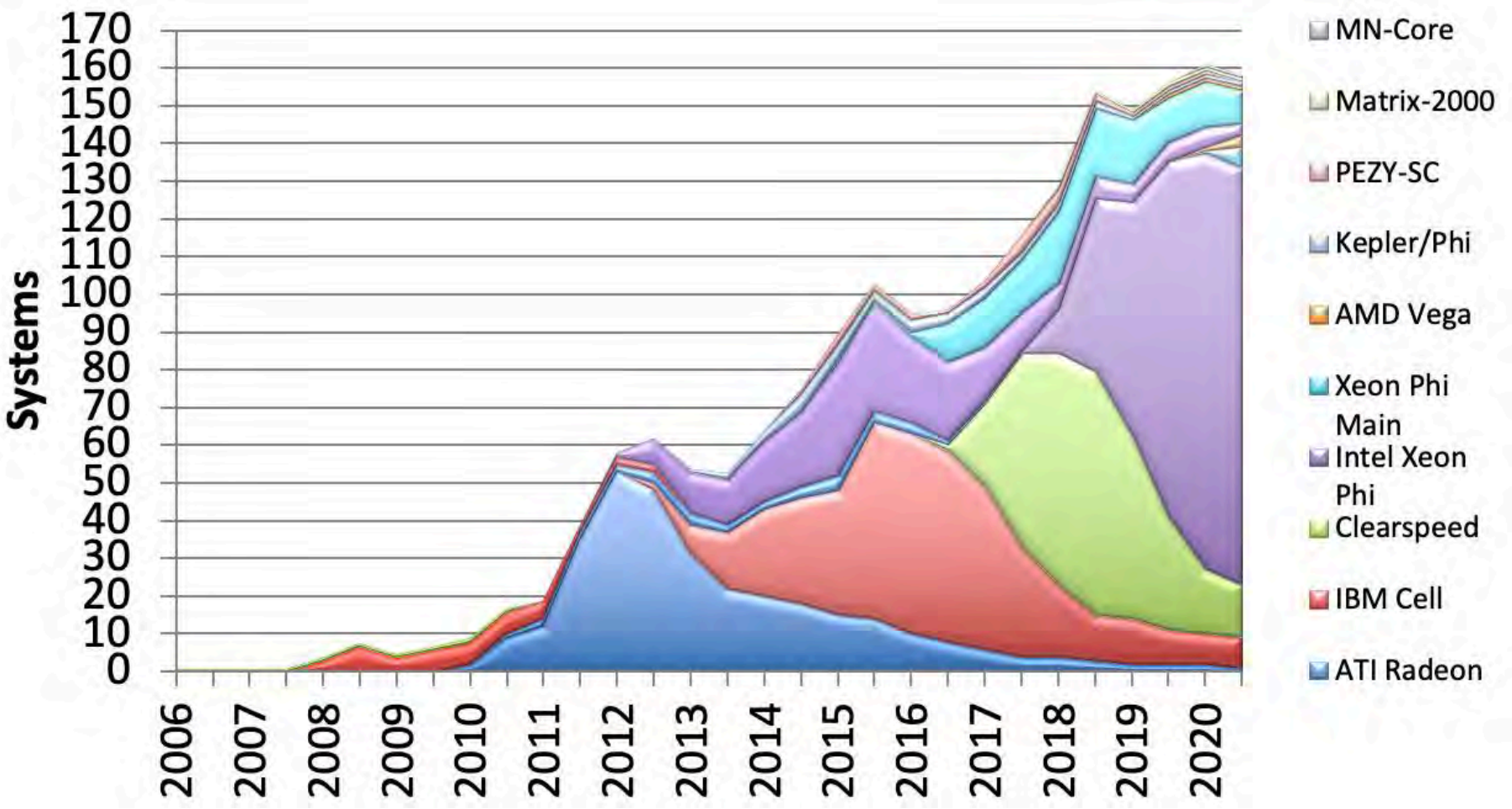
c/o Jens Domke (RIKEN, 2021)

Some “universals” of exascale computing

Architectural imperatives

- Reside “high” on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

Heterogeneity is taking over (top of) Top500

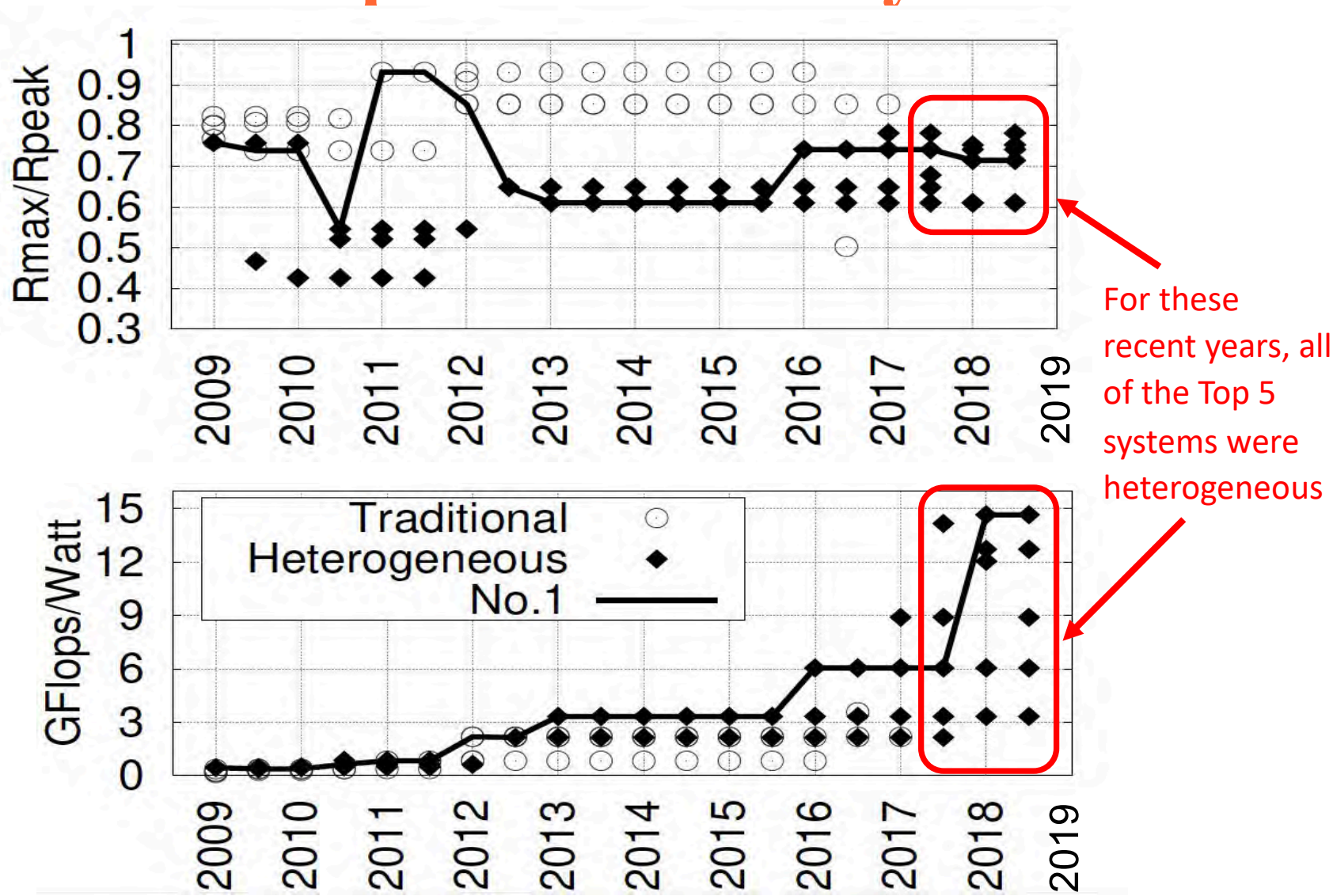


Nearly one-third of the Top500 systems exploit accelerators

- **disproportionally concentrated at the top of the list**

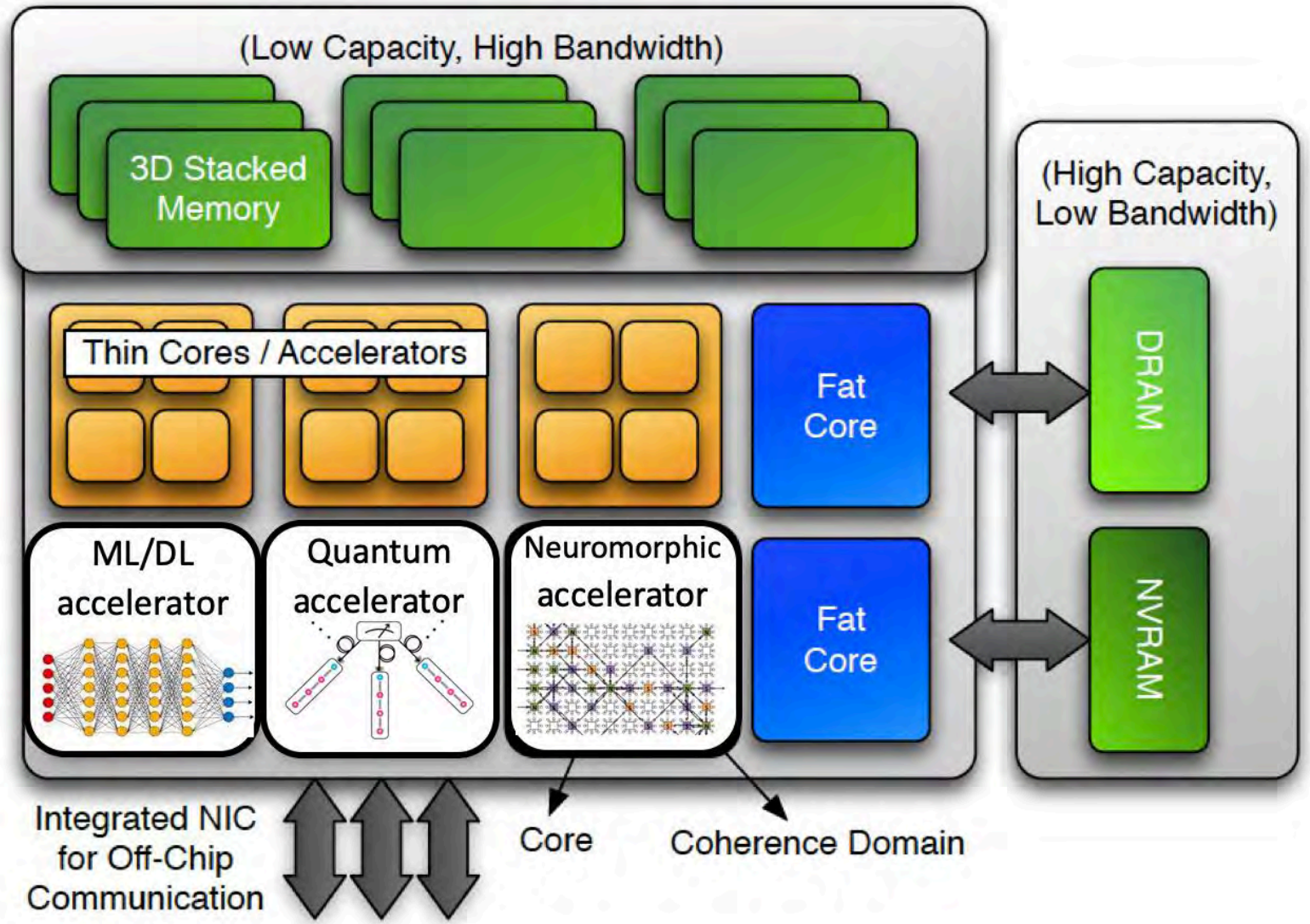
c/o Erich Strohmeier (LBNL, 2020)

Heterogenous HPL performance and power efficiency



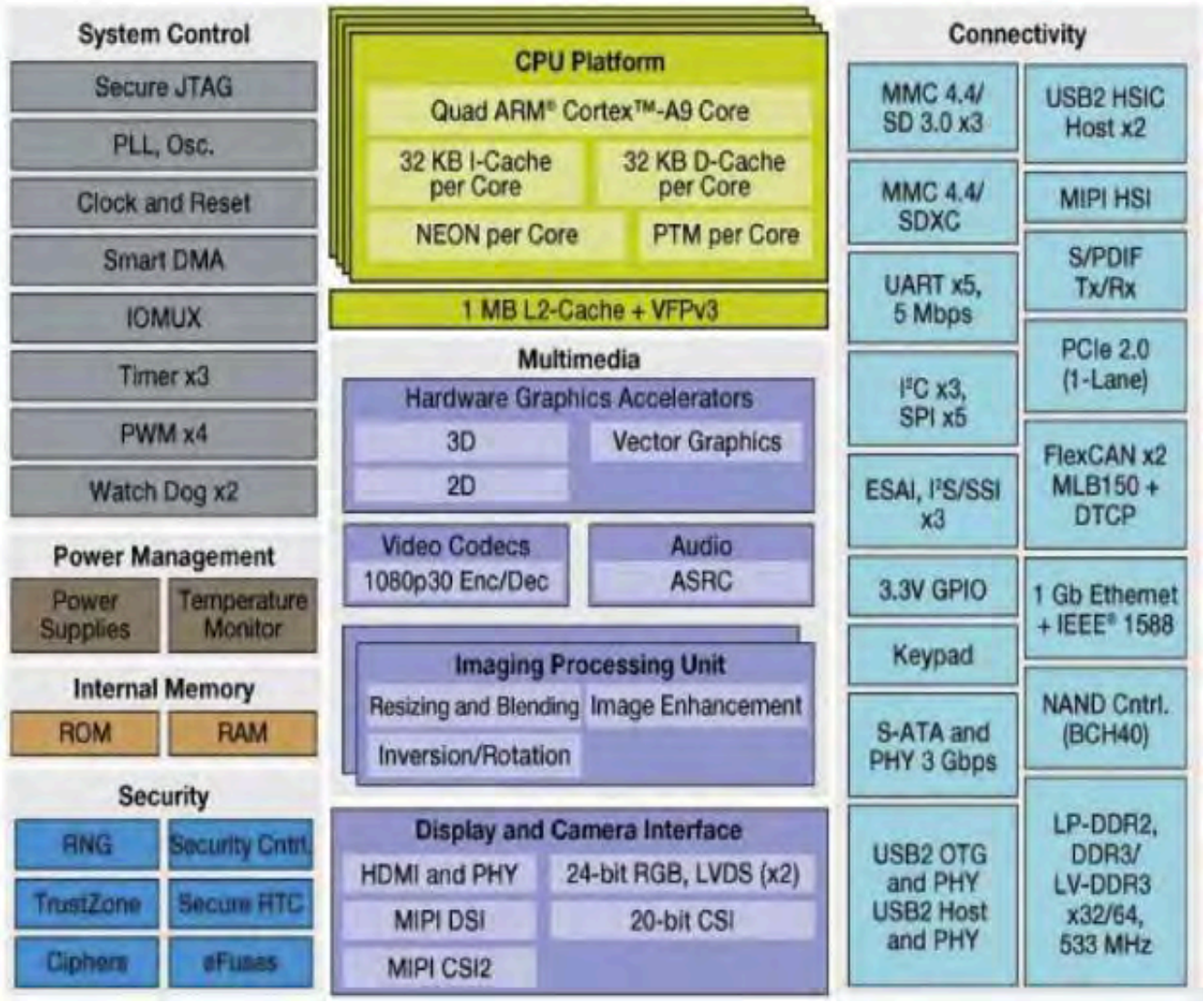
c/o H. Sim, S. Vazhkudai & A Khan (ORNL, 2020)

Exploit heterogeneity



after J. Ang et al (Sandia, 2014)

Heterogeneity in today's smart phone



Typical smart phone has 40+ special processors

c/o John Shalf (LBNL, 2021)

Some “universals” of exascale computing

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Exploit extra memory to reduce comm

SIAM J. MATRIX ANAL. & APPL.
Vol. 32, No. 3, pp. 866–901

MINIMIZING COMMUNICATION IN NUMERICAL LINEAR ALGEBRA*

GREY BALLARD†, JAMES DEMMEL‡, OLGA HOLTZ§, AND ODED SCHWARTZ¶

Abstract. In 1981 Hong and Kung proved a lower bound on the amount of communication (amount of data moved between a small, fast memory and large, slow memory) needed to perform dense, n-by-n matrix multiplication using the conventional $O(n^3)$ algorithm, where the input matrices were too large to fit in the small, fast memory. In 2004 Irony, Toledo, and Tiskin gave a new proof of this result and extended it to the parallel case (where communication means the amount of data moved between processors). In both cases the lower bound may be expressed as $\Omega(\#\text{arithmetic\_operations}/\sqrt{M})$, where M is the size of the fast memory (or local memory in the parallel case). Here we generalize these results to a much wider variety of algorithms, including LU factorization, Cholesky factorization, LDL^T factorization, QR factorization, the Gram–Schmidt algorithm, and algorithms for eigenvalues and singular values, i.e., essentially all direct methods of linear algebra. The proof works for dense or sparse matrices and for sequential or parallel algorithms. In addition to lower bounds on the amount of data moved (bandwidth cost), we get lower bounds on the number of messages required to move it (latency cost). We extend our lower bound technique to compositions of linear algebra operations (like computing powers of a matrix) to decide whether it is enough to call a sequence of simpler optimal algorithms (like matrix multiplication) to minimize communication, or whether we can do better. We give examples of both. We also show how to extend our lower bounds to certain graph-theoretic problems. We point out recently designed algorithms that attain many of these lower bounds.

Exploit extra memory to reduce comm

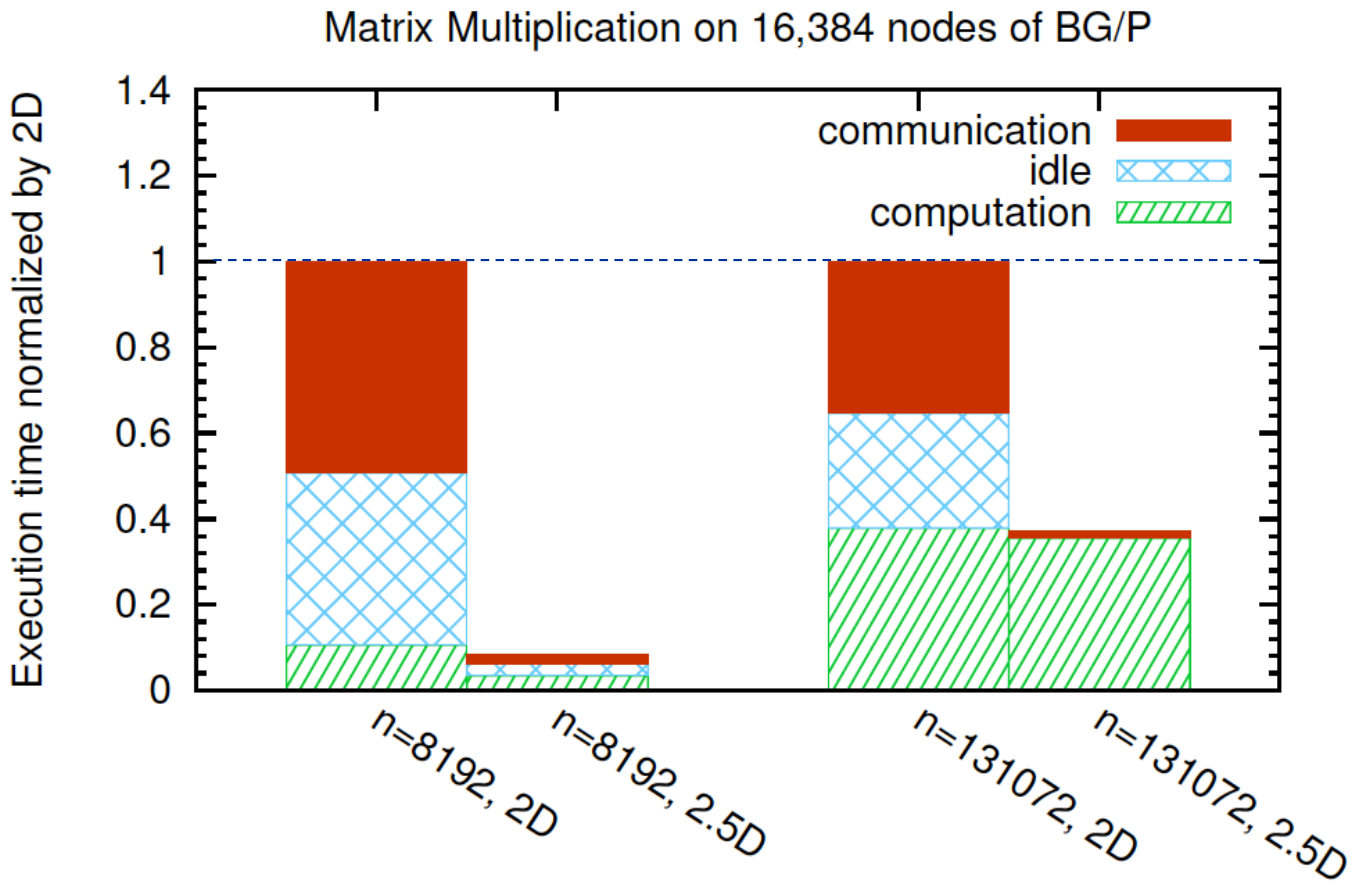


Figure 3.2. 2.5D matrix multiplication on BG/P, 16K nodes / 64K cores.

Some “universals” of exascale computing

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Perform extra flops to synchronize less

Parallel Computing 40 (2014) 224–238

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm

P. Ghysels [a,b,*], W. Vanroose [a]

[a] University of Antwerp, Department of Mathematics and Computer Science, Middelheimlaan 1, B-2020 Antwerp, Belgium
[b] Intel ExaScience Lab, Kapeldreef 75, B-3001 Leuven, Belgium

ARTICLE INFO

Article history:
Available online 24 June 2013

Keywords:
Conjugate gradients
Parallelization
Global communication
Latency hiding
Conjugate residuals

ABSTRACT

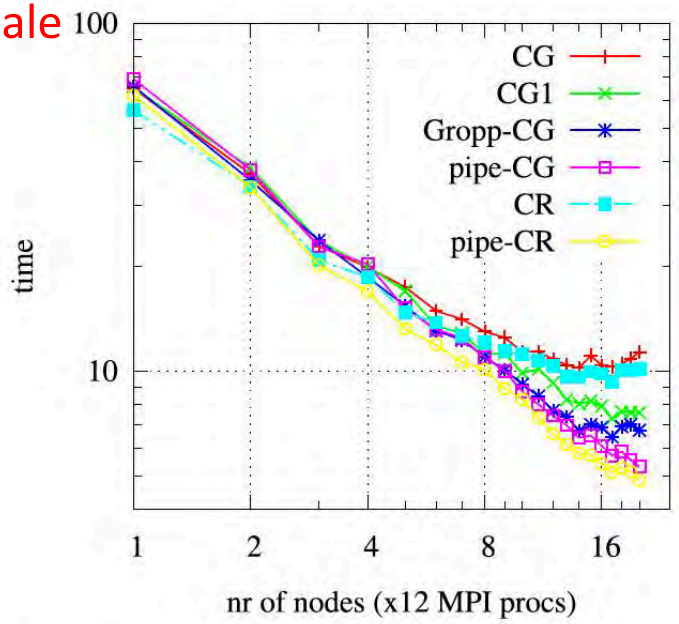
Scalability of Krylov subspace methods suffers from costly global synchronization steps that arise in dot-products and norm calculations on parallel machines. In this work, a modified preconditioned Conjugate Gradient (CG) method is presented that removes the costly global synchronization steps from the standard CG algorithm by only performing a single non-blocking reduction per iteration. This global communication phase can be overlapped by the matrix–vector product, which typically only requires local communication. The resulting algorithm will be referred to as *pipelined CG*. An alternative pipelined method, mathematically equivalent to the Conjugate Residual (CR) method that makes different trade-offs with regard to scalability and serial runtime is also considered. These methods are compared to a recently proposed asynchronous CG algorithm by Gropp. Extensive numerical experiments demonstrate the numerical stability of the methods. Moreover, it is shown that hiding the global synchronization step improves scalability on distributed memory machines using the message passing paradigm and leads to significant speedups compared to standard preconditioned CG.

Perform extra flops to synchronize less

Table 1
Overview of the different preconditioned CG and CR variations. Column *flops* lists the number of flops ($\times N$) for AXPYS and dot-products. The *time* column has the time spent in global all-reduce communication (G), in the matrix–vector product (SPMV) and the preconditioner (PC). Column *#global synchronizations* has the number of global communication phases per iteration. The *memory* column counts the number of vectors that need to be kept in memory (excluding x and b).

	Flops	Time (excl, AXPYS, DOTS)	#Glob syncs	Memory
CG	10	2G + SpMV + PC	2	4
Chron/Gear-CG	12	G + SpMV + PC	1	5
CR	12	2G + SpMV + PC	2	5
Pipe-CG	20	max(G, SpMV + PC)	1	9
Pipe-CR	16	max(G, SpMV) + PC	1	7
Gropp-CG	14	max(G, SpMV) + max(G, PC)	2	6

NB: log scale



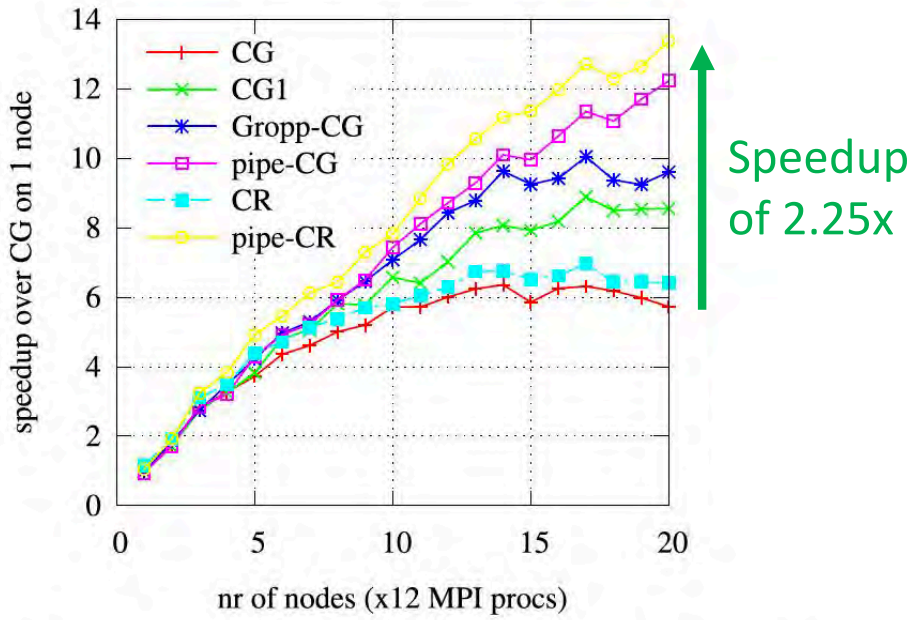


Fig. 3. Left: Time to solution for the 3*D* hydrostatic ice sheet flow simulation using $100 \times 100 \times 50$ *Q*1 finite elements. Right: Speedup as function of number of nodes over standard CG solver on a single node.

Some “universals” of exascale computing

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

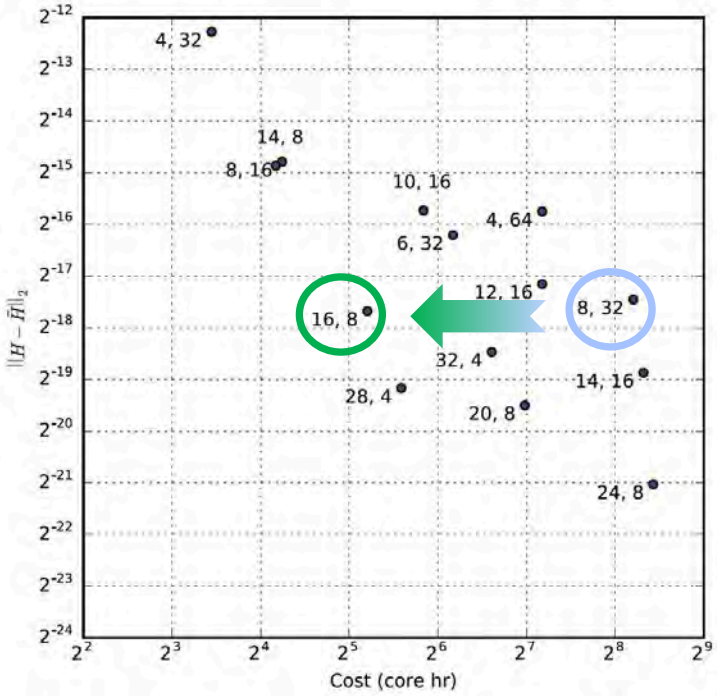
Use high-order discretizations for fewer DOFs

Efficiency of High Order Spectral Element Methods on Petascale Architectures

Maxwell Hutchinson[1(✉)], Alexander Heinecke[2], Hans Pabst[3], Greg Henry[4], Matteo Parsani[5], and David Keyes[5]

[1] Department of Physics, University of Chicago, Chicago, IL, USA
maxhutch@uchicago.edu
[2] Intel Corporation, Santa Clara, CA, USA
[3] Intel Semiconductor AG, Zurich, Switzerland
[4] Intel Corporation, Hillsboro, OR, USA
[5] Extreme Computing Research Center, KAUST, Thuwal 23955, Kingdom of Saudi Arabia

Abstract. High order methods for the solution of PDEs expose a trade-off between computational cost and accuracy on a per degree of freedom basis. In many cases, the cost increases due to higher arithmetic intensity while affecting data movement minimally. As architectures tend towards wider vector instructions and expect higher arithmetic intensities, the best order for a particular simulation may change.



Rediscretize from *32 spectral elements of order 8 on a side* to *8 spectral elements of order 16 on a side*

Same error in key functional:
- approx 4e-6

Savings in execution time:
- factor of 8

Use high-order discretizations for fewer DOFs

Four different dense linear algebra libraries compared on 15 different element orders for execution rate and memory transfer rate

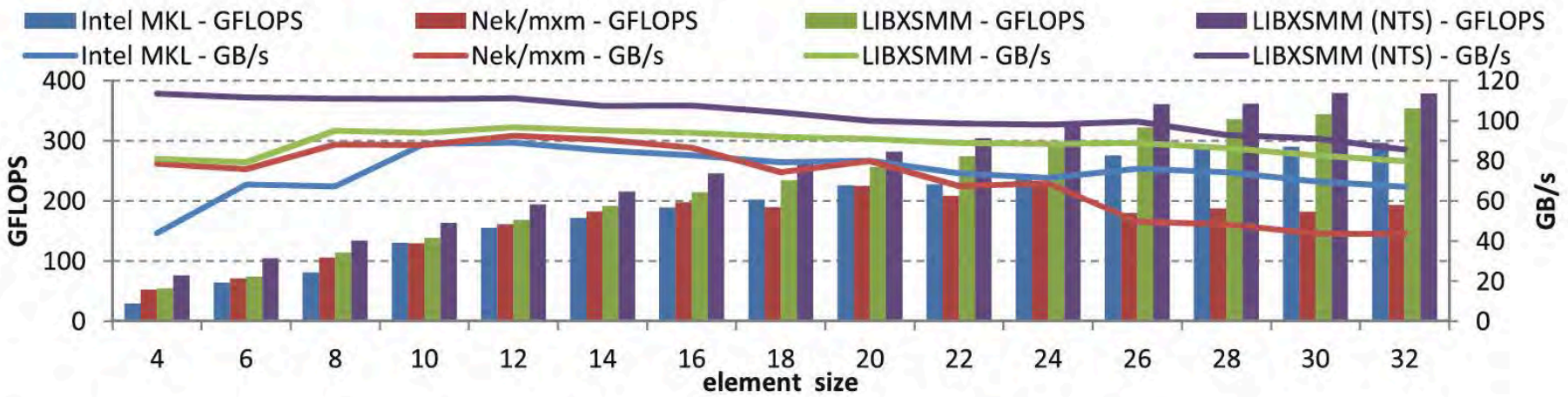


Fig. 1. Performance of the Helmholtz reproducer running on a single node of Shaheen for different implementation of small matrix multiplications. NTS denotes the usage of the non-temporal store optimized module.

Performance of all libraries improves up to 16th-order elements

LIBXSMM continues to improve up to 32nd-order elements

Some “universals” of exascale computing

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Adapt precision to accuracy requirements

Accelerating Geostatistical Modeling and Prediction With Mixed-Precision Computations: A High-Productivity Approach with PaRSEC

Sameh Abdulah, Qinglei Cao, Yu Pei, George Bosilca, Jack Dongarra, Marc G. Genton, David E. Keyes, Hatem Ltaief, and Ying Sun

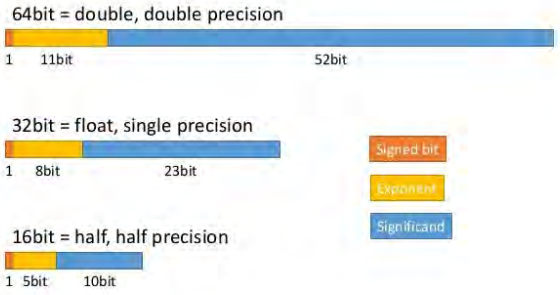
Abstract—Geostatistical modeling is an efficient technique for climate and environmental analysis and predicting desired quantities from geographically distributed data, based on statistical models and optimization of parameters. Spatial data presenting some known property of stationarity (or non-stationarity) requires a specific geostatistics kernel to handle. A primary computational kernel of stationary spatial statistics is the evaluation of the Gaussian maximum log-likelihood estimation (MLE) function, whose central data structure is a dense, symmetric, and positive definite covariance matrix of the dimension of the number of correlated observations. In the MLE approach considered herein, two essential operations are the application of its inverse and evaluation of its determinant. These can be rendered through the Cholesky decomposition and triangular solution. In this paper, we propose to reduce the precision of low correlated locations to single- or half- precision based on the distance. We migrate geostatistics to a three precision approximation by exploiting the mathematical structure of the dense covariance matrix. We illustrate application-expected accuracy worthy of double-precision from a majority half-precision computation, in a context where all single precision is by itself insufficient. We deploy PaRSEC runtime system with high productivity in mind to tackle the complexity and imbalance caused by the mixed three precisions. The PaRSEC delivers within a solo Cholesky factorization on-demand casting of precisions, while orchestrating tasks and data movement in a multi-GPU distributed-memory environment. We maintain the application-expected accuracy, while achieving against FP64-only operations up to $1.59X$ by mixing FP64/FP32 operations and $2.64X$ by mixing FP64/FP32/FP16 operations on 1536/4096 nodes of HAWK / Shaheen II and 128 nodes of Summit, respectively. This translates into up to 4.5 / 4.7 and 9.1 (mixed) PFlop/s sustained performance, respectively, demonstrating the effective synergism between PaRSEC dynamic runtime systems and challenging environmental HPC applications.

1 Introduction

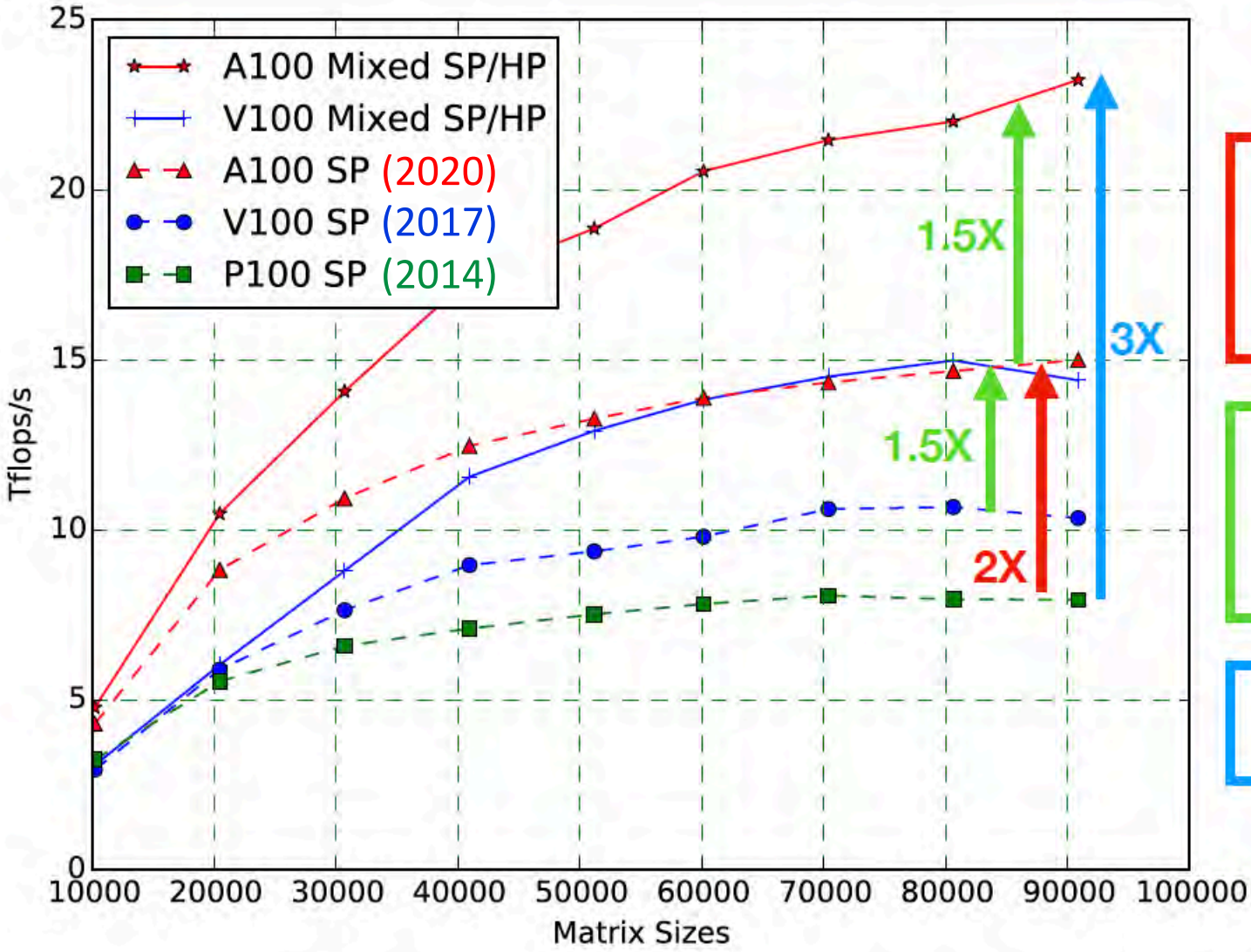
Geostatistics is a means of modeling and predicting desired quantities directly from data. It is based on statistical assumptions and optimization of parameters. It is complementary to first-principles modeling approaches rooted in conservation laws and typically expressed in PDEs. Alternative statistical approaches to predictions from first-principles methods, such as Monte Carlo sampling wrapped around simulations with a distribution of inputs, may be vastly more computationally expensive than sampling from an assumed parameterized distribution based on a much smaller number of simulations. Geostatistics is relied upon for economic and policy decisions for which billions of dollars or even lives are at stake, such as engineering safety margins into developments, mitigating hazardous air quality, locating fixed renewable energy resources, and planning agricultural yields or weather-dependent tourist revenues. Climate and weather predictions are among the principal workloads occupying supercomputers around the world, and even minor improvements for regular production applications pay large dividends. A wide variety of such predictive codes have opportunistically migrated or are migrating to mixed-precision environments; we describe a novel migration of one important class of such codes.

A main computational kernel of stationary spatial statistics considered herein is the evaluation of the Gaussian log-likelihood function, whose central data structure is a

Format of Floating points
IEEE754



Adapt precision to accuracy requirements



Red Arrow: speedup from hardware, same algorithm

Green Arrows: speedup from algorithm, same hardware

Blue Arrow: from both

Implicit question: Do we want to wait for NVIDIA Hopper (2023)?
Or do we want Hopper performance on NVIDIA Ampere today?

Adapt precision to accuracy requirements

Precision	Type	Bits Signif (t)	Exp	Range	$u = 2^{-t}$
half	bfloat16	8	8	$10^{\pm 38}$	3.9×10^{-3}
half	fp16	11	5	$10^{\pm 5}$	4.9×10^{-4}
single	fp32	24	8	$10^{\pm 38}$	6.0×10^{-8}
double	fp64	53	11	$10^{\pm 308}$	1.1×10^{-16}

fp64, fp32, fp16 defined by IEEE standard

Bfloat16: Google, Intel, ARM, NVIDIA

c/o Nick Higham (Manchester, 2021)

Some “universals” of exascale computing

Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Resilience in algorithms, not hardware

arXiv:1206.1390v1 [math.NA] 7 Jun 2012

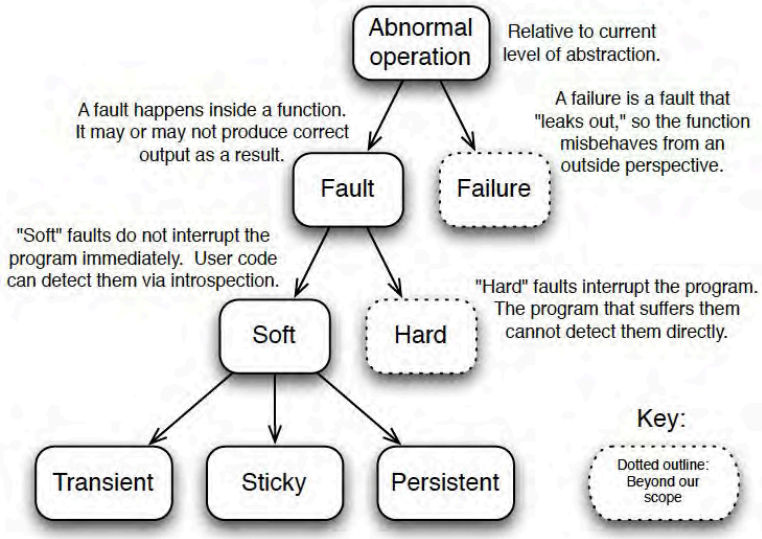
Fault-tolerant linear solvers via selective reliablity

Patrick G. Bridges, Kurt B. Ferreira, Michael A. Heroux, and Mark Hoemmen

October 31, 2018

Energy increasingly constrains modern computer hardware, yet protecting computations and data against errors costs energy. This holds at all scales, but especially for the largest parallel computers being built and planned today. As processor counts continue to grow, the cost of ensuring reliability consistently throughout an application will become unbearable. However, many algorithms only need reliability for certain data and phases of computation. This suggests an algorithm and system codesign approach. We show that if the system lets applications apply reliability selectively, we can develop algorithms that compute the right answer despite faults. These "fault-tolerant" iterative methods either converge eventually, at a rate that degrades gracefully with increased fault rate, or return a clear failure indication in the rare case that they cannot converge. Furthermore, they store most of their data unreliably, and spend most of their time in unreliable mode.

We demonstrate this for the specific case of detected but uncorrectable memory faults, which we argue are representative of all kinds of faults. We developed a cross-layer application / operating system framework that intercepts and reports uncorrectable memory faults to the application, rather than killing the application, as current operating systems do. The application in turn can mark memory allocations as subject to such faults. Using this framework, we wrote a fault-tolerant iterative linear solver using components from the Trilinos solvers library. Our solver exploits hybrid parallelism (MPI and threads). It performs just as well as other solvers if no faults occur, and converges where other solvers do not in the presence of faults. We show convergence results for representative test problems. Near-term future work will include performance tests.



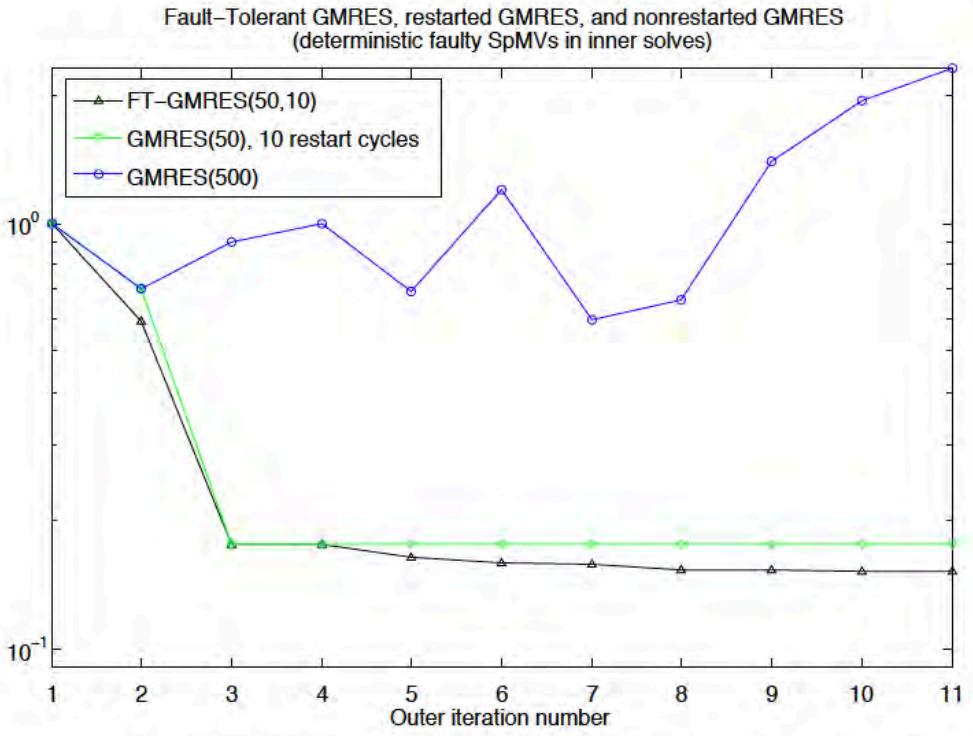
Key ideas:

- **Reliable computing is expensive**
- **Divide memory: reliable/unreliable**
- **Divide routines: reliable/unreliable**
- **Do most of the work in unreliable mode with reliable detection and correction**
- **Ex.: FT-GMRES with unreliable matvec or preconditioner**

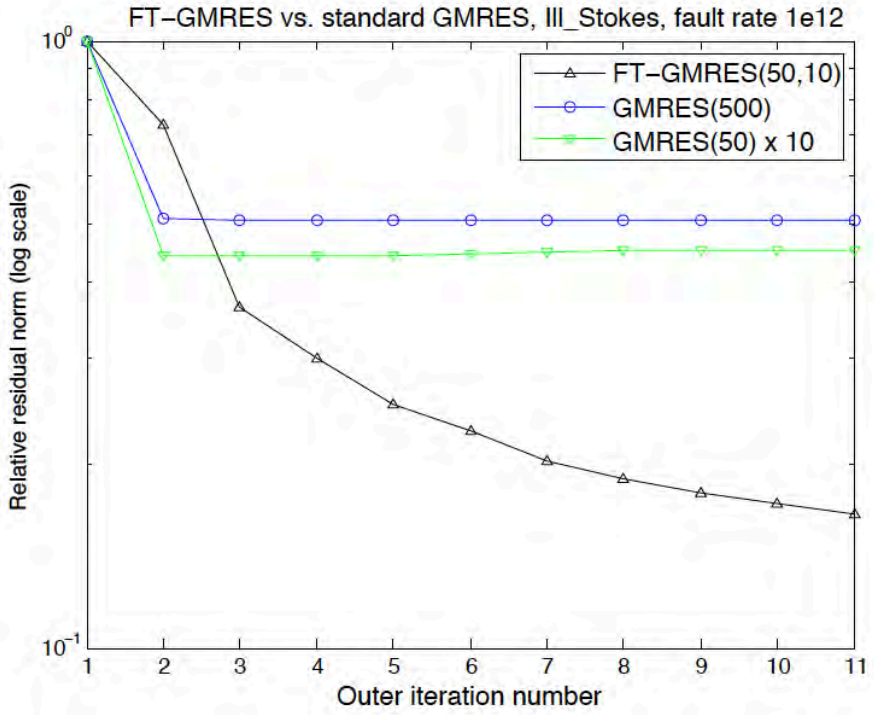
Resilience in algorithms, not hardware

Ill-conditioned Stokes problem

matvec unreliable
deterministically spaced faults



matvec & preconditioner unreliable
random faults



Some “universals” of exascale computing

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

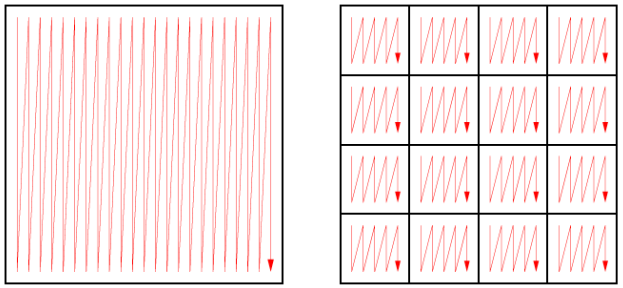
Employ dynamic scheduling

Solving the Generalized Symmetric Eigenvalue Problem using Tile Algorithms on Multicore Architectures[1]

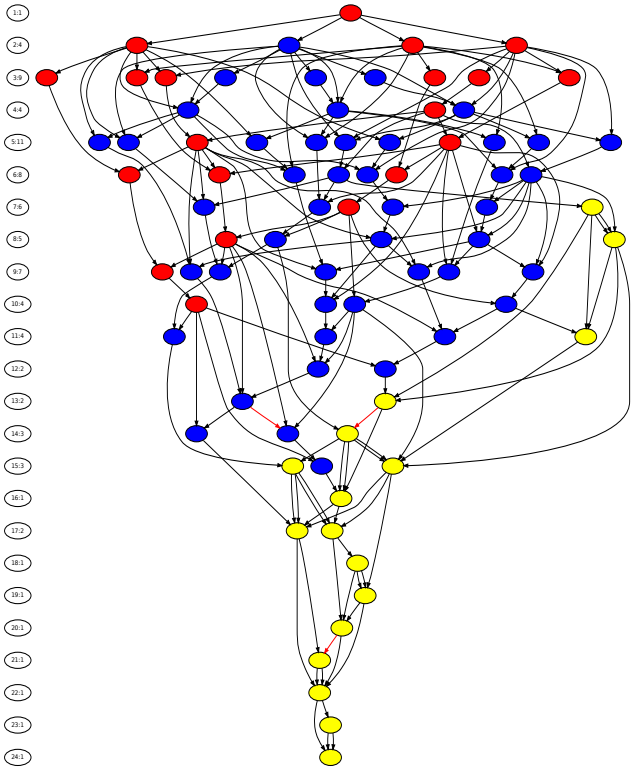
Hatem LTAIEF [a,2], Piotr LUSZCZEK [b], Azzam HAIDAR [b] and Jack DONGARRA [b]
[a] *KAUST Supercomputing Laboratory, Thuwal, Saudi Arabia*
E-mail: Hatem.Ltaief@kaust.edu.sa
[b] *Innovative Computing Laboratory, University of Tennessee, Knoxville TN USA*
Email: luszczek,haidar,dongarra@eecs.utk.edu
[c] *Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee*
[d] *School of Mathematics & School of Computer Science, University of Manchester*

Abstract. This paper proposes an efficient implementation of the generalized symmetric eigenvalue problem on multicore architecture. Based on a four-stage approach and tile algorithms, the original problem is first transformed into a standard symmetric eigenvalue problem by computing the Cholesky factorization of the right hand side symmetric definite positive matrix (first stage), and applying the inverse of the freshly computed triangular Cholesky factors to the original dense symmetric matrix of the problem (second stage). Calculating the eigenpairs of the resulting problem is then equivalent to the eigenpairs of the original problem. The computation proceeds by reducing the updated dense symmetric matrix to symmetric band form (third stage). The band structure is further reduced by applying a bulge chasing procedure, which annihilates the extra off-diagonal entries using orthogonal transformations (fourth stage). More details on the third and fourth stage can be found in Haidar et al. [Accepted at SC'11, November 2011]. The eigenvalues are then calculated from the tridiagonal form using the standard LAPACK QR algorithm (i.e., DTSEQR routine), while the complex and challenging eigenvector computations will be addressed in a companion paper. The tasks from the various stages can concurrently run in an out-of-order fashion. The data dependencies are cautiously tracked by the dynamic runtime system environment QUARK, which ensures the dependencies are not violated for numerical correctness purposes. The obtained tile four-stage generalized symmetric eigenvalue solver significantly outperforms the state-of-the-art numerical libraries (up to 21-fold speed up against multithreaded LAPACK with optimized multithreaded MKL BLAS and up to 4-fold speed up against the corresponding routine from the commercial numerical software Intel MKL) on four sockets twelve cores AMD system with a 24000×24000 matrix size.

Int Conf Par Comput (ParCo) 2011



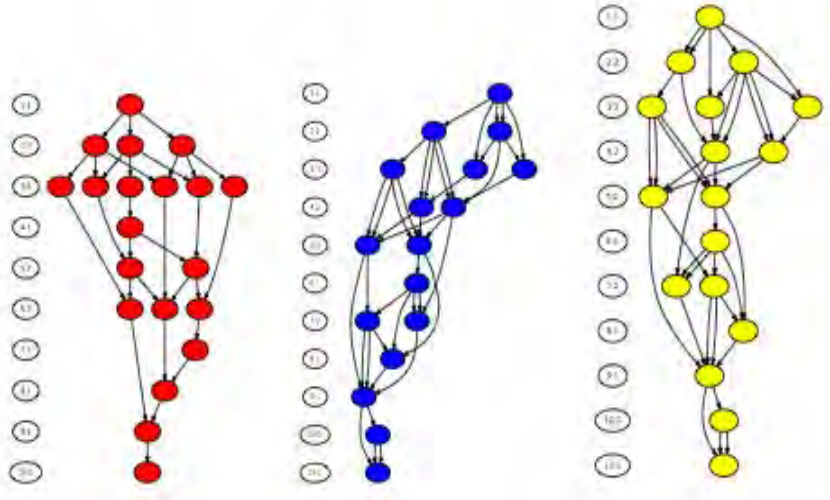
Task graph for the first 3 stages of a Generalized Symmetric EVP with 4 blocks

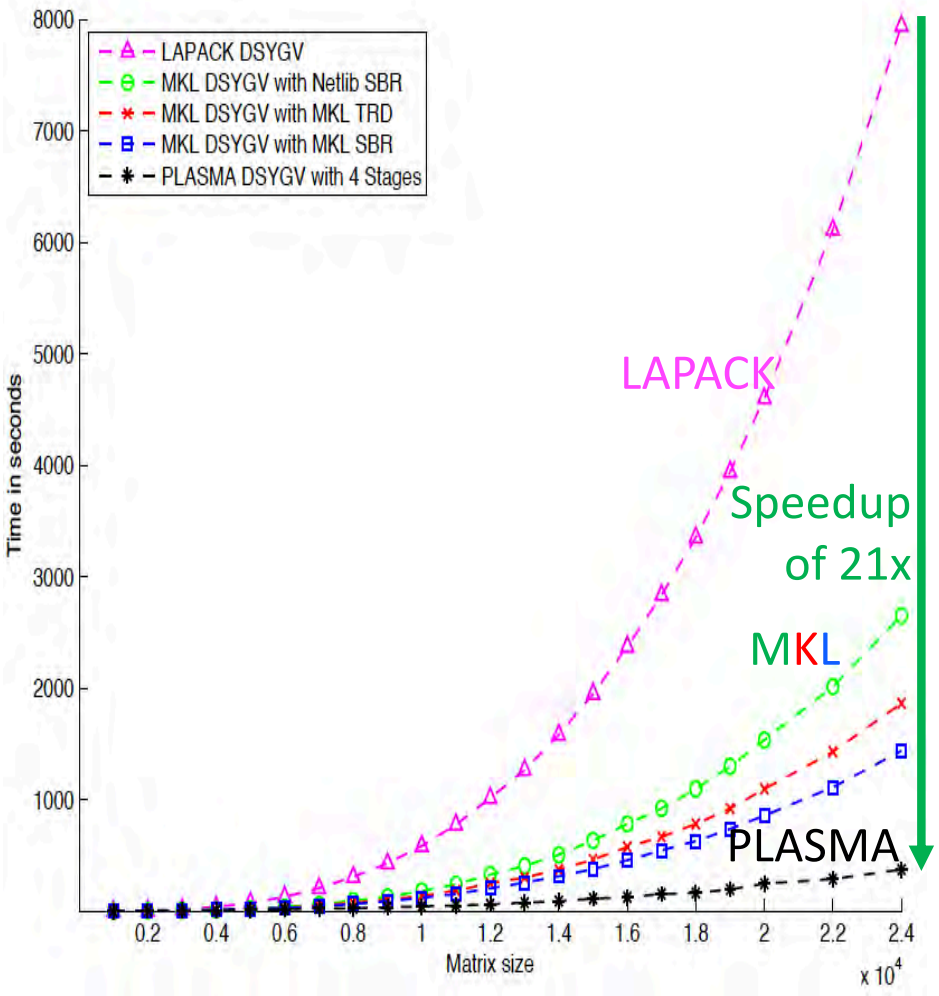


Employ dynamic scheduling

$$Ax = \lambda Bx$$

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





- **Remove artifactual synchronizations in the form of subroutine boundaries**
- **Remove artifactual orderings in the form of pre-scheduled loops**
- **Expose more concurrency**

Some “universals” of exascale computing

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Employ APIs to specialized back-ends

DOI:10.1145/3274770

Used in the design of the Internet and Unix, the layered services of the hourglass model have enabled viral adoption and deployment scalability.

BY MICAH BECK

On The Hourglass Model

THE HOURGLASS MODEL of layered systems architecture is a visual and conceptual representation of an approach to design that seeks to support a great diversity of applications and allow implementation using a great diversity of supporting services. At the center of the hourglass model is a distinguished layer in a stack of abstractions that is chosen as the sole means of accessing the lower-level resources of the system. This distinguished layer can be implemented using services that are considered as lying below it in the stack as well as other services and applications that are considered as lying above it. However, the components that lie above the distinguished layer cannot directly access the services that lie below it.

David Clark called the distinguished layer the "spanning layer" because in the Internet architecture it bridges the multiple local area network implementations that lie below it in the stack (see Figure 1). Clark defined the function of the spanning layer by its ability to " ... hide the detailed differences among these various technologies, and present a uniform service interface to the applications above" and identified the Internet Protocol as the spanning layer of the Internet (see Figure 2).[5] Arguably the spanning layer also includes other elements of the Internet Protocol Suite that access lower-layer services (such as ARP and DHCP).

The shape suggested by the hourglass model expresses the goal that the spanning layer should support various applications and be implementable using many possible supporting layers. Referring to the hourglass as a design tool also expresses the intuition that restricting the functionality of the spanning layer is instrumental in achieving these goals. The elements of the model are combined visually in the form of an hourglass shape, with the "thin waist" of the hourglass representing the restricted spanning layer, and its large upper and lower bells representing the multiplicity of applications and supporting layers, respectively.

The hourglass model is widely used in describing the design of the Internet, and can be found in many modern networking textbooks.[8] A similar principle has also been implicitly applied to the design of other successful spanning layers, notably the Unix operating system kernel interface by which we mean

» key insights

- **Adoption of a common service interface is a key to interoperability, portability, and "future-proofing" in the face of rapid technological change.**
- **The design of both the Internet and Unix followed the hourglass principle, leading to dominance in two software markets while also enabling disruptive innovation.**
- **Many successful interface designers adhere to a discipline of simplicity, generality, and limitation of the common interface.**
- **This article introduces the Deployment Scalability Tradeoff, a principle that seeks to explain the reason for the success of this discipline.**

Figure 1. The hourglass model.

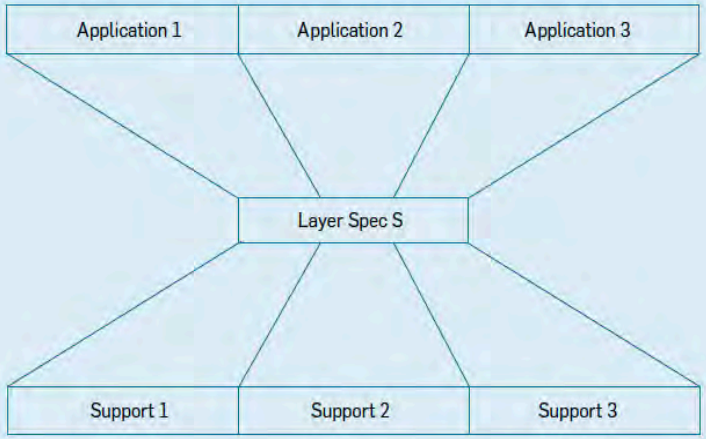
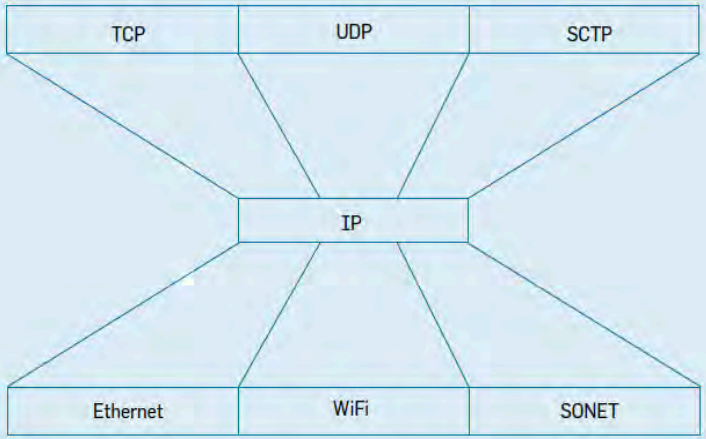
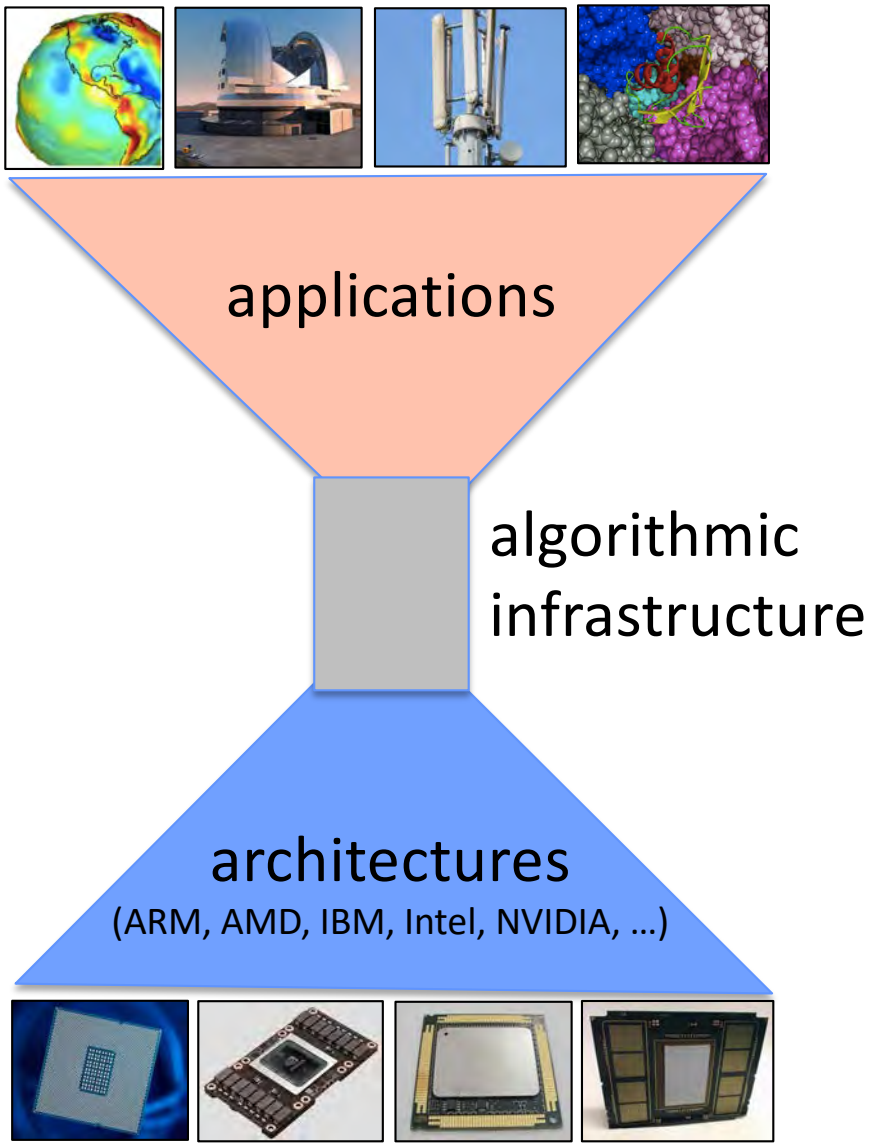


Figure 2. The Internet hourglass.



Employ APIs to specialized back-ends



- Tiling and recursive subdivision create large numbers of small problems that can be marshaled for batched operations on GPUs and MICs
  - amortize call overheads
  - polyalgorithmic approach based on block size
- Non-temporal stores, coalesced memory accesses, double-buffering, etc. reduce sensitivity to memory
- Code is complex
- Code is architecture-specific at the bottom
- Need to hide the support from the apps through an API

Some “universals” of exascale computing

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Exploit data sparsity

PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
numerical analysis, high performance computing

Keywords:
computational linear algebra, hierarchical matrices, exascale architectures

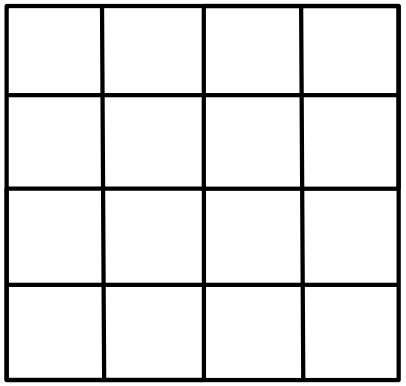
Author for correspondence:
D. E. Keyes
e-mail: david.keyes@kaust.edu.sa

Hierarchical Algorithms on Hierarchical Architectures

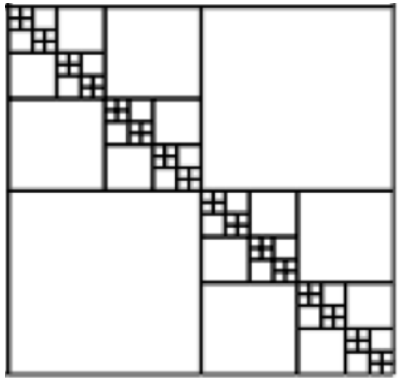
D. E. Keyes[1], H. Ltaief[1] and G. Turkiyyah[2]

[1]Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia
[2]Department of Computer Science, American University of Beirut 1107 2020, Lebanon

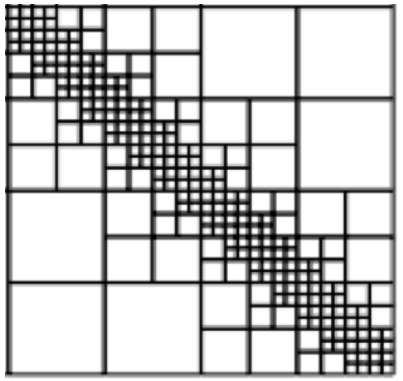
A traditional goal of algorithmic optimality, squeezing out flops, has been superseded by evolution in architecture. Flops no longer serve as a reasonable proxy for all aspects of complexity. Instead, algorithms must now squeeze memory, data transfers, and synchronizations, while extra flops on locally cached data represent only small costs in time and energy. Hierarchically low rank matrices realize a rarely achieved combination of optimal storage complexity and high computational intensity for a wide class of formally dense linear operators that arise in applications for which exascale computers are being constructed. They may be regarded as algebraic generalizations of the Fast Multipole Method. Methods based on these hierarchical data structures and their simpler cousins, tile low rank matrices, are well proportioned for early exascale computer architectures, which are provisioned for high processing power relative to memory capacity and memory bandwidth. They are ushering in a renaissance of computational linear algebra. A challenge is that emerging hardware architecture possesses hierarchies of its own that do not generally align with those of the algorithm. We describe modules of a software toolkit, Hierarchical Computations on Manycore Architectures (HiCMA), that illustrate these features and are intended as building blocks of applications, such as matrix-free higher-order methods in optimization and large-scale spatial statistics. Some modules of this open source project have been adopted in the software libraries of major vendors.



TLR



HLR
weakly admissible



HLR
strongly admissible

Complexities of rank-structured factorization

For a square dense matrix of $O(N)$:

- **"Straight"** LU **or** LDL^T
  - **Operations** $O(N^3)$
  - **Storage** $O(N^2)$
- **Tile low-rank** (Amestoy, Buttari, L'Excellent & Mary, *SISC*, 2016)*
  - **Operations** $O(k^{0.5} N^2)$
  - **Storage** $O(k^{0.5} N^{1.5})$
  - **for uniform blocks with size chosen optimally for max rank *k* of any compressed block, bounded number of uncompressed blocks per row**
- **Hierarchically low-rank** (Grasedyck & Hackbusch, *Computing*, 2003)
  - **Operations** $O(k^2 N \log^2 N)$
  - **Storage** $O(k N)$
  - **for strong admissibility, where *k* is max rank of any compressed block**

* First reported $O(k^{0.5} N^{2.5})$, then later $O(k^{0.5} N^2)$ for variant that reorders updates and recompression

Some “universals” of exascale computing

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Process “on the fly”

SIAM J. SCI. COMPUT.
Vol. 41, No. 4, pp. C339–C366

RANDOMIZED GPU ALGORITHMS FOR THE CONSTRUCTION OF HIERARCHICAL MATRICES FROM MATRIX-VECTOR OPERATIONS*

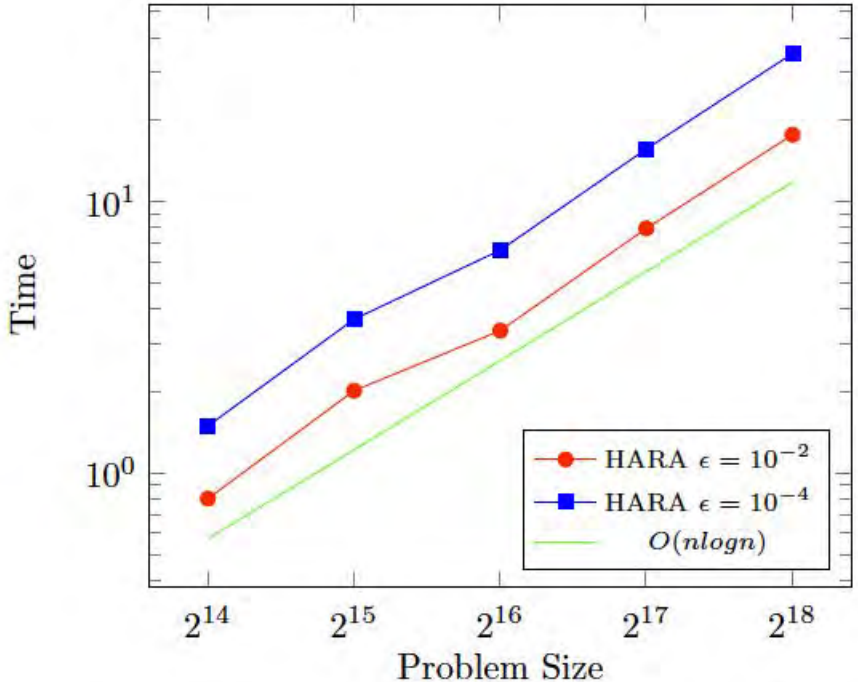
WAJIH BOUKARAM†, GEORGE TURKIYYAH‡, AND DAVID KEYES†

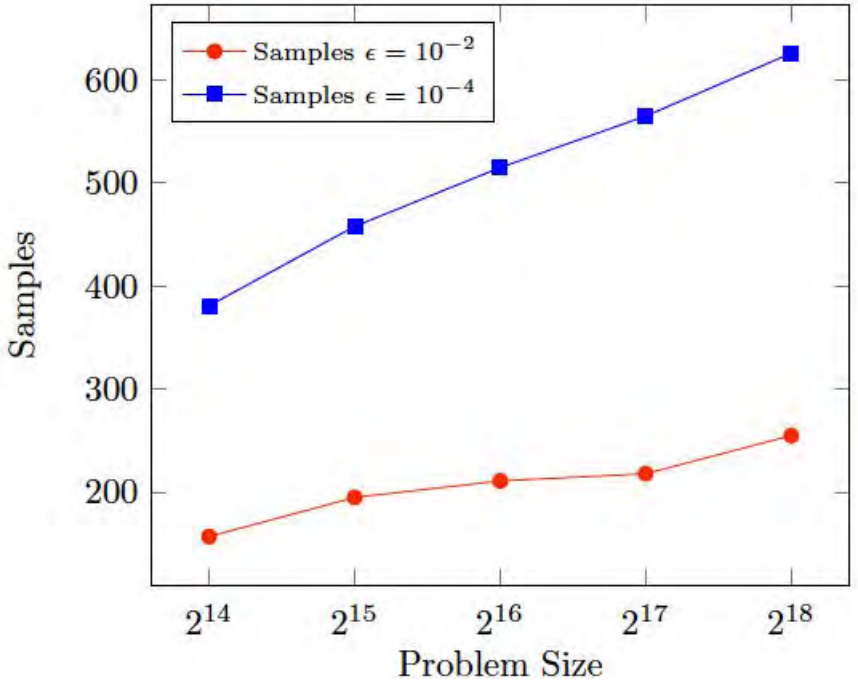
Abstract. Randomized algorithms for the generation of low rank approximations of large dense matrices have become popular methods in scientific computing and machine learning. In this paper, we extend the scope of these methods and present batched GPU randomized algorithms for the efficient generation of low rank representations of large sets of small dense matrices, as well as their generalization to the construction of hierarchically low rank symmetric $\mathcal{H}^2$ matrices with general partitioning structures. In both cases, the algorithms need to access the matrices only through matrix-vector multiplication operations which can be done in blocks to increase the arithmetic intensity and substantially boost the resulting performance. The batched GPU kernels are adaptive, allow nonuniform sizes in the matrices of the batch, and are more effective than SVD factorizations on matrices with fast decaying spectra. The hierarchical matrix generation consists of two phases, interleaved at every level of the matrix hierarchy. A first phase adaptively generates low rank approximations of matrix blocks through randomized matrix-vector sampling. A second phase accumulates and compresses these blocks into a hierarchical matrix that is incrementally constructed. The accumulation expresses the low rank blocks of a given level as a set of local low rank updates that are performed simultaneously on the whole matrix allowing high-performance batched kernels to be used in the compression operations. When the ranks of the blocks generated in the first phase are too large to be processed in a single operation, the low rank updates can be split into smaller-sized updates and applied in sequence. Assuming representative rank k, the resulting matrix has optimal $O(kN)$ asymptotic storage complexity because of the nested bases it uses. The ability to generate an $\mathcal{H}^2$ matrix from matrix-vector products allows us to support a general randomized matrix-matrix multiplication operation, an important kernel in hierarchical matrix computations. Numerical experiments demonstrate the high performance of the algorithms and their effectiveness in generating hierarchical matrices to a desired target accuracy.

$\mathcal{H}$ matrix-$\mathcal{H}$ matrix multiplication

- can be cast as the problem of constructing an $\mathcal{H}$-matrix from matvec operations
- we can do HGEMV operations efficiently on GPUs
  - HGEMV on multiple vectors is even more efficient
- HARA construction of product also performed efficiently on the GPU

Fast matvecs ⇒ fast approx inversions with Newton-Schulz





Some “universals” of exascale computing

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized “back-ends” while presenting high-level APIs to general users
- Exploit data sparsity to meet “curse of dimensionality” with “blessing of low rank”
- Process “on the fly” rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Co-design algorithms with hardware

Journal of Parallel and Distributed Computing 128 (2019) 137–150

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Fast parallel multidimensional FFT using advanced MPI

Lisandro Dalcin [a], Mikael Mortensen [b,*], David E. Keyes [a]

[a] Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia[1]
[b] Department of Mathematics, University of Oslo, Oslo, Norway[2]

HIGHLIGHTS

- Standard MPI features allow for compact implementation of parallel Fourier transforms.
- Data movement between memory subsystems and processor cores is reduced.
- Without special hardware support performance is on par with standard implementations.
- Potential for communication-hardware enhanced performance without code change.
- Less than a hundred lines of C code replace thousands in some implementations.

ARTICLE INFO

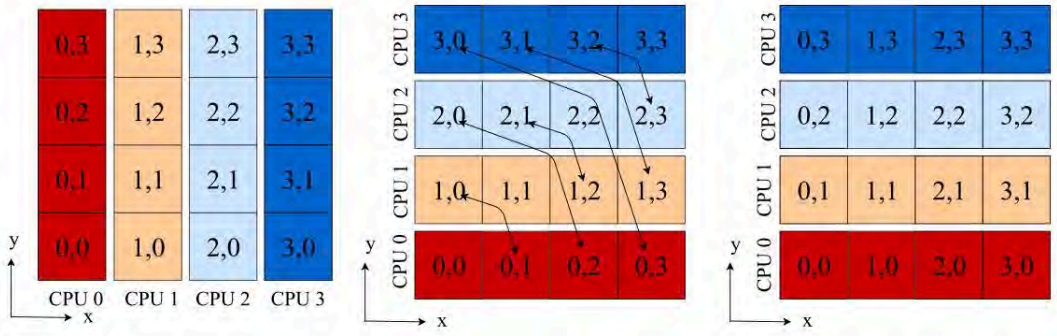
Article history:
Received 3 May 2018
Received in revised form 14 February 2019
Accepted 20 February 2019
Available online 11 March 2019

Keywords:
FFT
MPI
ALLTOALLW
Pencil
Slab

ABSTRACT

We present a new method for performing global redistributions of multidimensional arrays essential to parallel fast Fourier (or similar) transforms. Traditional methods use standard all-to-all collective communication of contiguous memory buffers, thus necessarily requiring local data realignment steps intermixed in-between redistribution and transform steps. Instead, our method takes advantage of subarray datatypes and generalized all-to-all scatter/gather from the MPI-2 standard to communicate discontiguous memory buffers, effectively eliminating the need for local data realignments. Despite generalized all-to-all communication of discontiguous data being generally slower, our proposal economizes in local work. For a range of strong and weak scaling tests, we found the overall performance of our method to be on par and often better than well-established libraries like MPI-FFTW, P3DFFT, and 2DECOMP&FFT. We provide compact routines implemented at the highest possible level using the MPI bindings for the C programming language. These routines apply to any global redistribution, over any two directions of a multidimensional array, decomposed on arbitrary Cartesian processor grids (1D slabs, 2D pencils, or even higher-dimensional decompositions). The high level implementation makes the code easy to read, maintain, and eventually extend. Our approach enables for future speedups from optimizations in the internal datatype handling engines within MPI implementations.

Co-design algorithms with hardware



(a) Original array. (b) After local transpose. (c) After Alltoall.

Fig. 2. Illustration of the slab decomposition. In (a) we see the original global array from Fig. 1a projected on the *xy*-plane. Each slab is divided into 4 smaller chunks that are to be communicated with the other processors and each chunk is identified with a pair of integer labels. In (b) we see the layout for slabs and chunks after performing a local transpose to align data in *x*-direction. Bidirectional arrows represent a subsequent all-to-all exchange among processors. In (c) we see the layout of the final global array after redistribution. Note that the relative position (within the global array) of each chunk in (a) is preserved in (c), however most chunks have migrated to a different processor.

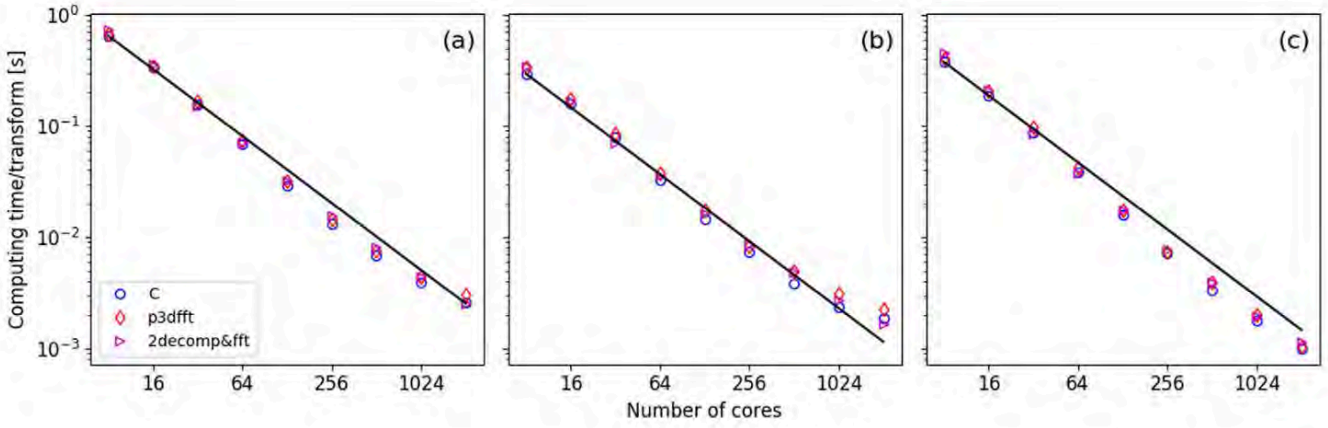


Fig. 7. Strong scaling with pencil decomposition of complete real-to-complex/complex-to-real FFTs on a mesh that has global shape 512^3. Showing fastest times measured. Subplot (a) shows total time, whereas (b) and (c) show times for global redistribution and FFTs, respectively.

Some "universals" of exascale computing

Architectural imperatives

- Reside "high" on the memory hierarchy, close to the processing elements
- Rely on SIMD/SIMT-amenable batches of tasks at fine scale
- Reduce synchrony in frequency and/or span
- Reduce communication in number and/or volume of messages
- Exploit heterogeneity in processing, memory, and networking elements

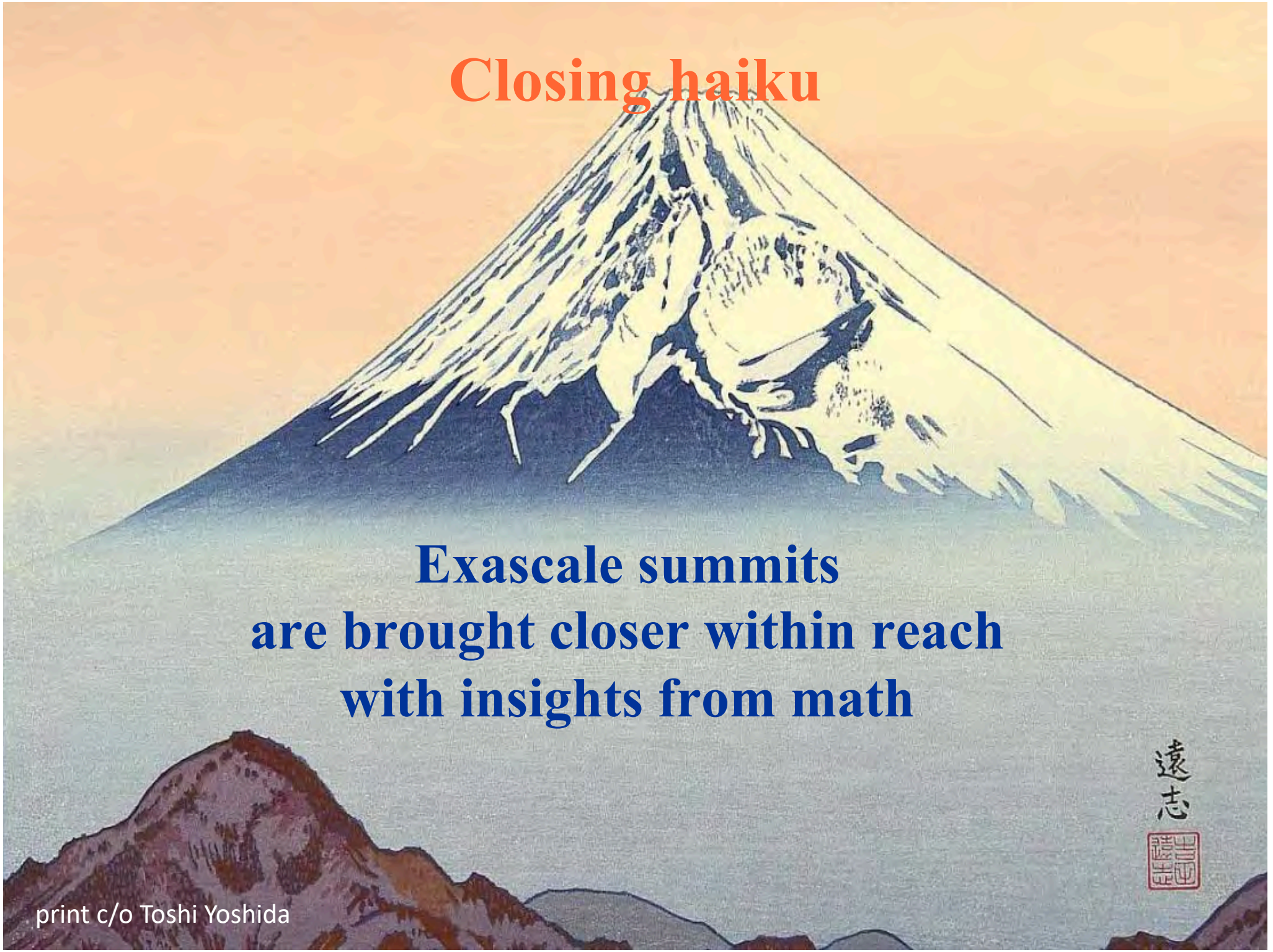
Strategies in practice

- Exploit extra memory to reduce communication volume
- Perform extra flops to require fewer global operations
- Use high-order discretizations to manipulate fewer DOFs (w/more ops per DOF)
- Adapt floating point precision to output accuracy requirements
- Take more resilience into algorithm space, out of hardware/systems space

Strategies in progress

- Employ dynamic scheduling capabilities, e.g., dynamic runtime systems based DAGs
- Code to specialized "back-ends" while presenting high-level APIs to general users
- Exploit data sparsity to meet "curse of dimensionality" with "blessing of low rank"
- Process "on the fly" rather than storing all at once (esp. large dense matrices)
- Co-design algorithms with hardware, incl. computing in the network or in memory

Closing haiku



Exascale summits
are brought closer within reach
with insights from math

print c/o Toshi Yoshida


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