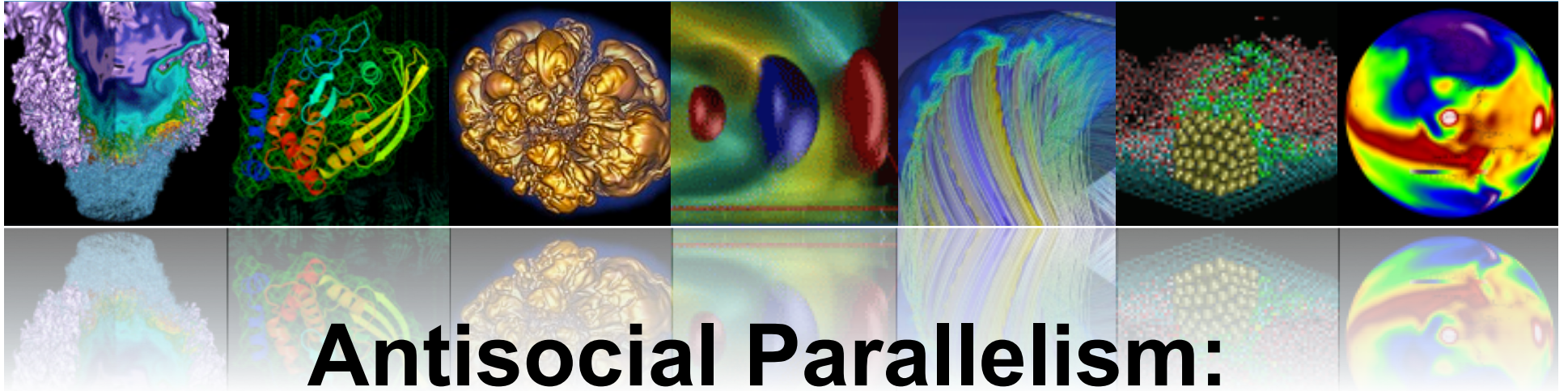




Antisocial Parallelism: *Avoiding, Hiding and Managing Communication*

Kathy Yelick

**Associate Laboratory Director of Computing Sciences
Lawrence Berkeley National Laboratory**

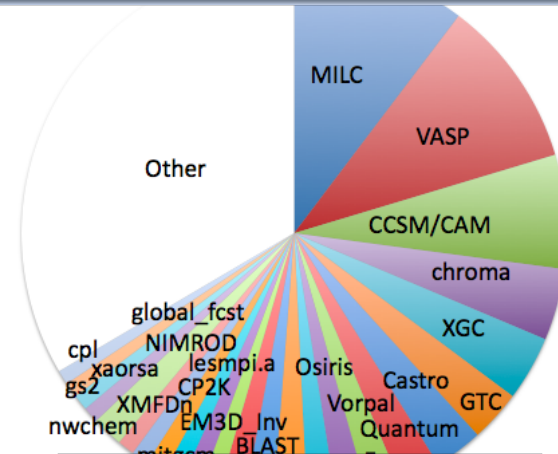
EECS Professor, UC Berkeley



The National Energy Research Scientific Computing Center Enables Science



4500 users ~600 applications



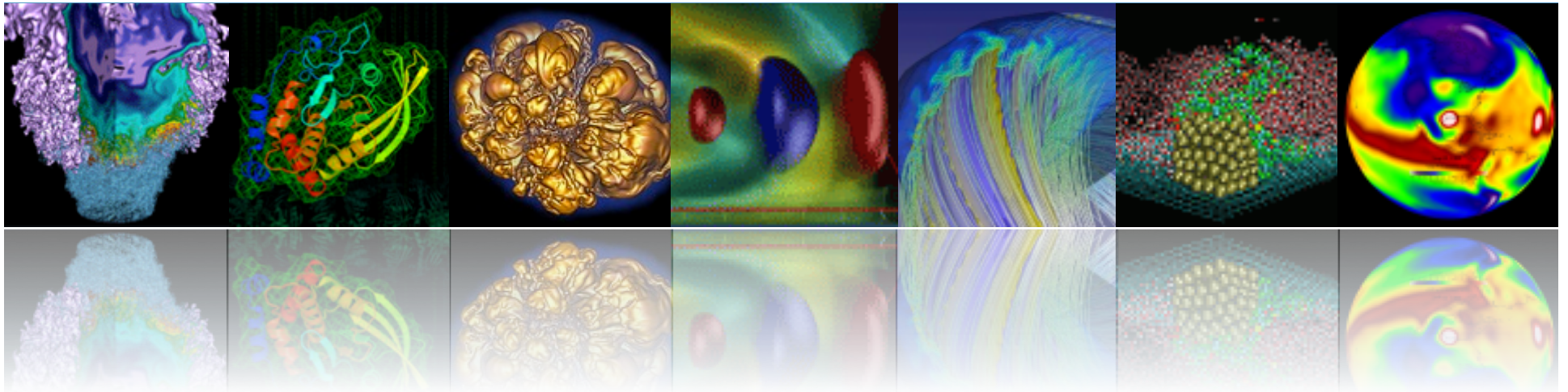
25 applications =
2/3s of workload

1500+ publications per year



Petaflop and Petabyte systems for science

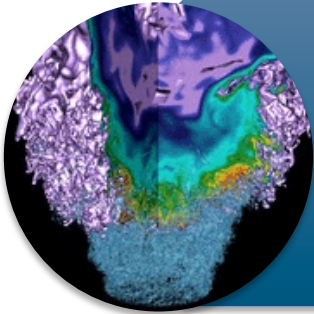




4 Myths of Supercomputing

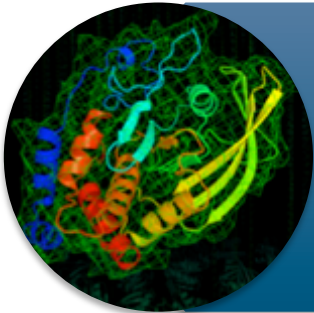


Myth #1: Supercomputers are only needed for a few hero simulations



Science at Scale

Petascale to Exascale Simulations



Science through Volume

Thousands to Millions of Simulations

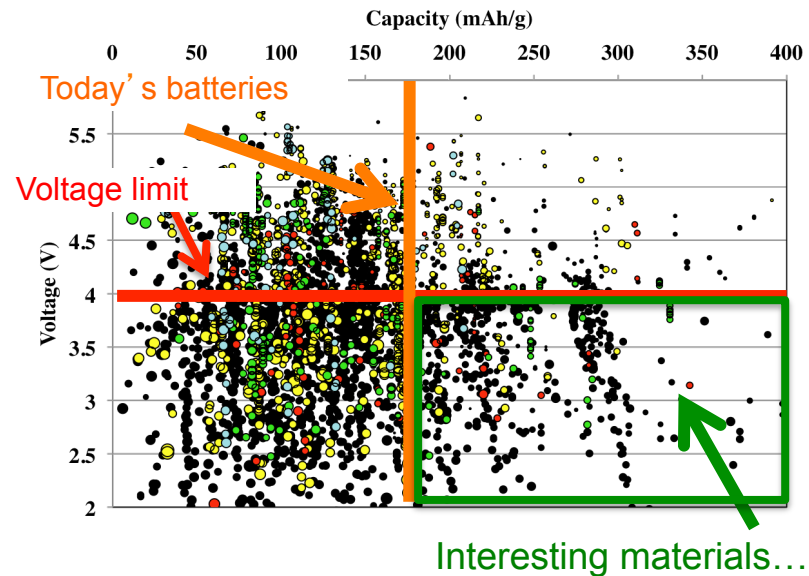
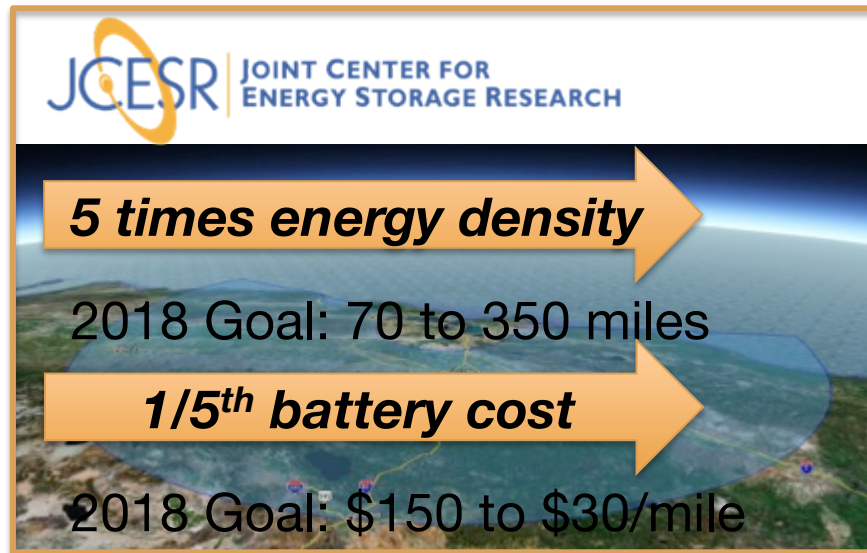


Science in Data

Petabytes to Exabytes of Data



Materials Genome Needs Millions of Simulations and Higher Fidelity



- Need more computing
 - 10s of thousands of simulations used to screen potential materials
 - Need millions of simulations and fidelity for new class of materials, study in extreme environments, etc.

- Data services raise impact
 - Results from 10s of thousands of simulations web-searchable
 - Materials Project has >3,000 users
 - Goal: cut in half 18 years time from discovery to market



PIs: Gerd Ceder, MIT and Kristin Persson, LBNL



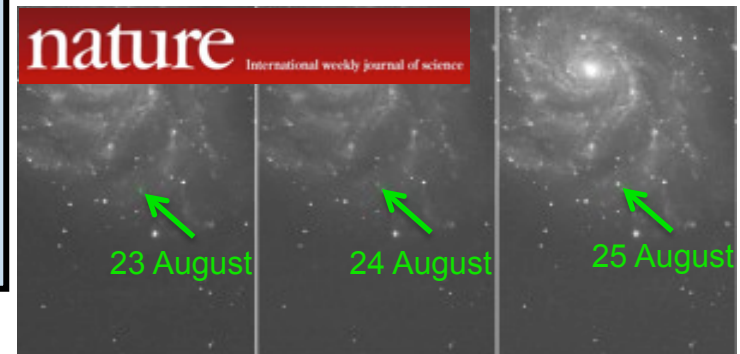
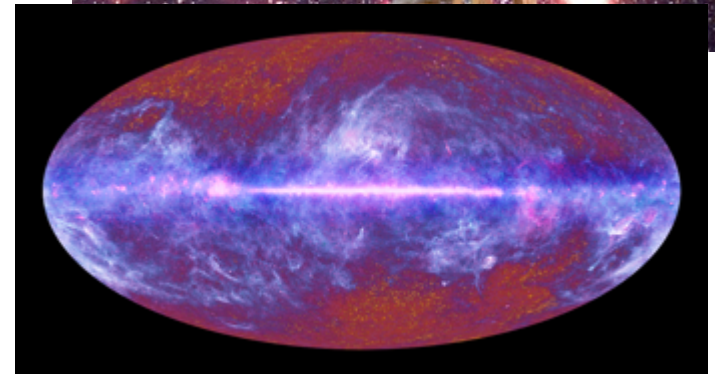


Myth #2: Supercomputing and Big Data are Unrelated

- NERSC imports 1 petabyte / month
- Runs pipelines archives, and serves (astro, genomics, neutrino)
- Simulation intimately connected

Example: Astrophysicists discover early nearby supernova

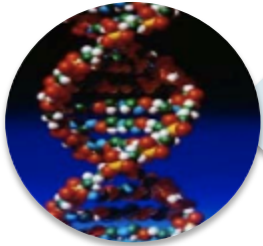
- Palomar Transient Factory runs analysis algorithms on ~300GB/night delivered by ESnet “science network”
- Rare young supernova: within 11 hours of exploding, 20M light years away
- Telescopes world-wide redirected



Saul Perlmutter Nobel prize used NERSC simulations; George Smoot Nobel used CMB, currently analyzed at NERSC and on the Science watchlist for 2013; Peter Nugent was LBNL lead on Supernova discovery

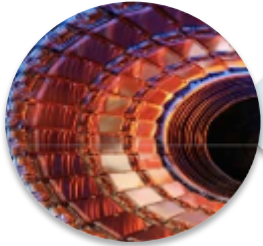


Data Explosion is Everywhere in Science



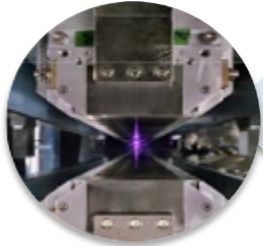
Genomics:

- Sequencer volume up 12x, cost down 10x, over next 3 years



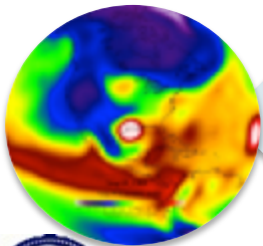
High Energy Physics

- LHC produces petabytes/year, rates up 3-5x in 5 years



Light Sources

- Many detectors outpacing Moore's Law curve

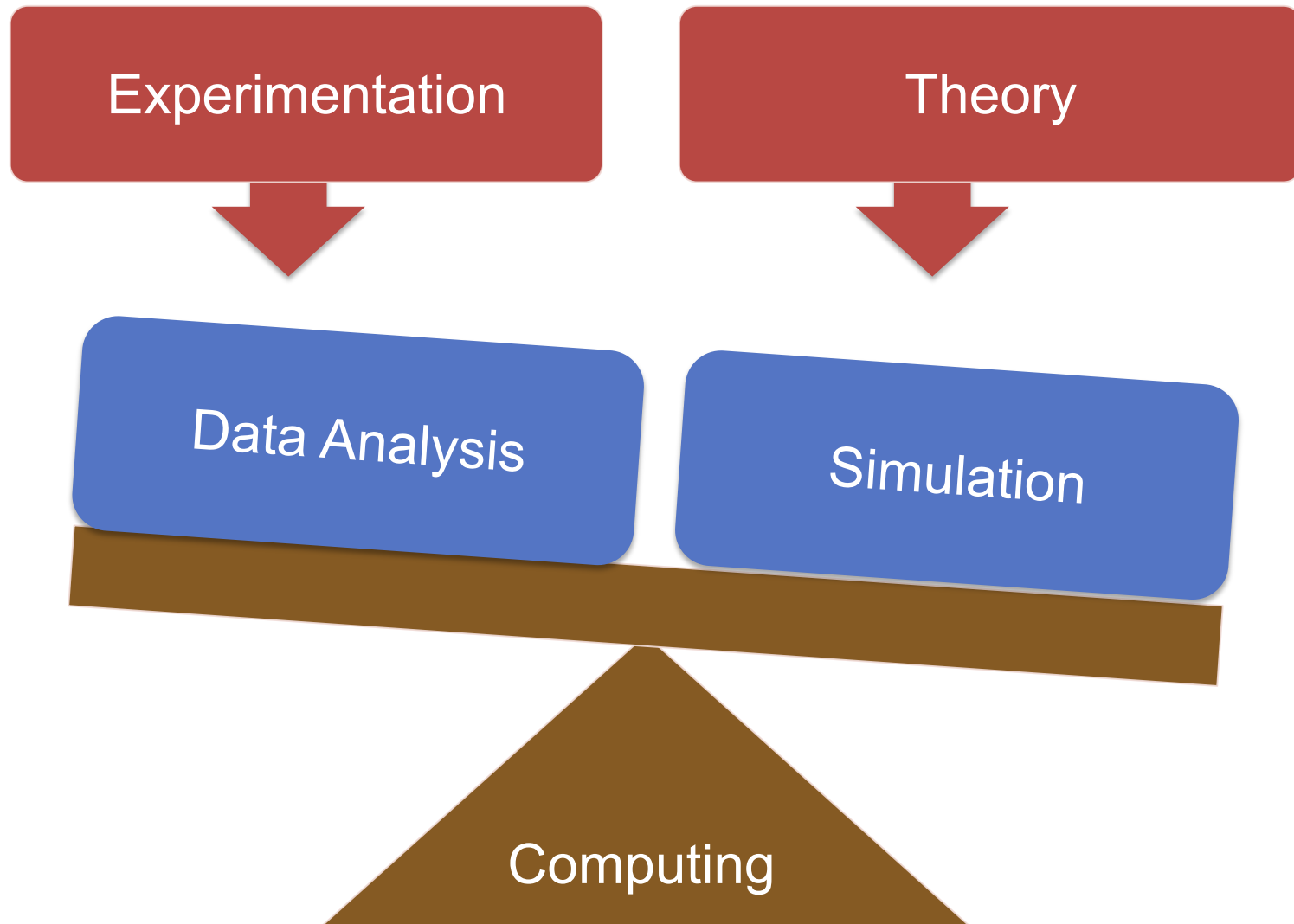


Climate

- By 2020, climate data expected to be hundreds of exabytes

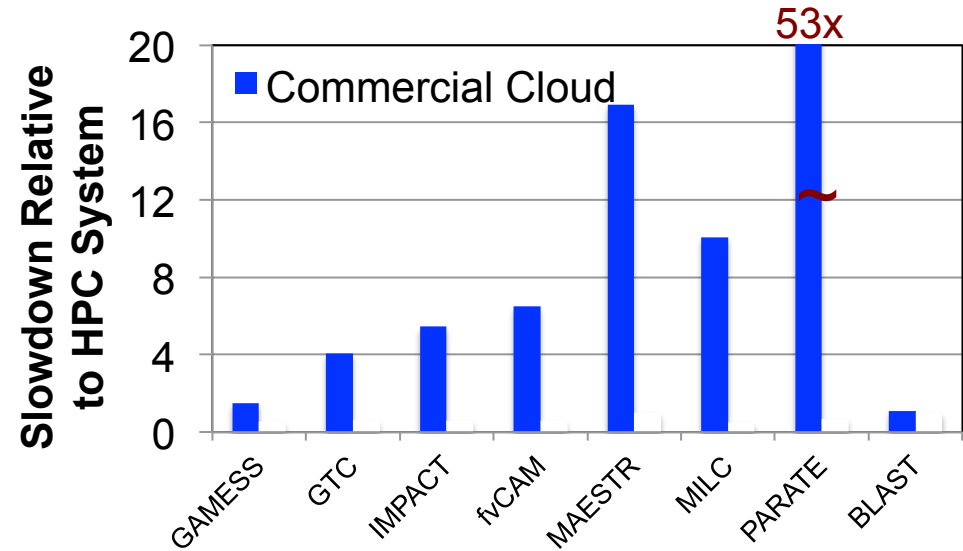


Data Analysis and Simulation Both Require More Computing



Myth #3: Supercomputers are Expensive, Clouds are Cheap

Component	Annual Cost (rough estimate)
Cloud cost on apps (ave 5x slowdown)	~\$900M
Cloud cost (1.38B core hours)	\$181M
NERSC Budget	\$57M
NERSC HPC HW	~\$20M



To buy raw NERSC core hours costs more than NERSC budget

- Even ignoring the measured performance slowdown
- **Doesn't include consulting staff, account management, licenses, bandwidth, software support: ~2/3 of NERSC's Budget**

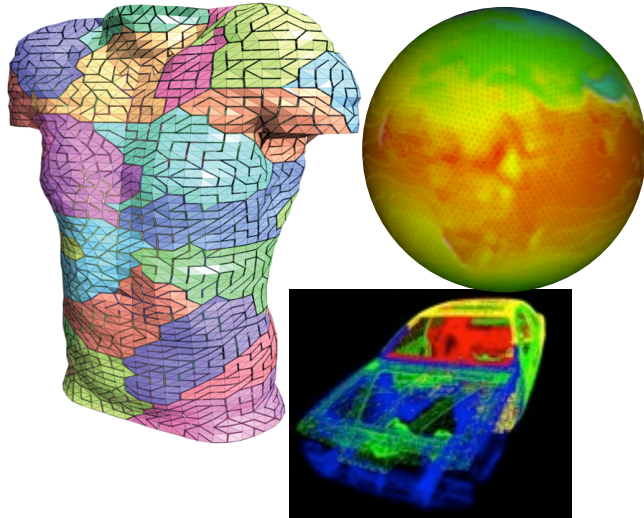
Why?

- NERSC runs at higher utilization (> 90%) and **no profit**.
- NERSC cost/core hours dropped 10x (1000%) from 2007 to 2011, while Amazon pricing dropped 15% in the same period



Myth #4: Data Analysis is “easy parallelism,” simulation is “hard”

More Regular



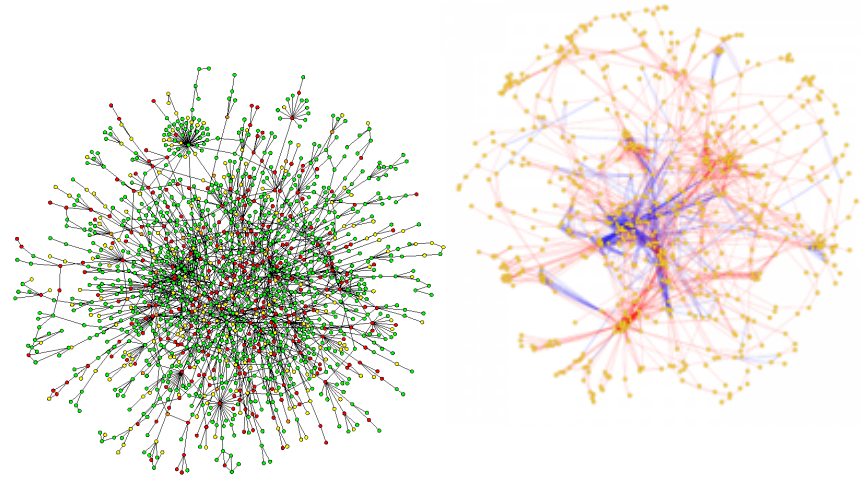
Message Passing Programming

Divide up domain in pieces
Compute one piece
Send/Receive data from others

MPI, and many libraries



More Irregular

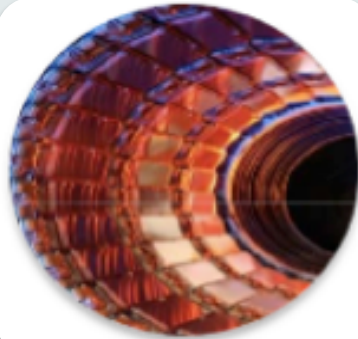


Global Address Space Programming

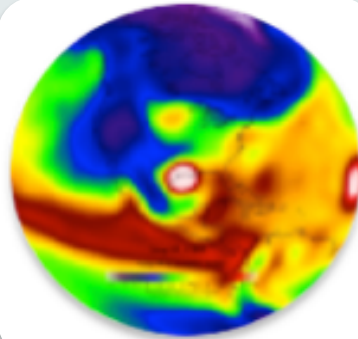
Each start computing
Grab whatever / whenever

UPC, CAF, X10, Chapel, GlobalArrays

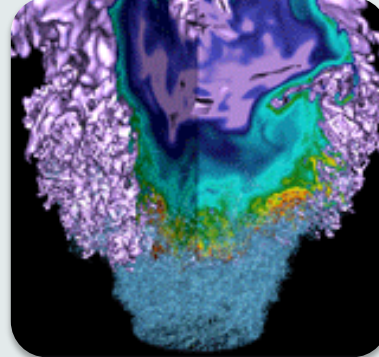
Science Problems Fit Across the “Irregularity” Spectrum



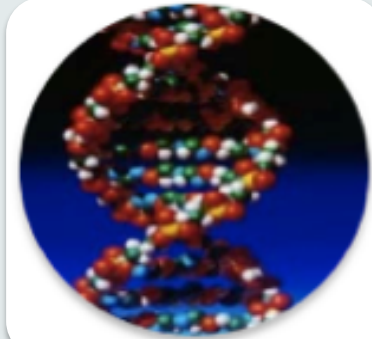
Massive
Independent
Jobs for
Analysis and
Simulations



Nearest
Neighbor
Simulations



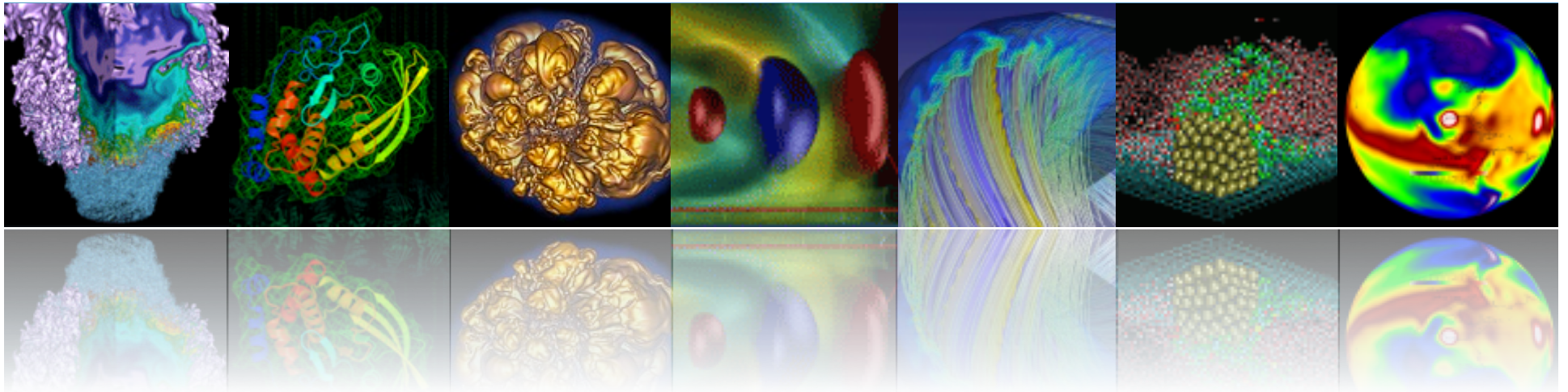
All-to-All
Simulations



Random
access, large
data Analysis

... often they fit in multiple categories

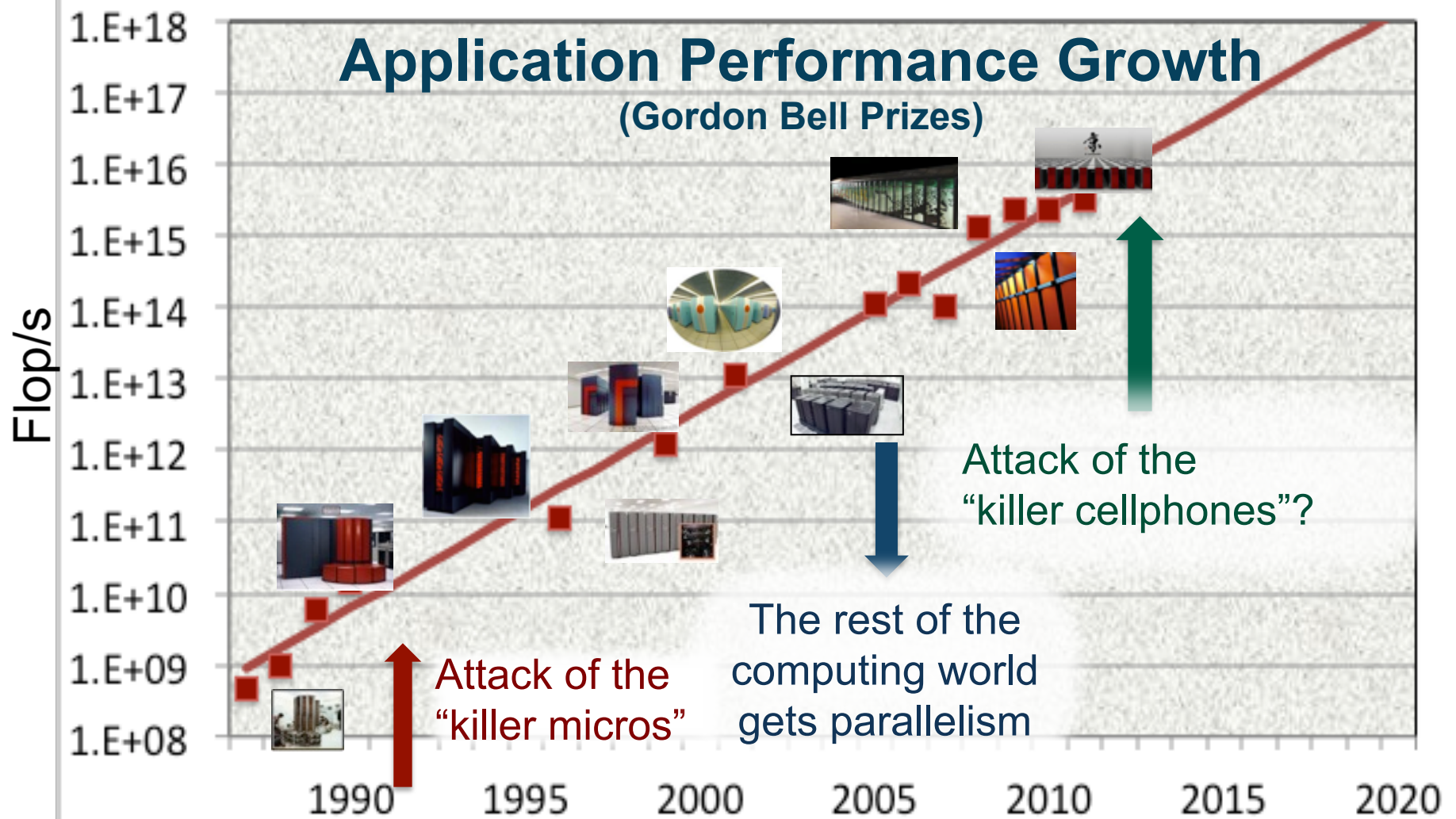




**More performance is needed through
experimental and theoretical science**



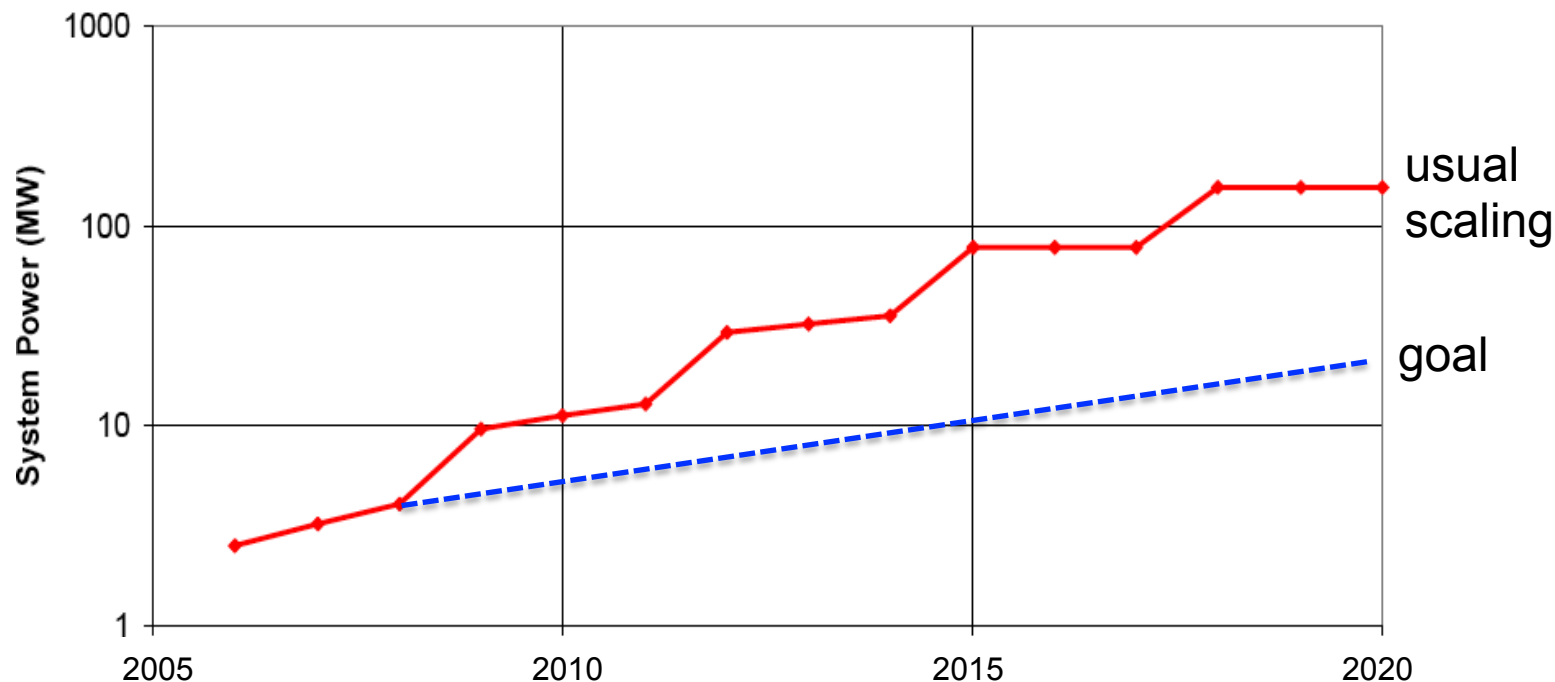
Computational Science has Moved through Difficult Technology Transitions



Energy Efficient Computing is Key to Performance Growth

At \$1M per MW, energy costs are substantial

- 1 petaflop in 2010 used 3 MW
- 1 exaflop in 2018 would use 100+ MW with “Moore’s Law” scaling



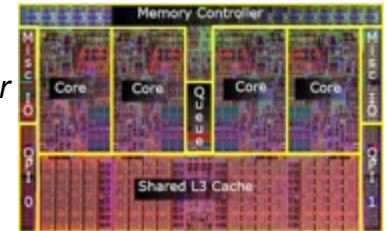
This problem doesn't change if we were to build 1000 1-Petaflop machines instead of 1 Exasflop machine. It affects every university department cluster and cloud data center.



Challenge: Node Programming, Heterogeneity

- **Case for heterogeneity**
 - Many small cores and SIMD for energy efficiency; few CPUs for OS / speed
- **Local store, explicitly managed memory**
 - More efficient (get only what you need) and simpler hardware
- **Split memory between CPU and “Accelerators”**
 - Driven by market history and simplicity, but may not last
 - Communication: The bus is a significant bottleneck.
- **Co-Processor interface between CPU and Accelerator**
 - Default is on CPU, only run “parallel” code in limited regions
 - Why are the minority CPUs in charge?

Cell phone processor
(0.1 Watt, 4 Gflop/s)



Server processor
(100 Watts, 50 Gflop/s)

Avoid vicious cycle: Programming model should be designed for future, not for current/past constraints



Challenge: Communication is expensive

Communication is expensive...
... time and energy

Cost components:

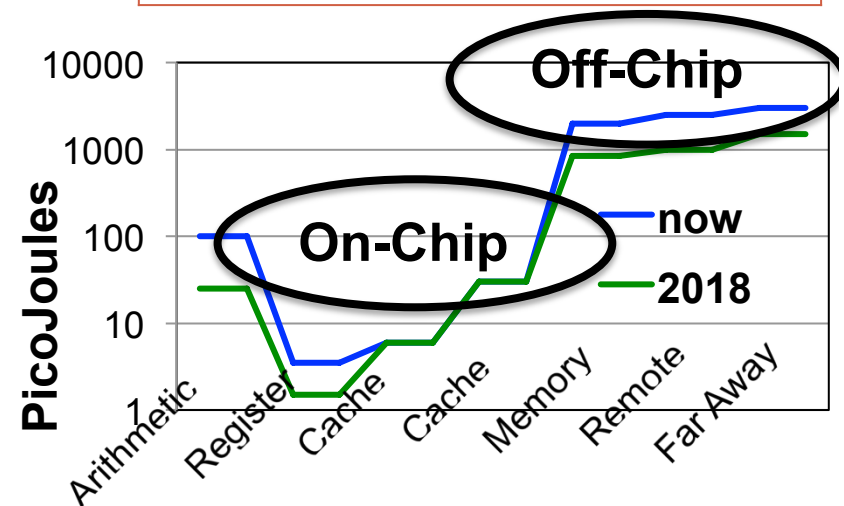
- **Bandwidth:** # of words
- **Latency:** # messages

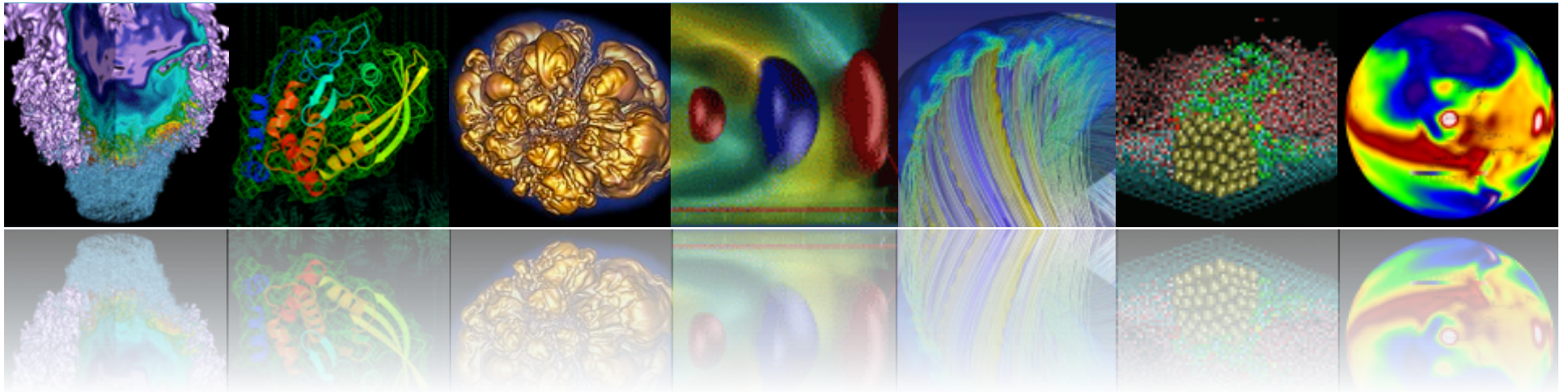
Strategies

- Overlap: hide latency
- Avoid: new algorithms to reduce bandwidth (at least)

Hard to change: Latency is physics; bandwidth is money!

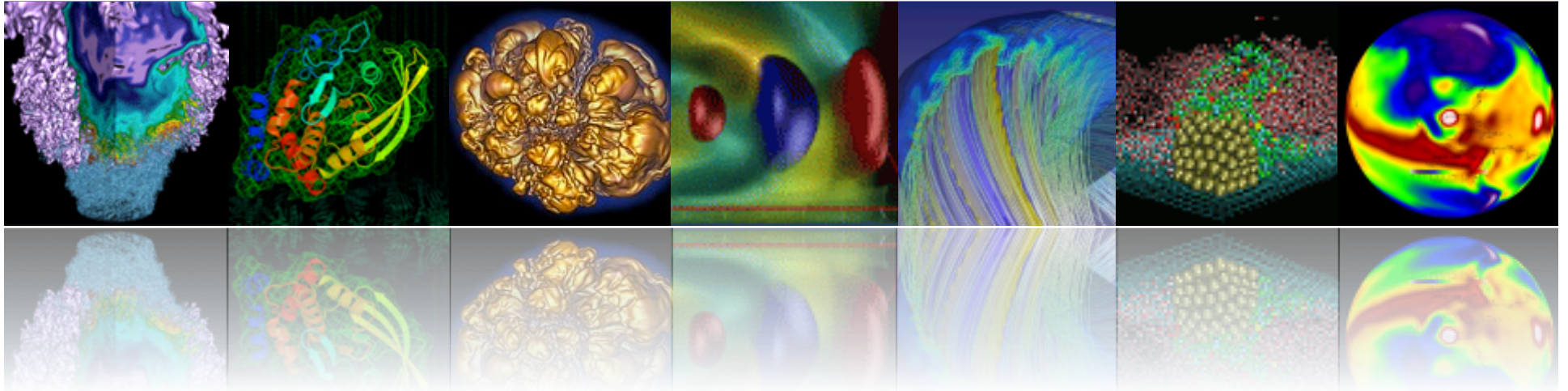
Annual improvements			
Flops		BW	Latency
59%	Network	26%	15%
	DRAM	23%	5%





Getting to Exascale





Lesson #1: Co-Design architectures for Science

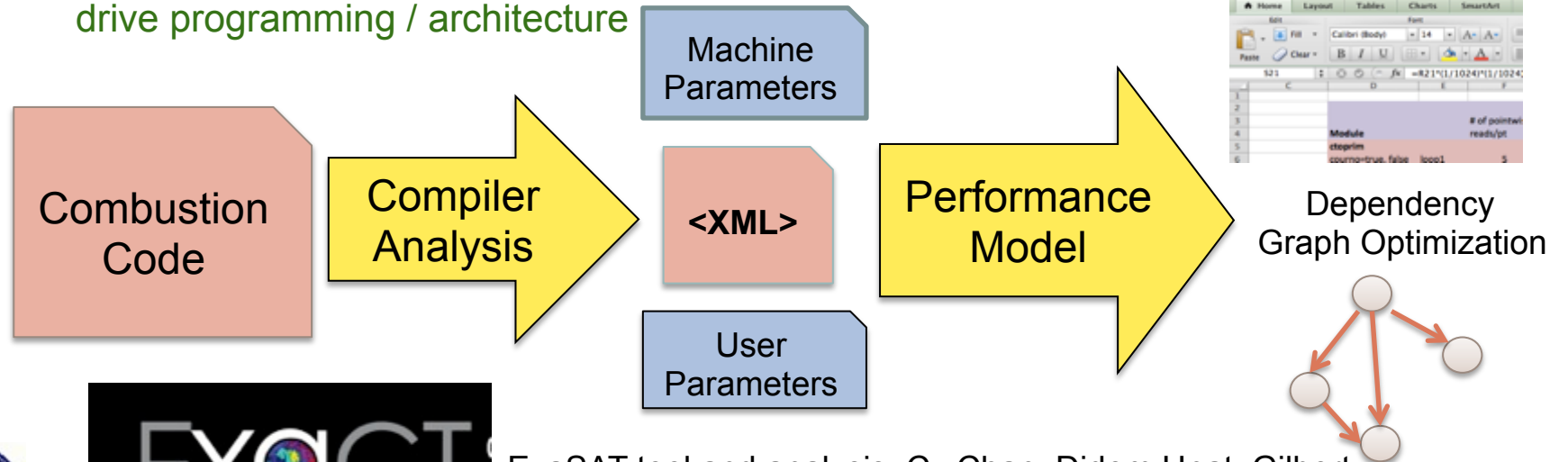
Keeping in mind market pressures



Application Strategy Based on Co-Design

Goal: Use scientific requirements to guide architecture and system software and use technology capabilities to design algorithms and software

- **Each centers has a focused application domain**
 - ExaCT (Combustion), ExMatEx (Material), CESAR (Nuclear energy)
- **Different types of “proxy apps”**
 - Scaled down versions of “full” code
 - Selects parts/patterns from code to drive programming / architecture

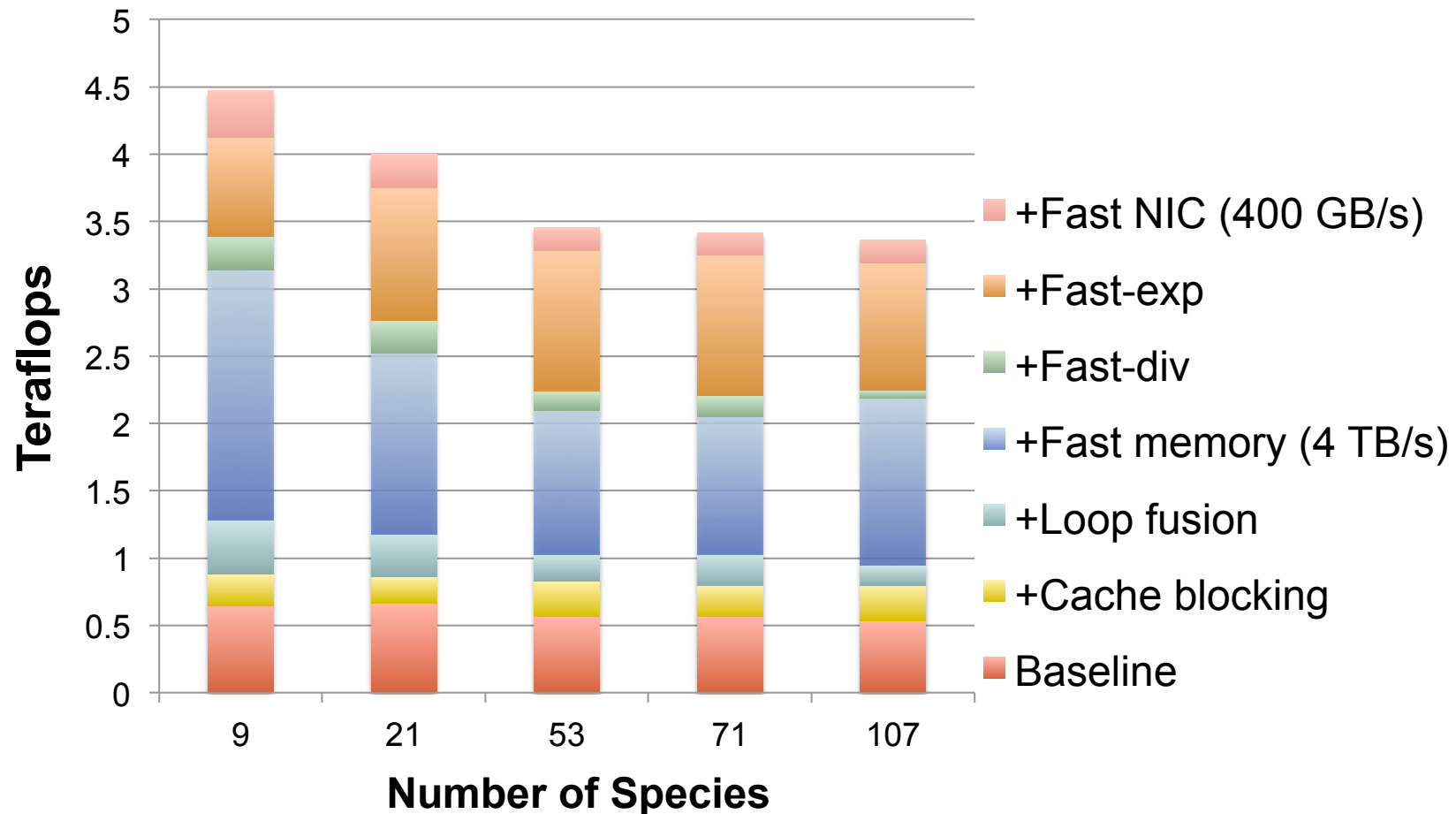


ExaSAT tool and analysis: Cy Chan, Didem Unat, Gilbert Hendry, John Bell and John Shalf



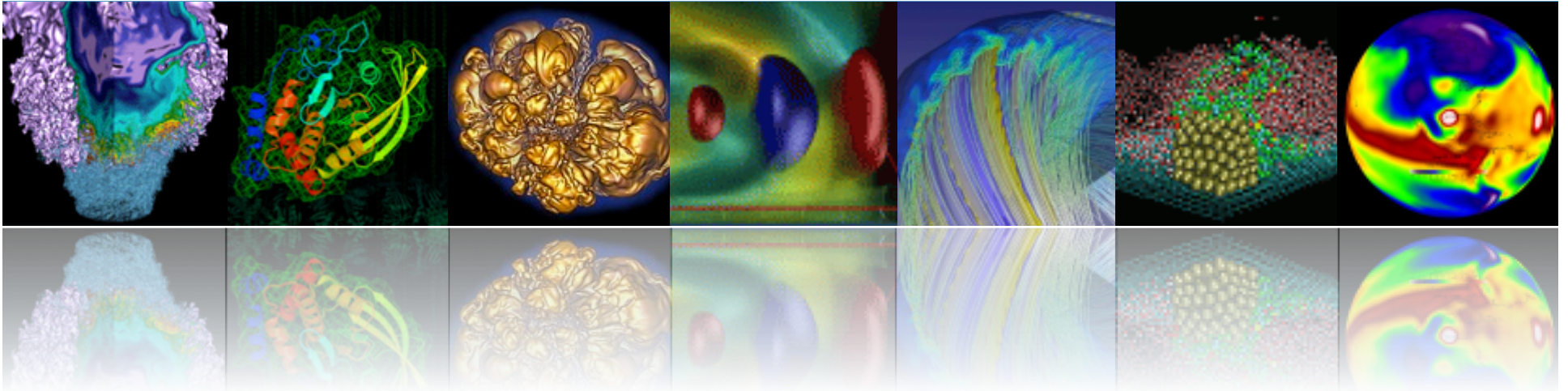
Co-Design Analysis for Combustion

Estimated Performance Improvements



- Hardware and software need to change together

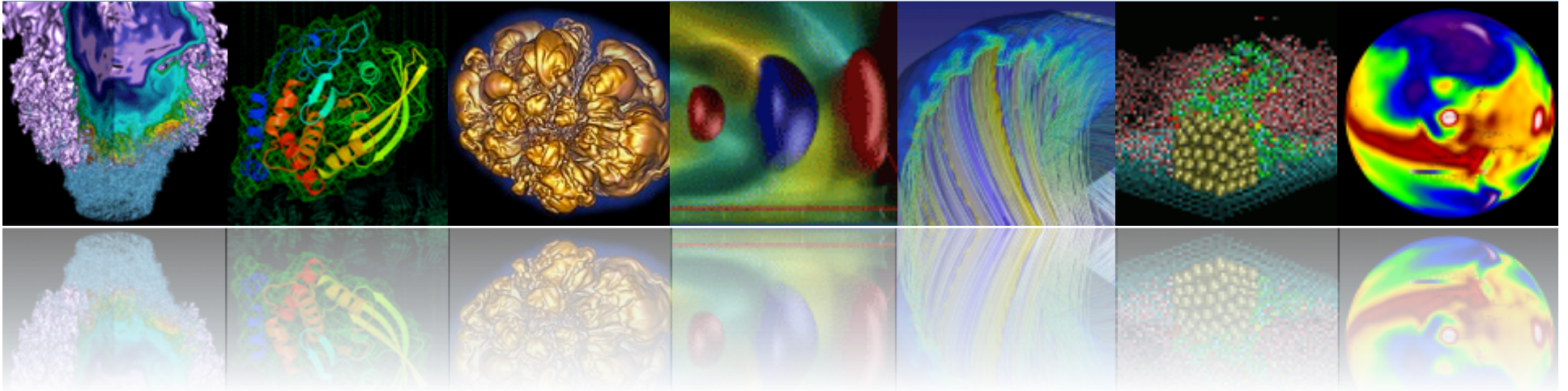




**Essentially, all models are wrong, but
some are useful.**

-- George E. Box, Statistician





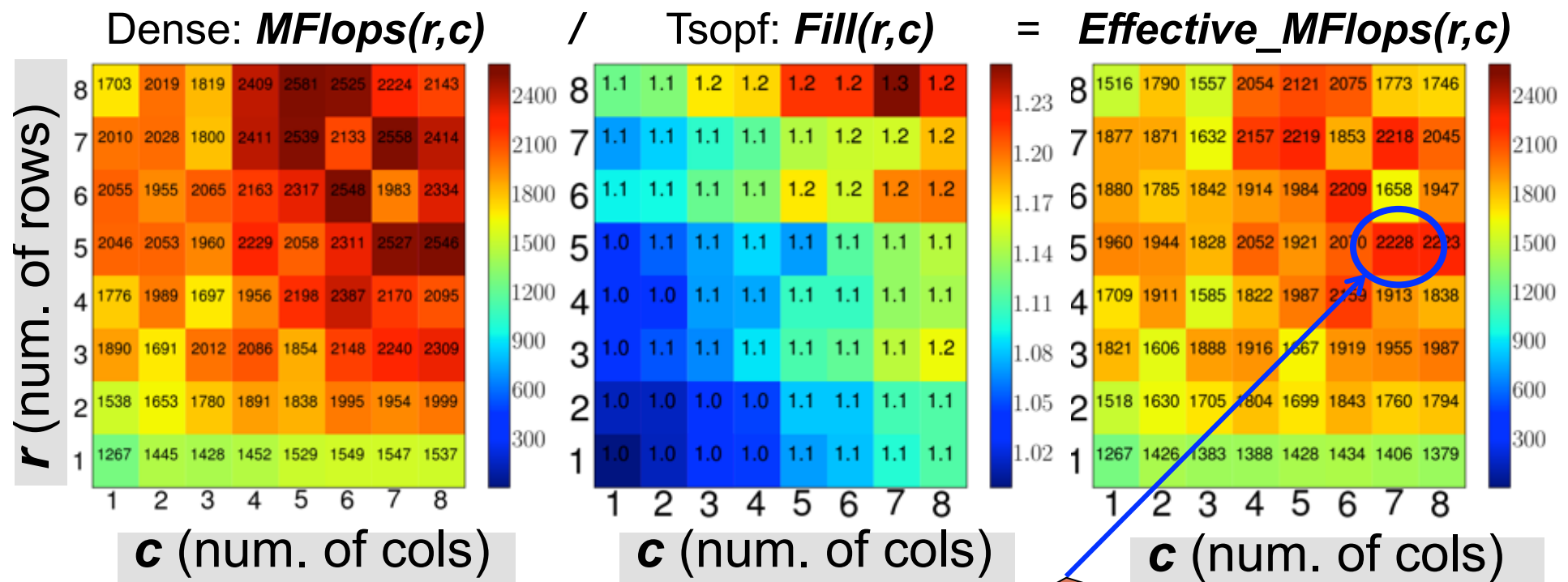
**Lesson #2: Let computers, not humans,
tune for modern architectures code**

But beware of trusting compilers



Finding Good Performance is like finding the Needle in a Haystack

OSKI sparse matrix library: offline search + online evaluation:
adding zeros can reduce storage in blocked **format**

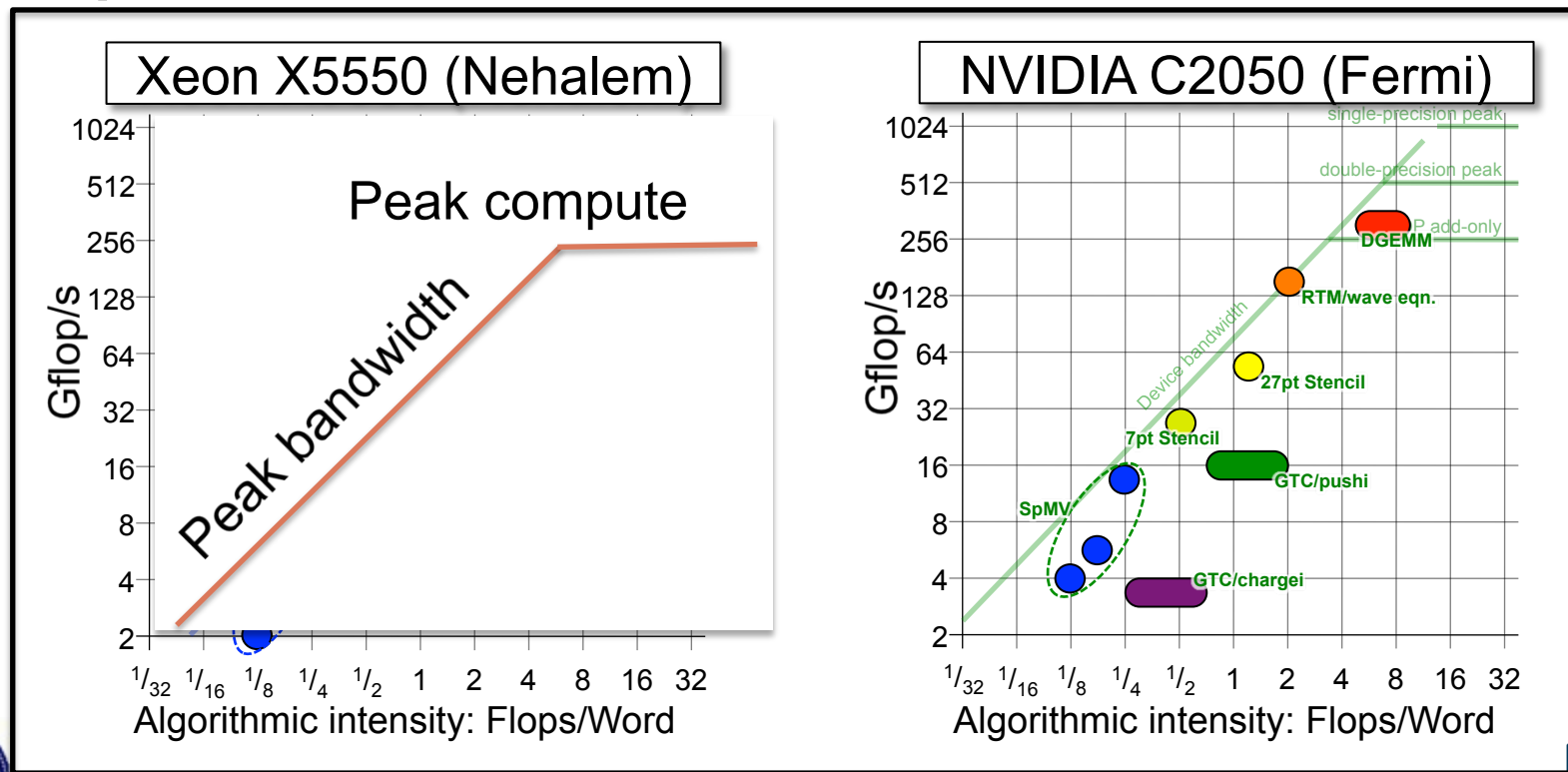


Work by Im, Vuduc, Williams, Kamil, Ho, Demmel, Yelick...



Autotuning: Write Code Generators

- Autotuners are code generators plus search
- Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- Popular in libraries: Atlas, FFTW, OSKI,...

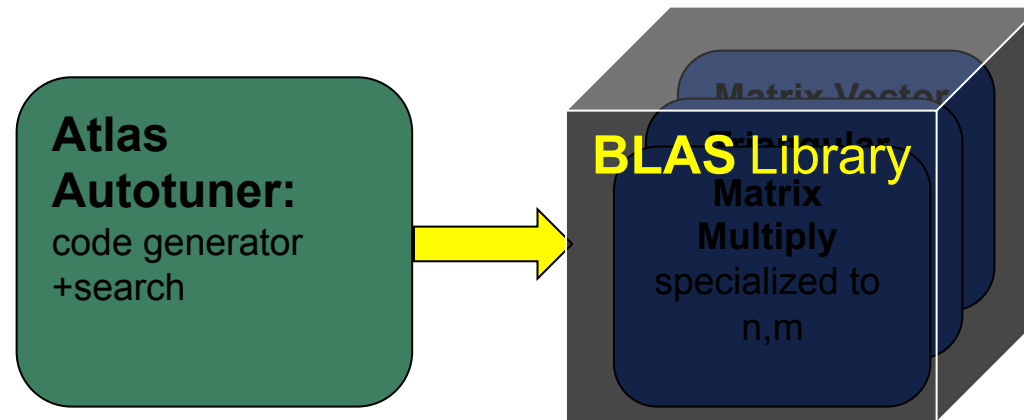


Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,...

24



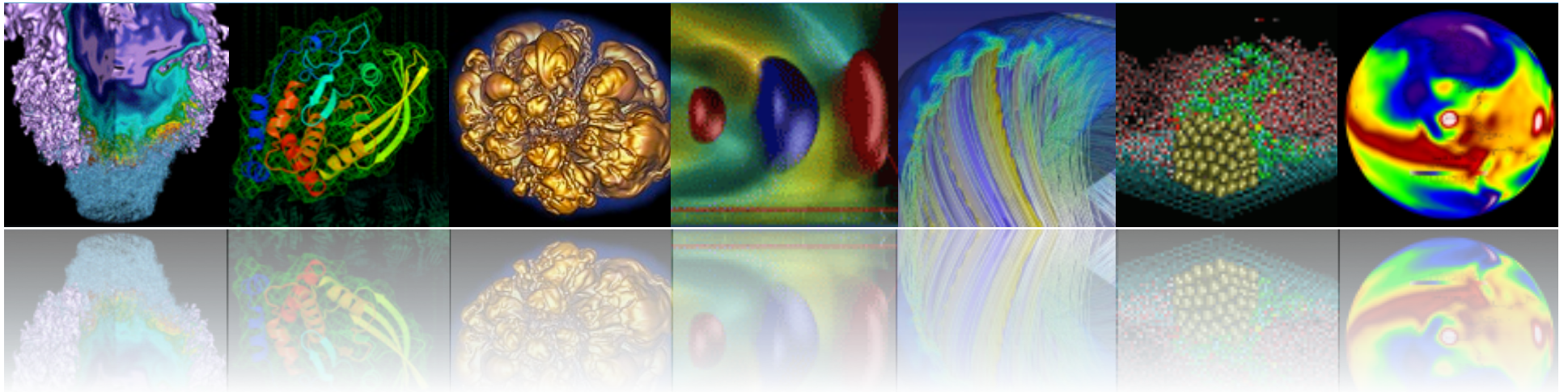
Approaches to Autotuning



How do we produce all of these (correct) versions?

- Using scripts (Python, perl, C,..)
- Transform high level representation (FFTW, Spiral)
- Compiling a domain-specific language (D-TEC)
- Compiling a general-purpose language (X-Tune)
- Dynamic compilation of a domain-specific (SEJITS)



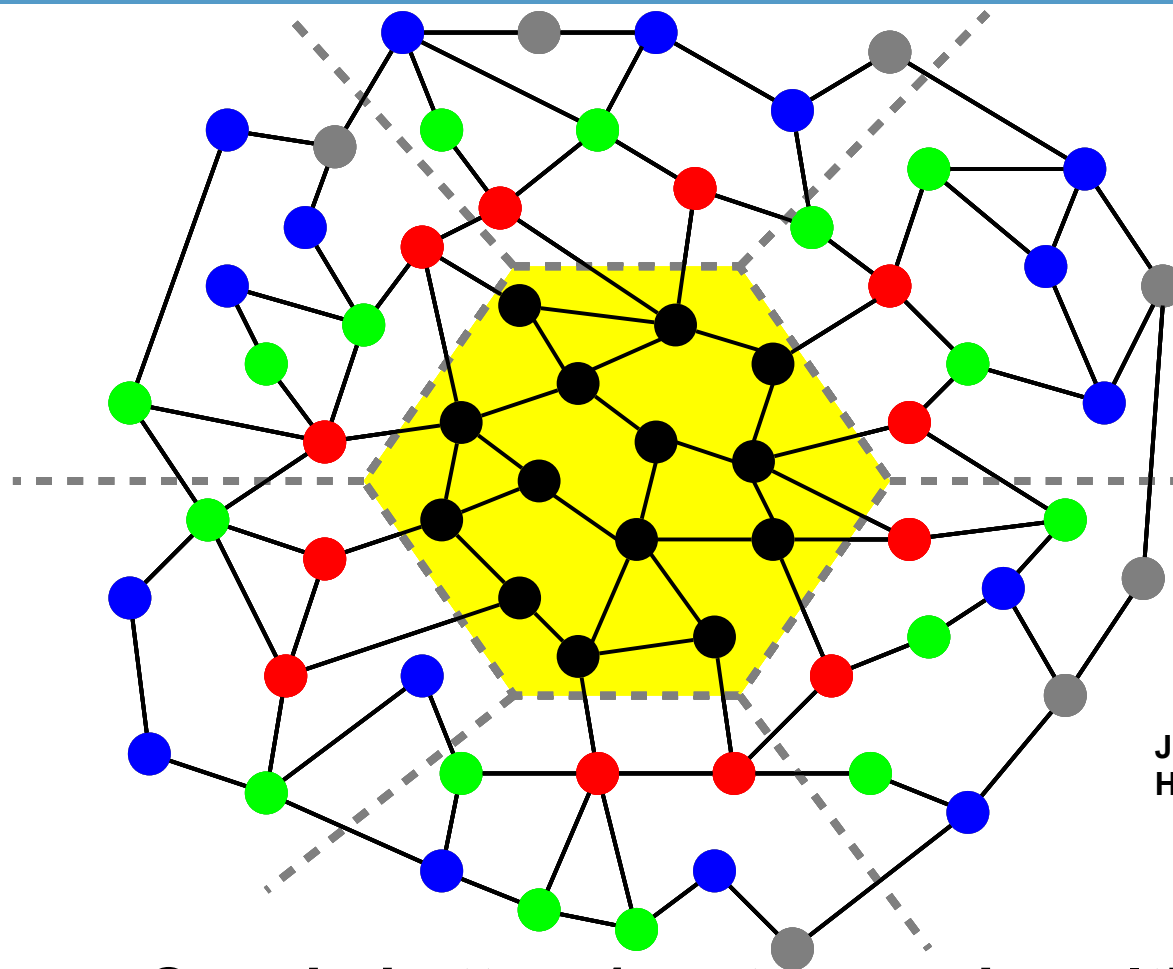


Lesson #3: Target Higher Level Loops

Harder than inner loops....



Iterative Solves are Dominated by Sparse Matrix-Vector Multiply (nearest neighbor on graph)



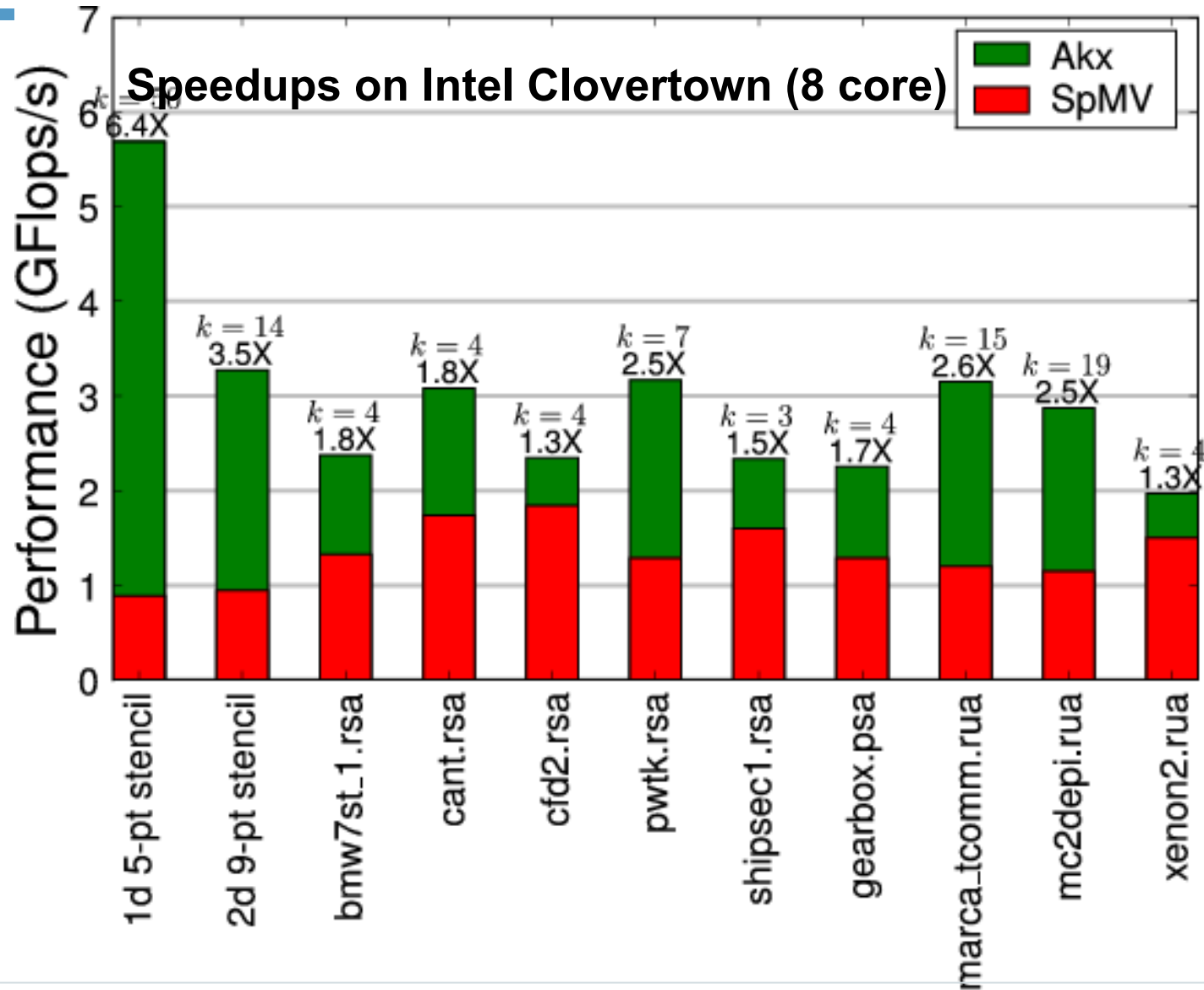
For implicit memory management (caches) uses a TSP algorithm for layout

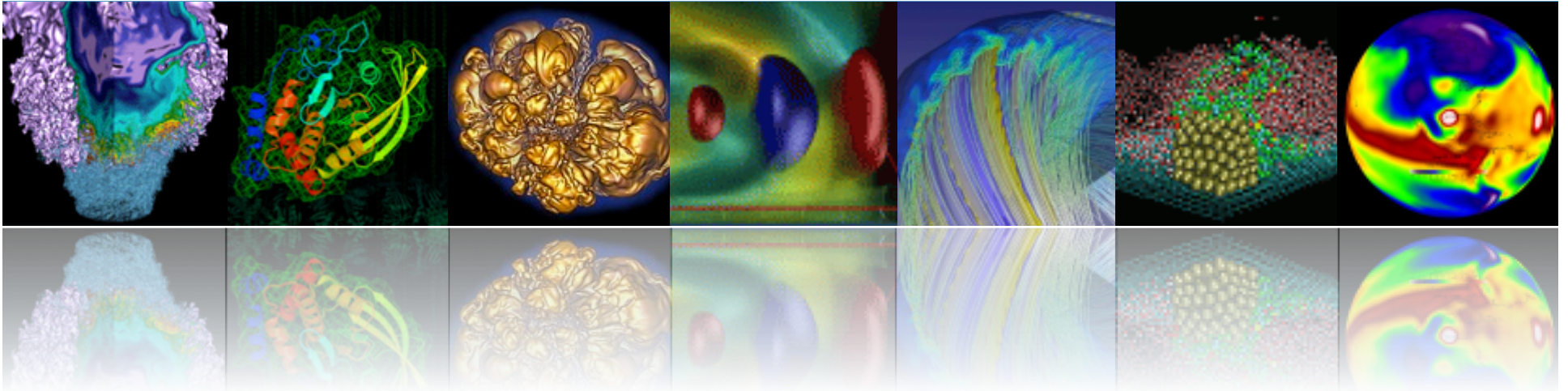
Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

- **Can do better: 1 matrix read, multiple multiplies**
 - Serial: $O(1)$ moves of data moves vs. $O(k)$
 - Parallel: $O(\log p)$ messages vs. $O(k \log p)$



Bigger Kernel (A^kx) Runs at Faster Speed than Simpler (Ax)



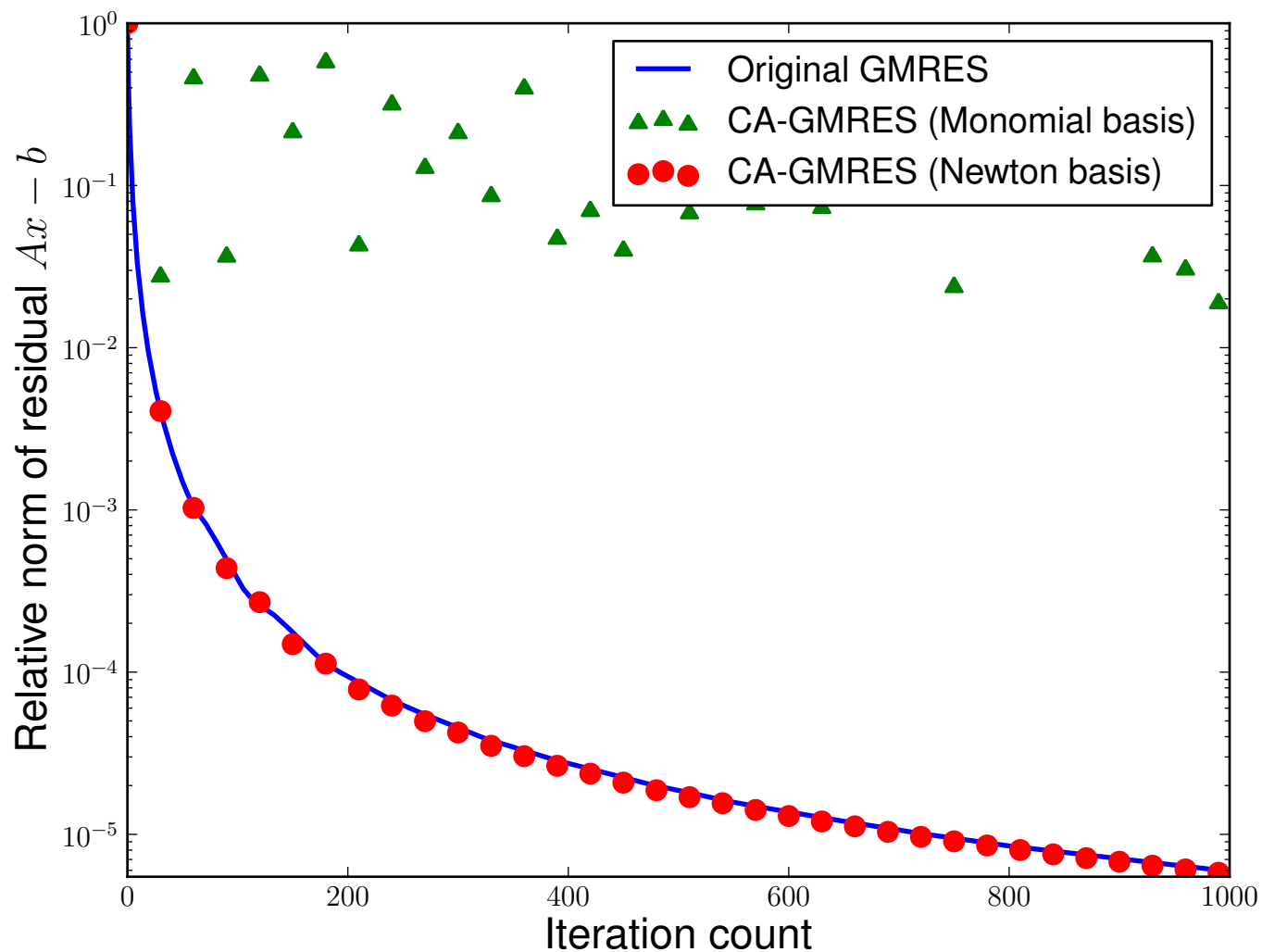


Lesson #4: Understand Numerics (Or work with someone who does)

**Don't be afraid to change to a
“different” right answer**



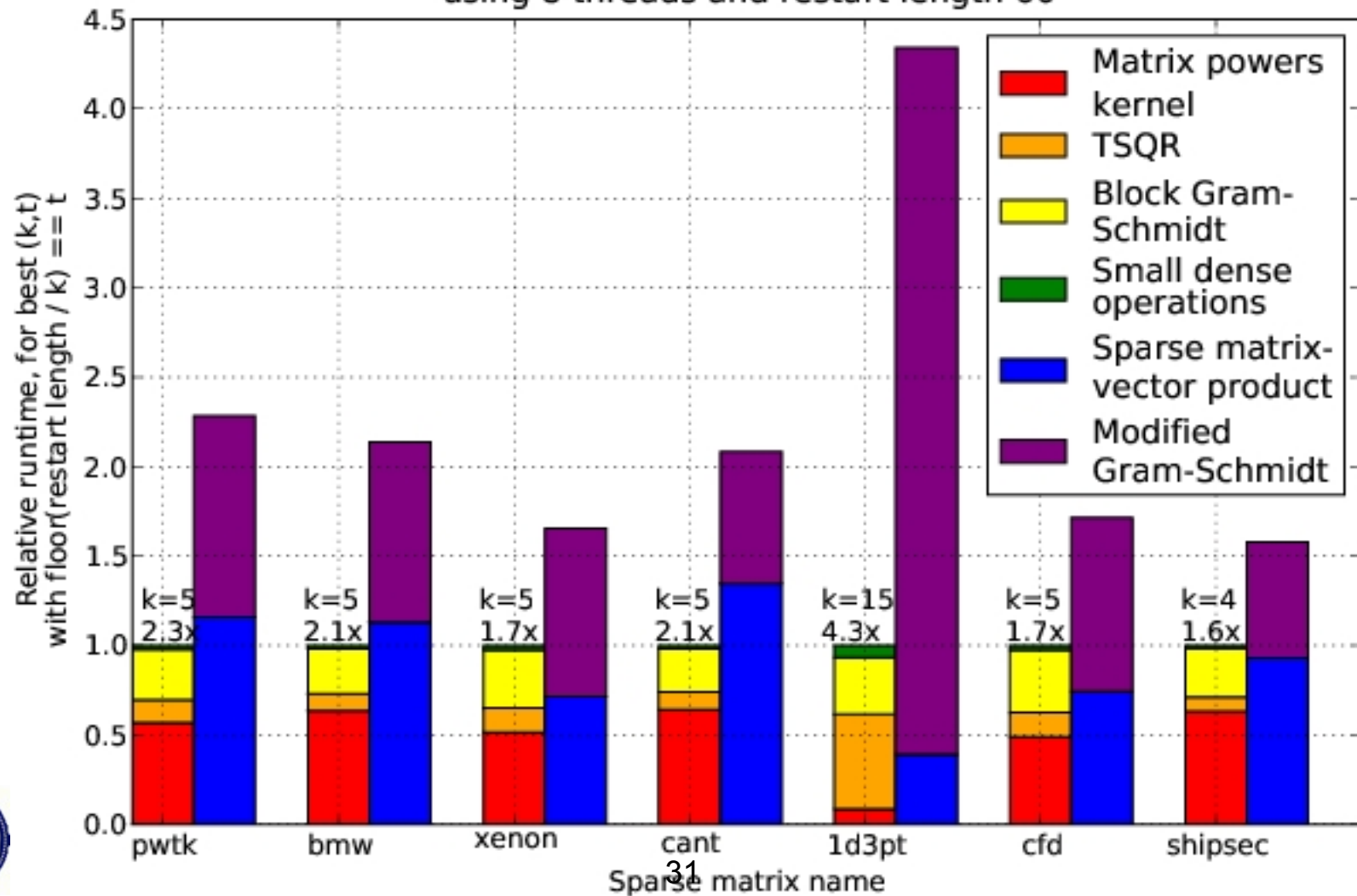
Matrix Powers Kernel (and TSQR) in GMRES

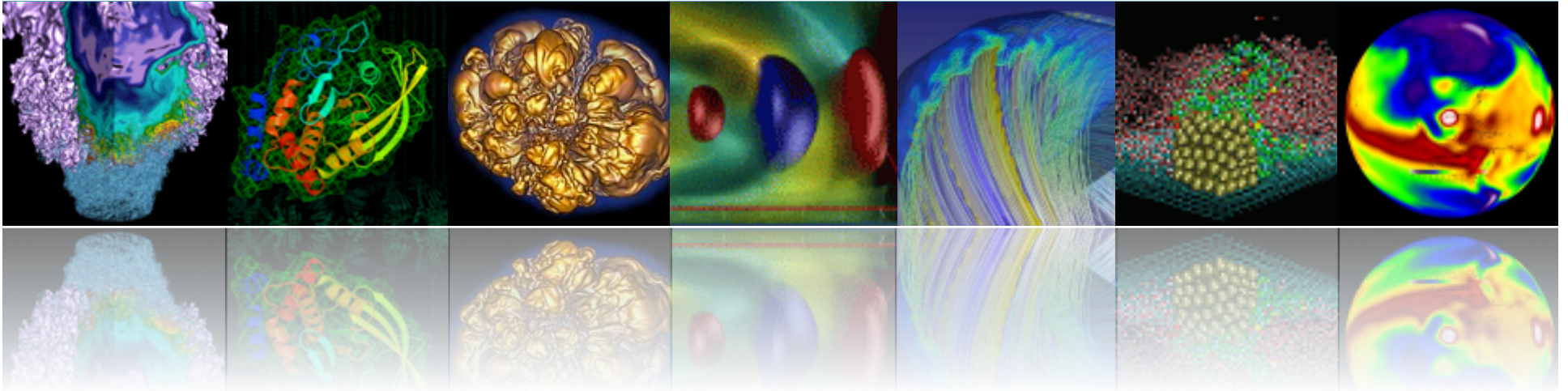


Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60





Lesson #5: Never Waste Fast Memory

Don't get hung up on the “owner computes” rule.



Beyond Domain Decomposition

2.5D Matrix Multiply on BG/P, 16K nodes / 64K cores

$c = 16$ copies

Matrix multiplication on 16,384 nodes of BG/P

Surprises:

- Even Matrix Multiply had room for improvement
- Idea: make copies of C matrix (as in prior 3D algorithm, but not as many)
- Result is provably optimal in communication

Lesson: Never waste fast memory

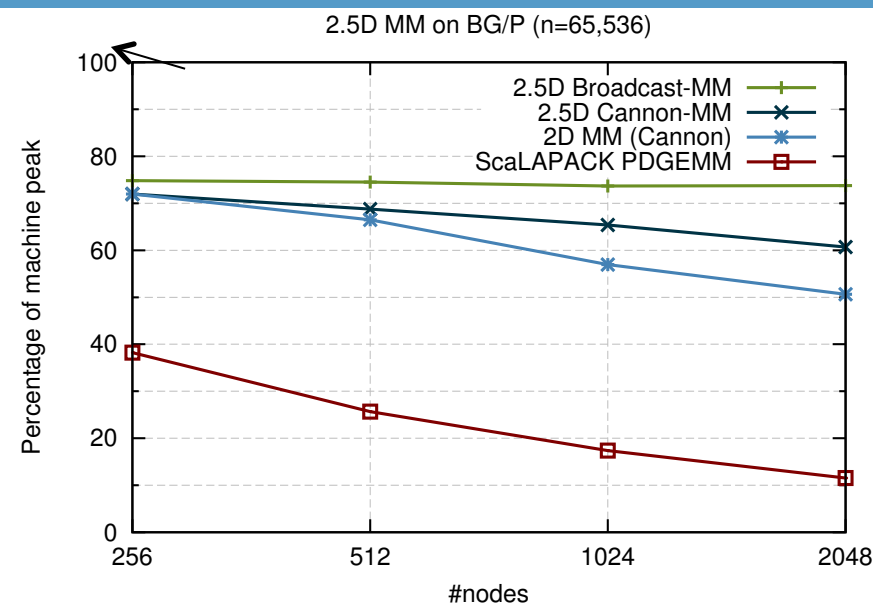
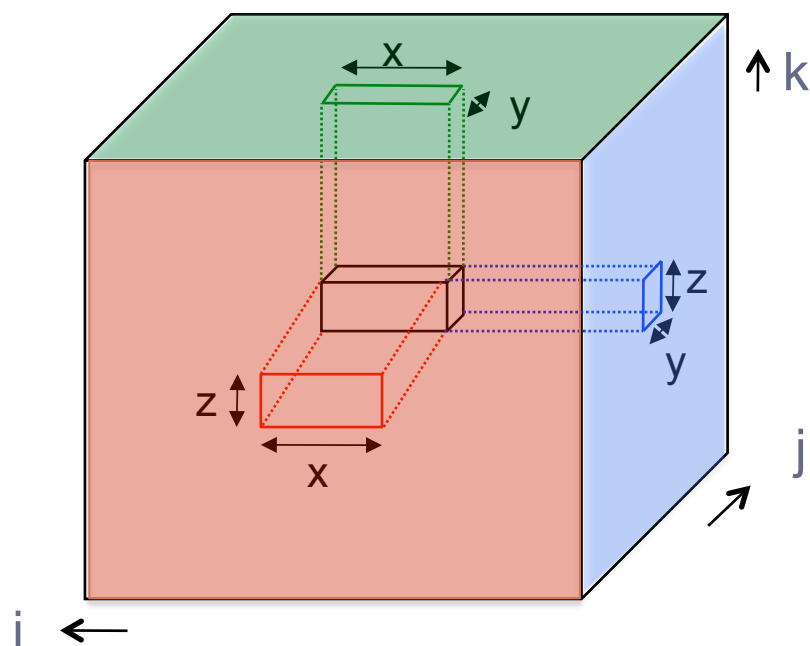
Can we generalize for compiler writers?



EuroPar'11 (Solomonik, Demmel)
SC'11 paper (Solomonik, Bhatele, Demmel)



Towards Communication-Avoiding Compilers

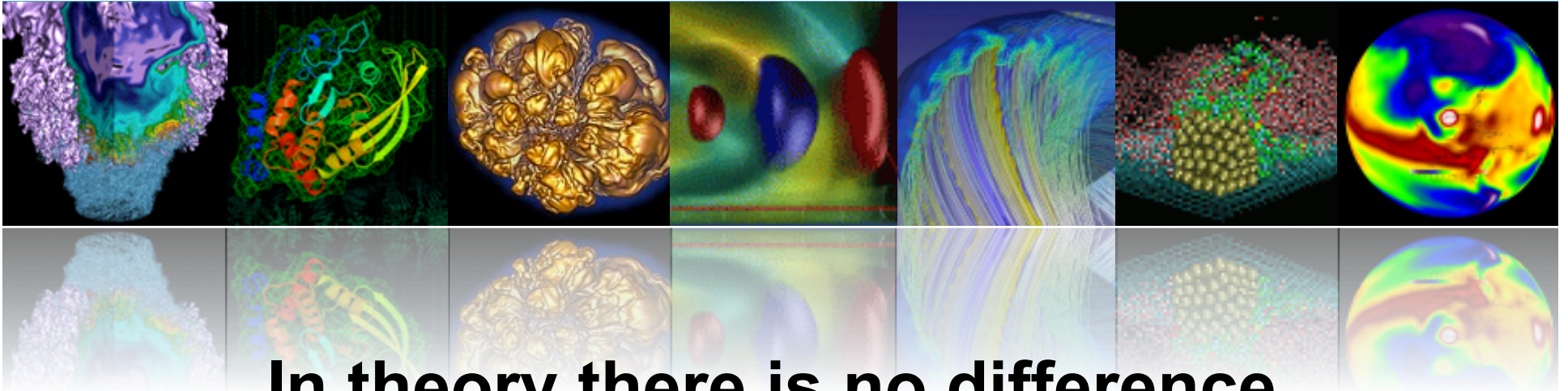


Matrix Multiplication code has a 3D iteration space
Each point in the space is a constant computation ($\ast/\ast$)

for i, for j, for k $C[i,j] \dots A[i,k] \dots B[k,j] \dots$

These are not just “avoiding,” they are “communication-optimal”





**In theory there is no difference
between theory and practice, but in
practice there is.**

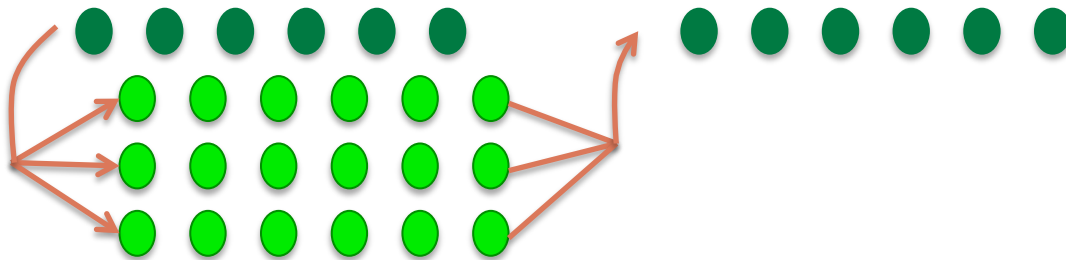
***-- Jan L. A. van de Snepscheut, Computer Scientist
or***

-- Yogi Berra, Baseball player and manager



Generalizing Communication Optimal Transformations to Arbitrary Loop Nests

1.5D N-Body: Replicate and Reduce



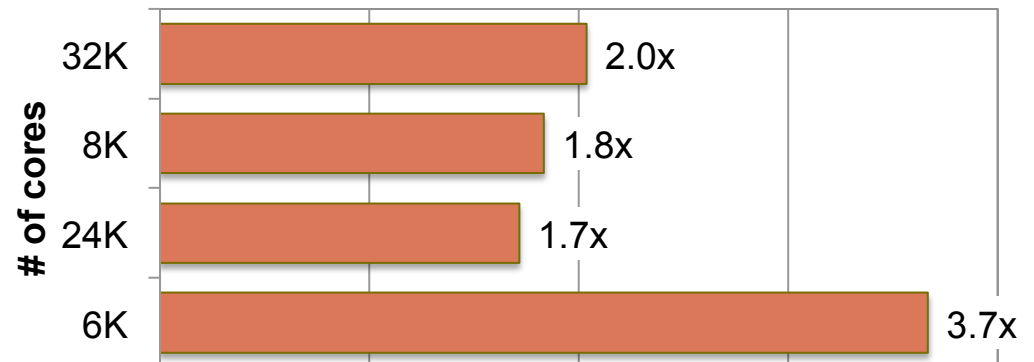
The same idea (replicate and reduce) can be used on (direct) N-Body code:

1D decomposition $\rightarrow$ “1.5D”

Does this work in general?

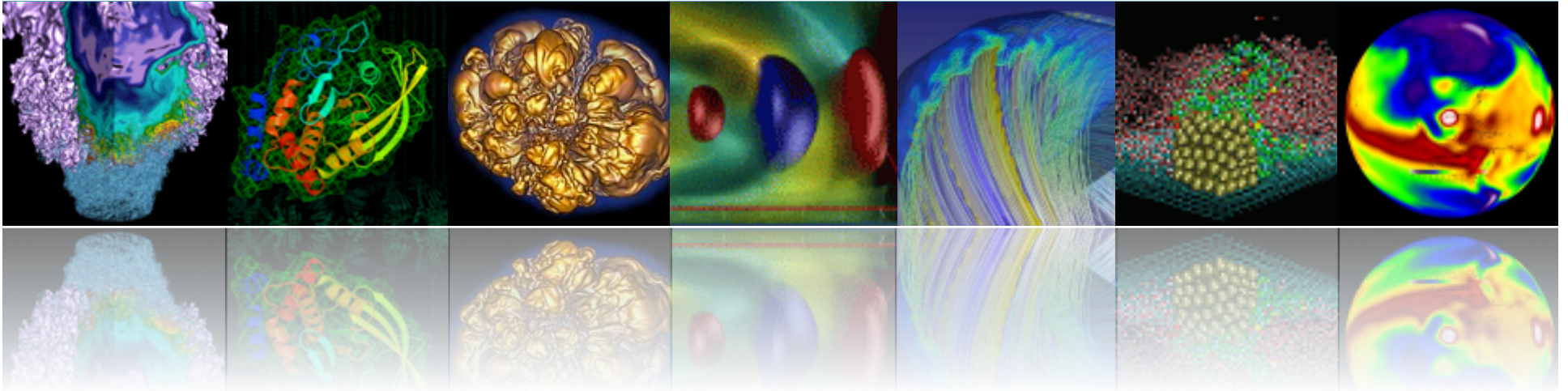
- *Yes, for certain loops and array expressions*
- *Relies on basic result in group theory*
- *Compiler work TBD*

Speedup of 1.5D N-Body over 1D



IPDPS'13 paper (Driscoll, Georganas, Koanantakool, Solomonik, Yelick)





Lesson #6: Overlap and Pipeline Communication

***Sometimes runs contrary to common
wisdom on message aggregation***



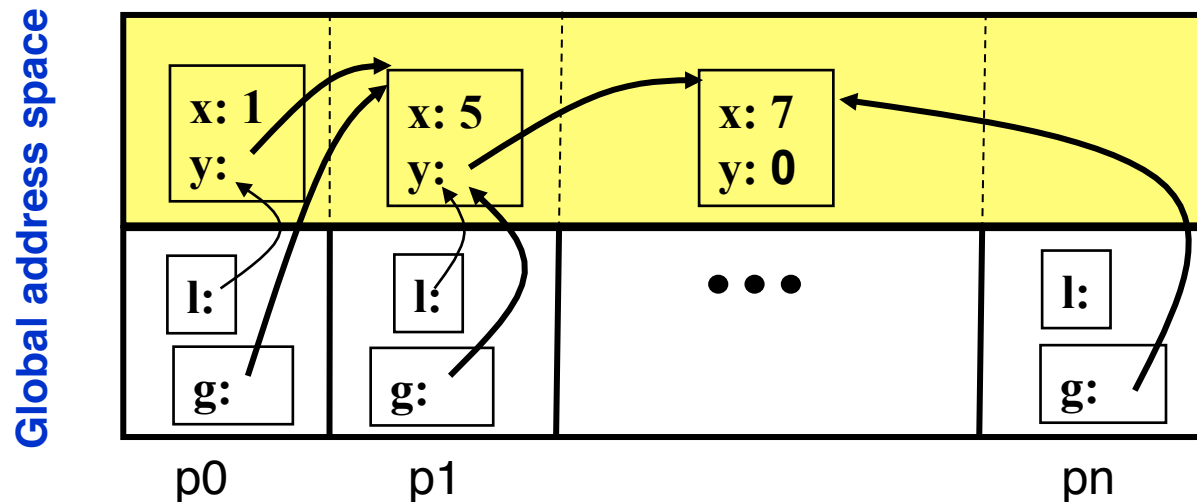
Partitioned Global Address Space for Convenience and Performance

PGAS: Partitioned Global Address Space

Global address space: directly read/write remote data

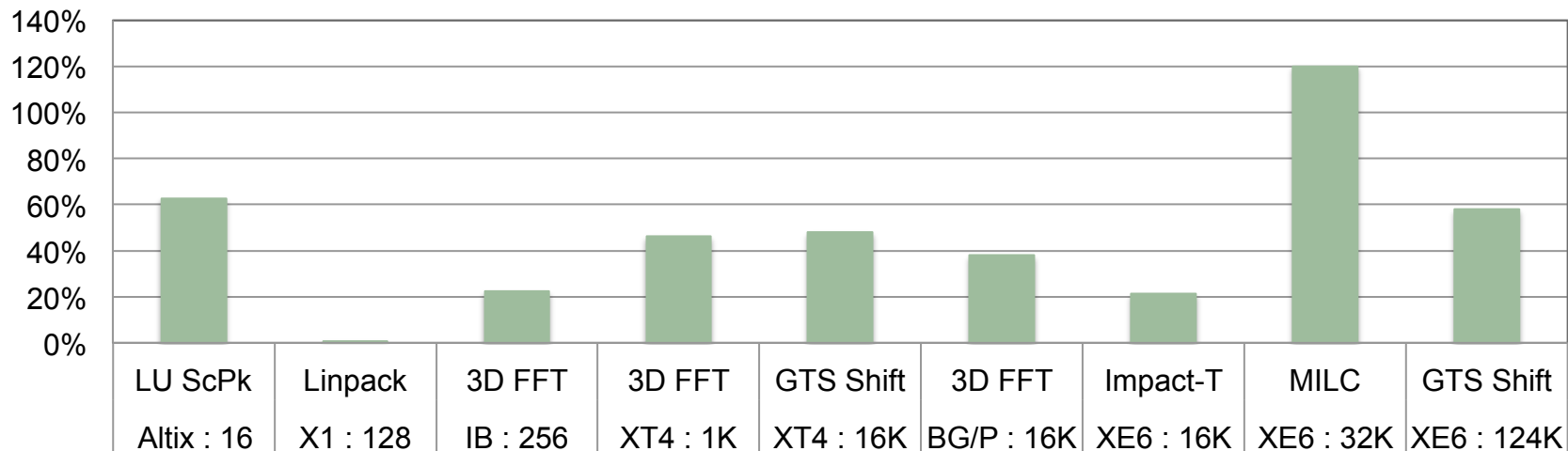
Partitioned: data is designated as local or global

- Affinity control
- Never say “receive”
- Note: not cache coherent shared mem



PGAS's One-sided communication has performance advantages

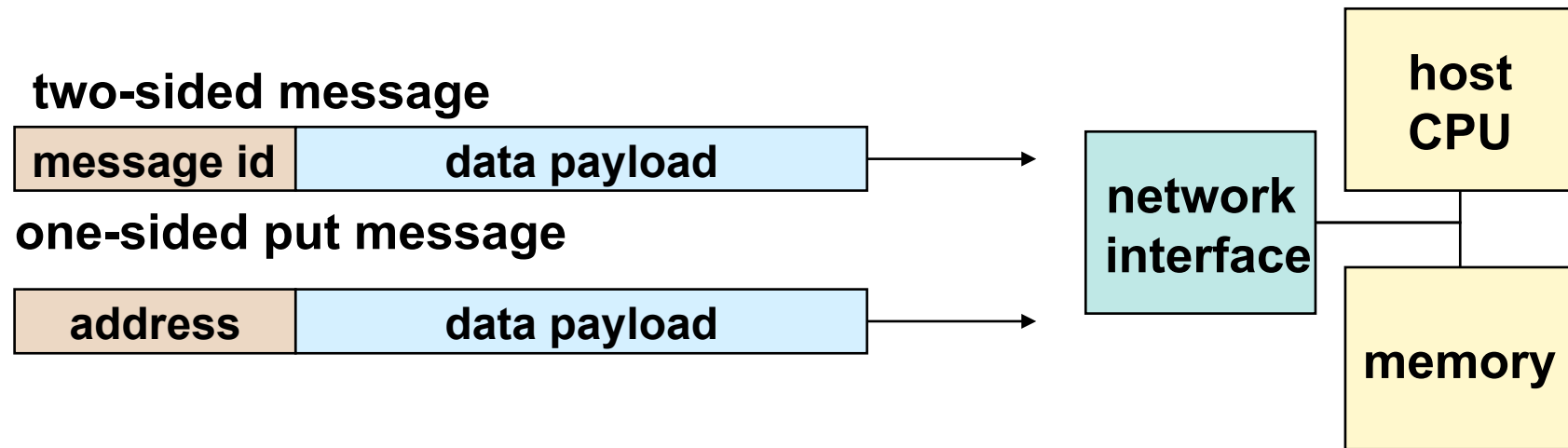
Speedup of PGAS over MPI



- **Performance advantages for PGAS over MPI from**
- **Lower latency and overhead**
- **Better pipeline (overlapping communication with communication)**
- **Overlapping communication with computation**
- **Use of bisection bandwidth**



Avoiding Synchronization in Communication



- **Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data**
 - Wildly popular in HPC, but cumbersome in some applications
 - Couples data transfer with synchronization
- **Using global address space decouples synchronization**
 - Pay for what you need!
 - Note: Global Addressing $\neq$ Cache Coherent Shared memory



One-sided communication is a mechanism that works everywhere

PGAS is a programming model

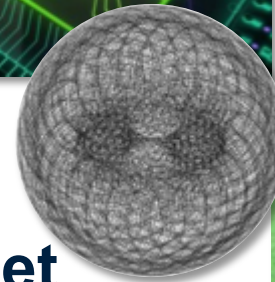
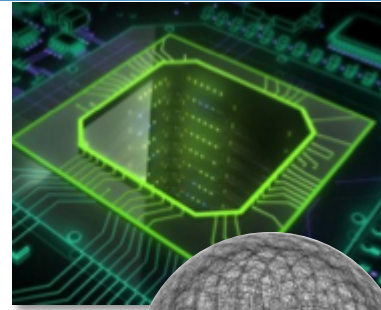
```
*p1 = *p2 + 1;  
A[i] = B[i];
```

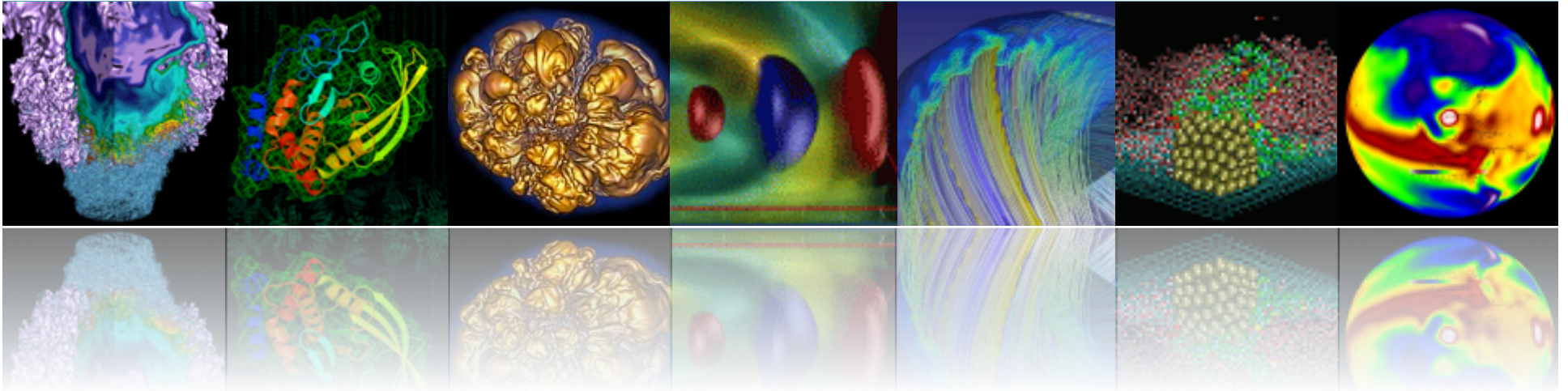
```
upc_memput(A, B, 64);
```

Uses 1-sided communication: put/get

This Direct Memory Access (DMA) also appears in:

- Fast one-sided network communication (RDMA, Remote DMA)
- Move data to/from accelerators
- Move data to/from I/O system (Flash, disks,...)
- Movement of data in/out of local-store (scratchpad) memory





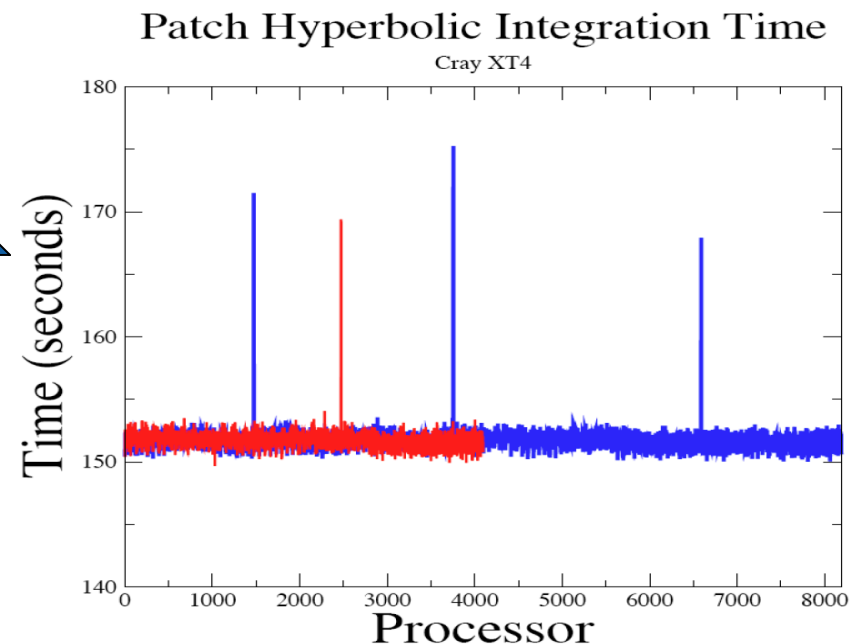
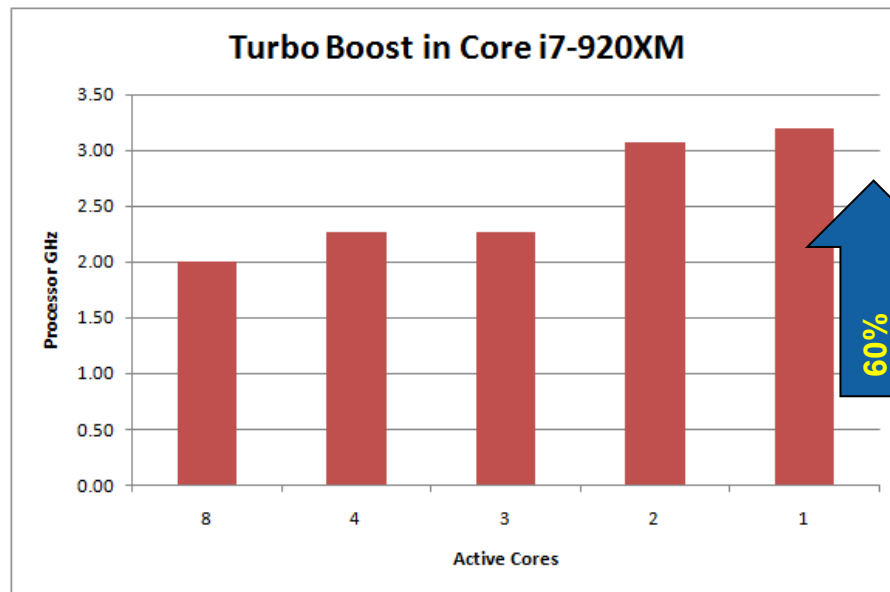
Lesson #7: Avoid Synchronization, a Particular form of Communication

Use One-sided Communication



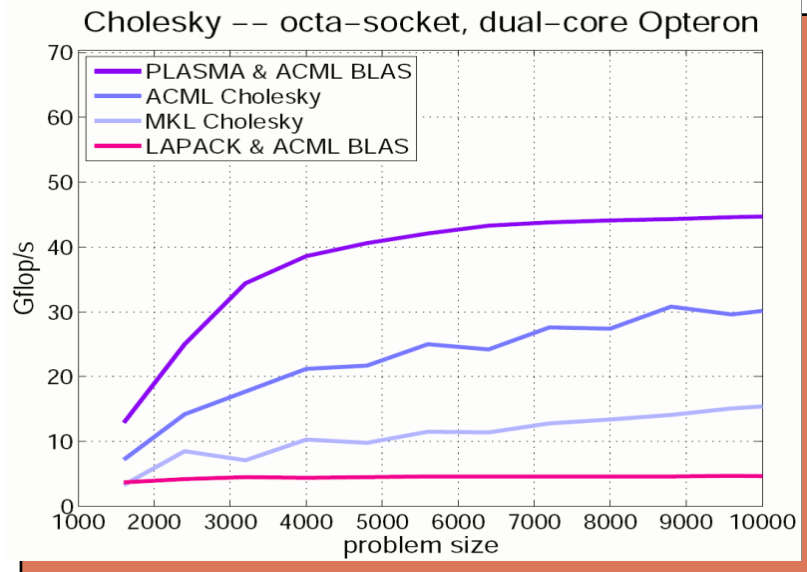
Reasons to avoid synchronization

- **Processors do not run at the same speed**
 - Never did, due to caches
 - Power / temperature management makes this worse

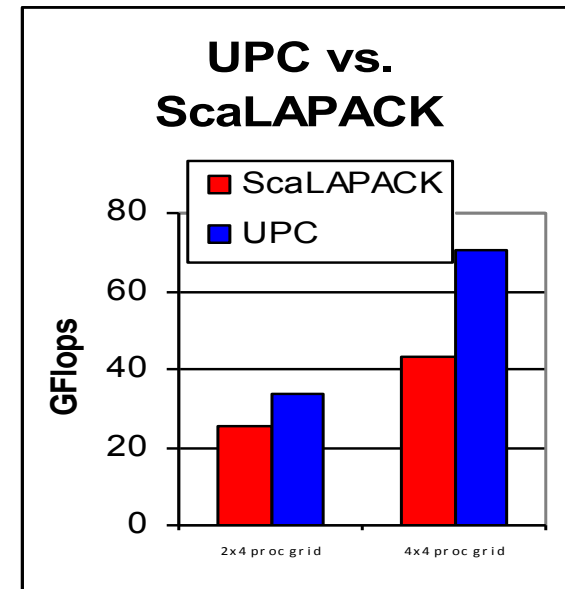


DAG Scheduling Outperforms Bulk-Synchronous Style

PLASMA on shared memory



UPC on partitioned memory



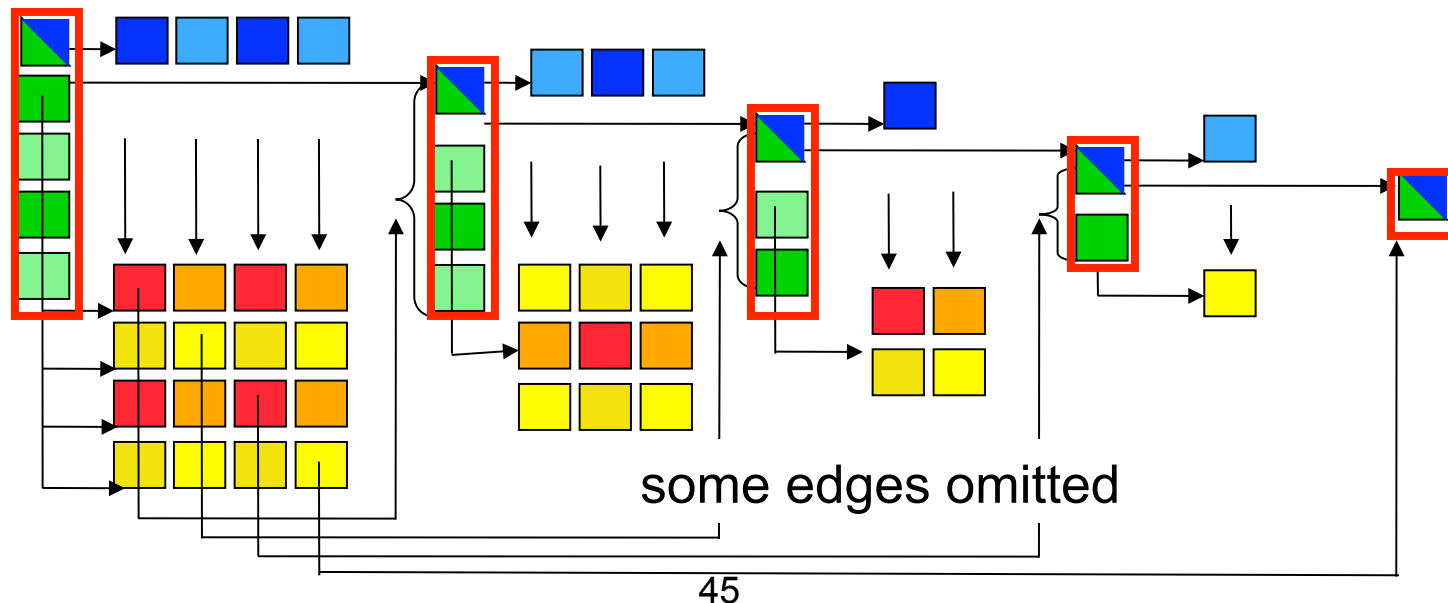
UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding

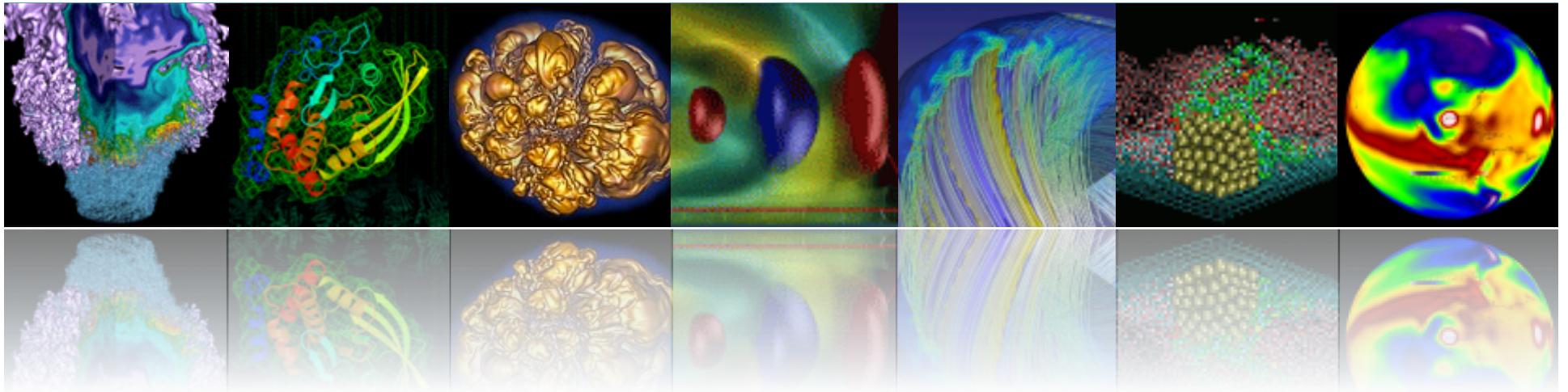
- New problem in partitioned memory: allocator deadlock
- Can run on of memory locally due tounlucky execution order



Event Driven LU in UPC

- Assignment of work is static; schedule is dynamic
- Ordering needs to be imposed on the schedule
 - Critical path operation: Panel Factorization
- General issue: dynamic scheduling in partitioned memory
 - Can deadlock in memory allocation
 - “memory constrained” lookahead



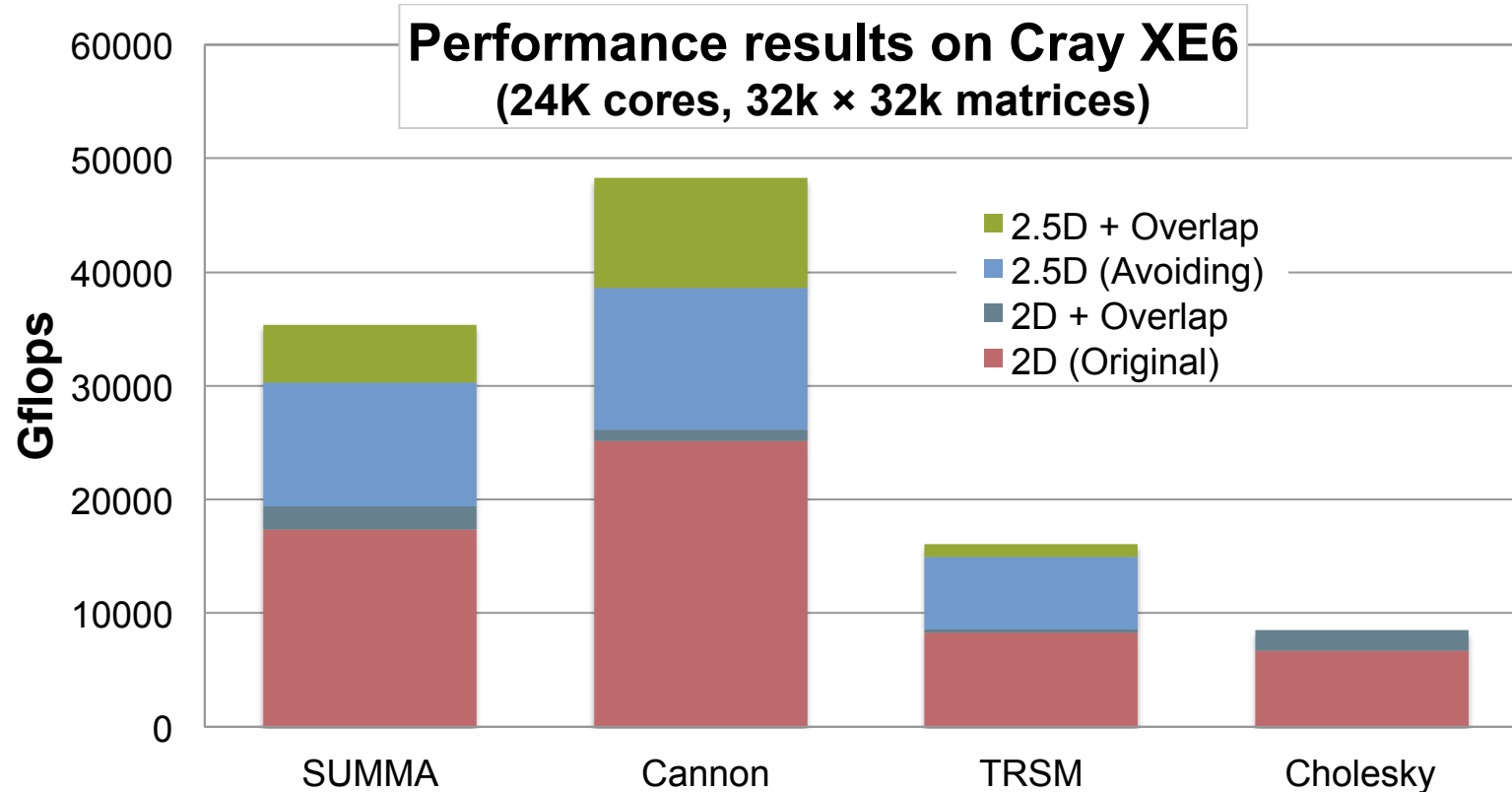


Lesson #8: Combine Techniques

These are not entirely orthogonal, and there are many interesting trade-offs.



Communication Overlap Complements Avoidance



Even with communication-optimal algorithms (minimized bandwidth) there are still benefits to overlap and other things that speed up networks



SC'12 paper (Georganas, González-Domínguez, Solomonik, Zheng, Touriño, Yelick)



Lessons in Antisocial Parallelism

- 1. Compress Data Structures**
- 2. Target Higher Level Loops**
- 3. Understand theory / numerics**
- 4. Replicate data**
- 5. Understand theory / lower bounds**
- 6. Aggregate communication**
- 7. Overlap communication**
- 8. Use one-sided communication**
- 9. Synchronization strength reduction**
- 10. Combine the techniques**

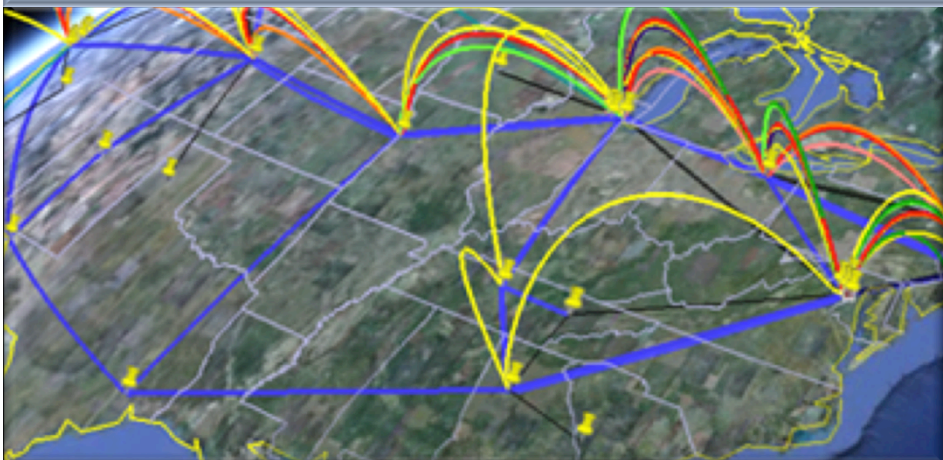


Computing Sciences at Berkeley Lab

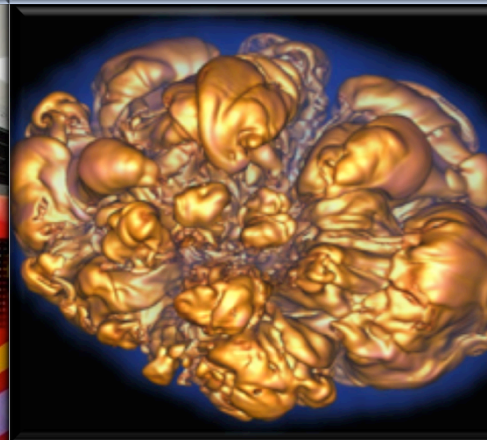
NERSC Facility



ESnet Facility



Computational Research



Applied Mathematics



Computational Science

$$\begin{aligned} \frac{dVar(t)}{dt} &= \\ &= -K^2 Fg + gK^2 F ds - \int_{s_2(t)}^{s_1(t)+S} (-g^{-1}F_s)_s - K \\ &= -\int_{s_1(t)}^{s_2(t)} (g^{-1}F_s)_s ds + \int_{s_2(t)}^{s_1(t)+S} (g^{-1}F_s)_s ds \\ &= -\left[g^{-1}F_s \mid_{s_2(t)} - g^{-1}F_s \mid_{s_1(t)} \right] + \left[g^{-1}F_s \mid_{s_1(t)+S} - \right. \\ &= -2(g^{-1}F_K K_s) \mid_{s_2(t)} + 2(g^{-1}F_K K_s) \mid_{s_1(t)} \end{aligned}$$

Computer Science