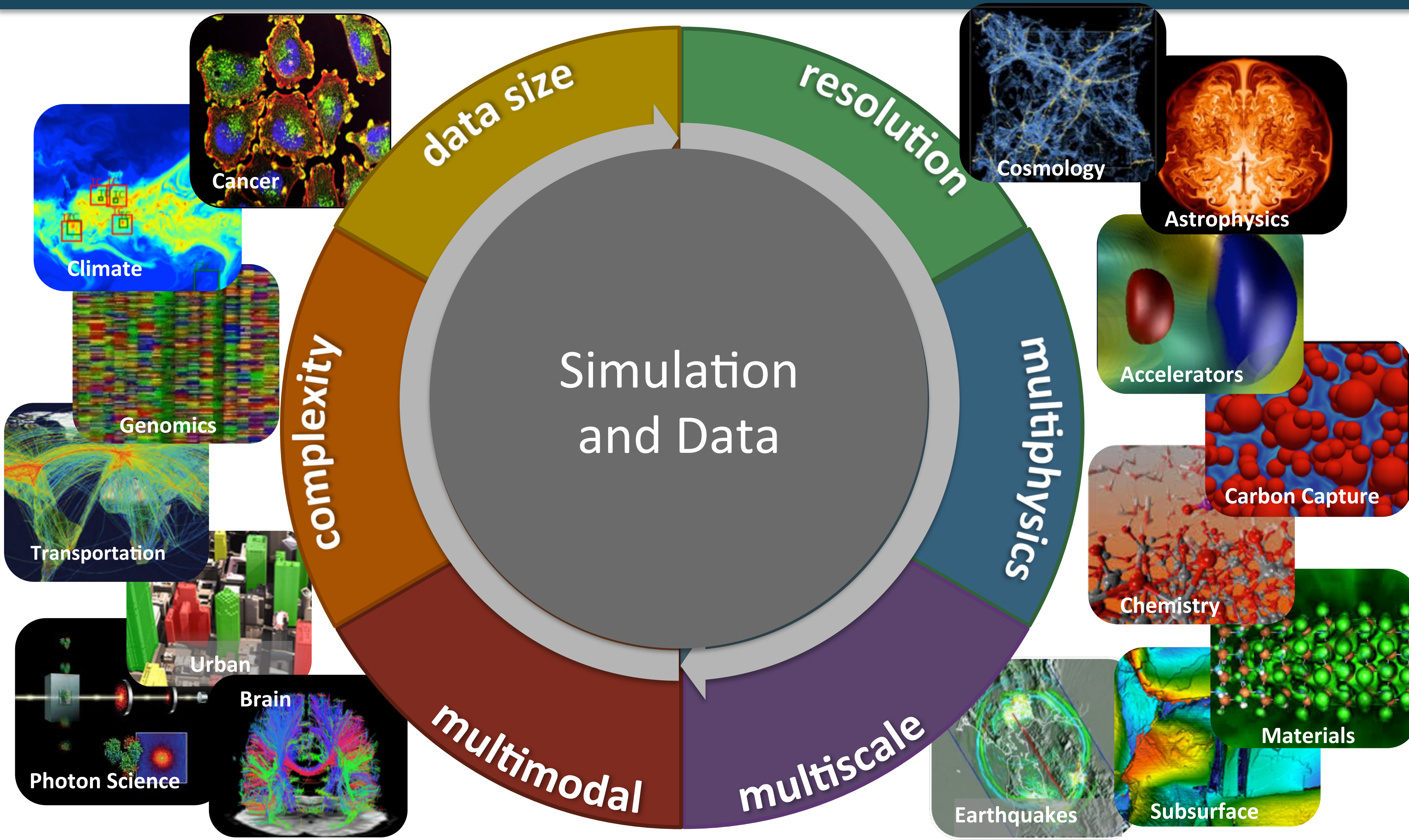
Antisocial Parallelism: Avoiding, Hiding and Managing Communication

Kathy Yelick
Professor of Electrical Engineering and Computer Sciences
U.C. Berkeley

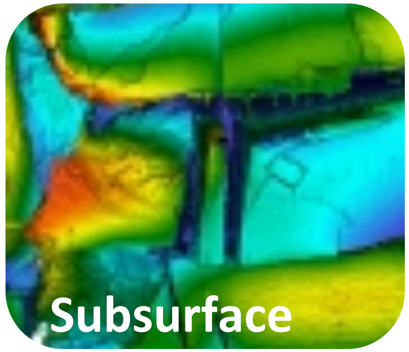
Associate Laboratory Director
Computing Sciences
Lawrence Berkeley National Laboratory

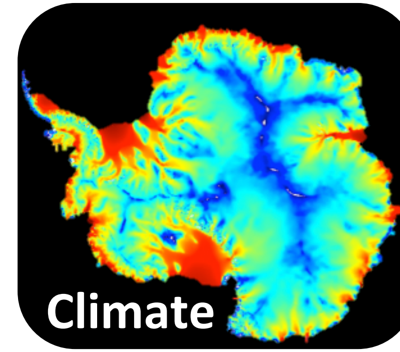
Exascale computing, combined with state-of-the-art mathematical models, algorithms, software techniques and data will enable breakthrough science

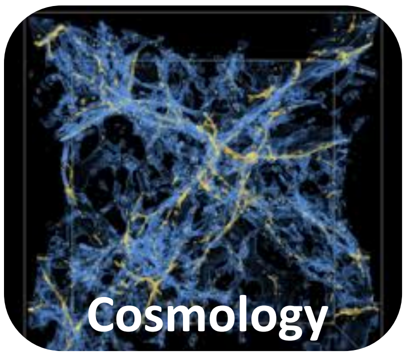
The Science Challenges at Exascale

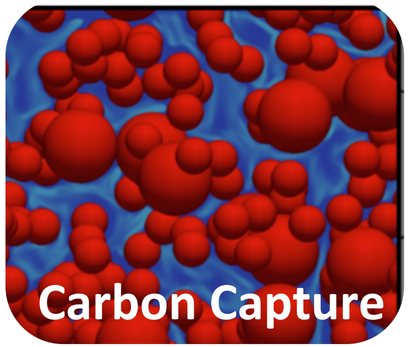


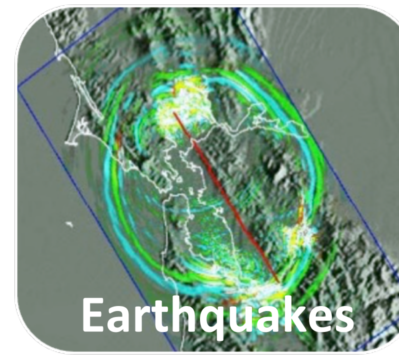
Berkeley Lab Priorities in Exascale Science

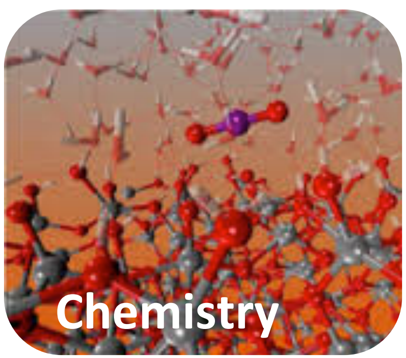


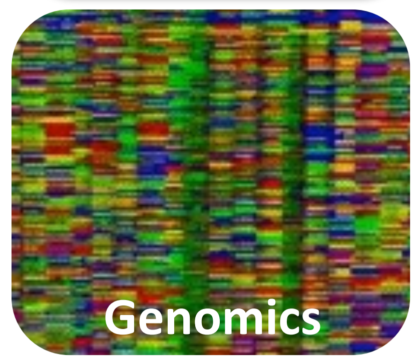


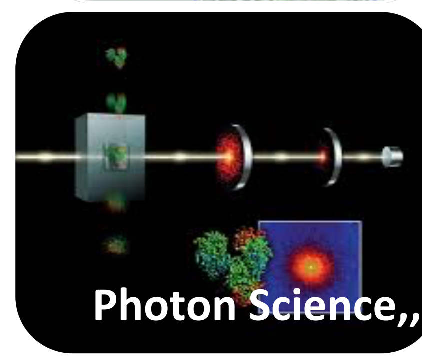




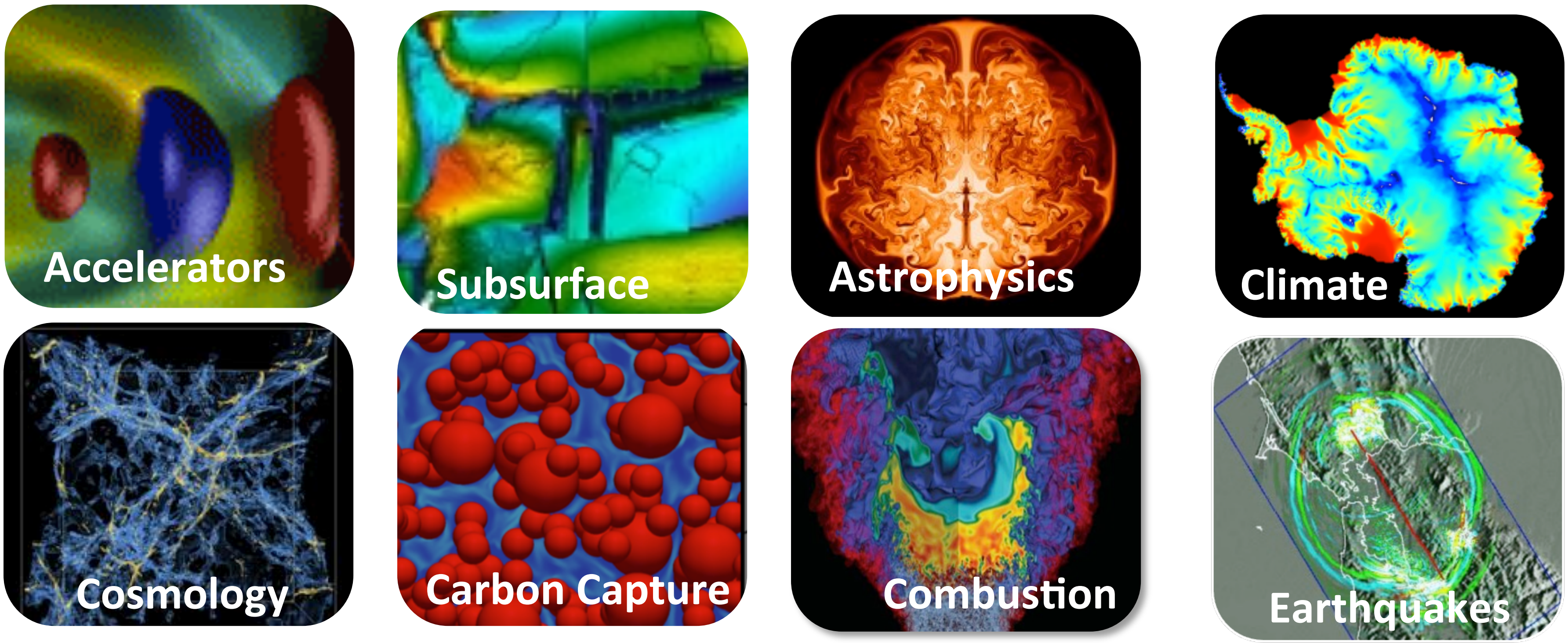








Berkeley Lab Priorities in Exascale Science



All the above will use Adaptive Mesh Refinement (AMR) mathematics and software, a method pioneered at Berkeley Lab

Computing challenges at the exascale

Computing is energy-constrained

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

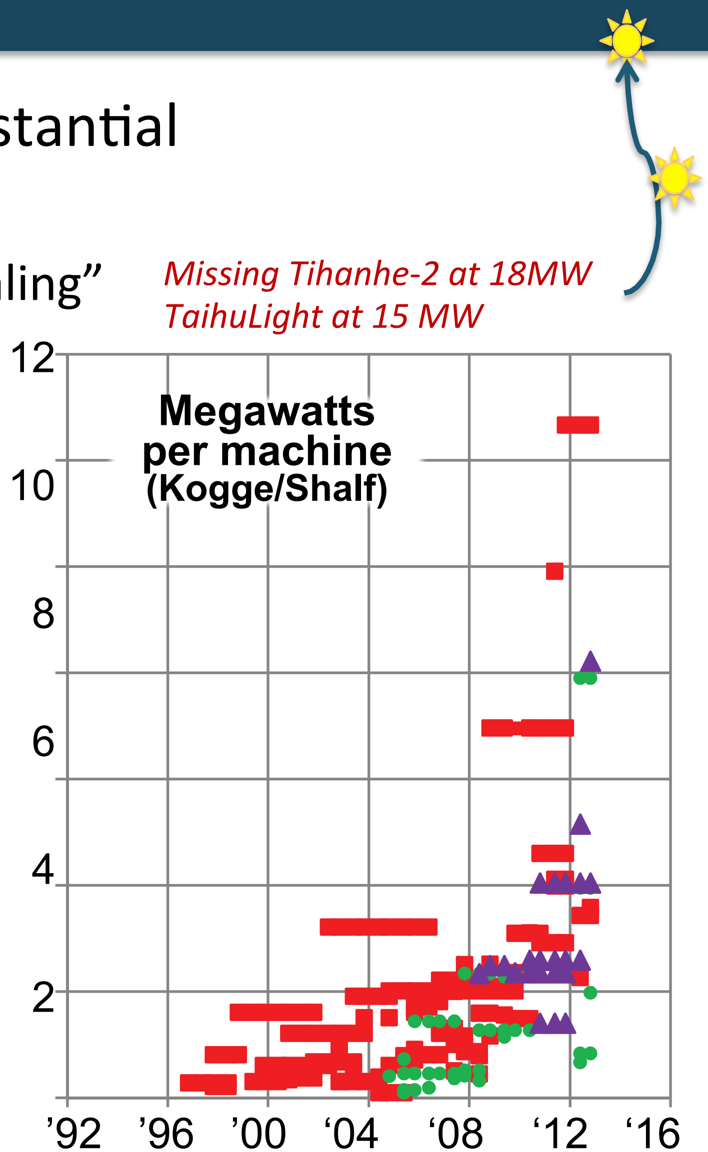
- 1 petaflop in 2008 used 3 MW
- 1 exaflop in 2018 at 200 MW “usual **chip** scaling”

Goal: 1 Exaflop in 20 MW
= 20 pJ / operation

Note: The 20 pJ / operation is

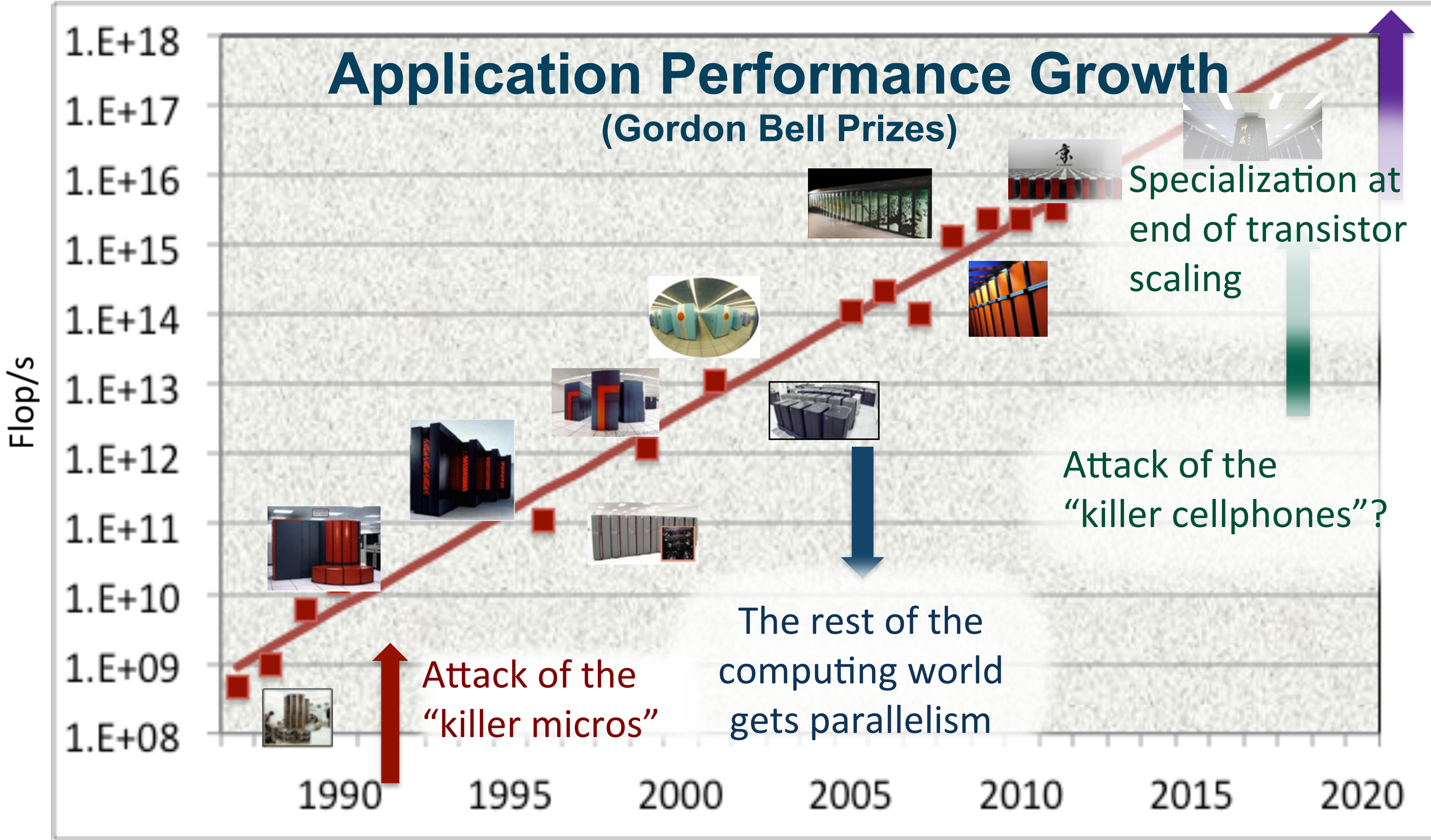
- **Independent of machine size**
- **Independent of # cores used per application**
- **But “operations” need to be useful ones**

Missing Tihanhe-2 at 18MW
TaihuLight at 15 MW



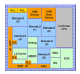
What Limits Computer Performance?

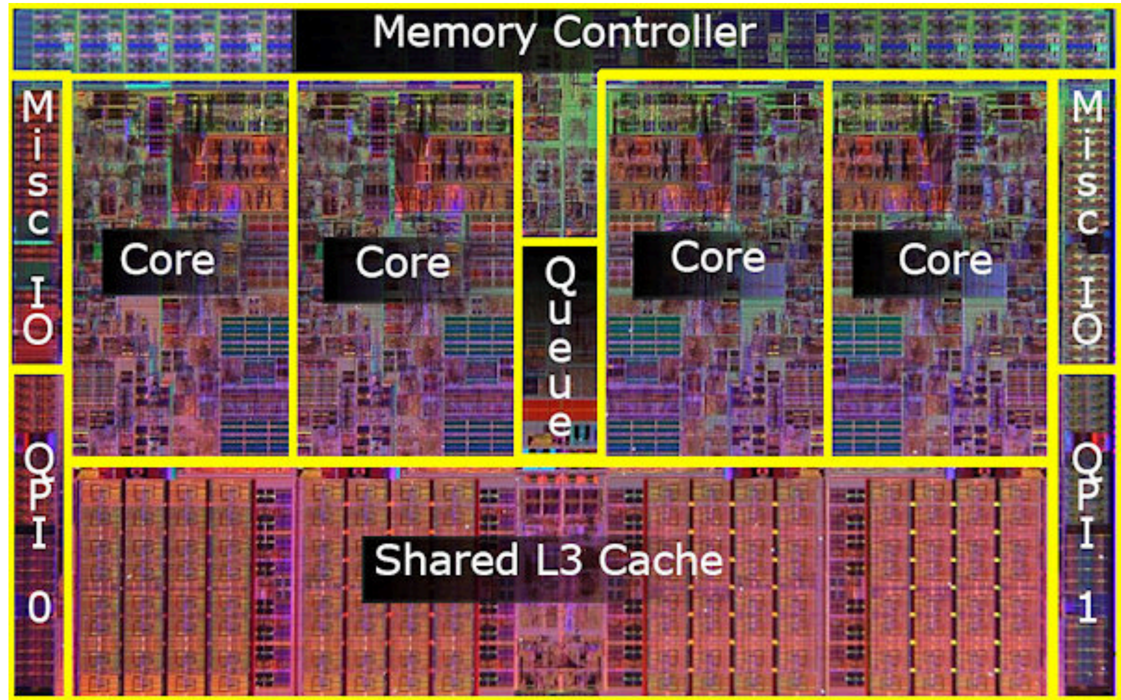
Computational Science has Moved through Difficult Technology Transitions



Lightweight Cores are the Future

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



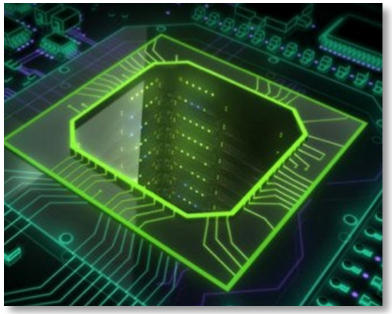


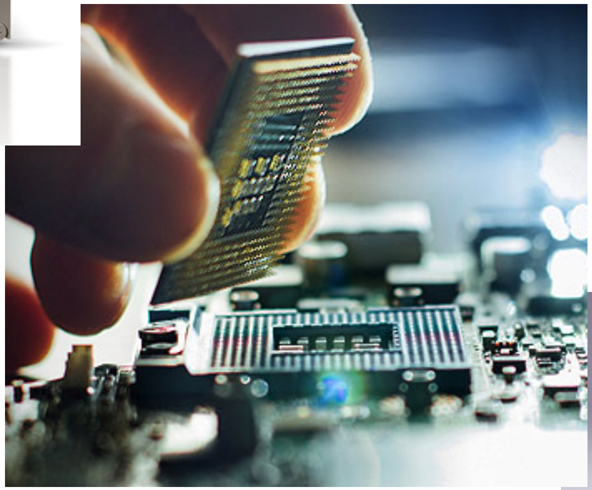
Server processor (100 Watts, 50 Gflop/s)

- **Small, simple cores are energy and area efficient**
  - 10-100x more energy efficient
- **Encourage "parallel thinking" in algorithms and software**

Specialization: End Game for Moore's Law

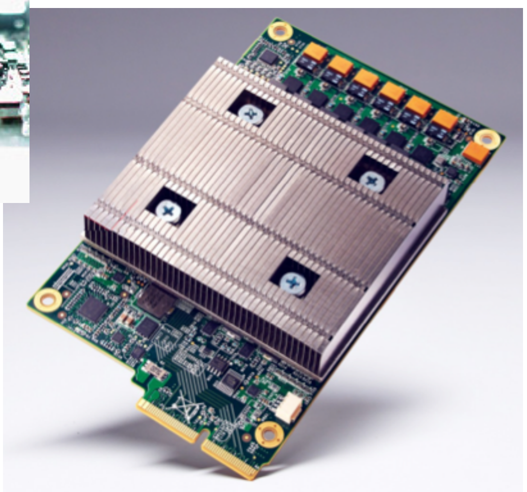
NVIDIA builds deep learning appliance with P100 Tesla's





Intel buys deep learning startup, Nervana

FPGAs

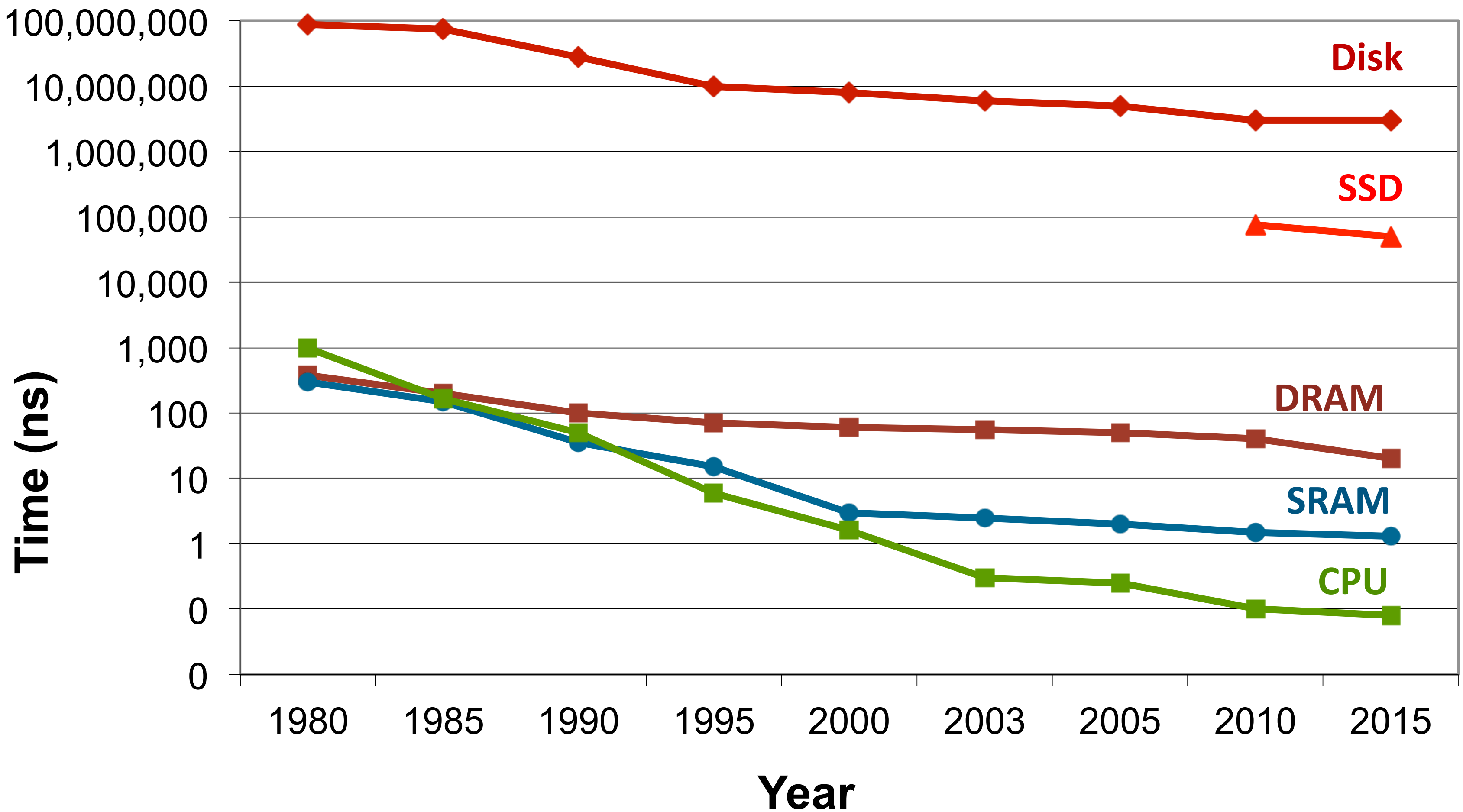


Google designs its own Tensor Processing Unit (TPU)

What's the most expensive operation on a computer?

Data Movement is Expensive

CPU cycle time vs memory access time



Source: http://csapp.cs.cmu.edu/2e/figures.html, http://csapp.cs.cmu.edu/3e/figures.html

Data Movement is Expensive

Hierarchical energy costs.

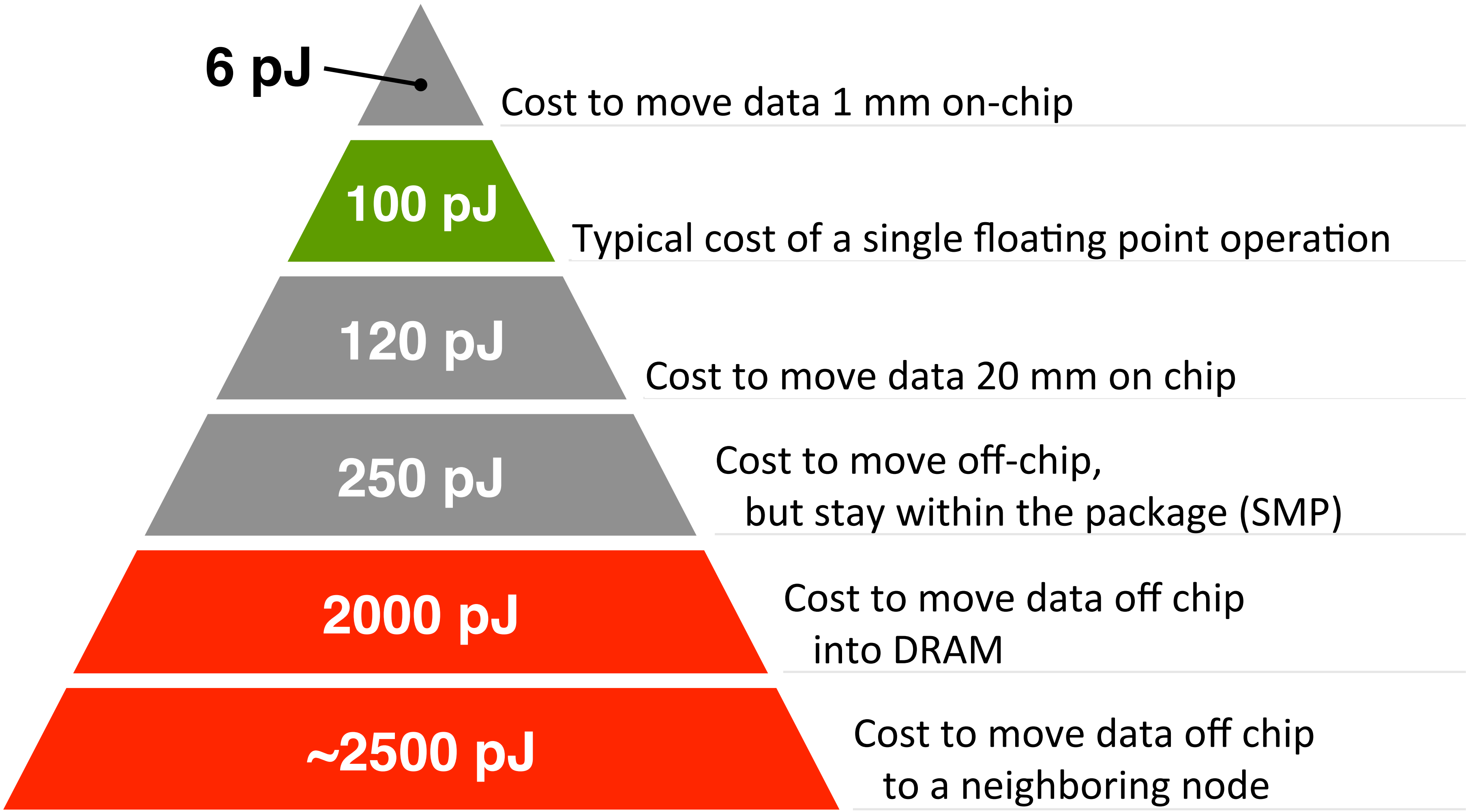
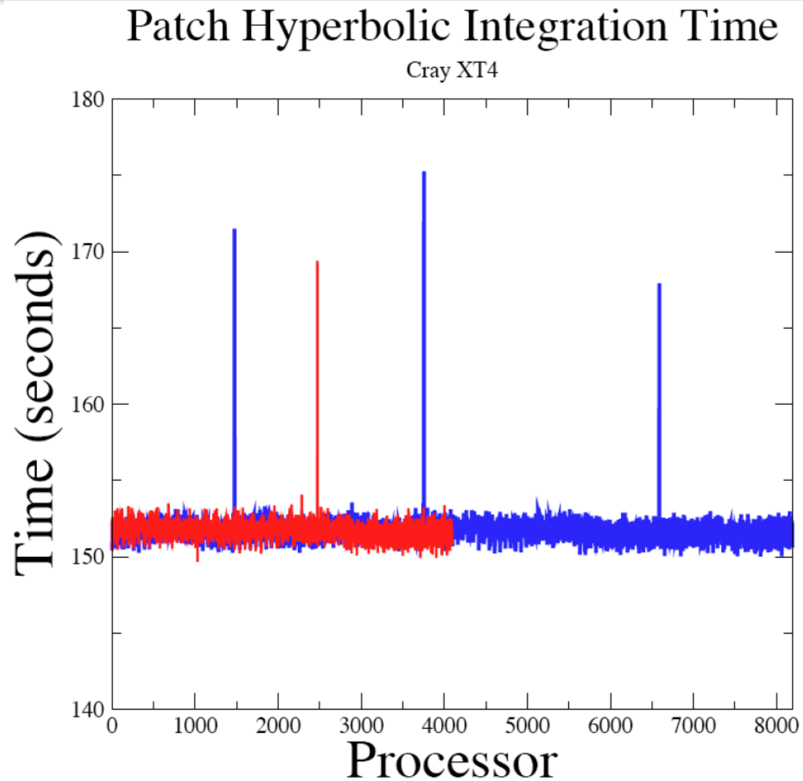


Image: http://slideplayer.com/slide/7541288/

Synchronization is Expensive

- **Machines will have Frequent Faults and "Performance Instability"**
- **Do all applications become "irregular"?**
- **Locality-Load balance trade-off**
  - Most work on dynamic scheduling is inside a shared memory node
  - Largest variability will be between nodes



Brian van Straalen, DOE Exascale Research Conference, April 16-18, 2012. *Impact of persistent ECC memory faults.*

Programming languages and compilers for exascale

The biggest concern for Exascale application developers is the need to write and maintain multiple versions of their software and the uncertainty over what the architectures will be.

Why develop new languages?

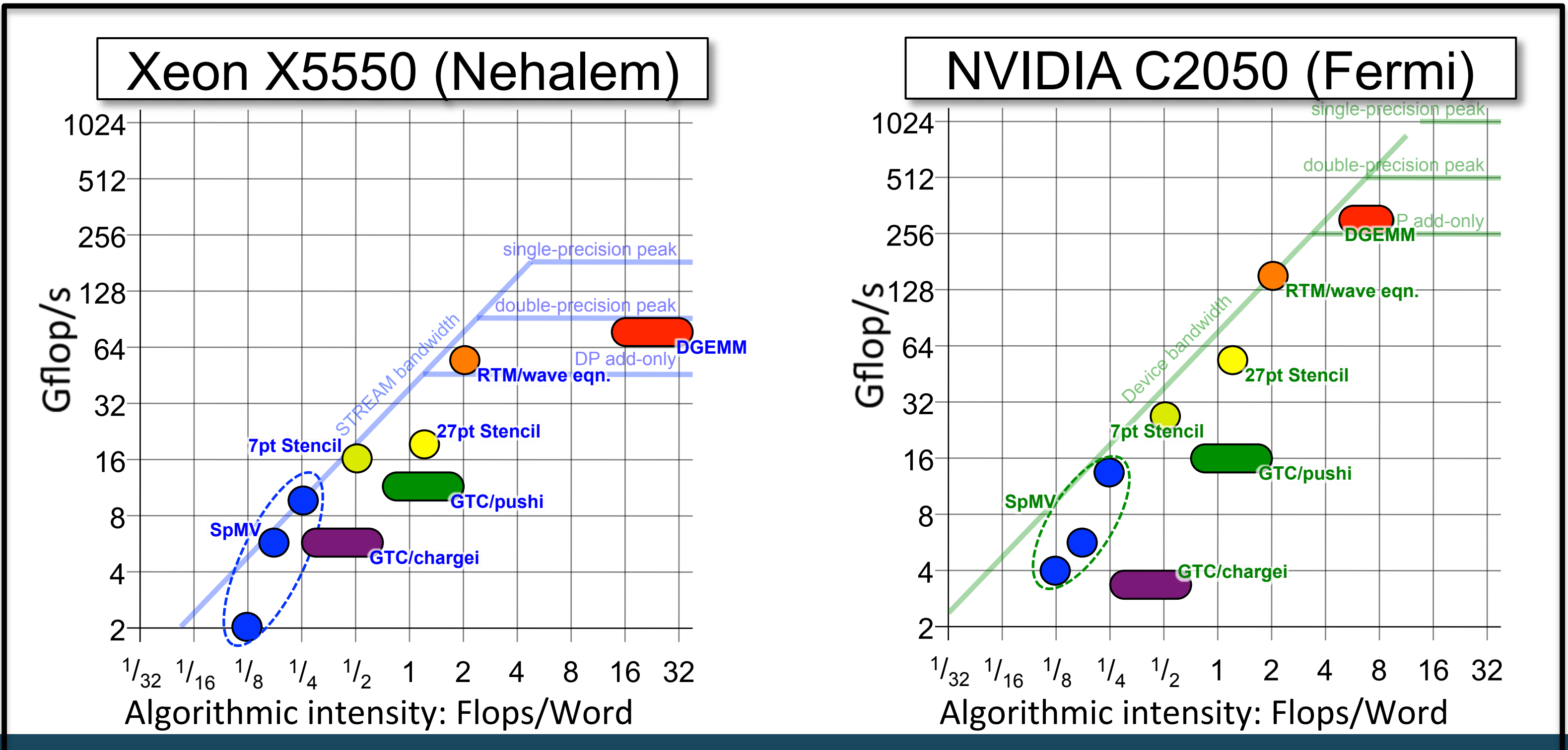
- **Productivity: higher level syntax**
  - We need a language
- **Correctness: static analysis can eliminate errors**
  - We need a compiler (front-end)
- **Performance: optimizations**
  - We need a compiler (back-end)

Language design enforces clarity in concepts

- **But you need to "know your audience"**
  - Need to rewrite installed base of code (anti-productivity)
  - Risk of compiler disappearing (maintainability)
  - Syntax matters (familiarity)
- **Language adoption is often about its libraries**

Programming for diverse (specialized) architectures

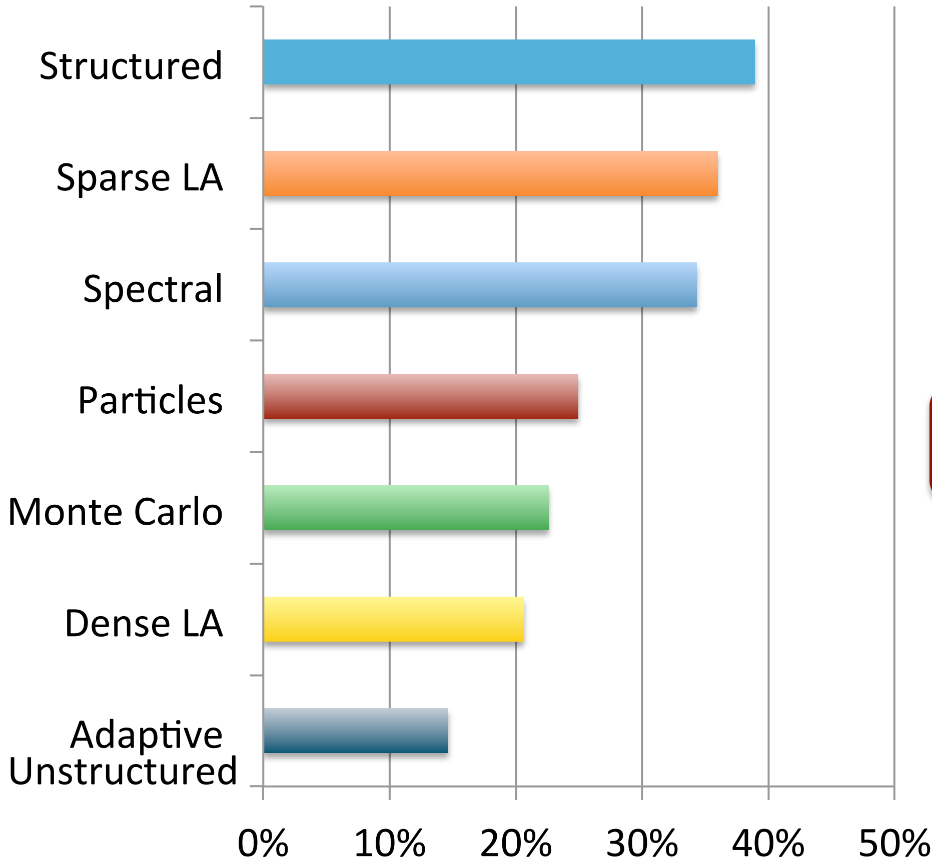
- **Two "hard" compiler problems:**
  - dependence analysis and — Domain-Specific Languages help with this
  - accurate performance models — Autotuning avoids this problem
- **Autotuners are code generators plus search**



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

Libraries vs. DSLs (domain-specific languages)

NERSC survey: what motifs do they use?



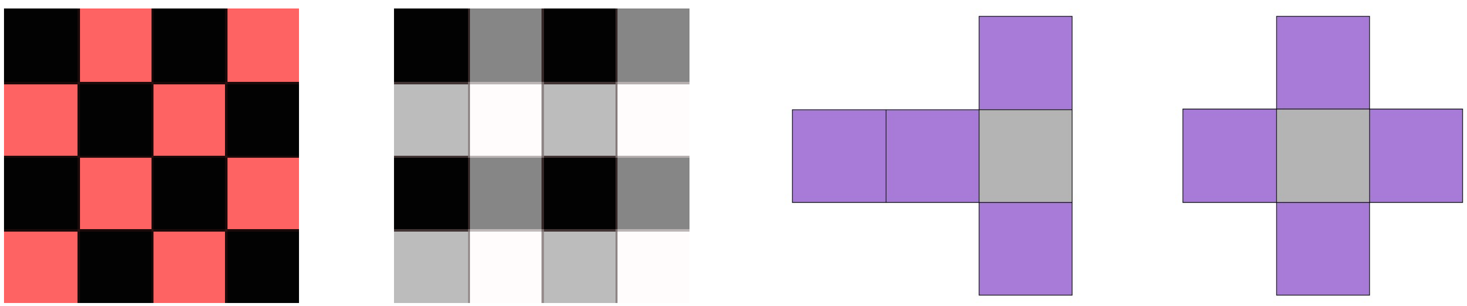
What code generators do we have?

Dense Linear Algebra	**Atlas**
Spectral Algorithms	FFTW, Spiral
Sparse Linear Algebra	OSKI
Structured Grids	TBD
Unstructured Grids	
Particle Methods	
Monte Carlo	

Stencils are both the most important motifs and a gap in our tools

Approach: Small Compiler for Small Language

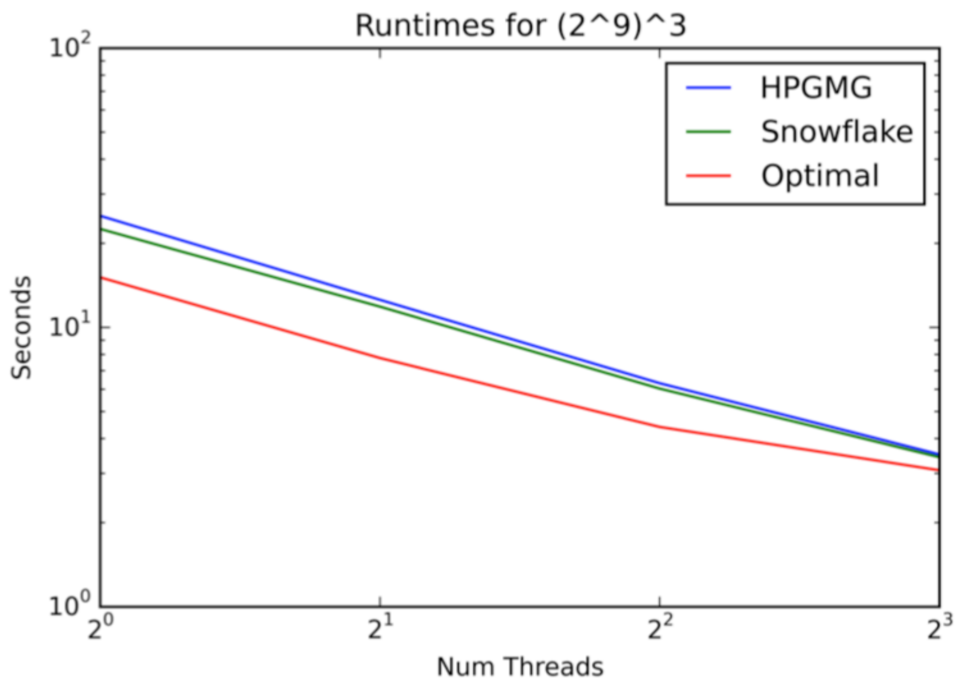
- **Snowflake: A DSL for Science Stencils**
  - Domain calculus inspired by Titanium, UPC++, and AMR in general

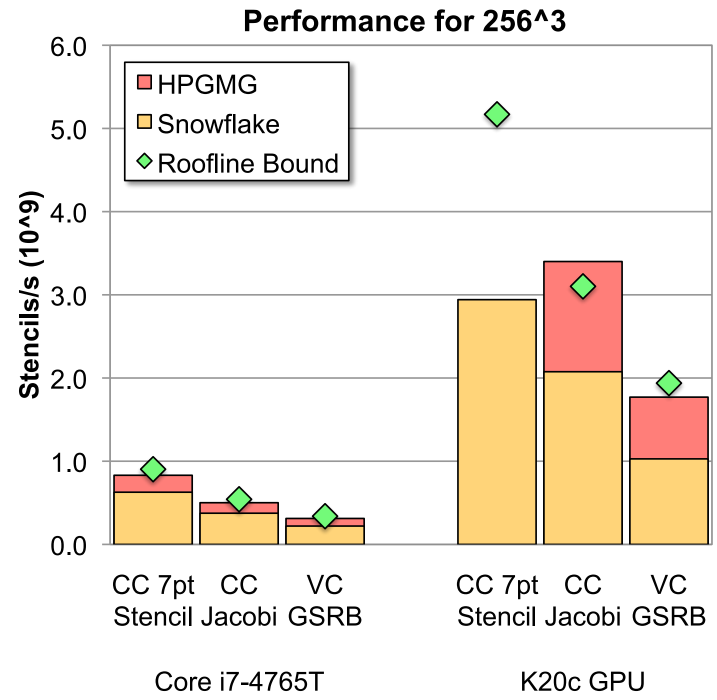


(a) Red-Black tiling (b) 4-color tiling (c) Asymmetric stencil used near mesh boundary (d) 5-point Jacobi stencil

- **Complex stencils: red/black, asymmetric**
- **Update-in-place while preserving provable parallelism**
- **Complex boundary conditions: key to Adaptive Meshes**

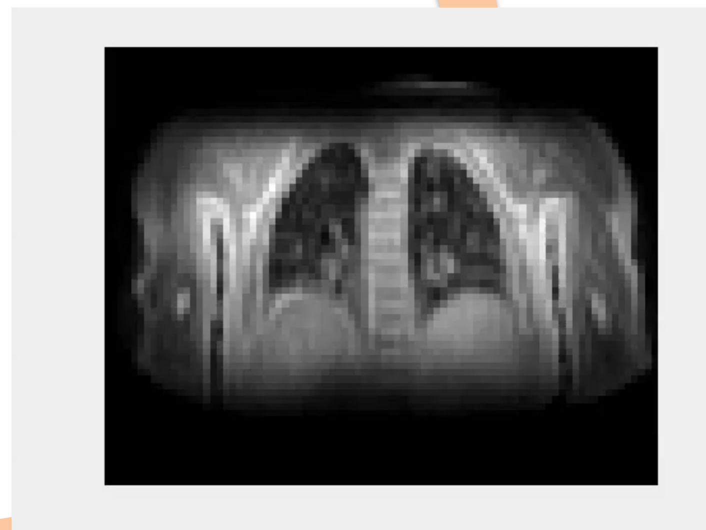
Snowflake performance





- **Performance on the HPGMG application benchmark using all the features of Snowflake**
- **Competitive with hand-optimized performance**
- **Within 2x of optimal roofline**

Real-Time MRI Challenge



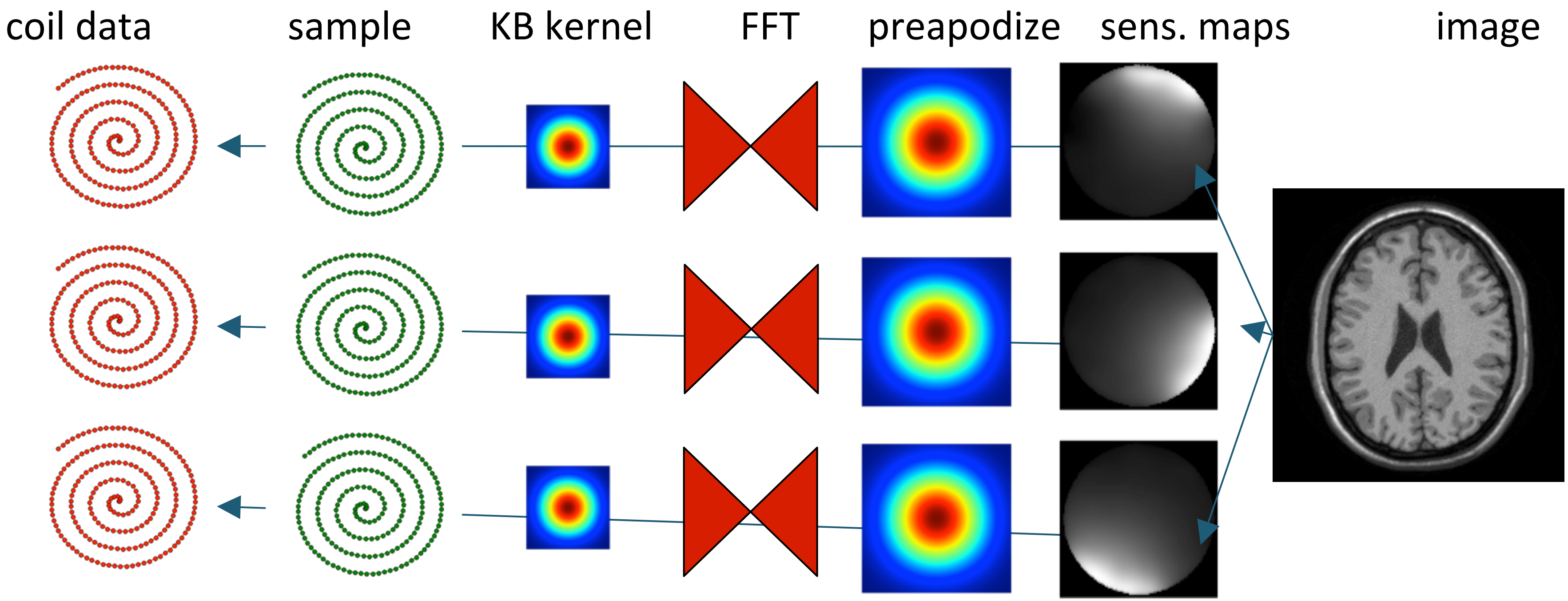
Time (min)	Architecture
6.15	KNL
5.42	Ivy Bridge
4.47	Broadwell
4.31	Kepler
4.12	Haswell
3.71	Broadwell
3.16	Kepler
0.94	Pascal

3 min goal

Michael Driscoll HPC optimization

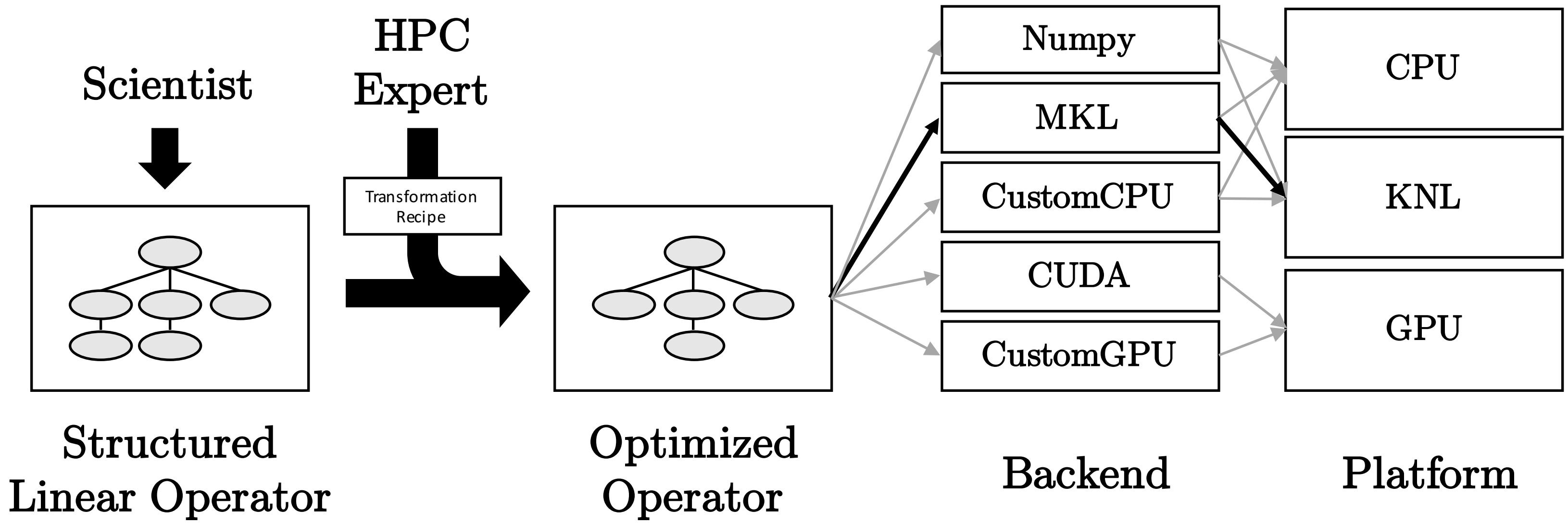
Compressed Sensing Approach by Mike Lustig et al
MRI results Wenwen Jiang

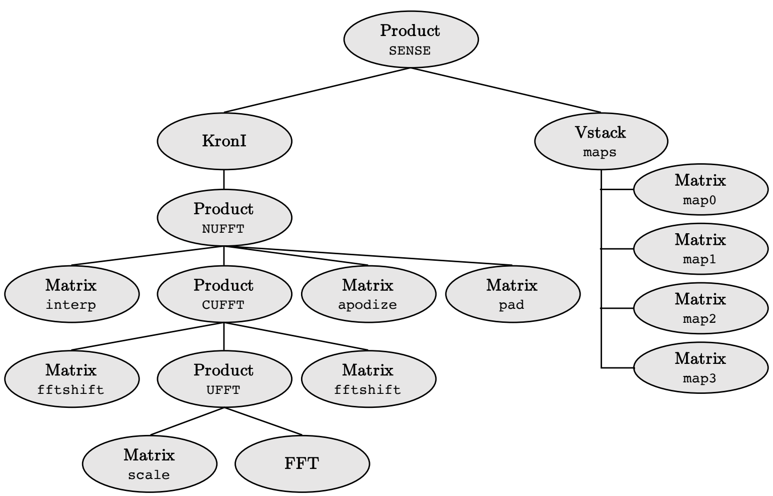
Matrix-free (loop optimization) vs. Matrix-full

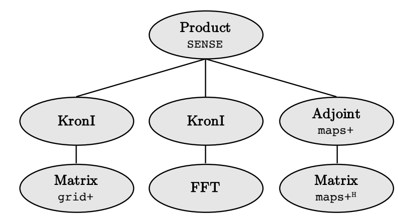


Loops	Structured matrices	Matrices
Operators as loop nests	Operators as matrices with structure that compiler can optimize	Operators as arbitrary sparse matrices

Domain-specific library with runtime optimizations

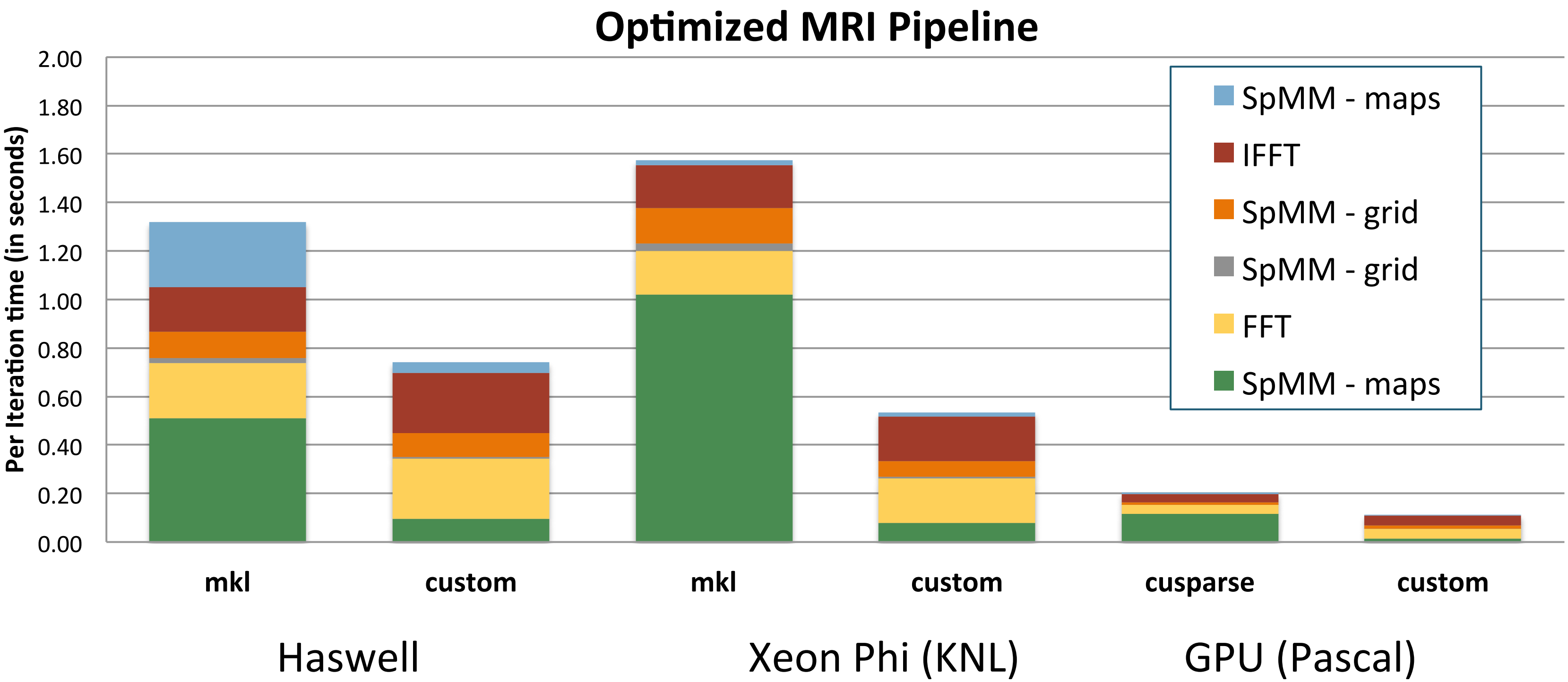






Tree transformation with matrix pattern knowledge

Python-Based Domain-Specific Language (EDSL)



- **Original Numpy code on Haswell: 87 sec/iteration**
- **Runtime optimization reorganize tree of operators (matrices + FFTs) cognizant of matrix structure**
- **Library or custom matrix kernels**

Compiler challenges

Auto-vectorization for vector processors

Auto-SIMDization

Auto-parallelization for SMPs

Auto-parallelism for attached accelerators

Auto-parallelization for HPC

Architecture difficulty (related to granularity of parallelism)

Autotuner code generation

Domain specific (in degrees of specificity)

General purpose strongly typed

General purpose loosely typed

Language difficulty

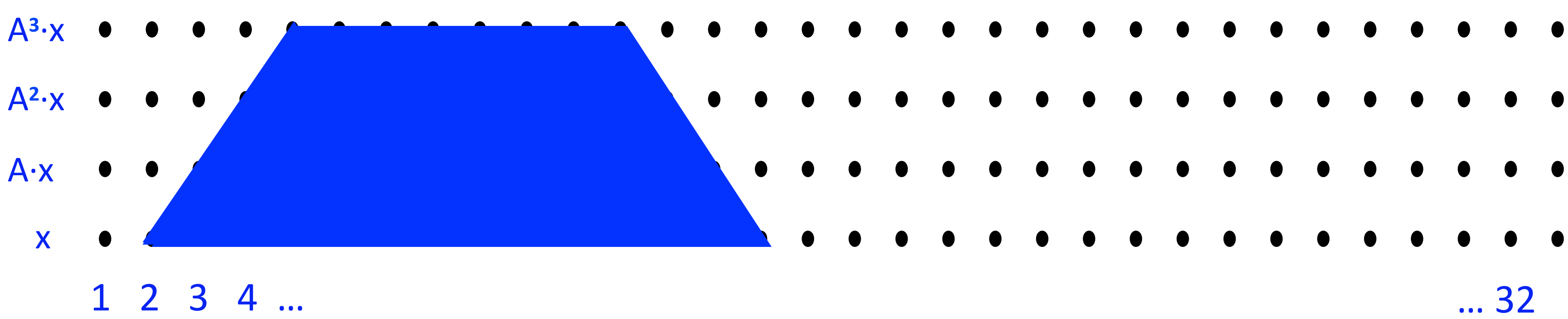
Avoiding Communication in Iterative Solvers

- **Consider Sparse Iterative Methods for *Ax*=*b***
  - Krylov Subspace Methods: GMRES, CG,…
- **Solve time dominated by:**
  - Sparse matrix-vector multiple (SPMV)
    - Which even on one processor is dominated by "communication" time to read the matrix
  - Global collectives (reductions)
    - Global latency-limited
- **Can we lower the communication costs?**
  - Latency: reduce # messages by computing multiple reductions at once
  - Bandwidth to memory, i.e., compute Ax, A^2x, … A^kx with one read of A

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$

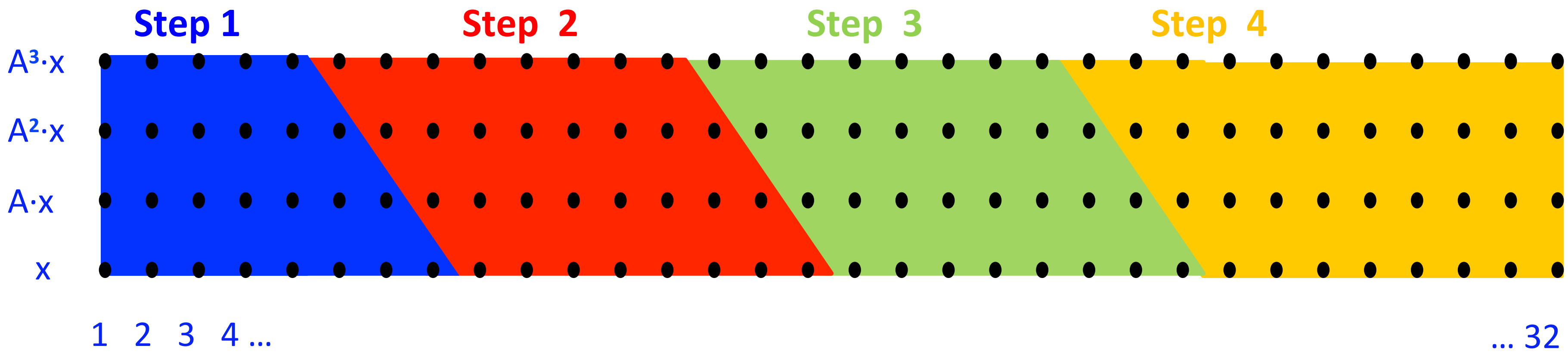


- Idea: pick up part of A and x that fit in fast memory, compute each of k products
- Example: A tridiagonal matrix (a 1D "grid"), n=32, k=3
- General idea works for any "well-partitioned" A

Communication Avoiding Kernels (Sequential case)

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- **Sequential Algorithm**

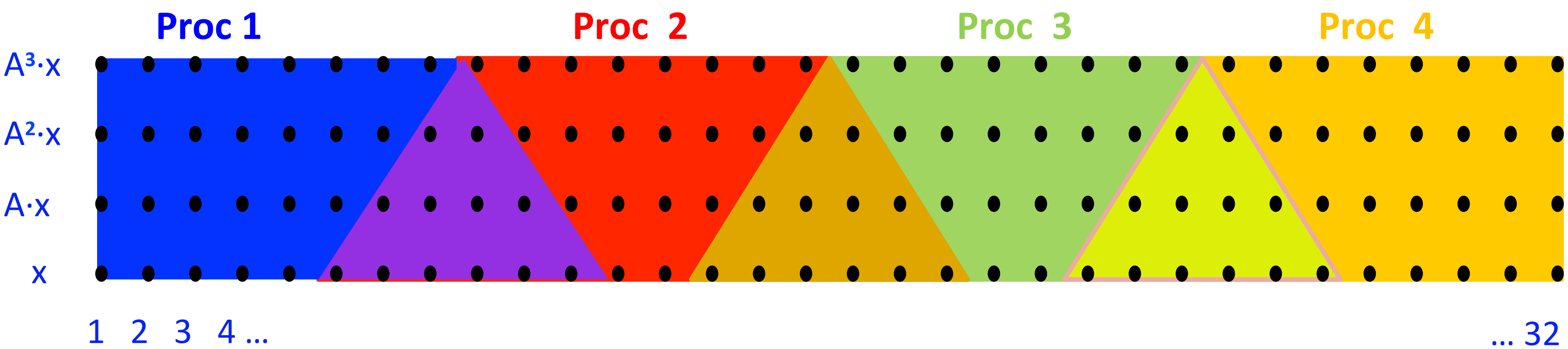


- Example: A tridiagonal, n=32, k=3
- Saves bandwidth (one read of A&x for k steps)
- Saves latency (number of independent read events)

Communication Avoiding Kernels: (Parallel case)

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

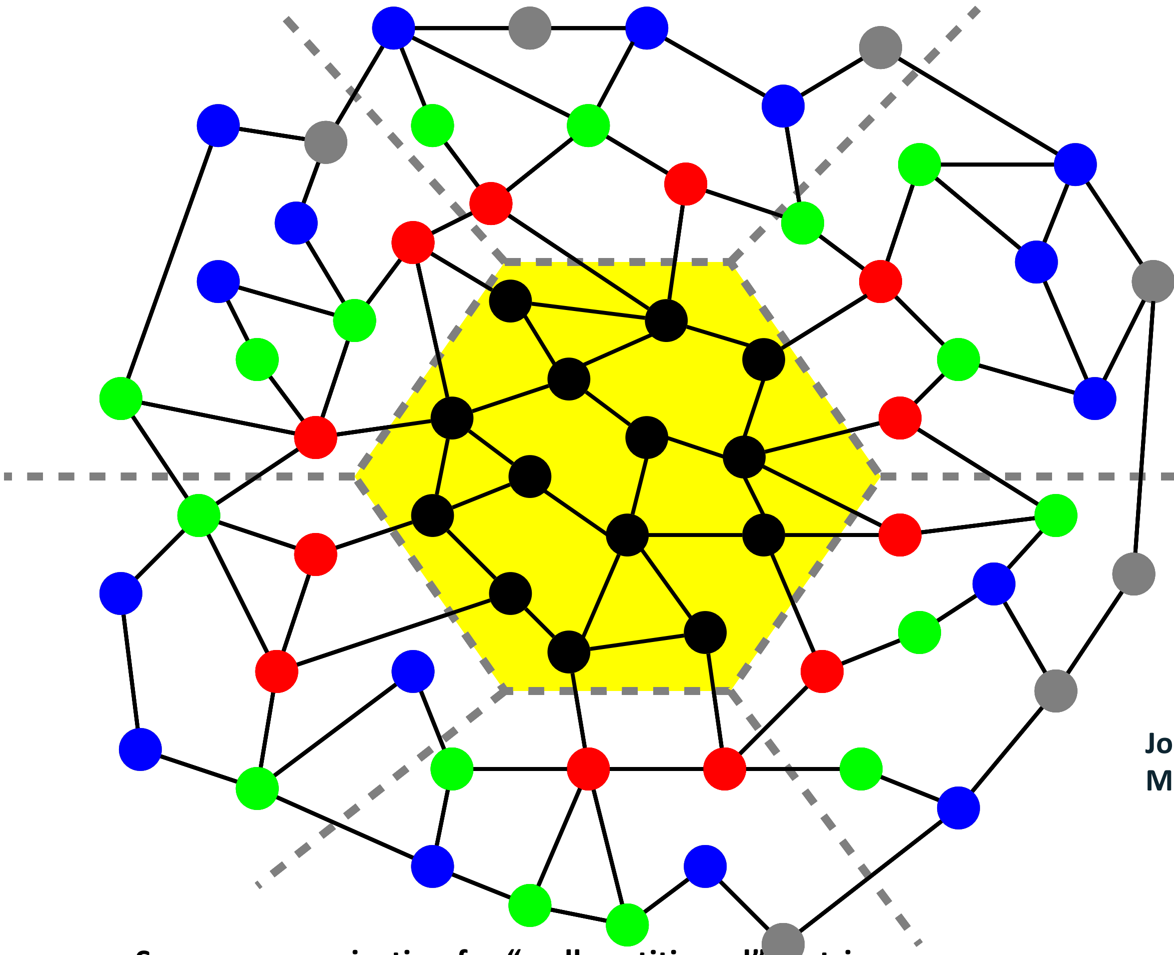
- Replace k iterations of $y = A\cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- **Parallel Algorithm**



- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid
- Saves latency (# of messages); Not bandwidth

But adds redundant computation

Matrix Powers Kernel on a General Matrix

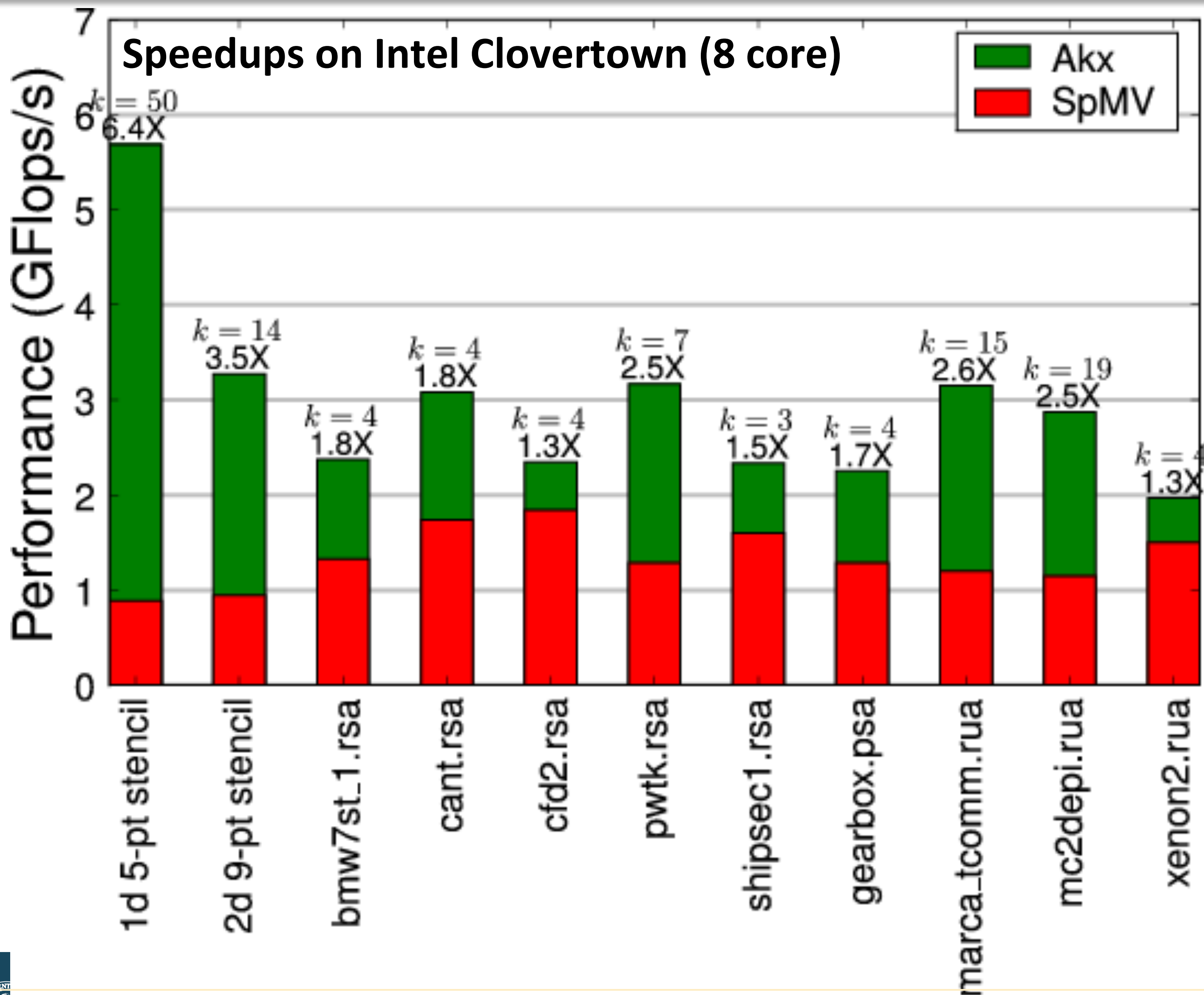


For implicit memory management (caches) uses a TSP algorithm for layout

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

- **Saves communication for "well partitioned" matrices**
  - Serial: O(1) moves of data moves vs. O(k)
  - Parallel: O(log p) messages vs. O(k log p)

$A^k x$ has **higher performance than Ax**



Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick

Minimizing Communication of GMRES to solve Ax=b

- **GMRES: find x in span{b,Ab,…,A^kb} minimizing || Ax-b $||_2$**

Standard GMRES	Communication-avoiding GMRES

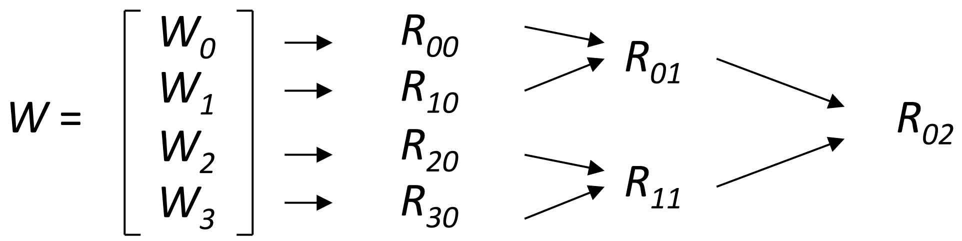
for i=1 to k
  w = A · v(i-1) … *SpMV*
  MGS(w, v(0),…,v(i-1))
  update v(i), H
endfor
solve LSQ problem with H
	W = [v, Av, A^2v, … , A^kv]
	[Q,R] = TSQR(W)
	  … *"Tall Skinny QR"*
	build H from R
	solve LSQ problem with H

Sequential case: #words moved decreases by a factor of k
Parallel case: #messages decreases by a factor of k

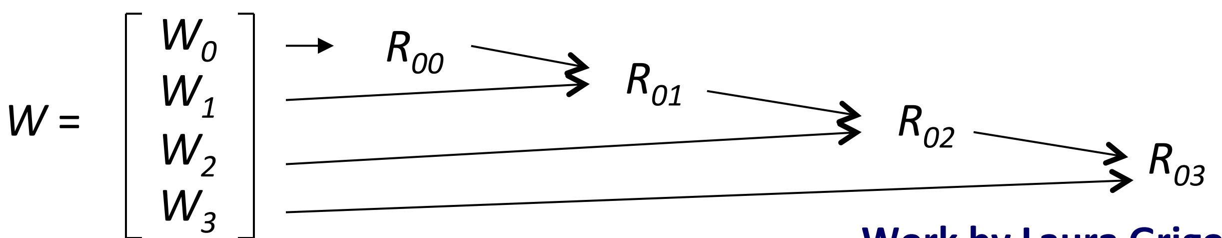
- Oops – W from power method, precision lost!

TSQR: An Architecture-Dependent Algorithm

Parallel:

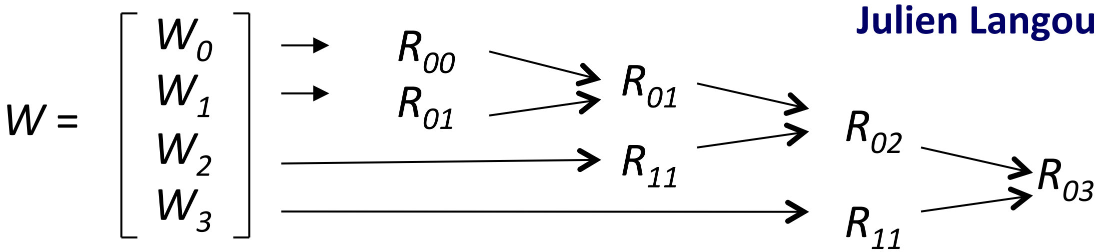


Sequential:



Work by Laura Grigori, Jim Demmel, Mark Hoemmen, Julien Langou

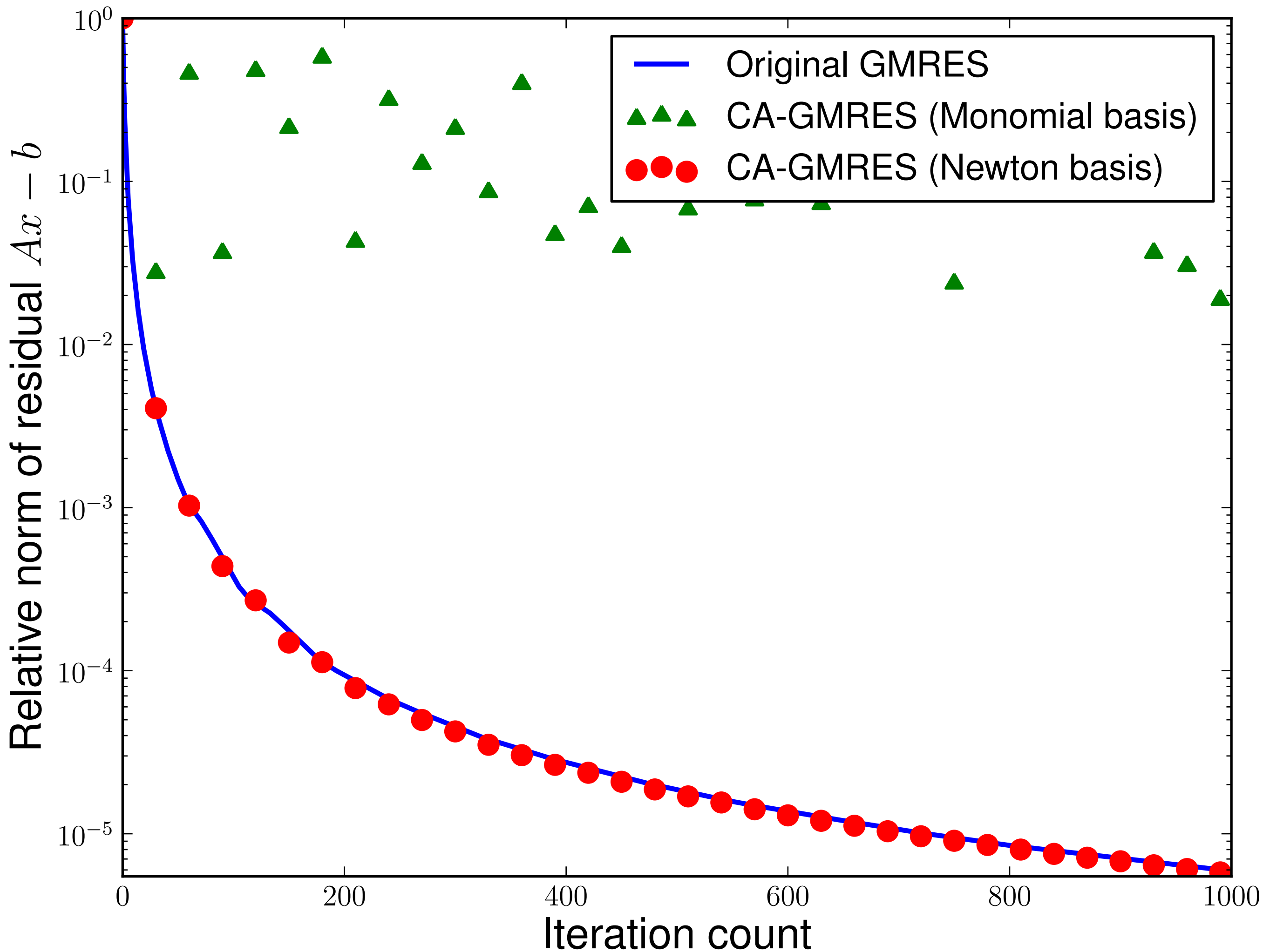
Dual Core:



Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can choose reduction tree dynamically

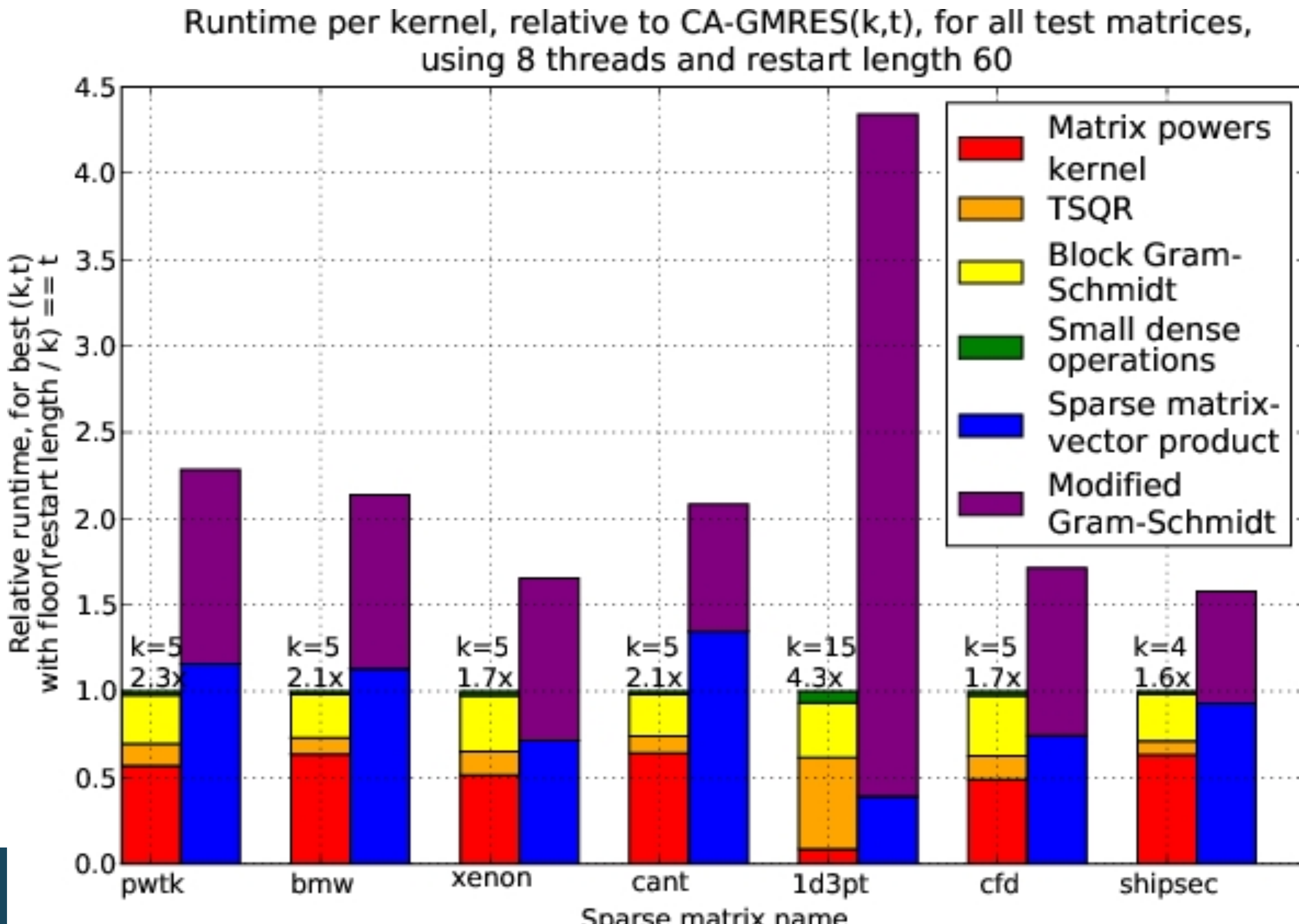
Matrix Powers Kernel (and TSQR) in GMRES



Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick

Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown



DSLs popular outside scientific computing

Developed for Image Processing

- 10+ FTEs developing Halide
- 50+ FTEs use it; > 20 kLOC

HPGMG (Multigrid on Halide)

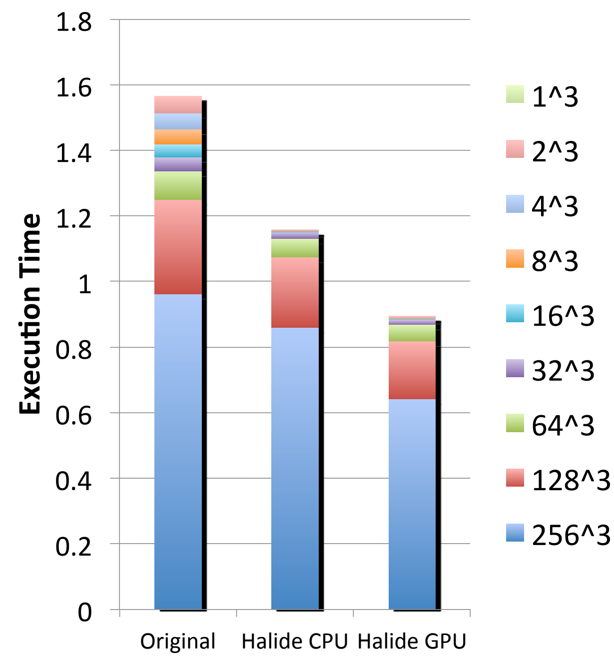
- **Halide Algorithm by domain expert**

```
Func Ax_n("Ax_n"), lambda("lambda"), chebyshev("chebyshev");
Var i("i"),j("j"),k("k");
Ax_n(i,j,k) =  a*alpha(i,j,k)*x_n(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k)*(x_n(i,j,k) + x_n(i-1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j,k)  *(valid(i,j-1,k)*(x_n(i,j,k) + x_n(i,j-1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k)  *(valid(i,j,k-1)*(x_n(i,j,k) + x_n(i,j,k-1)) - 2.0f*x_n(i,j,k))
  + beta_i(i+1,j,k)*(valid(i+1,j,k)*(x_n(i,j,k) + x_n(i+1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j+1,k)*(valid(i,j+1,k)*(x_n(i,j,k) + x_n(i,j+1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k+1)*(valid(i,j,k+1)*(x_n(i,j,k) + x_n(i,j,k+1)) - 2.0f*x_n(i,j,k)));
lambda(i,j,k) = 1.0f / (a*alpha(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k) - 2.0f)
  + beta_j(i,j,k)  *(valid(i,j-1,k) - 2.0f)
  + beta_k(i,j,k)  *(valid(i,j,k-1) - 2.0f)
  + beta_i(i+1,j,k)*(valid(i+1,j,k) - 2.0f)
  + beta_j(i,j+1,k)*(valid(i,j+1,k) - 2.0f)
  + beta_k(i,j,k+1)*(valid(i,j,k+1) - 2.0f)));
chebyshev(i,j,k) = x_n(i,j,k) + c1*(x_n(i,j,k)-x_nm1(i,j,k))+
                   c2*lambda(i,j,k)*(rhs(i,j,k)-Ax_n(i,j,k));
```

- **Halide Schedule either**
  - Auto-generated by autotuning with opentuner
  - Or hand created by an optimization expert

Halide performance

- Autogenerated schedule for CPU
- Hand created schedule for GPU
- No change to the algorithm



Open Problem: Compiling for communication optimality…

… with irregular loop nests and sparsity

Beyond Domain Decomposition

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

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

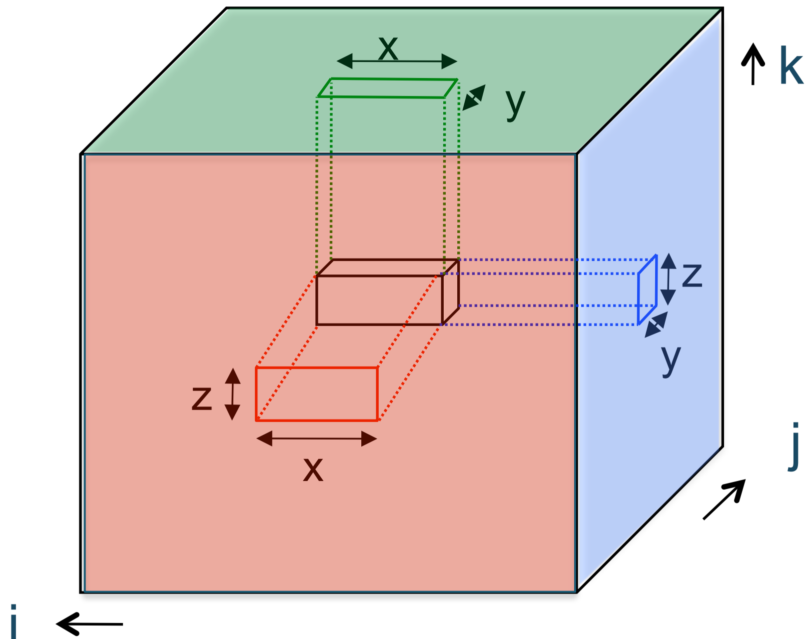
And don't get hung up on the owner computes rule

Can we generalize for compiler writers?

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

Deconstructing 2.5D Matrix Multiply

Solomonick & Demmel



- **Tiling the iteration space**
- **2D algorithm: never chop k dim**
- **2.5 or 3D: Assume + is associative; chop k, which is → replication of C matrix**

Matrix Multiplication code has a 3D iteration space
Each point in the space is a constant computation (*/+)

```
for i
  for j
    for k
          C[i,j] … A[i,k] … B[k,j] …
```

Beyond Domain Decomposition

x += …

x += …

x += …

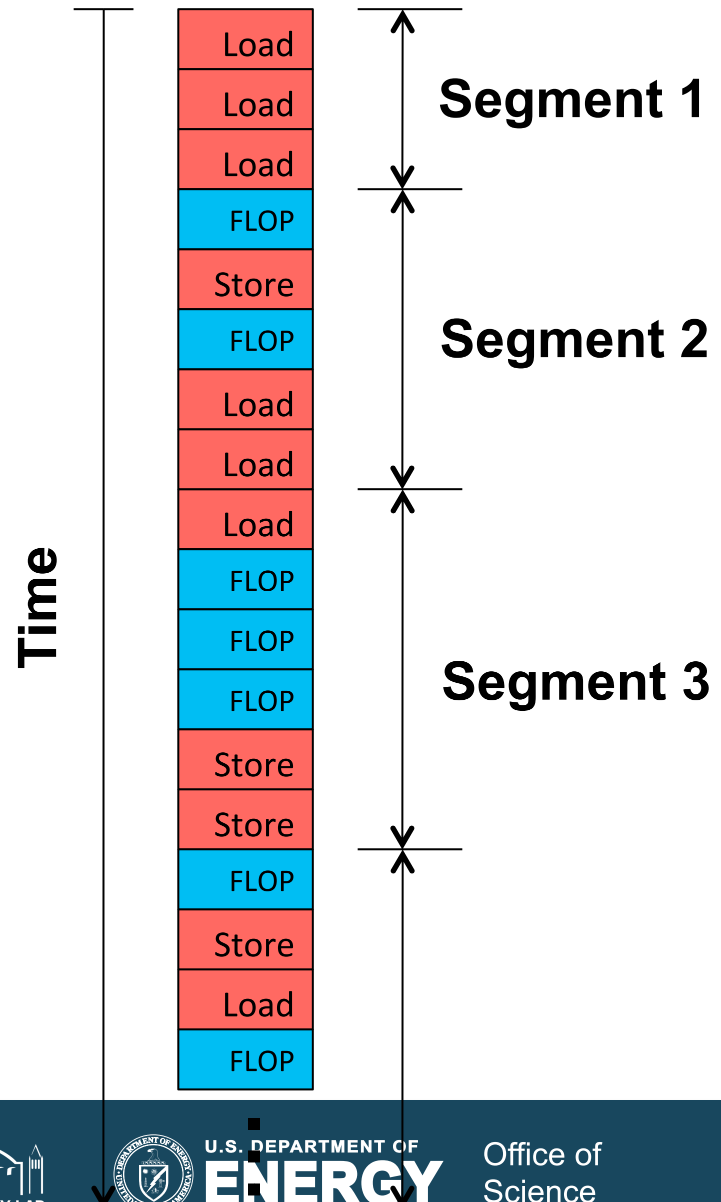
x += …

Using x for C[i,j] here

- **Much of the work on compilers is based on owner-computes (domain decomposition)**
  - For MM: Divide C into chunks, schedule movement of A/B
  - Data-driven domain decomposition partitions data; but we can partition work instead
- **Ways to compute C "pencil"**
  1. Serially
  2. Parallel reduction
  3. Parallel asynchronous (atomic) updates
  4. Or any hybrid of these

  Standard vectorization trick
- **For what types / operators does this work?**
  - "+" is associative for 1,2 rest of RHS is "simple"
  - and commutative for 3

Lower Bound: What is the minimum amount of communication required?



- **Matrix Multiply Proof from Irony/Toledo/Tiskin (2004)**
- **Assume fast memory of size M**
- **How much work (F) can we do with O(M) data?**

- **For matrix multiplication, this uses a result from Loomis an Whitney (1949)**

  # cubes in 3D set = Volume of 3D set

  ≤ (area(A shadow) * area(B shadow) * area(C shadow)) $^{1/2}$

Generalizing Communication Lower Bounds and Optimal Algorithms

- **For serial matmul, we know #words_moved = $\Omega(n^3/M^{1/2})$, attained by tile sizes $M^{1/2}$ x $M^{1/2}$**

- **Thm (Christ,Demmel,Knight,Scanlon,Yelick):** *For any program that "smells like" nested loops, accessing arrays with subscripts that are linear functions of the loop indices*

  #words_moved = Ω (#iterations/M^e)

  for some e we can determine

- **Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes**
  - E.g., index expressions are just subsets of indices
- **Long term goal: All compilers should generate communication optimal code from nested loops**

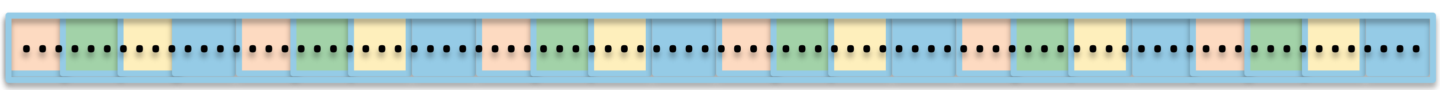
Using .5D ideas on N-body

- **n particles, k-way interaction.**
  - Molecules, stars in galaxies, etc.
- **Most common: 2-way N-body**

for t timesteps
    forall $i_1, \ldots, i_k$
        force[i_1] += interact(particle[i_1], …, particle[i_k])
    forall i
        move(particle[i], force[i])

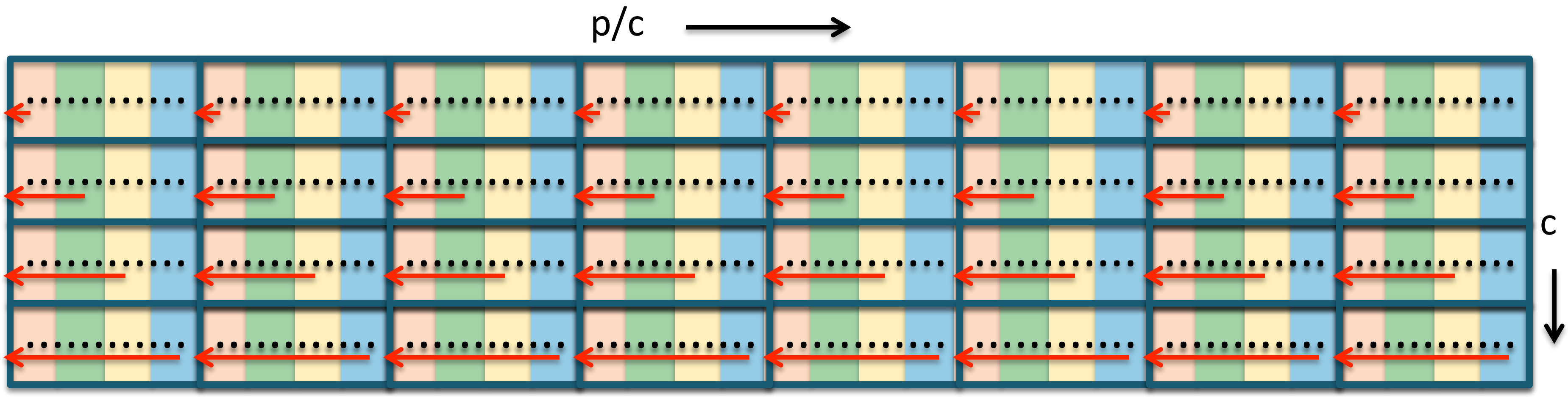
$O(n^k)$.

- **Best algorithm is to divide n particles into p groups??**



No!

Communication Avoiding 2-way N-body
(using a "1.5D" decomposition)



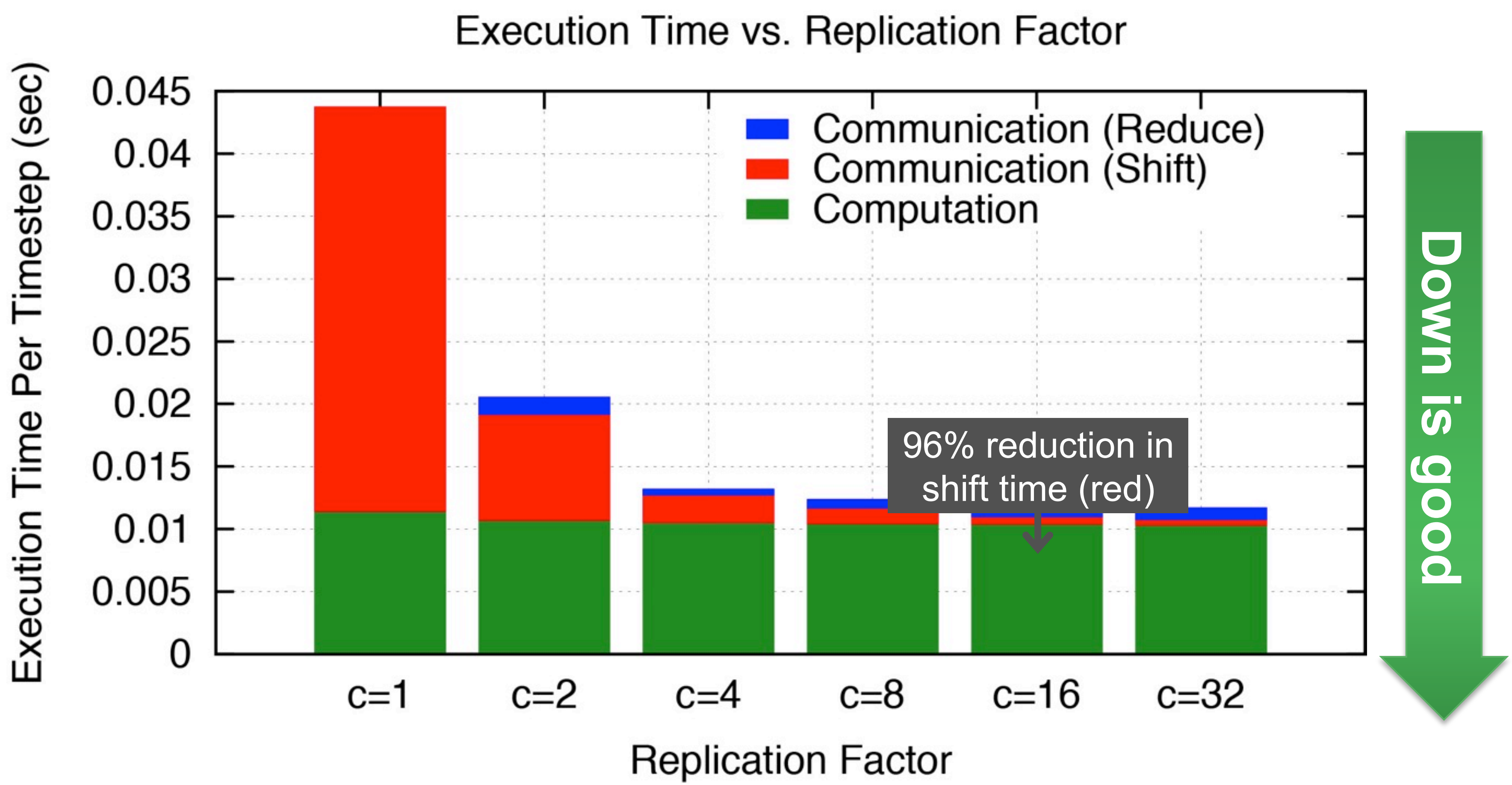
- **Divide p into c groups**
- **Replicate particles across groups**
- **Repeat: shift copy of n/(p*c) particles to the left within a group**
- **Reduce across c to produce final value for each particle**

Total Communication: O(log(p/c) + log c) messages,

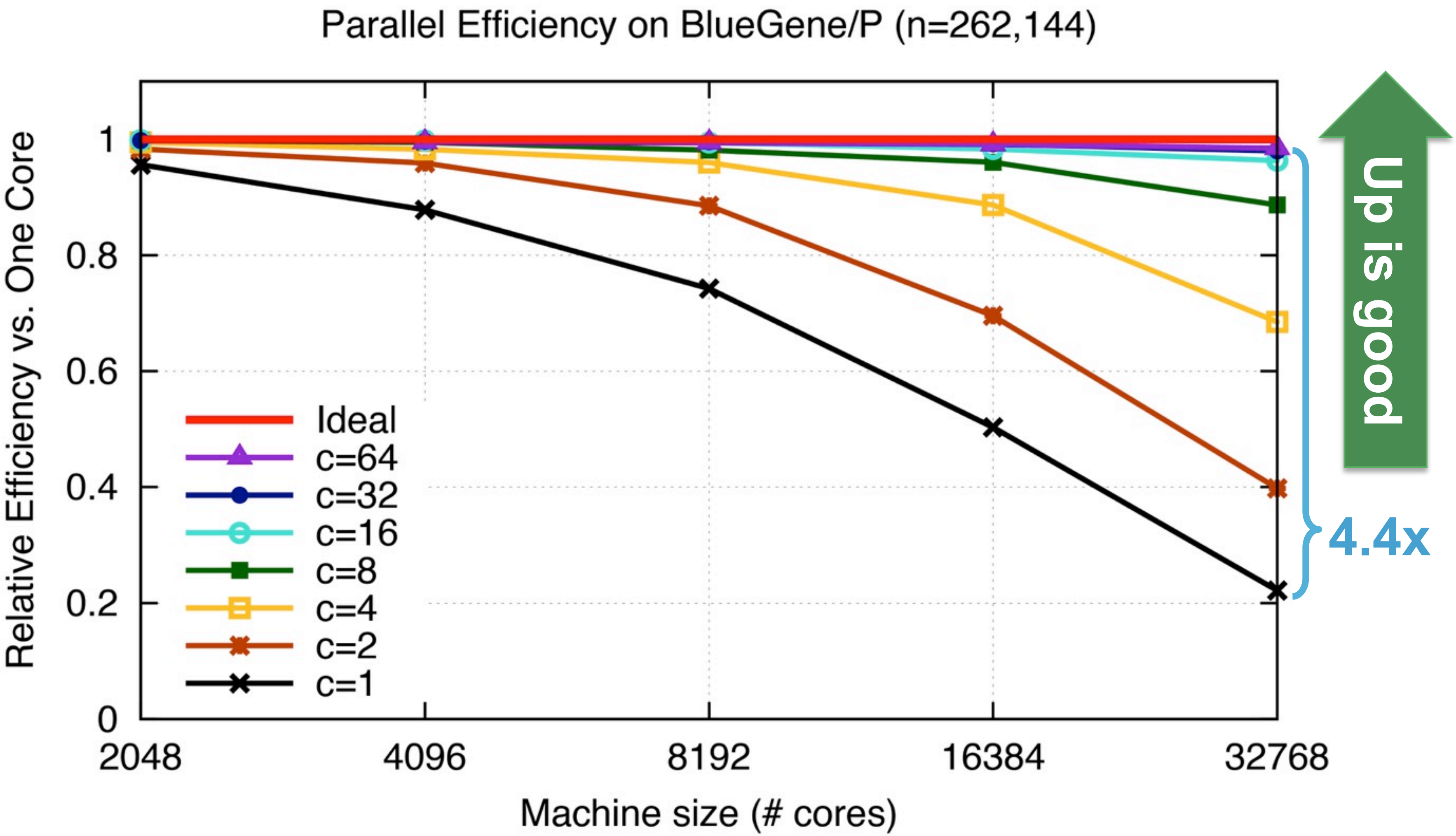
O(n*(c/p+1/c)) words

Less Communication..

Cray XE6; n=24K particles, p=6K cores

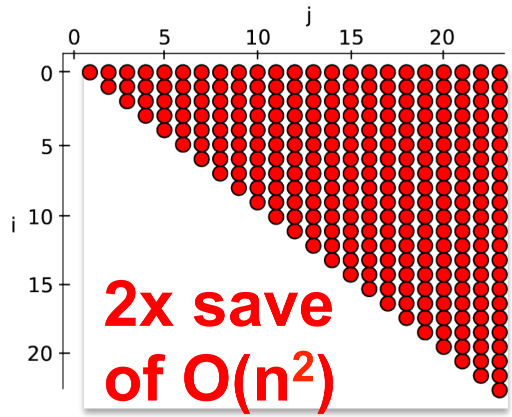


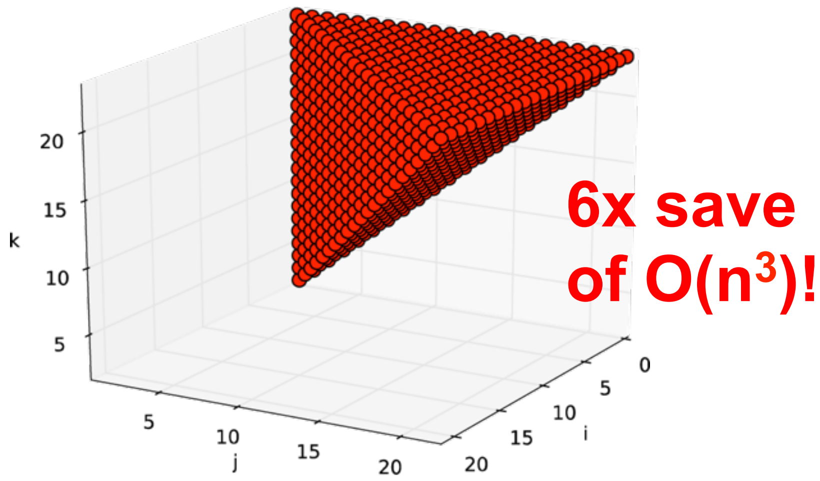
Strong Scaling of 1.5D N-body



Challenge: Symmetry & Load Balance

- Force symmetry ($f_{ij} = -f_{ji}$) saves computation
- 2-body force matrix vs 3-body force cube

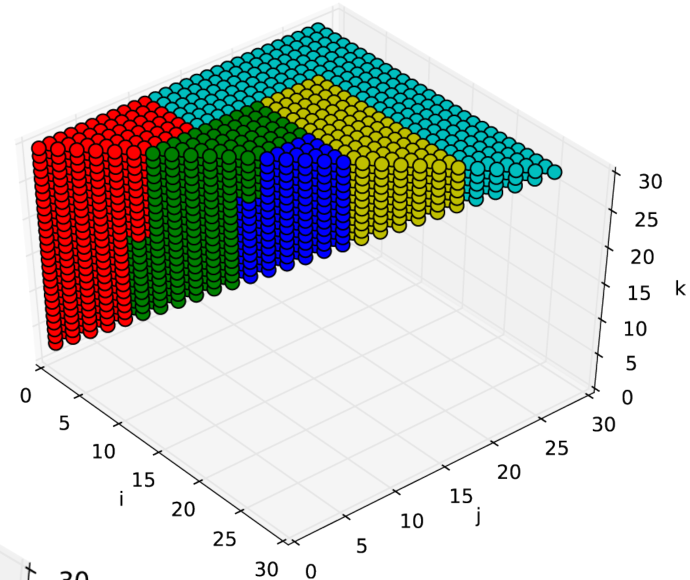




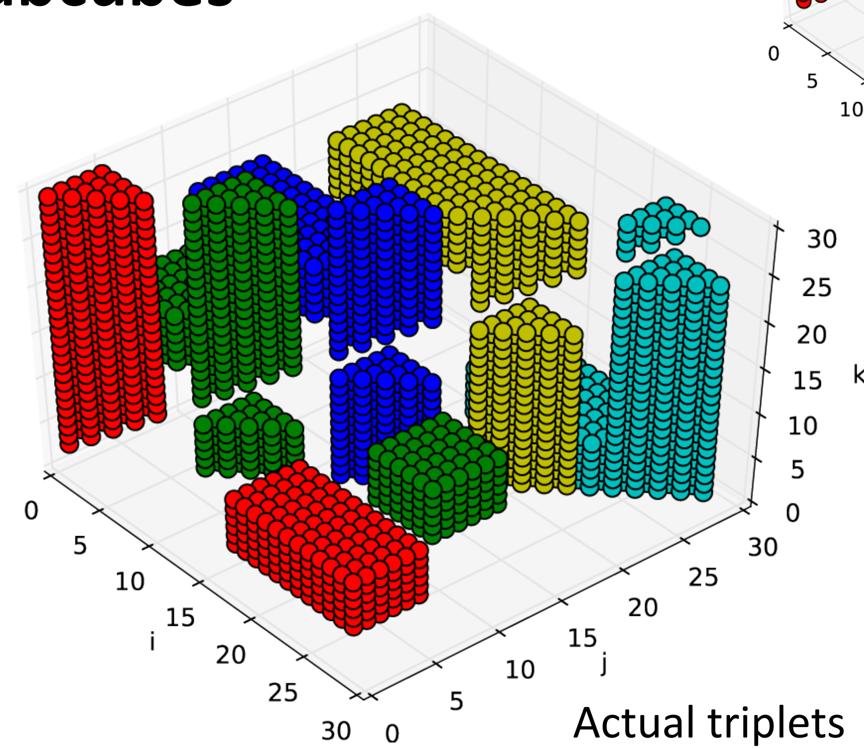
- How to divide work equally?

Communication-Avoiding 3-body

- **p=5 (in colors)**
- **6 particles per processor**
- **5x5 subcubes**



Equivalent triplets in the big tetrahedron

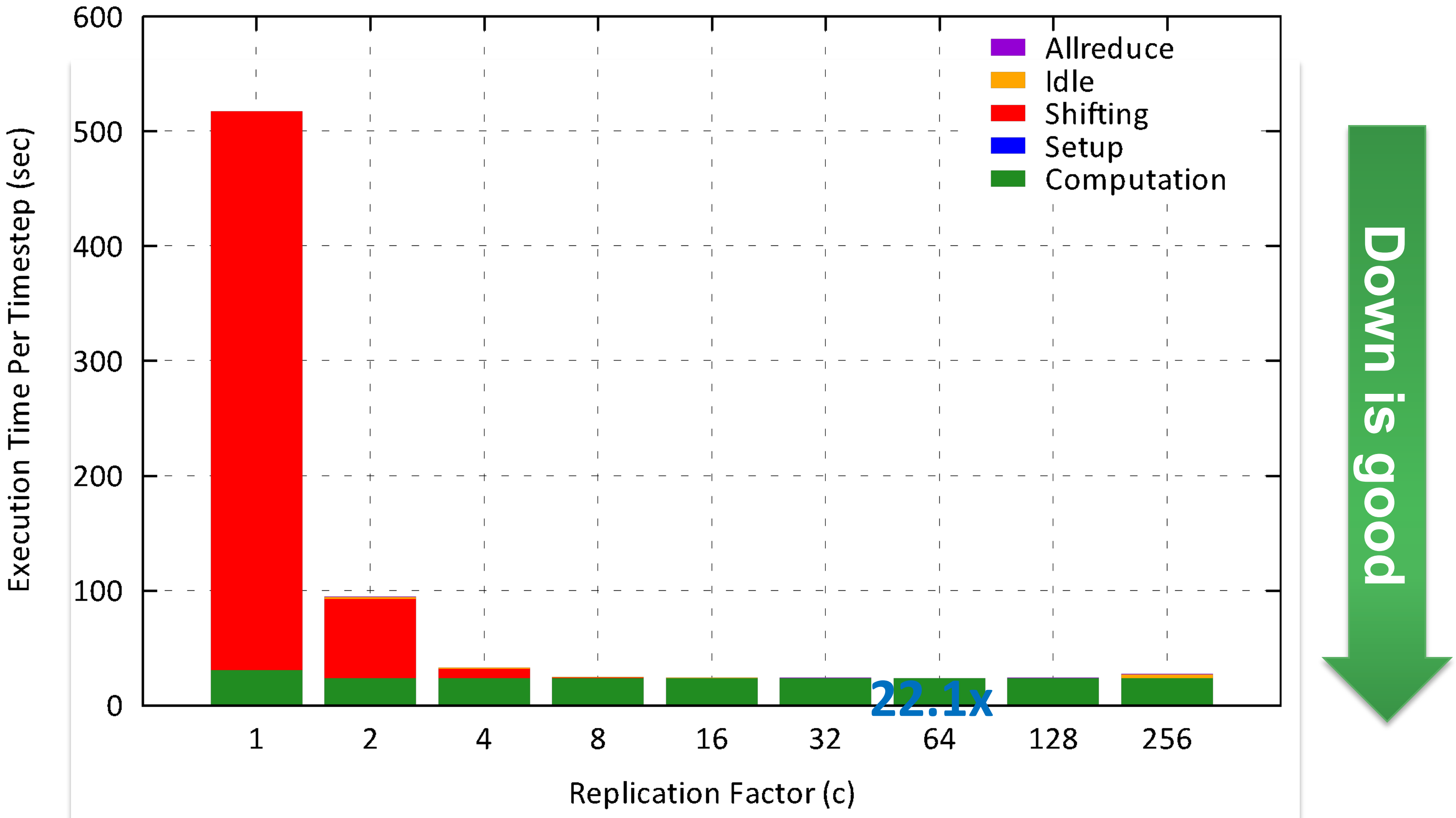


Actual triplets

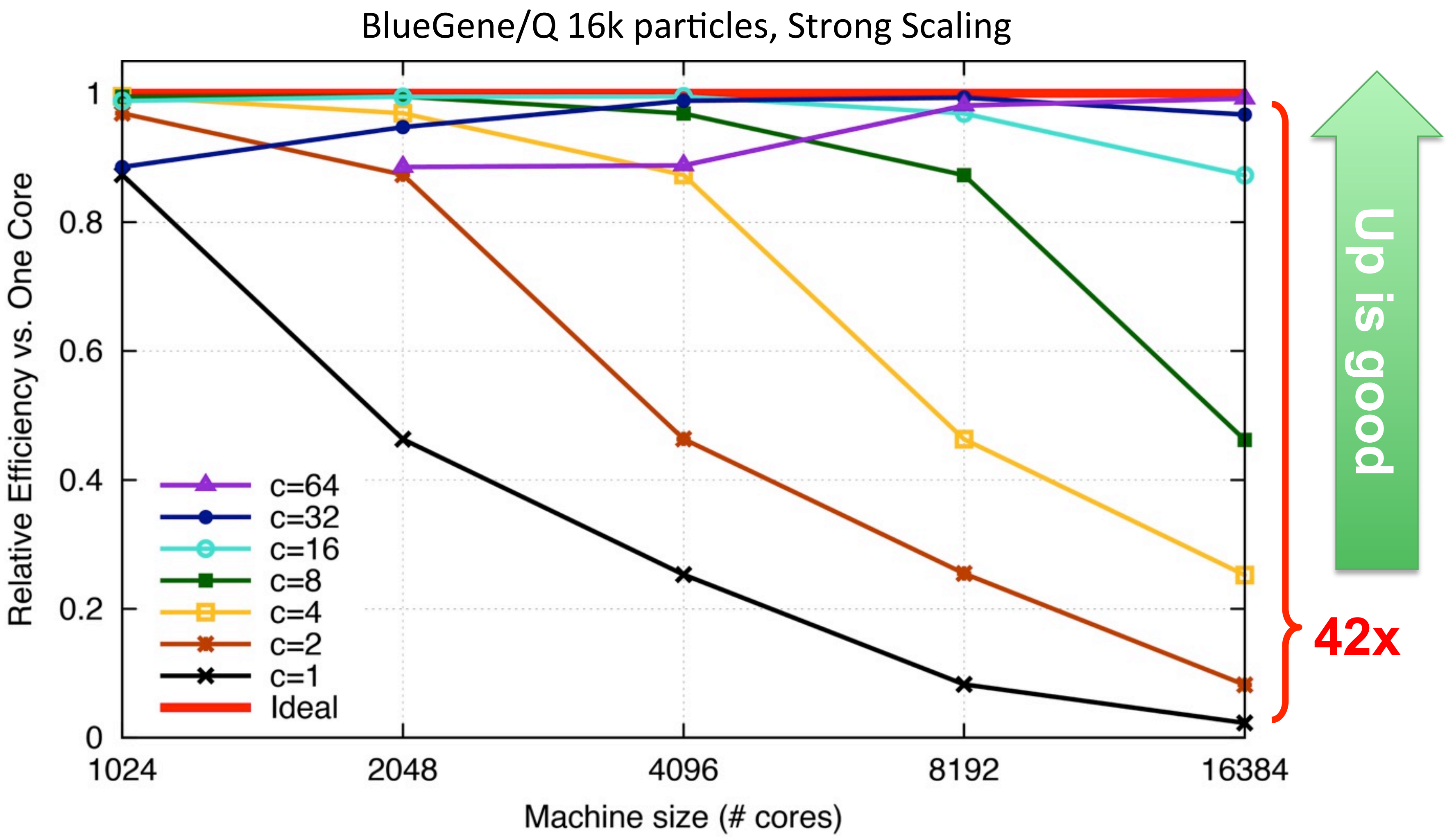
Communication optimal. Replication by c decreases #messages by c^3 and #words by c^2

3-Way N-Body Speedup

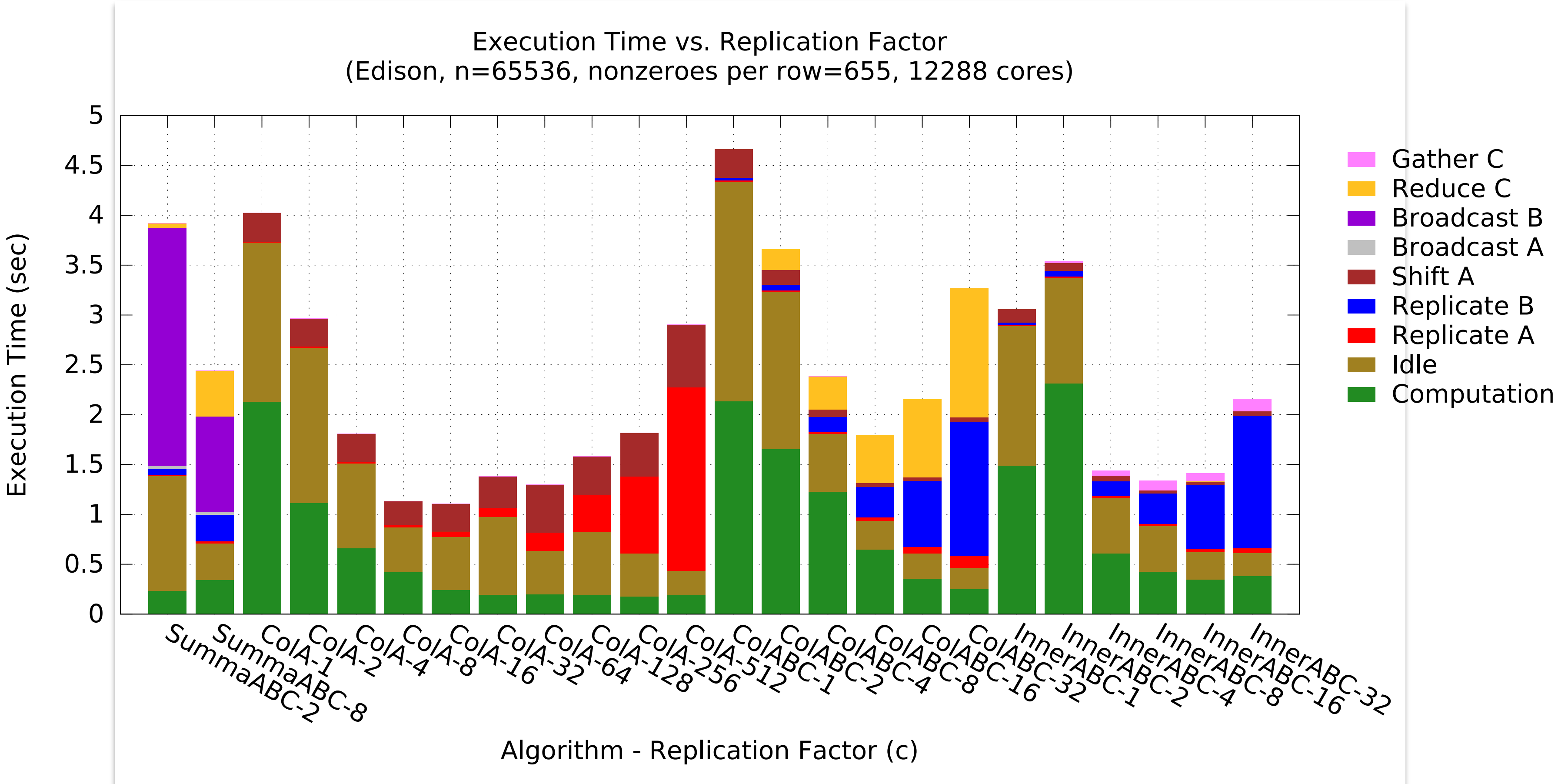
- **Cray XC30, 24k cores, 24k particles**



Perfect Strong Scaling



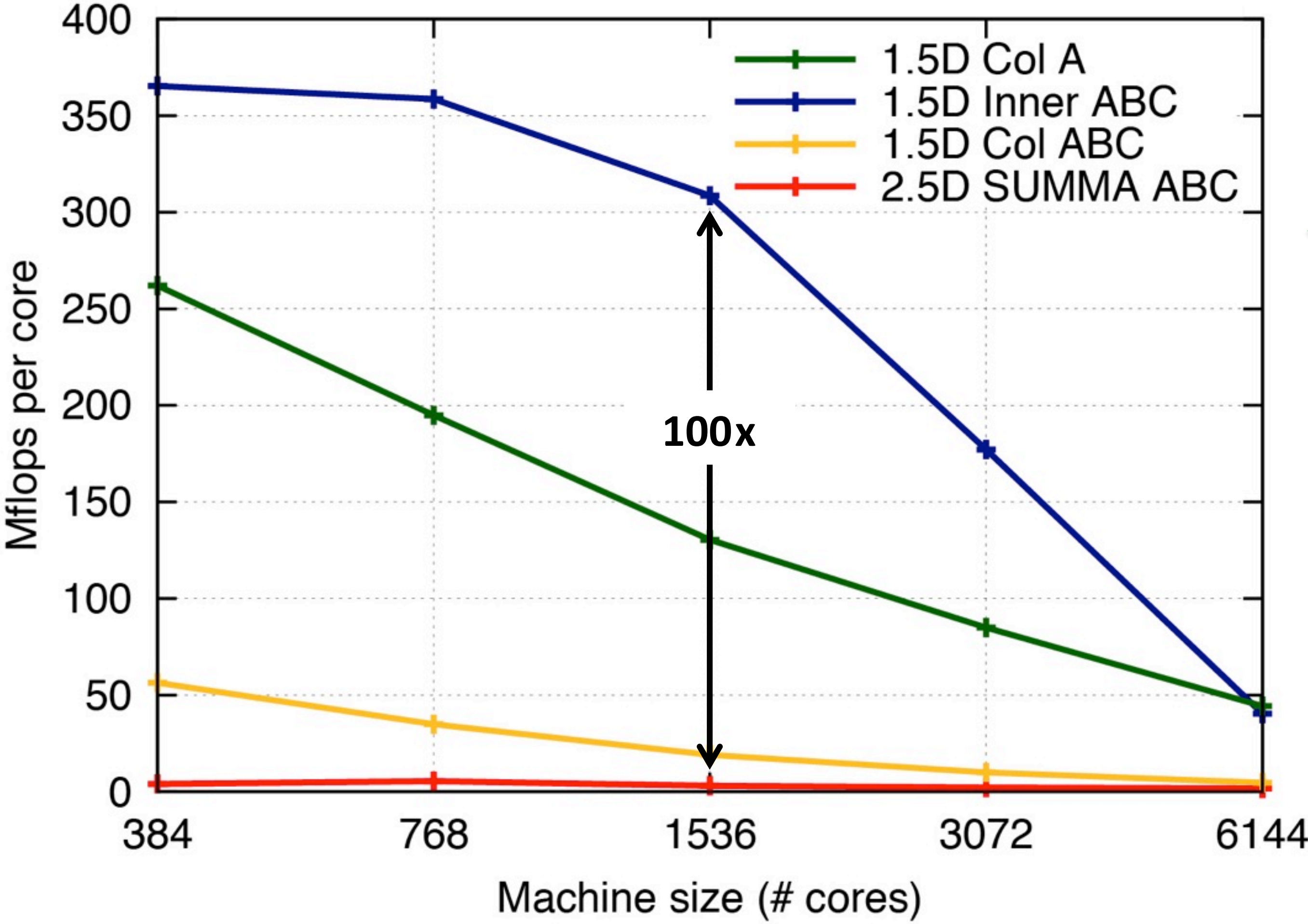
Sparse-Dense Matrix Multiply Too!

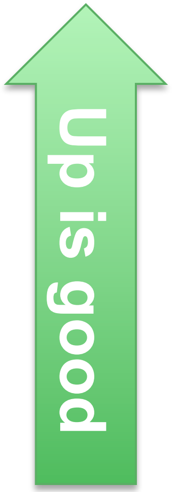


- **Variety of algorithms that divide in or 2 dimensions**

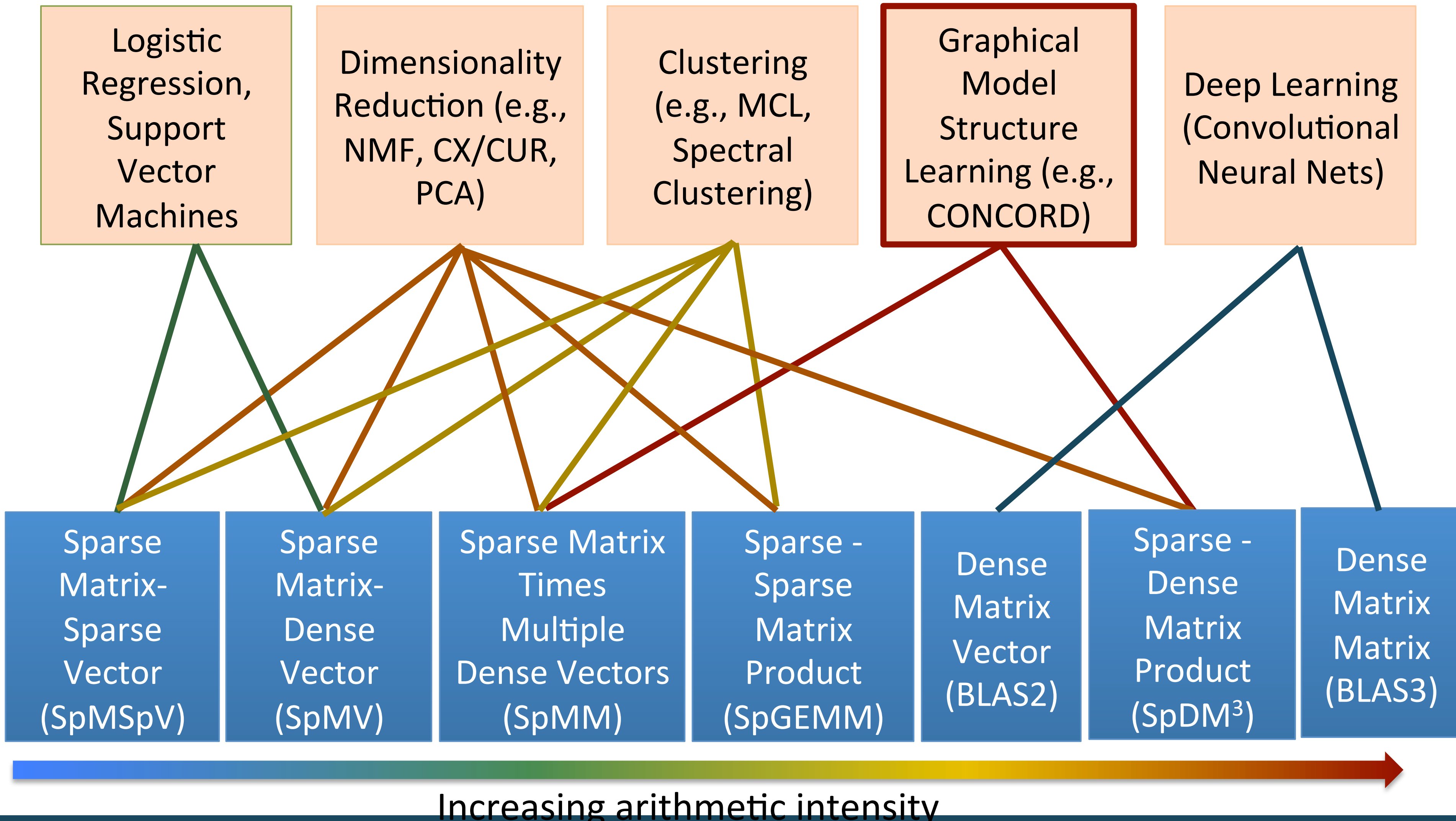
100x Improvement

- $A^{66k \times 172k}$, $B^{172k \times 66k}$, 0.0038% nnz, Cray XC30

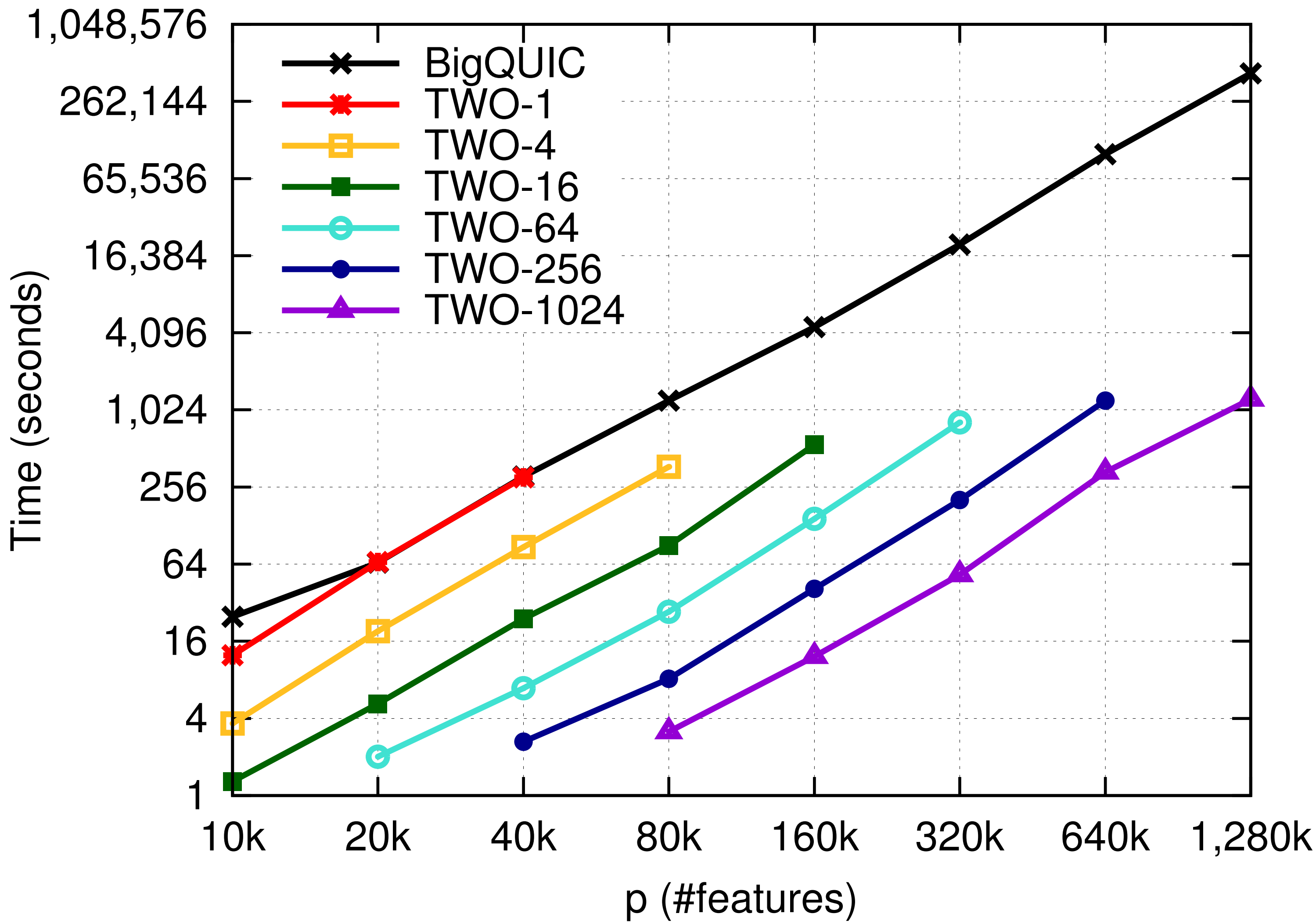




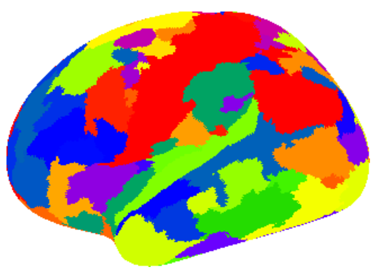
Linear Algebra is important to Machine Learning too!



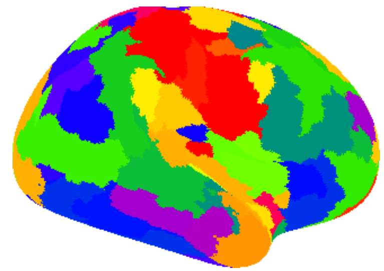
Inverse Covariance Matrix Estimation (CONCORD)



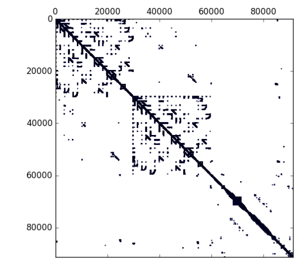
HP-CONCORD on Brain fMRI data



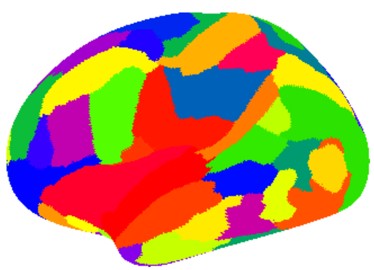
$\lambda_1 = 0.48,\ \lambda_2 = 0.39,\ \epsilon = 3,$
$\%$ of best score $= 100$



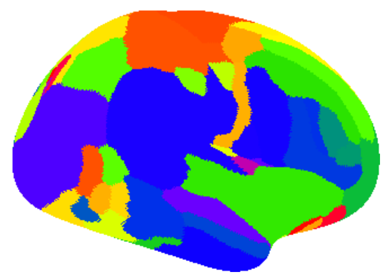
$\lambda_1 = 0.5,\ \lambda_2 = 0.39,\ \epsilon = 3,$
$\%$ of best score $= 100$



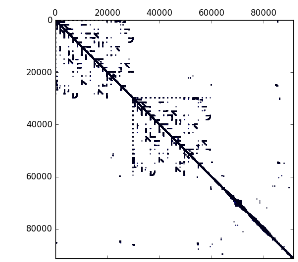
$\lambda_1 = 0.48,\ \lambda_2 = 0.39,\ \epsilon = 3,$
$\%$ of best score $= 100$



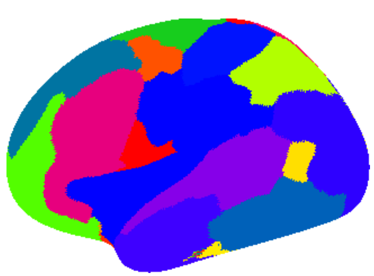
$\lambda_1 = 0.64,\ \lambda_2 = 0.13,\ k = 1,$
$\%$ of best score $= 75.03$



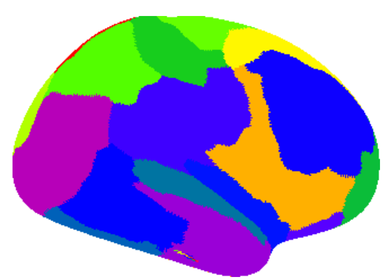
$\lambda_1 = 0.5425,\ \lambda_2 = 0.39,\ k = 0,$
$\%$ of best score $= 73.45$



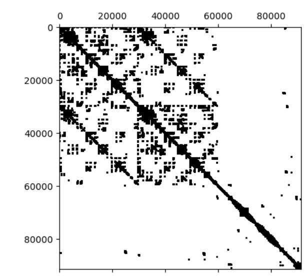
$\lambda_1 = 0.64,\ \lambda_2 = 0.13,\ k = 1,$
$\%$ of best score $= 75.03$



$t = 99.9,\ k = 4,$
$\%$ of best score $= 32.24$



$t = 99.9,\ k = 3,$
$\%$ of best score $= 32.45$



$t = 99.9,\ k = 4$
$\%$ of best score $= 32.24$

Problem 3: Runtimes

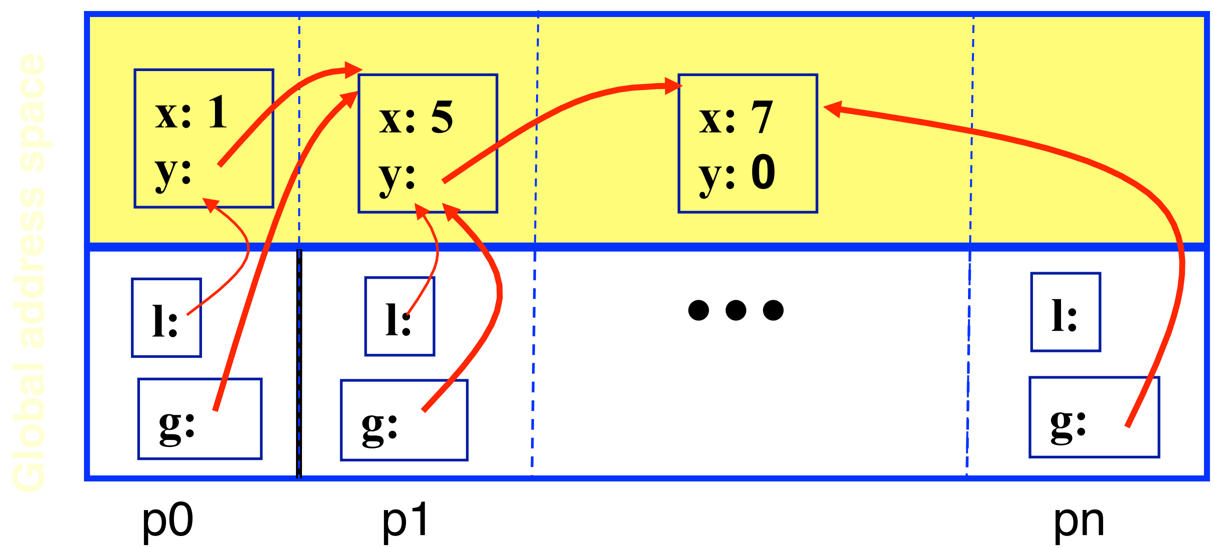
PGAS: A programming model for exascale

- ***Global address space:*** thread may directly read/write remote data using an address (pointers and arrays)

  ... = *gp;   ga[i] = ...

- *Partitioned:* data is designated as local or global

  shared int [ ] ga;   and   upc_malloc (...)



Examples:
UPC
UPC++

A programming model can influence how programmers think

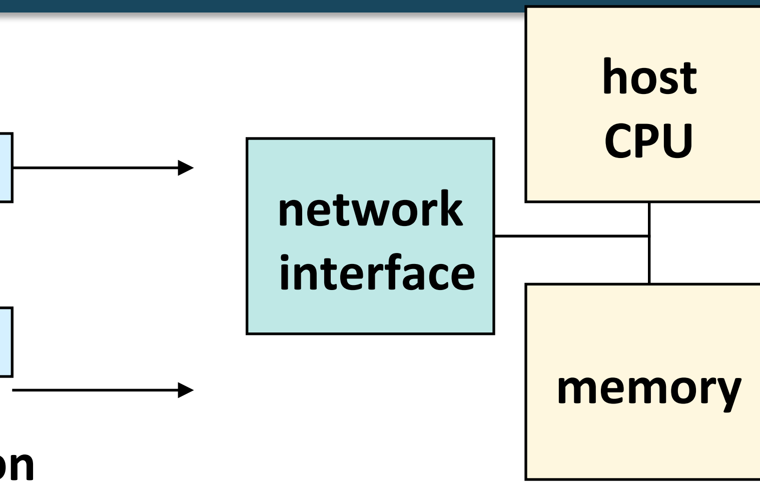
One-Sided Communication is Closer to Hardware

one-sided message

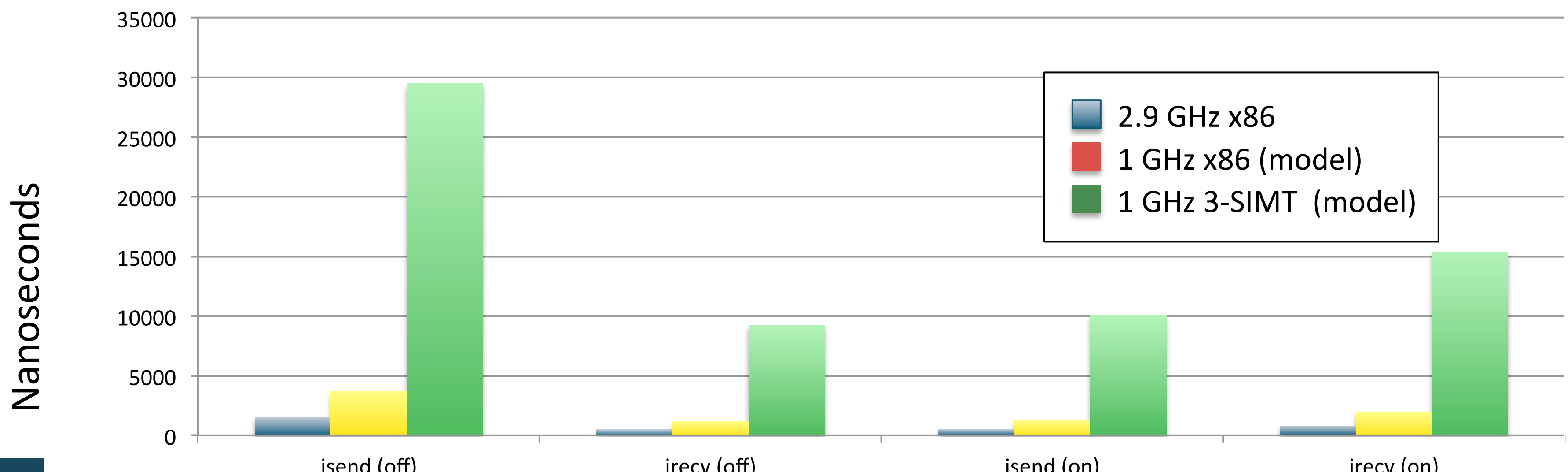
address	data payload

two-sided put message

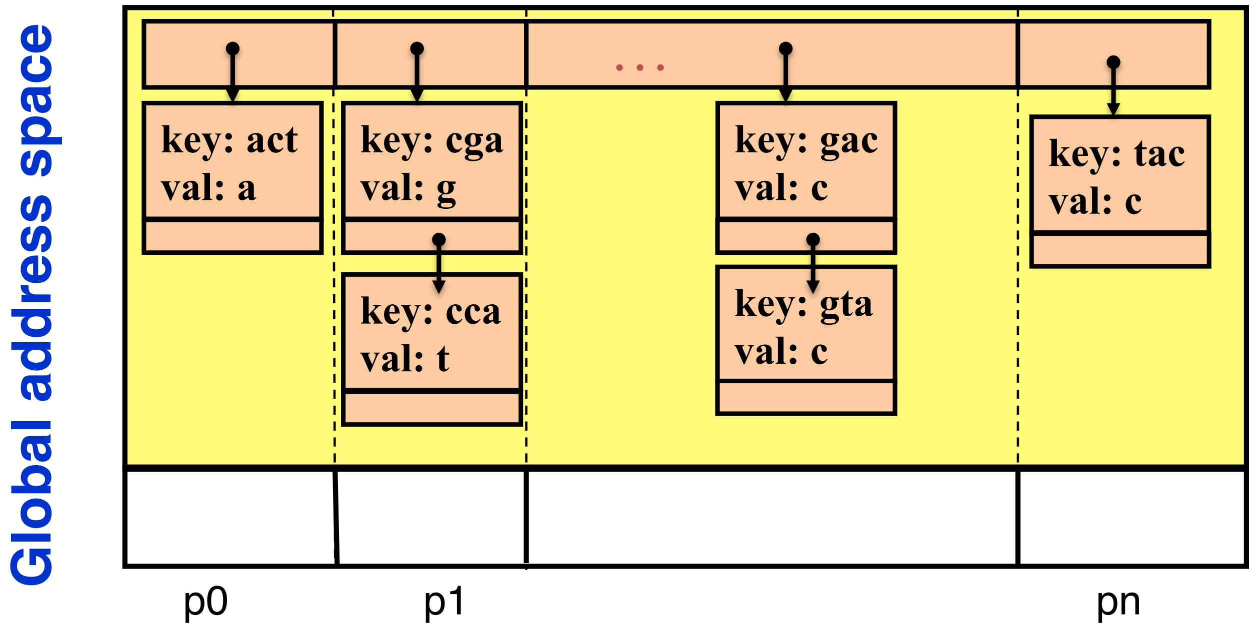
message ID	data payload



- **Hardware does 1-sided communication**
- **Overhead for send/receive messaging is worse at exascale**



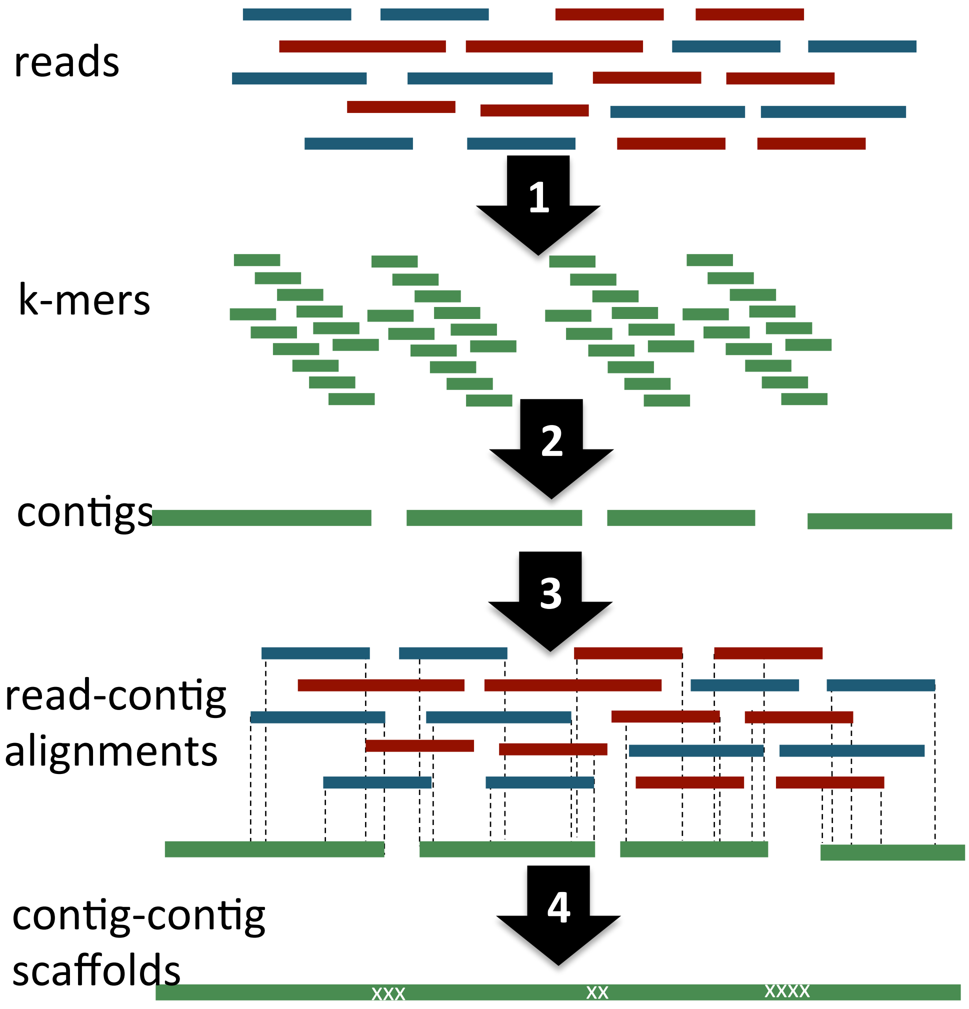
What we love about Partitioned Global Address Space Programming (PGAS)



Key: Never cache remote data (trivially coherent)

- **Convenience**
  – Build large shared structures
  – Read and write data "anywhere" (global), "anytime" (asynchronous) and without the other thread (one-sided)
- **Performance control**
  – Explicit control over data layout, direct use of RDMA hardware

HipMer is all about the runtime and data structures



1) *K-mer Analysis*

(synchronous) irregular all-to-all

2) *Contig Generation*

asynchronous remote insert (aggregate and overlap) and get

3) *Alignment*

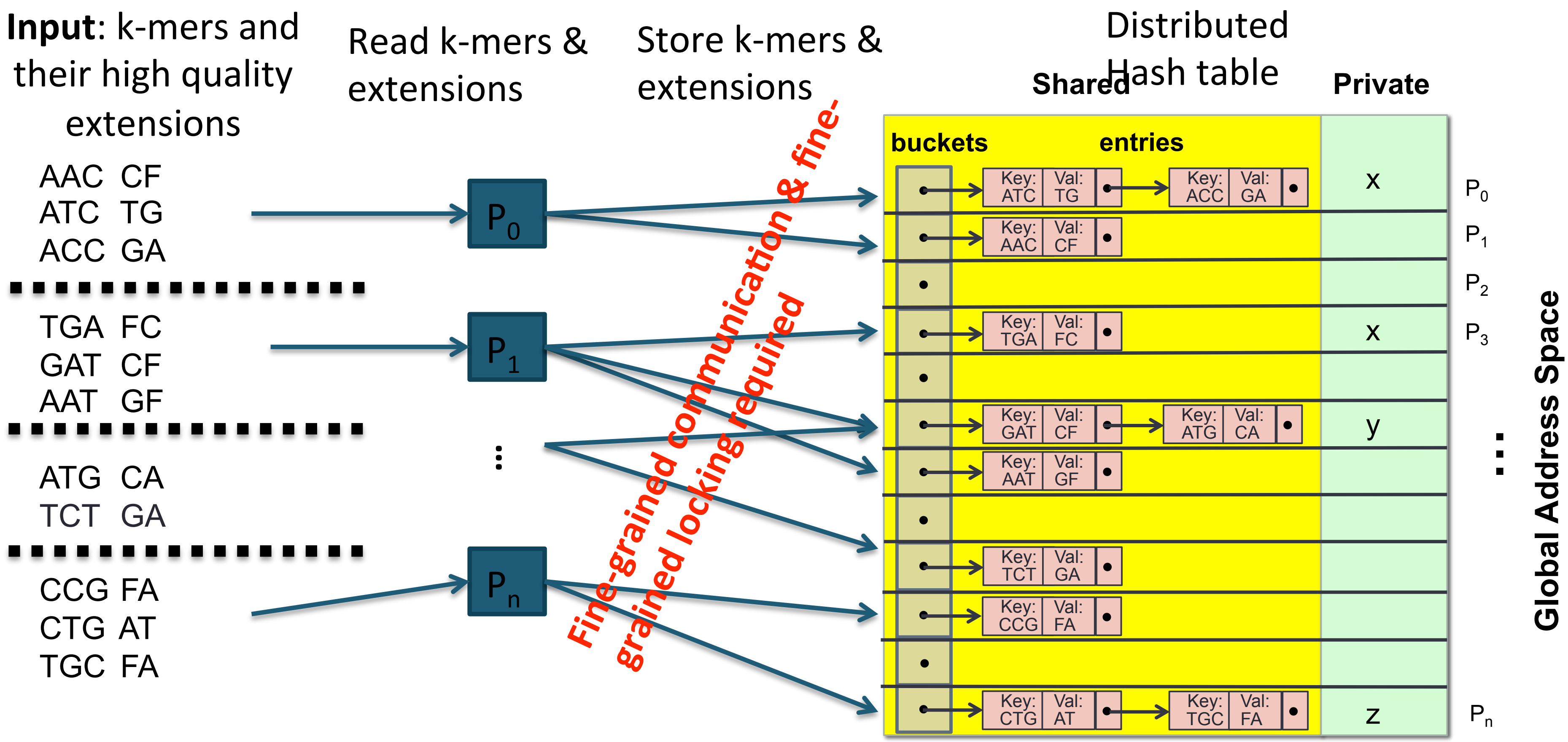
asynchronous remote insert and lookup (software caching)

4) *Scaffolding & Gap Closing*

asynchronous remote insert and lookup (software caching)

Graph algorithms (hash tables) in genome assembly

Graph construction, traversal, and all later stages are written in UPC to take advantage of its global address space



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

Communication Hurts!