Avoiding, Hiding and Managing Communication at the Exascale

Kathy Yelick
Associate Laboratory Director
Computing Sciences

Computer Science Professor
UC Berkeley

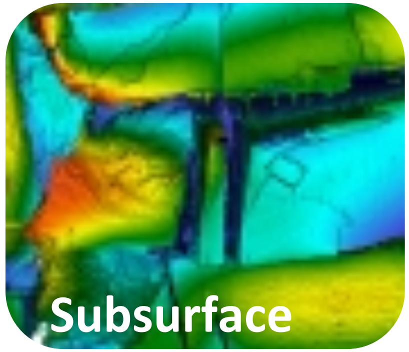
Deploy Exascale Systems at NERSC

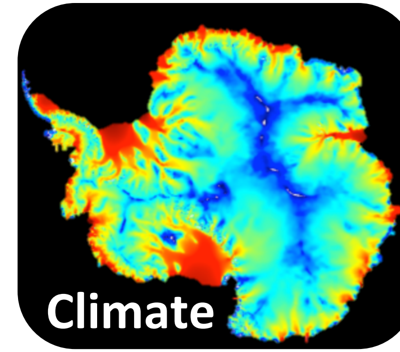
NERSC: the broadest, most open, widely used computing center in DOE

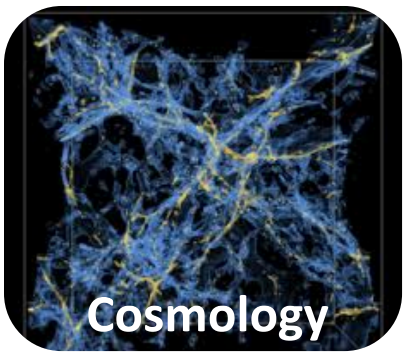
- 7000 users, 2000 publications / year, 700 codes, 5 associated Nobels
- Cori system delivers petascale science for simulation and data

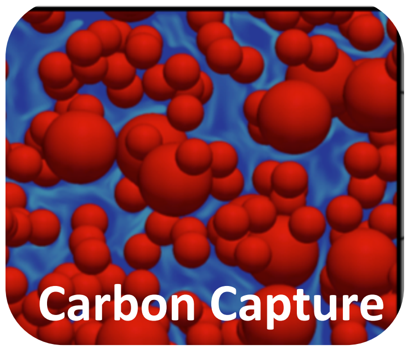
Exascale goal: broad scientific impact through exascale at NERSC

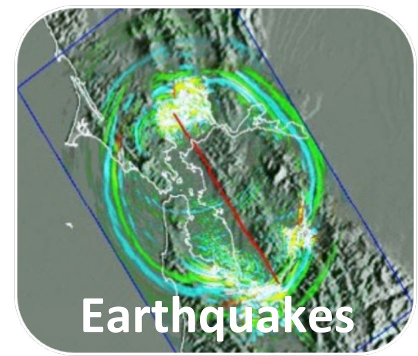
Berkeley Lab Priorities in Exascale Science

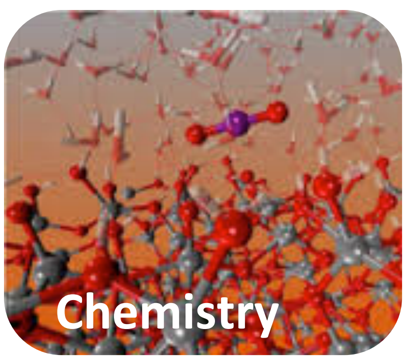


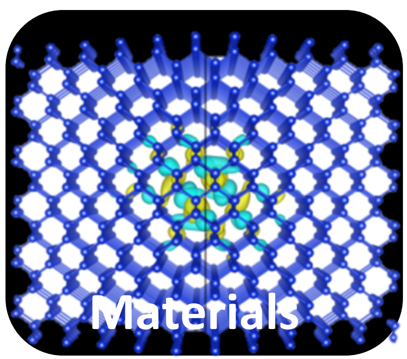


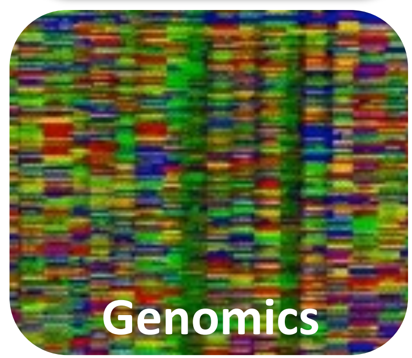


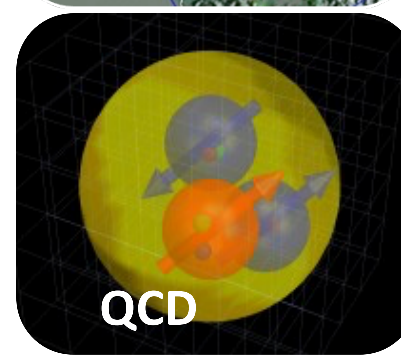




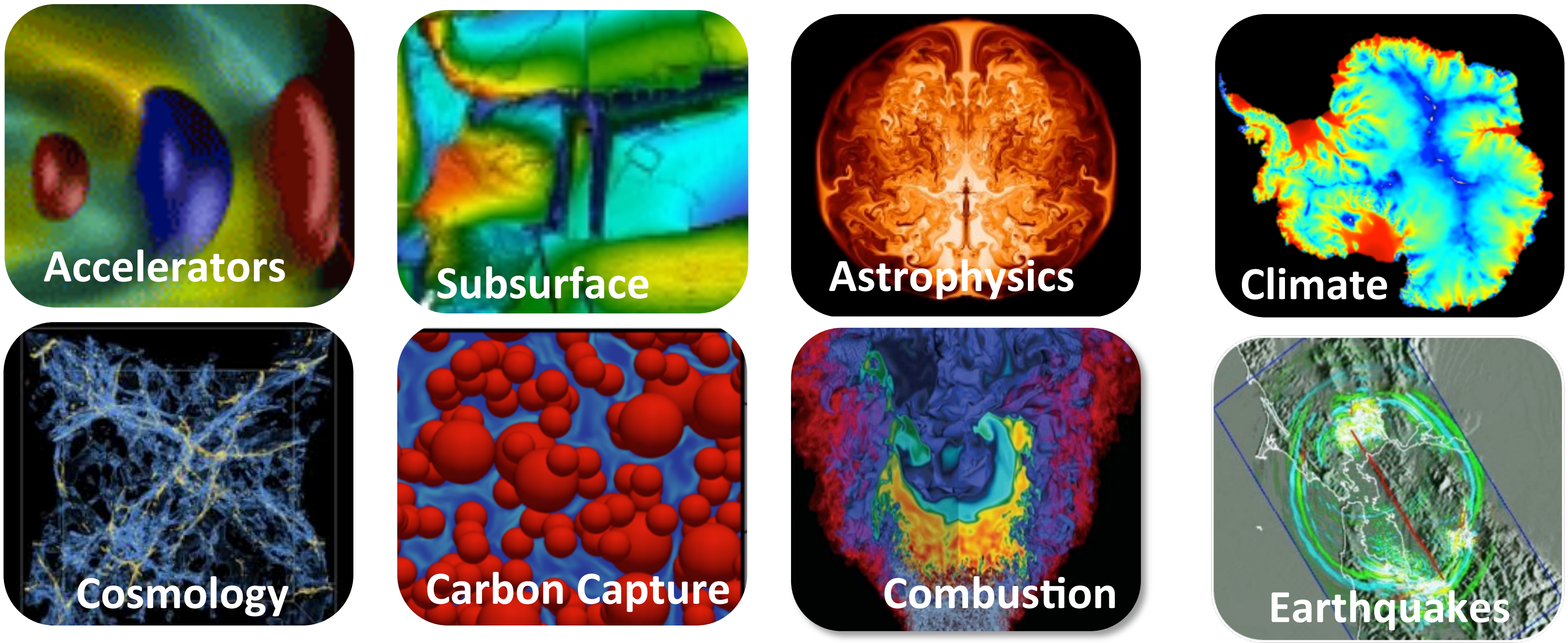






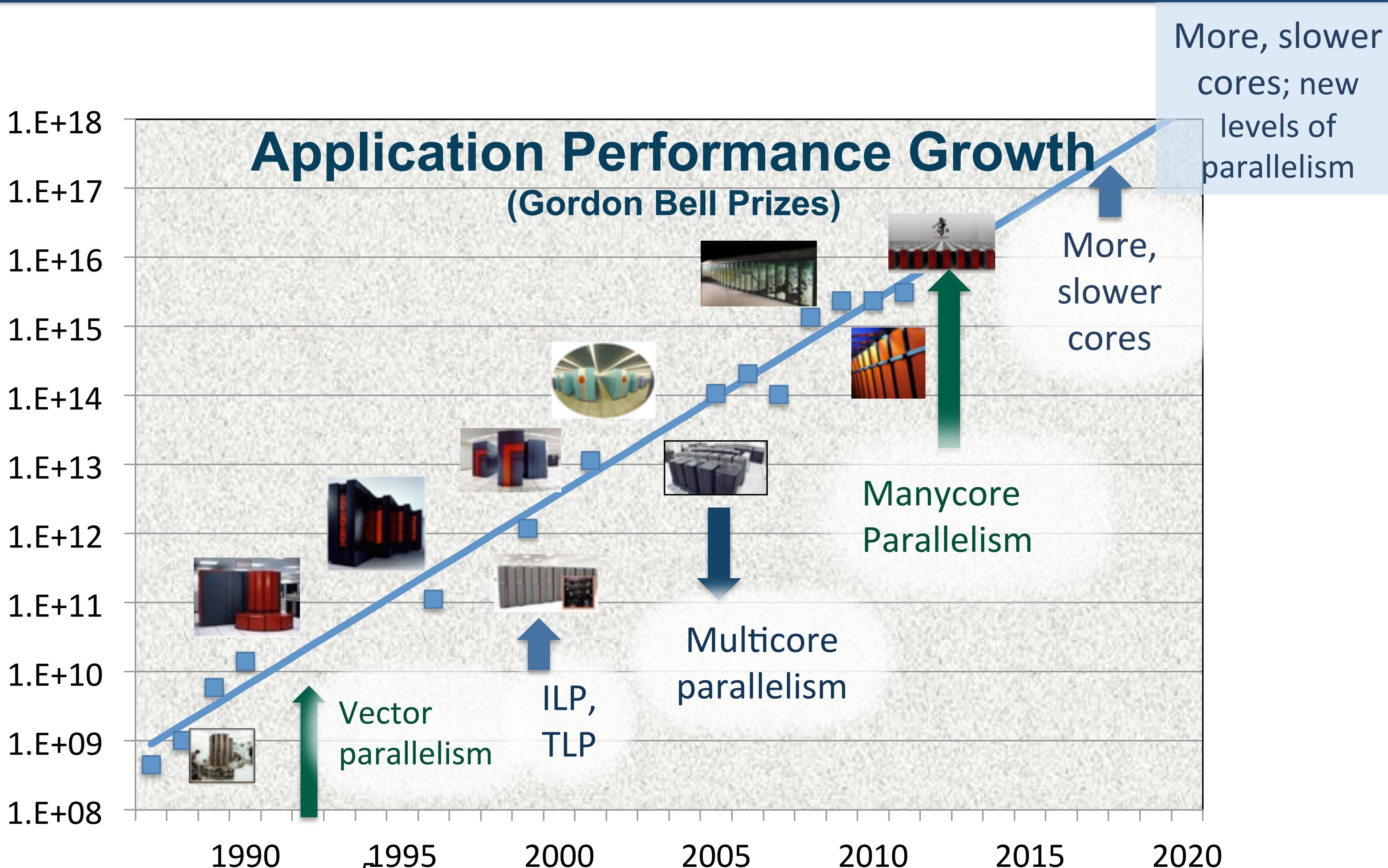


Berkeley Lab Priorities in Exascale Science



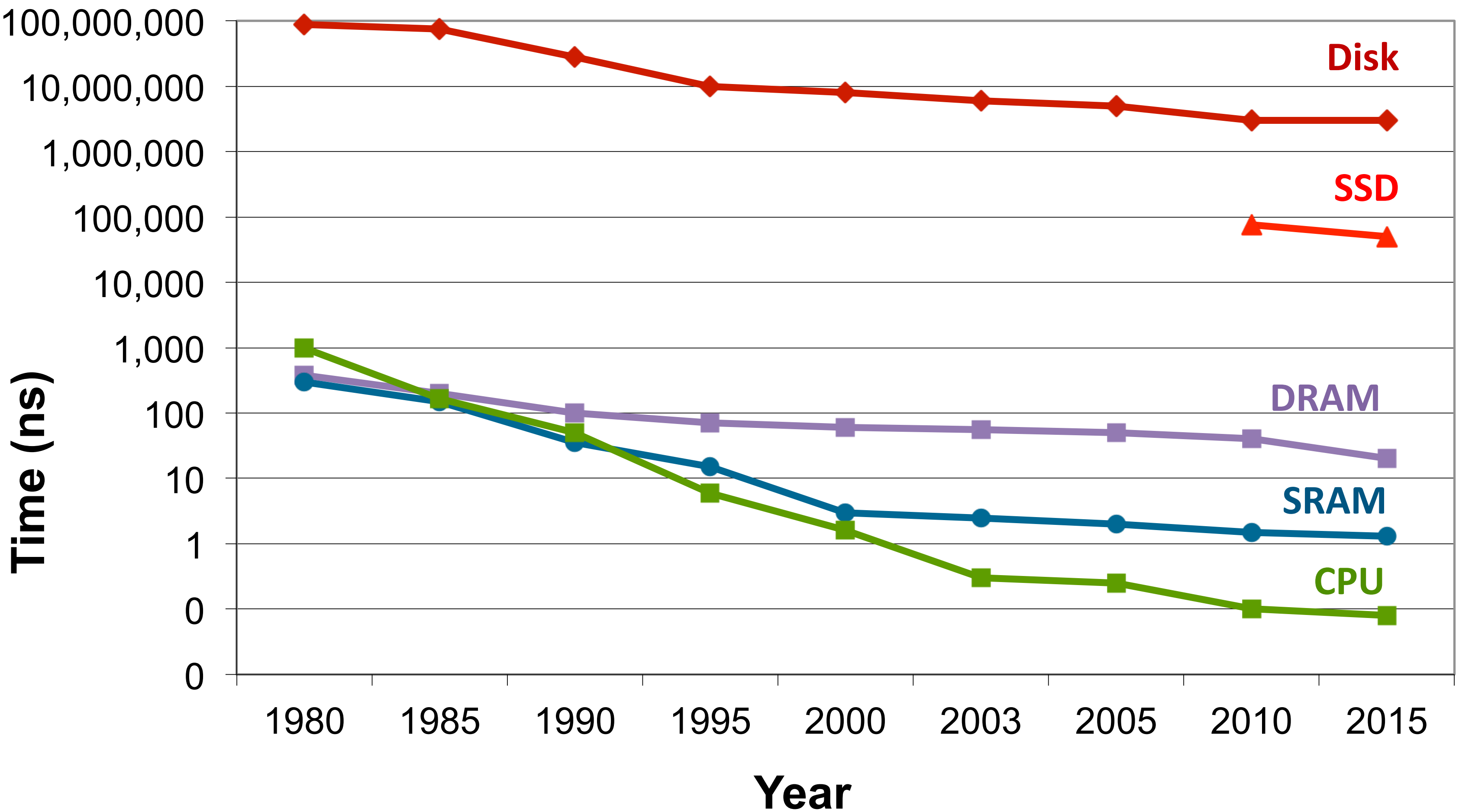
All the above will use Adaptive Mesh Refinement (AMR) mathematics and software, a method pioneered at Berkeley Lab

Hardware Trend: More Parallelism at Lower Levels



Data Movement is Expensive

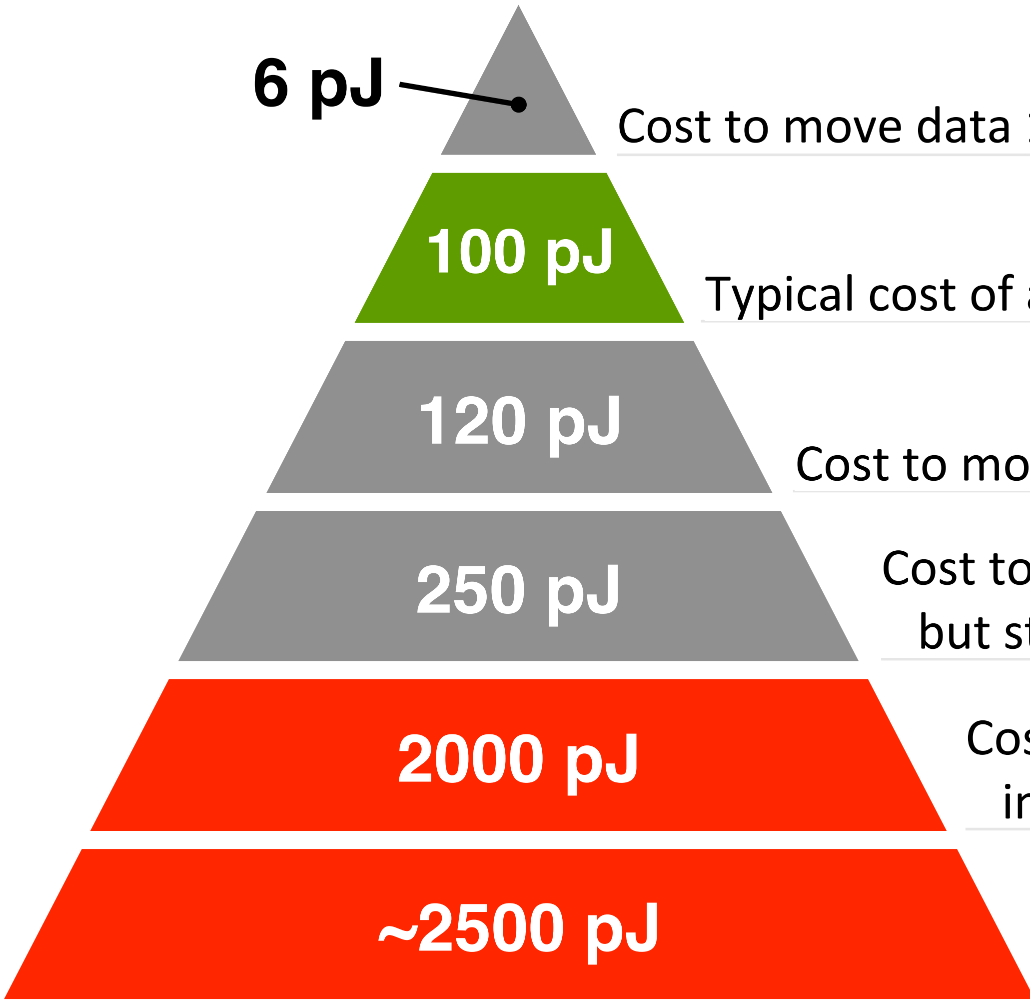
CPU cycle time vs memory access time



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

Data Movement is Expensive

Hierarchical energy costs.



Cost to move data 1 mm on-chip

Typical cost of a single floating point operation

Cost to move data 20 mm on chip

Cost to move off-chip,
but stay within the package (SMP)

Cost to move data off chip
into DRAM

Cost to move data off chip
to a neighboring node

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

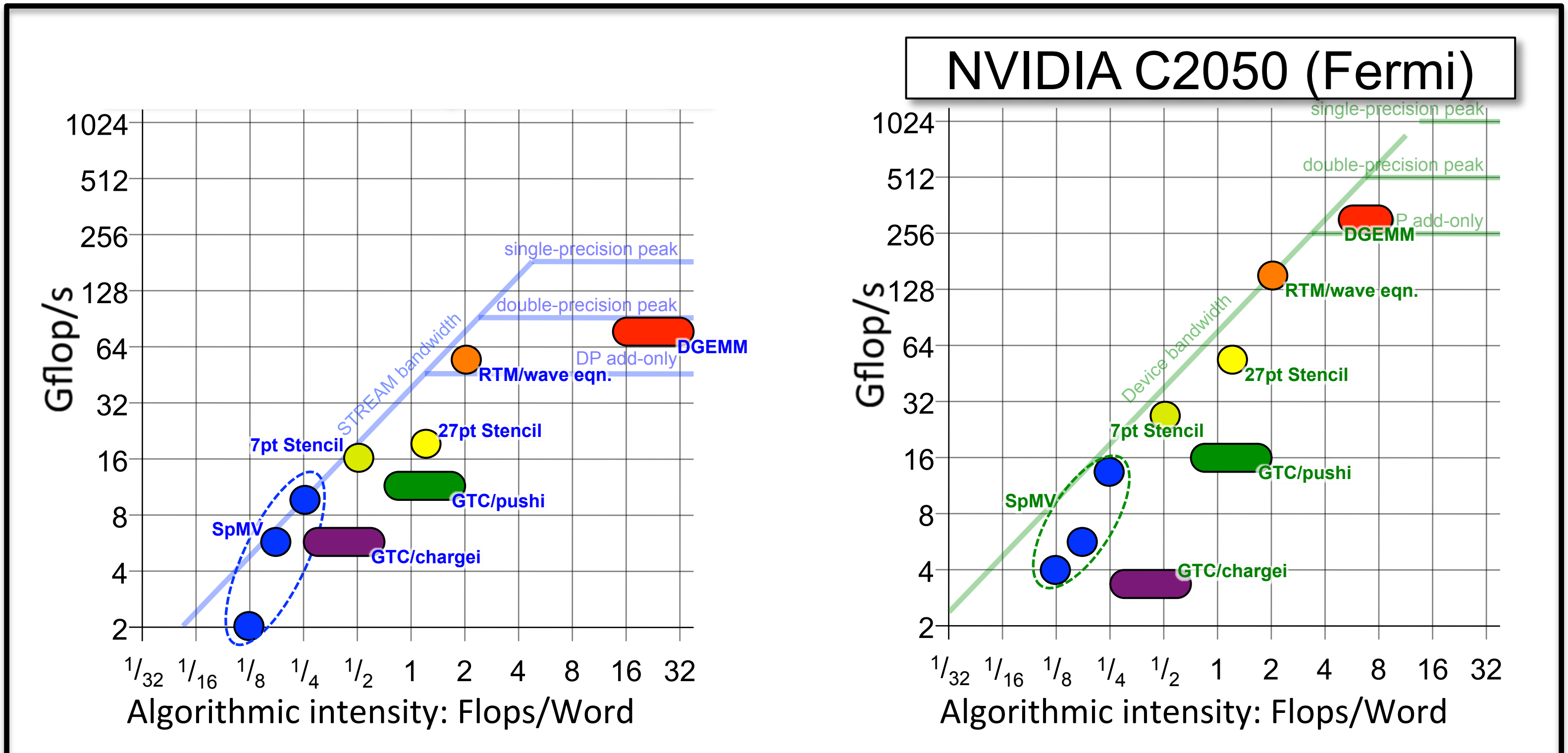
Summary

- More parallelism
- Diversity of processors / accelerators
- Communication (data movement) is expensive

Compilers and Autotuning

Autotuning: Write Code Generators

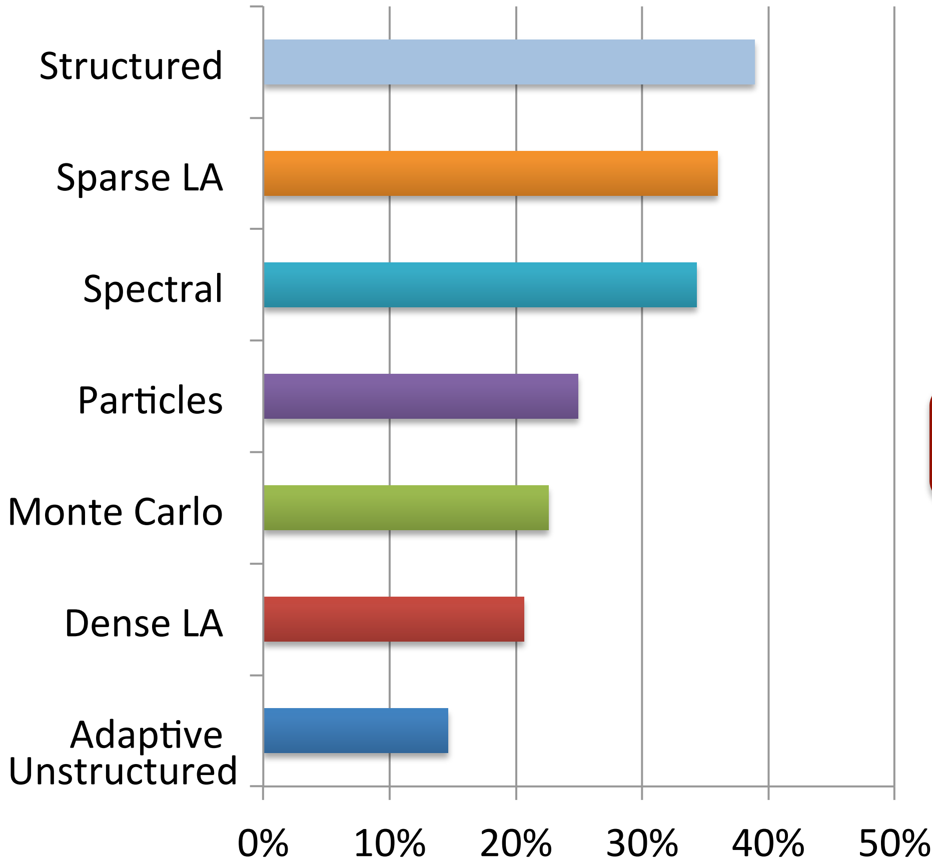
- **Two "unsolved" compiler problems:**
  - dependence analysis and — Domain-Specific Languages help with this
  - ✔ accurate performance models — Autotuning avoids this problem
- **Autotuners are code generators plus search**



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

Libraries vs. DSLs (domain-specific languages)

NERSC survey: what motifs do they use?



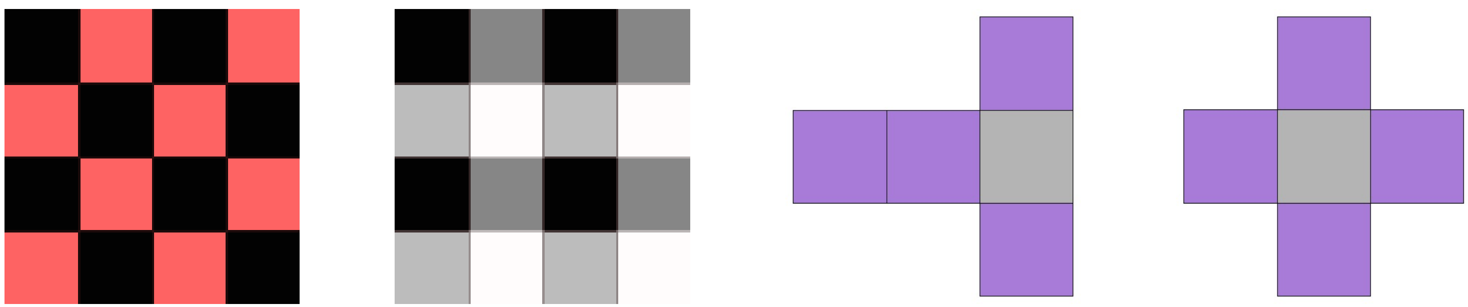
What code generators do we have?

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

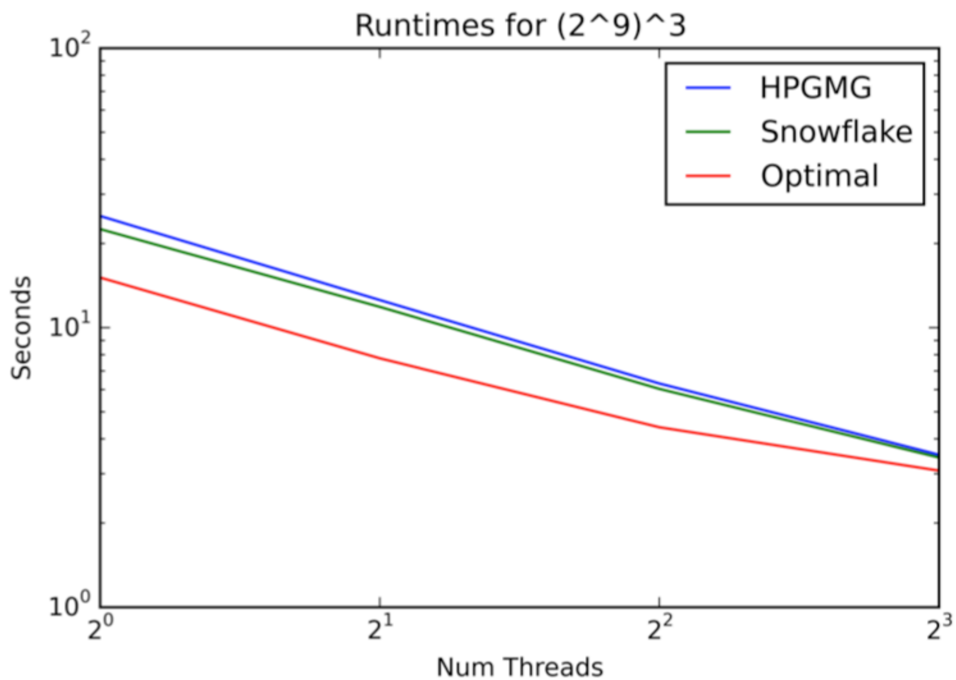
(b) 4-color tiling

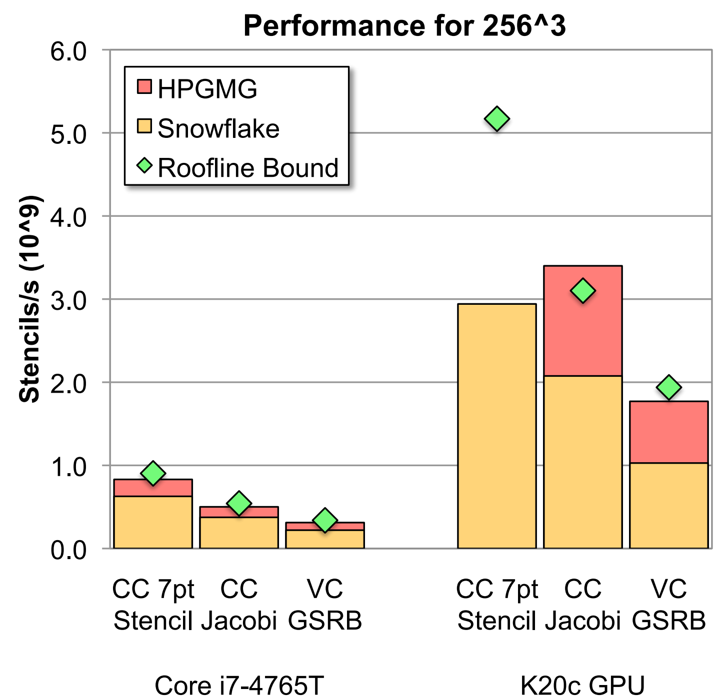
(c) Asymmetric stencil used near mesh boundary

(d) 5-point Jacobi stencil

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

Snowflake Performance





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

Nathan Zhang, C. Markley, S. Williams, A. Fox, K. Yelick

DSLs popular outside scientific computing

Developed for Image Processing

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

HPGMG (Multigrid on Halide)

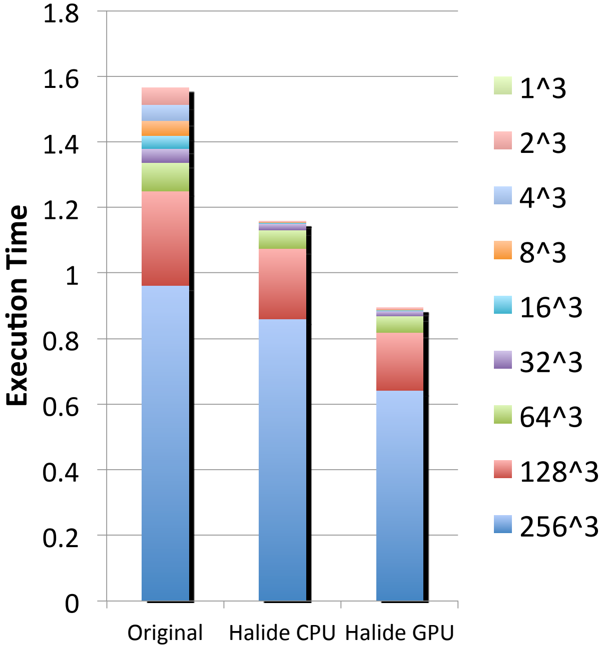
- **Halide Algorithm by domain expert**

```
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



Algorithms for the Hardware

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

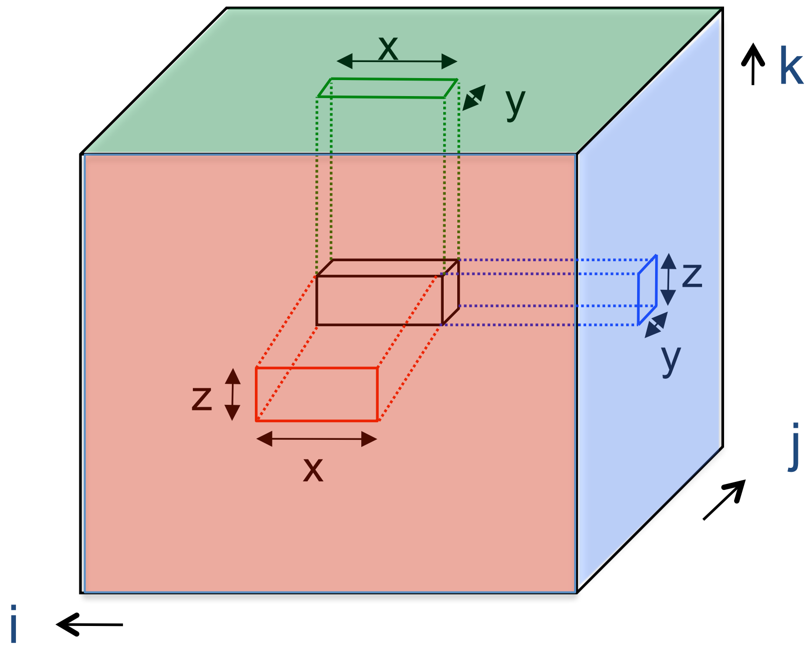
Lesson: Never waste fast memory

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

Can we generalize for compiler writers?

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] …

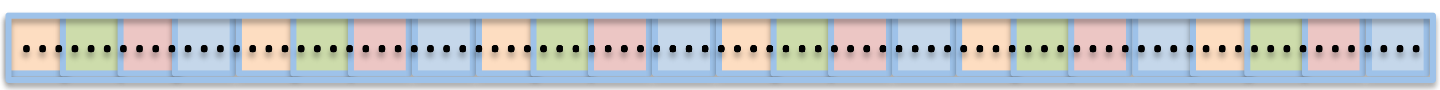
Using .5D ideas on N-body

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

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

$O(n^k)$.

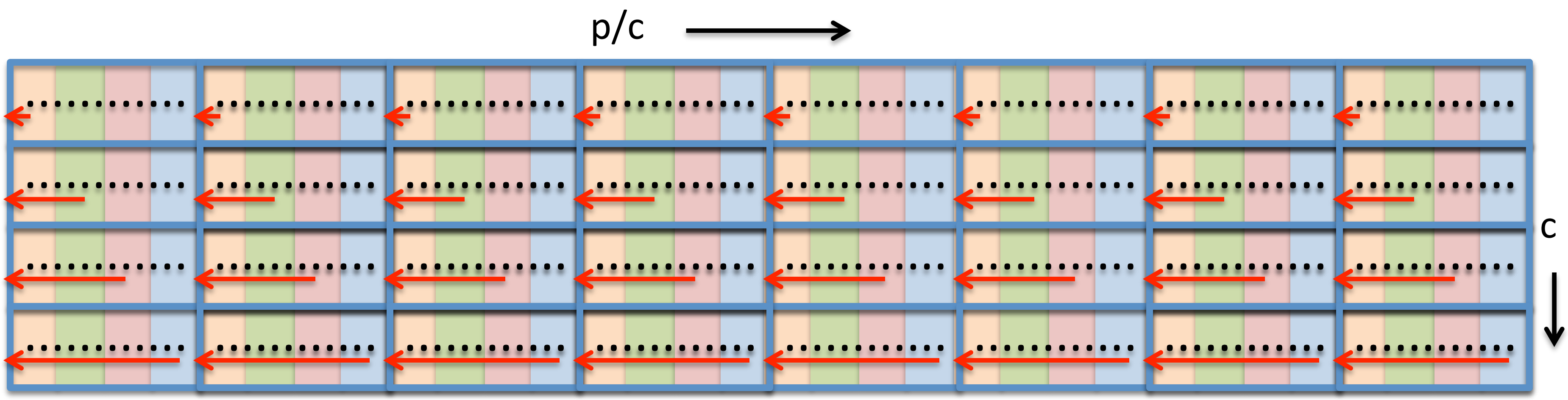
- **Best algorithm is to divide n particles into p groups??**



No!

Penporn Koanantakool, K. Yelick

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**

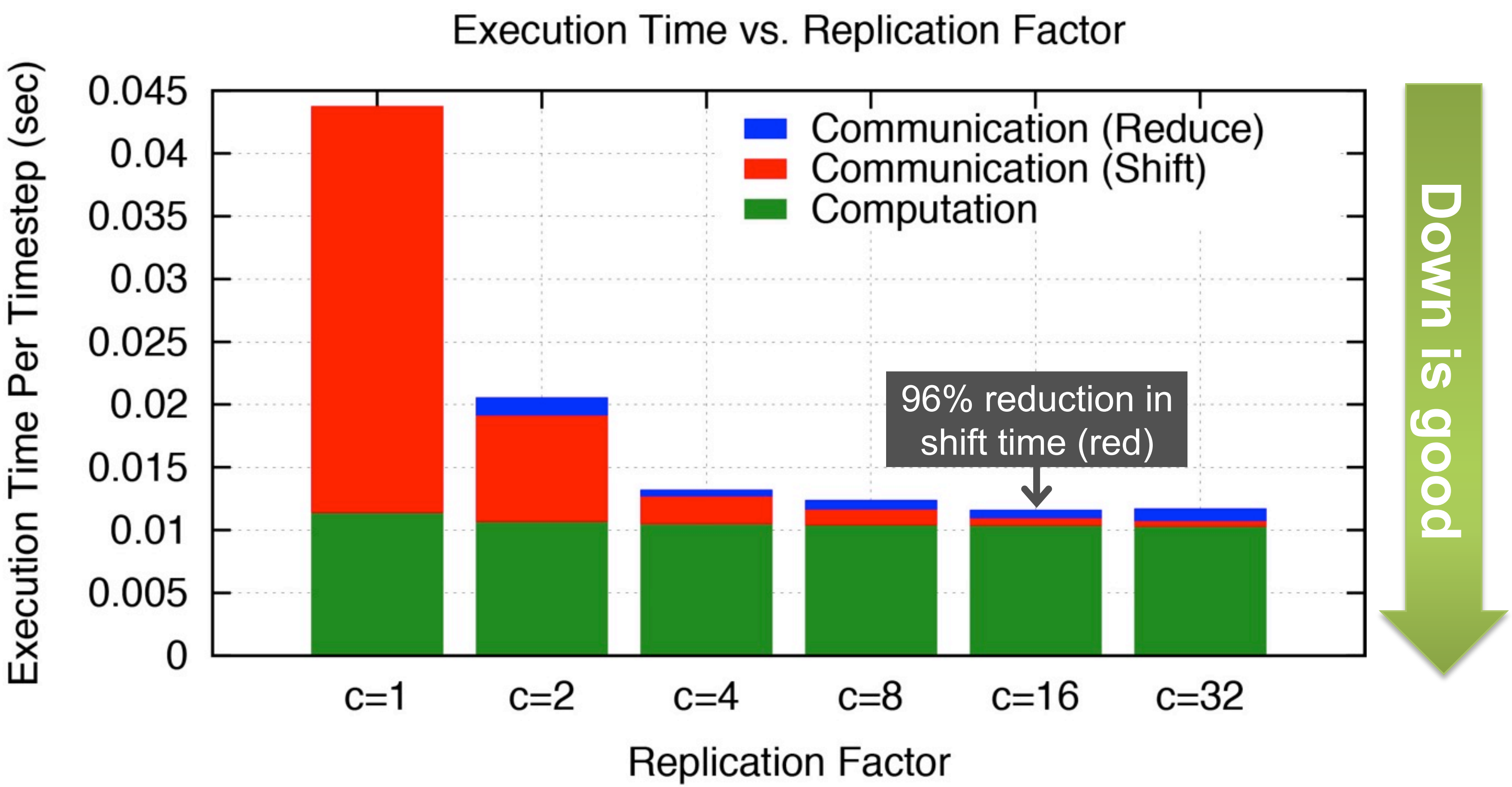
Total Communication: O(log(p/c) + log c) messages,

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

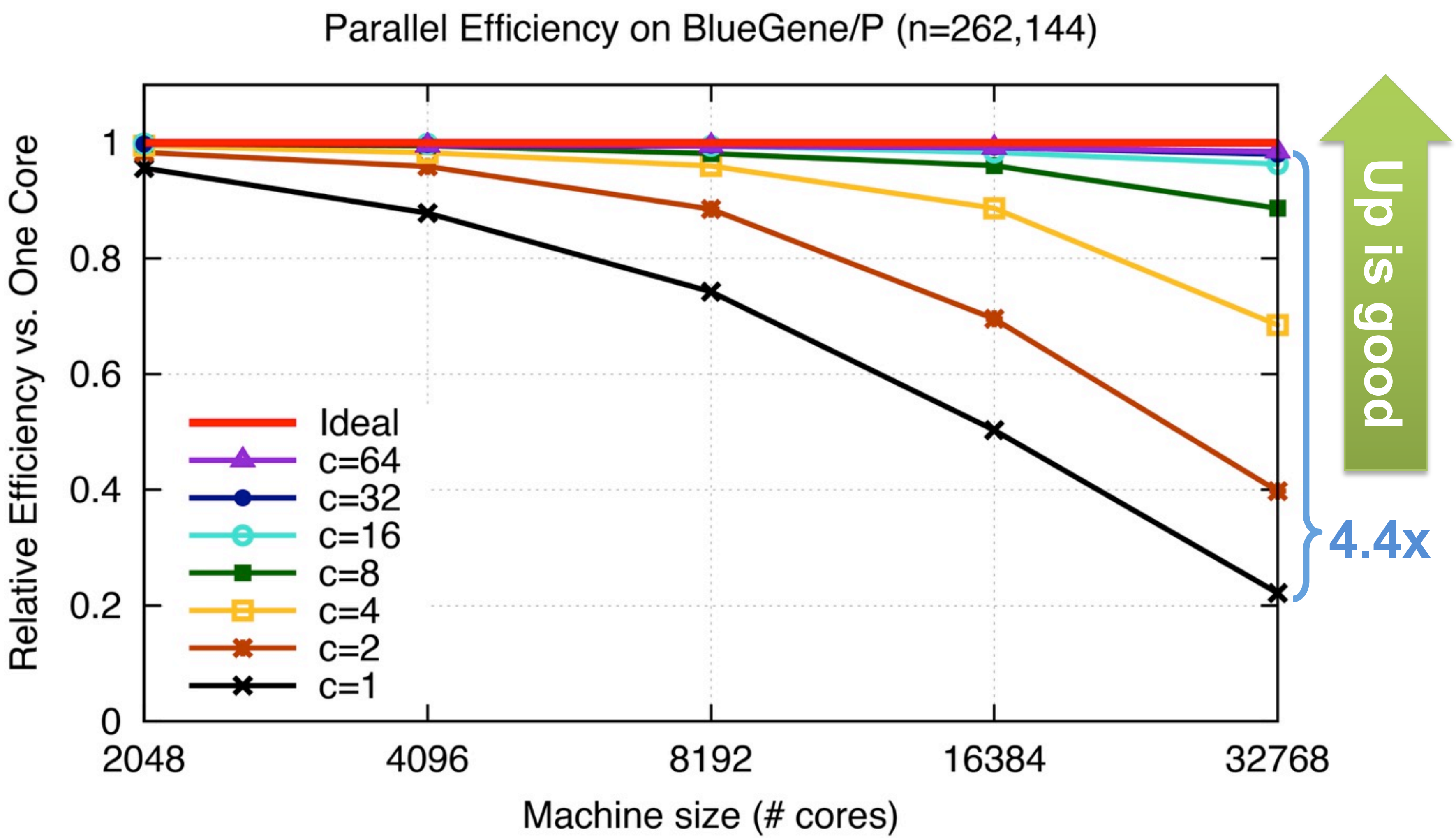
Driscoll, Georganas, Koanantakool, Solomonik, Yelick

Less Communication..

- **Cray XE-6; n=24K particles, p=6K cores**

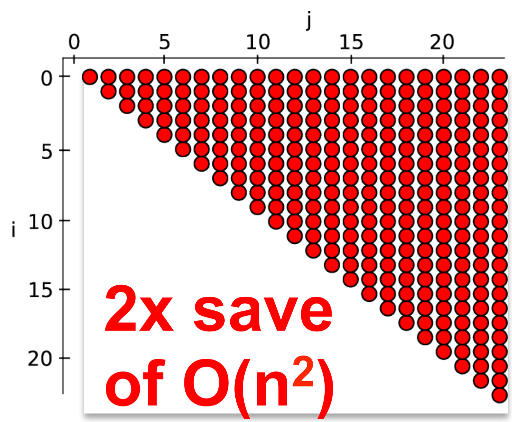


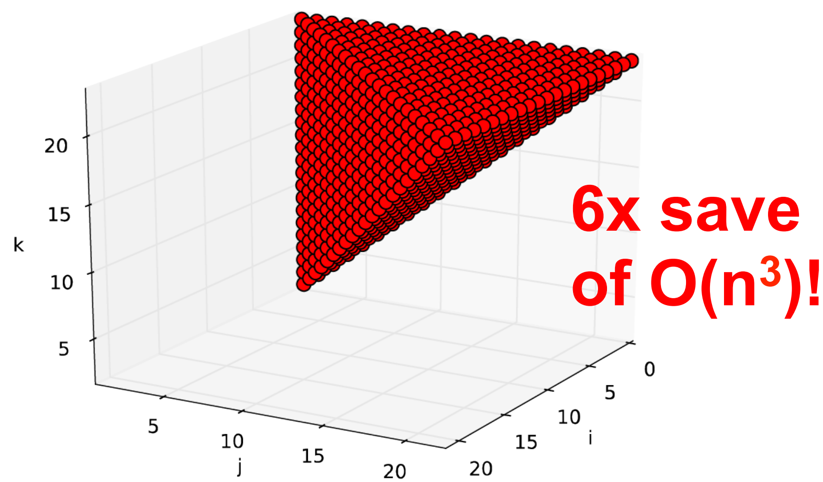
Strong Scaling



Challenge: Symmetry & Load Balance

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

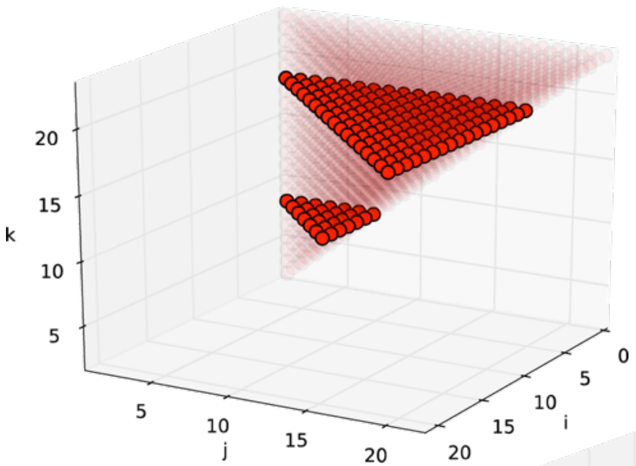


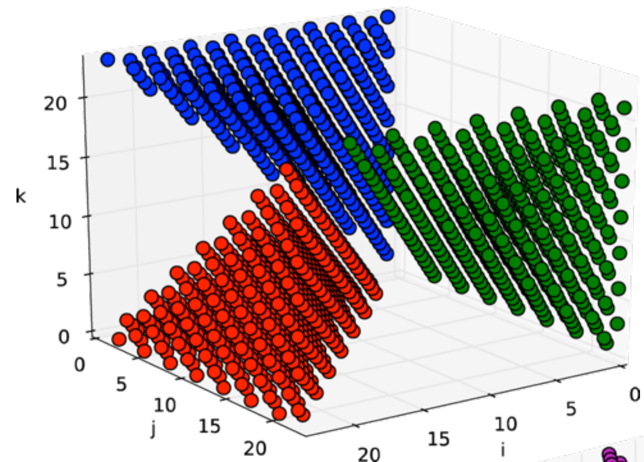


- How to divide work equally?

Koanantakool & Yelick

All-triplets 3-body: Challenges





[Sumanth et al. 2007]

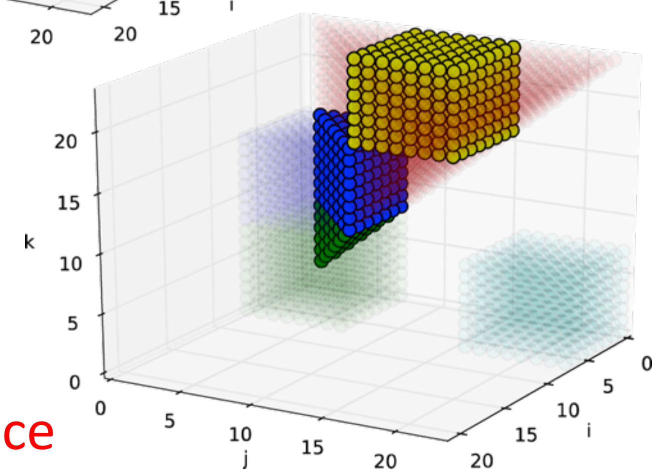
Symmetry

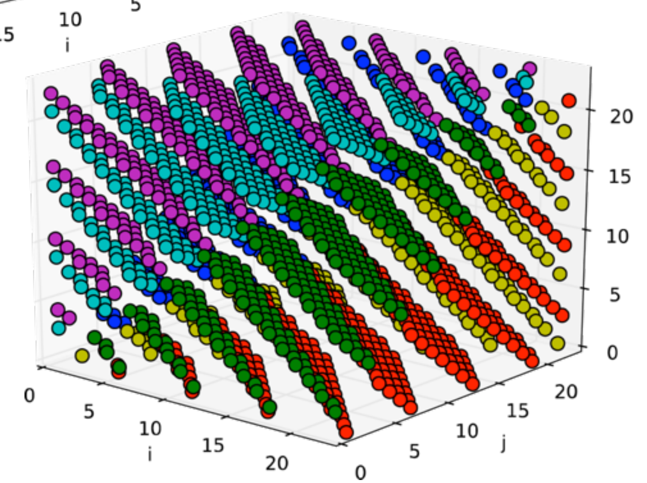
Load balance

Communication?

[Li et al. 2006]

[Li et al. 2008]





Symmetry

Load balance

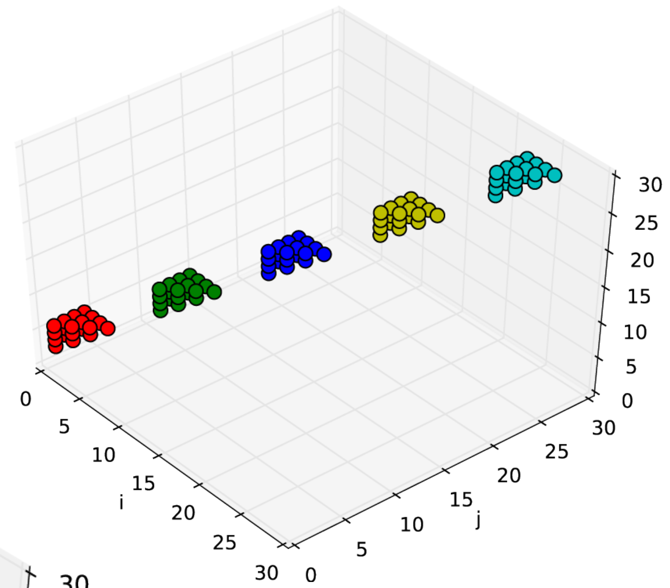
Communication?

* Colors have no special meaning -- for illustration purpose only.

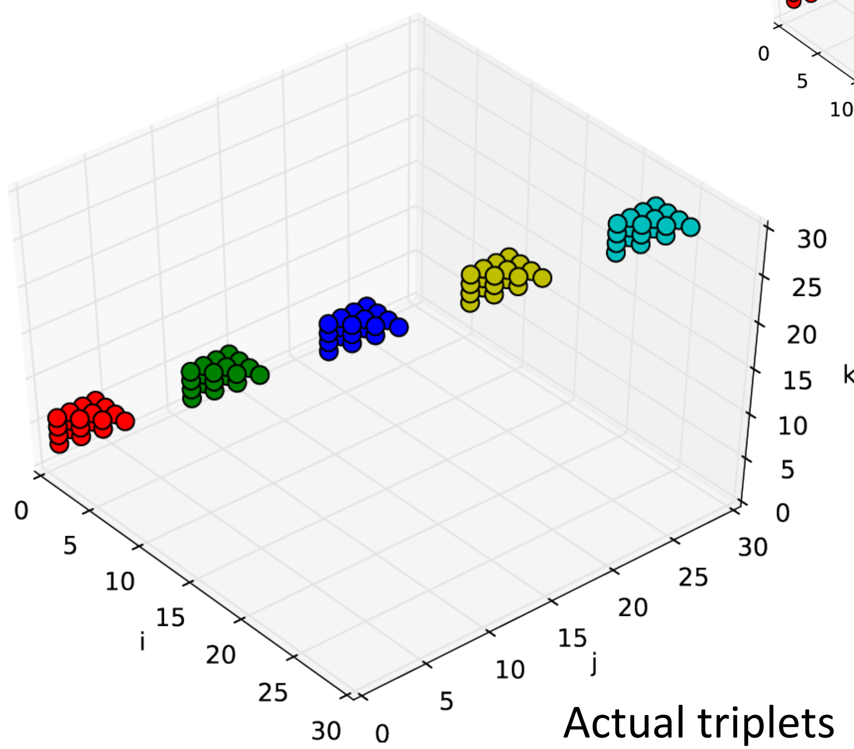
CA 3-body

[Koanantakool and Yelick 2014]

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



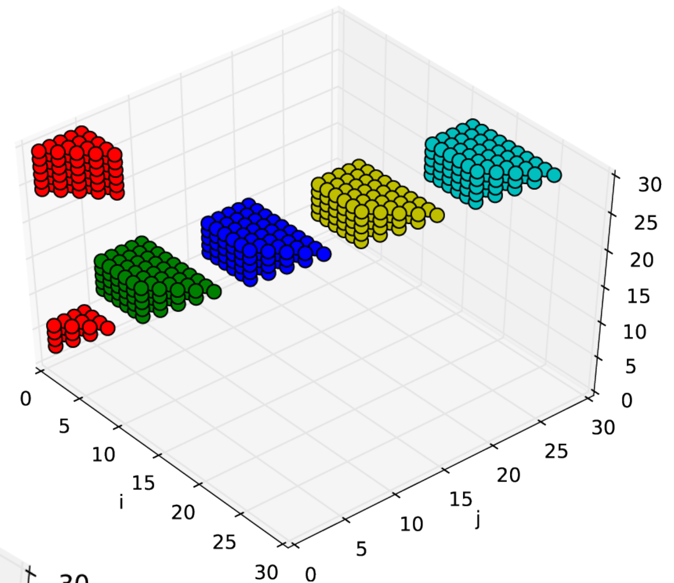
Equivalent triplets in the big tetrahedron



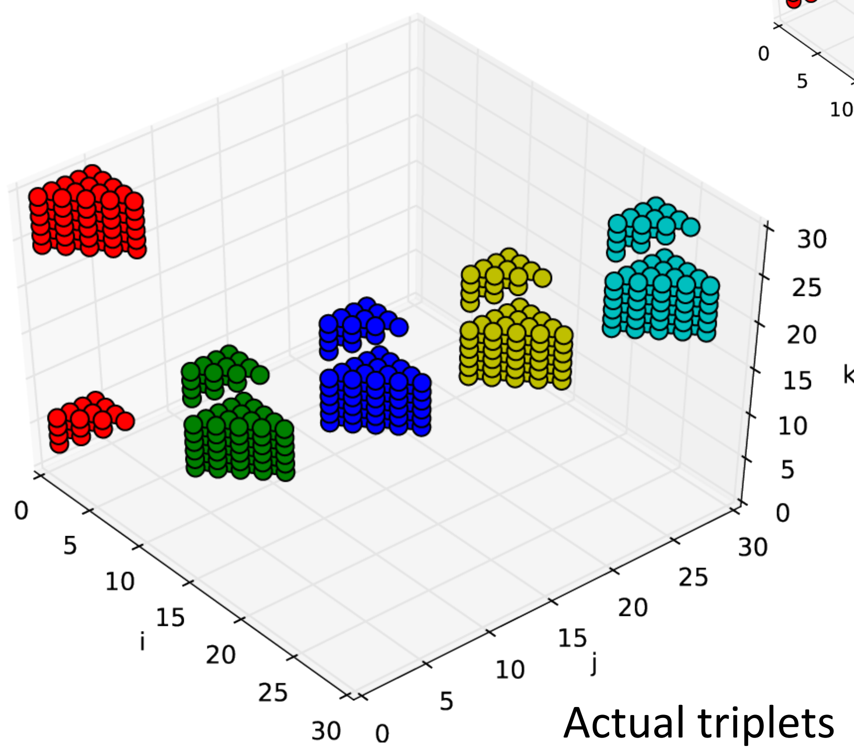
Actual triplets

CA 3-body

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



Equivalent triplets in the big tetrahedron

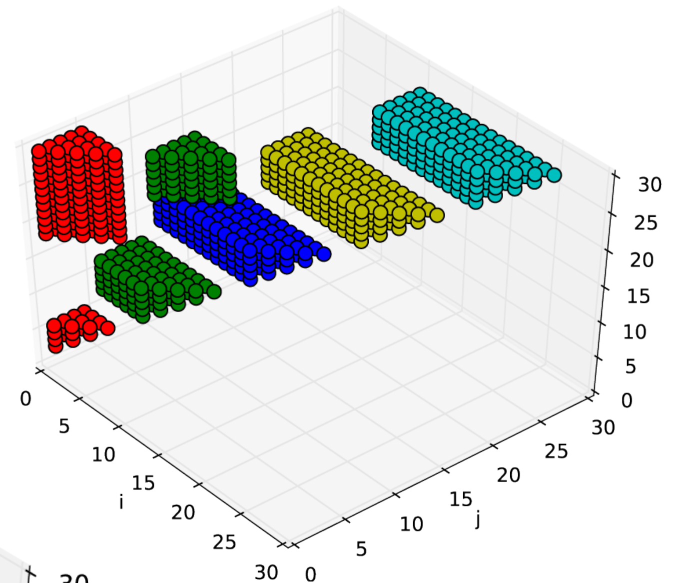


Actual triplets

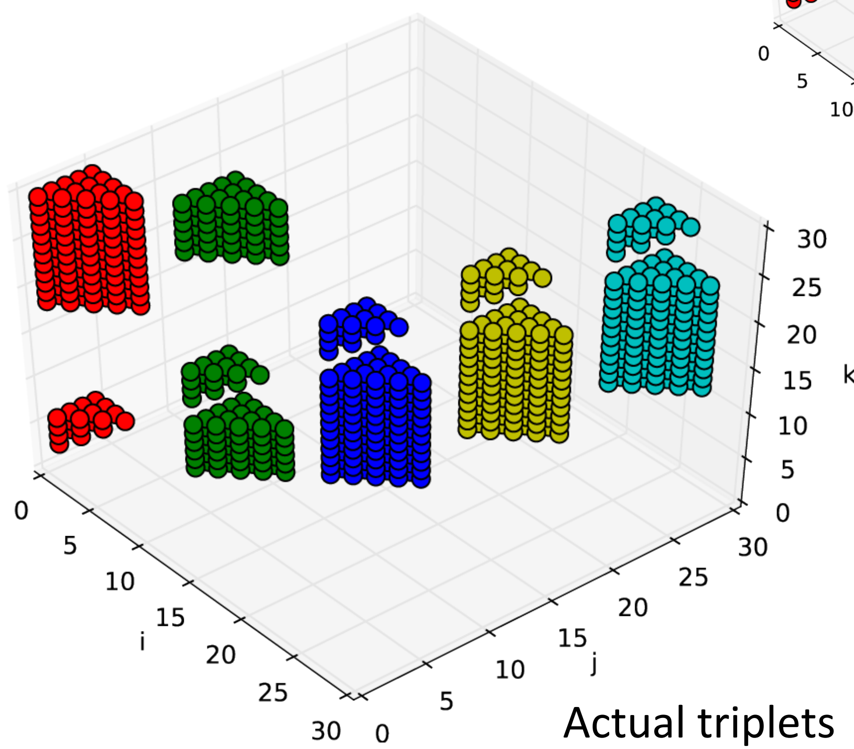
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

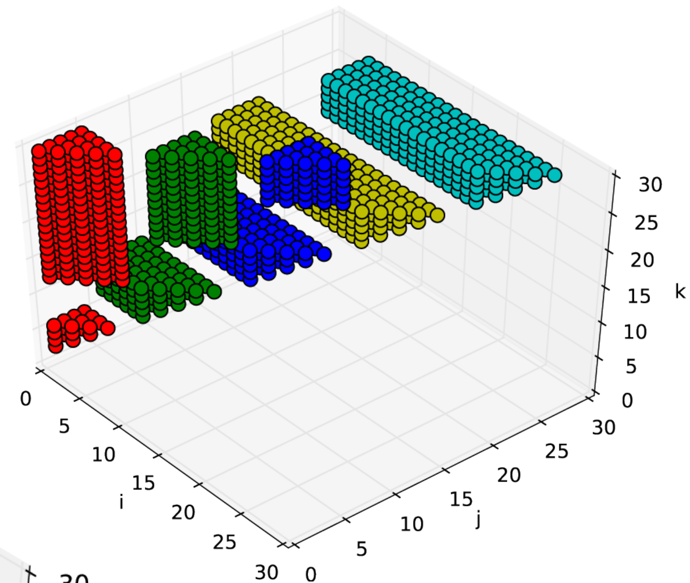


Actual triplets

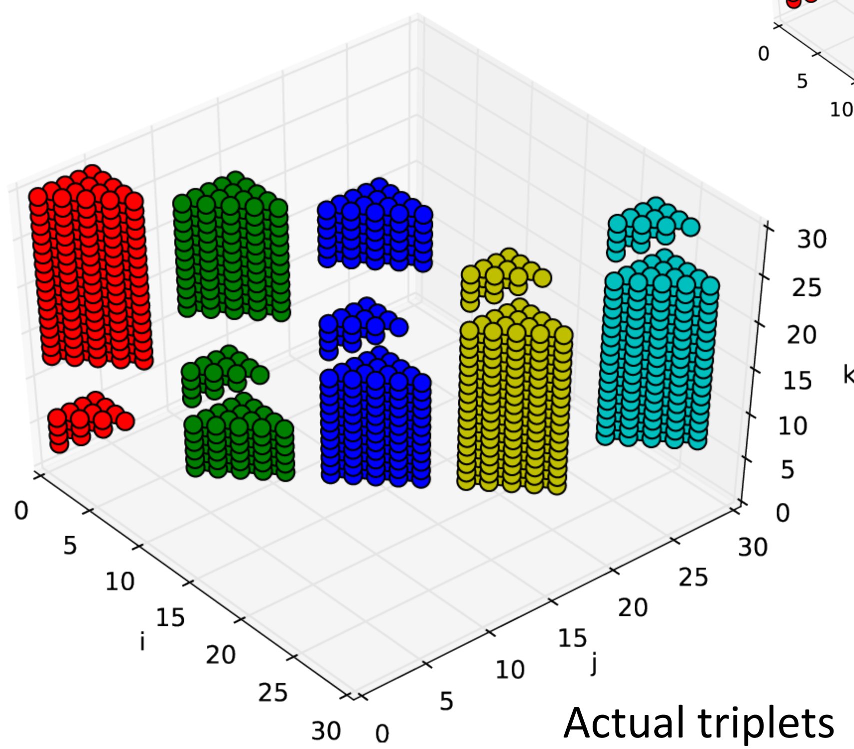
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

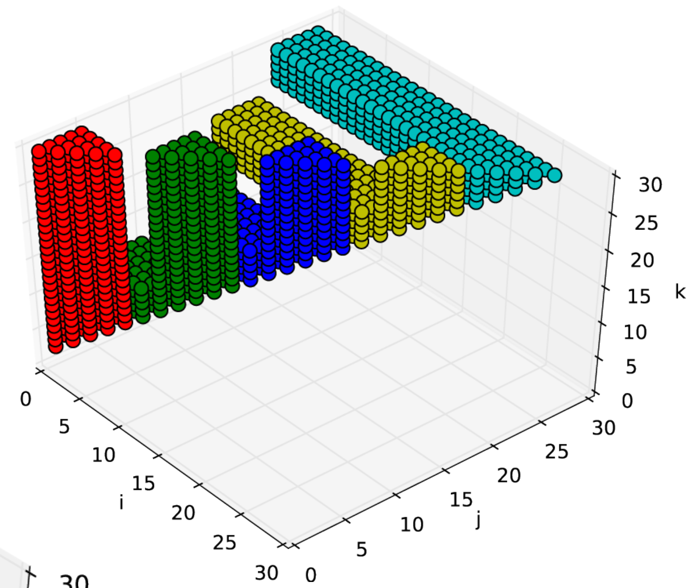


Actual triplets

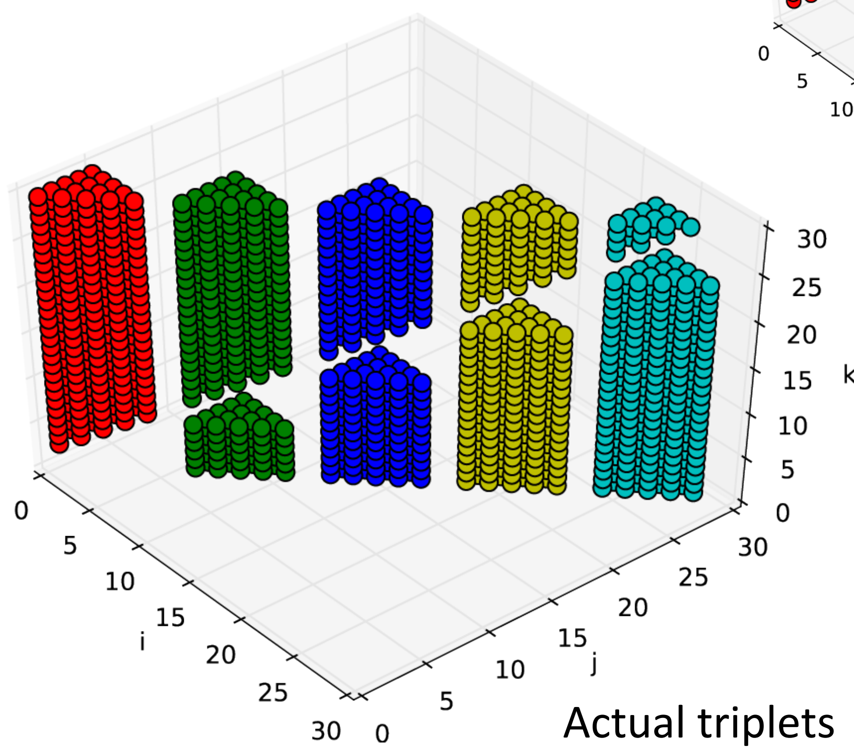
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

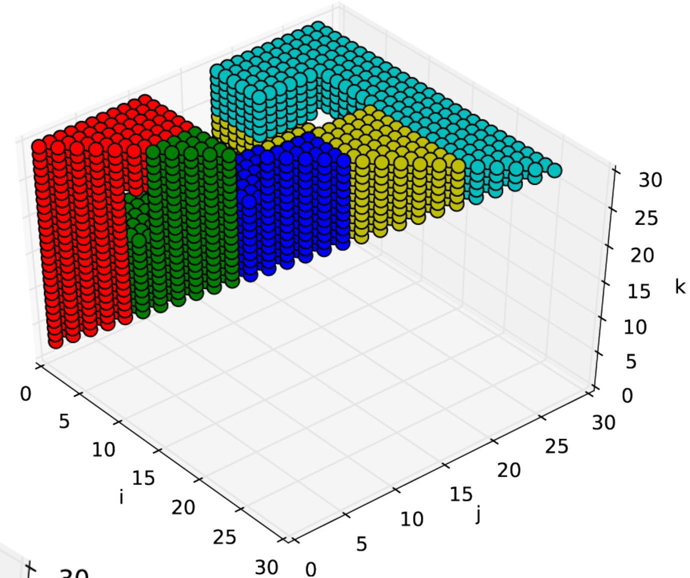


Actual triplets

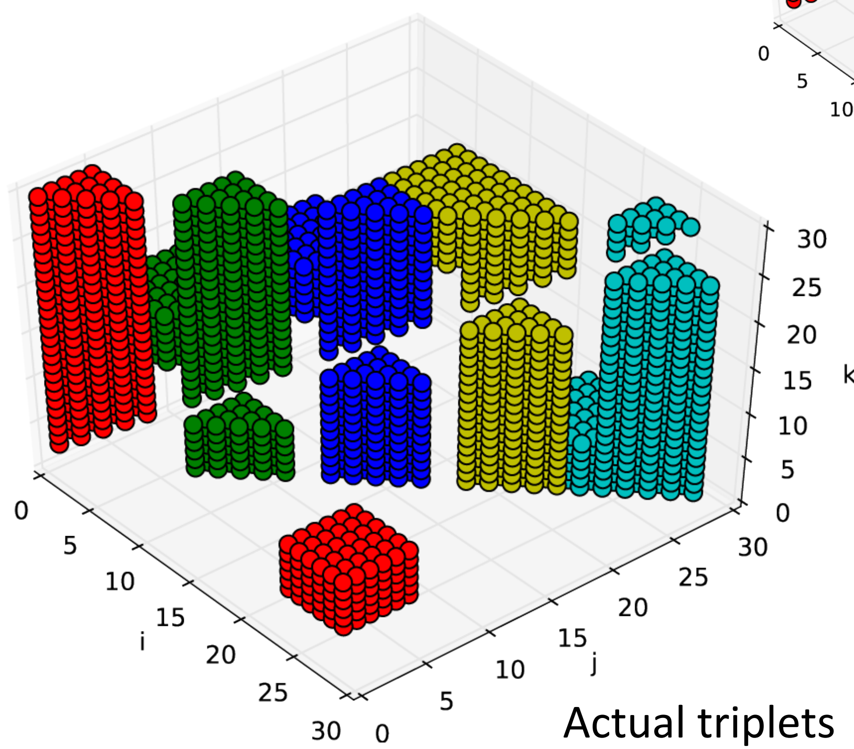
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

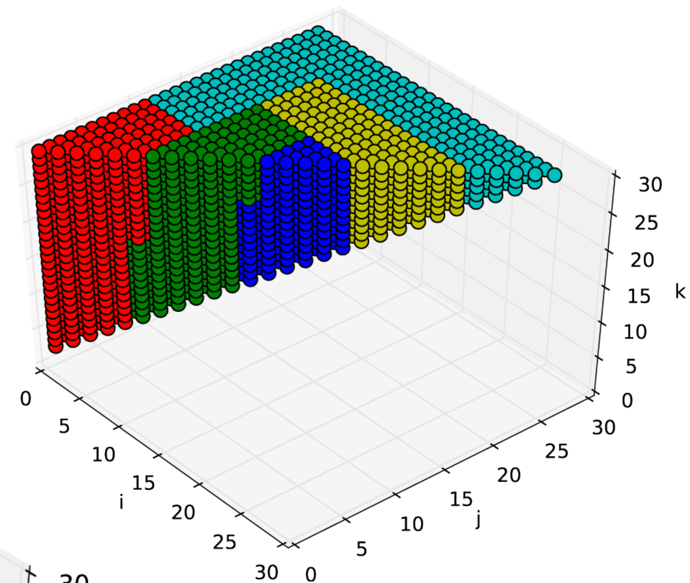


Actual triplets

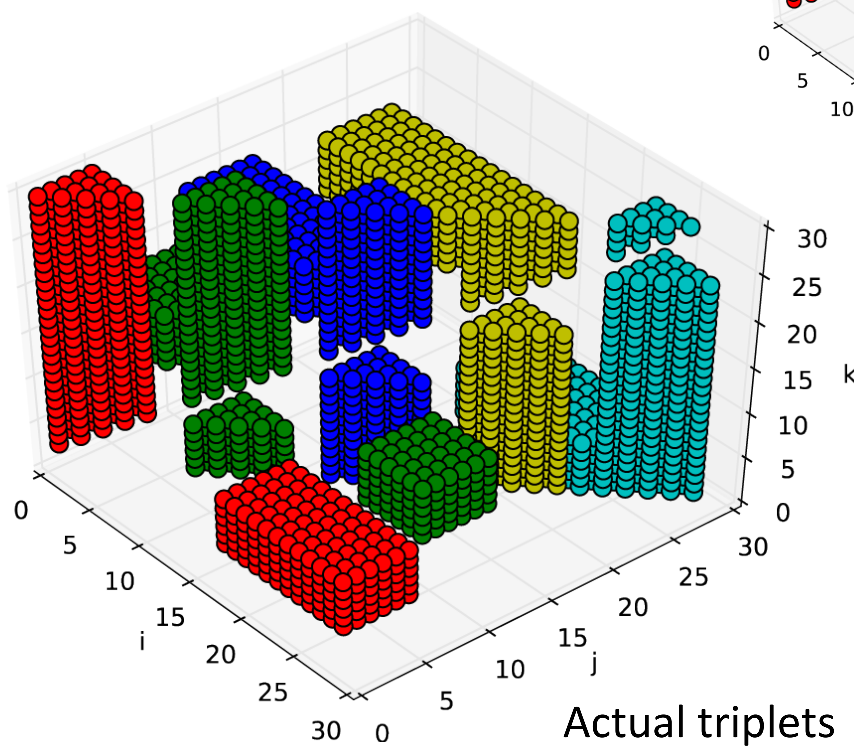
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

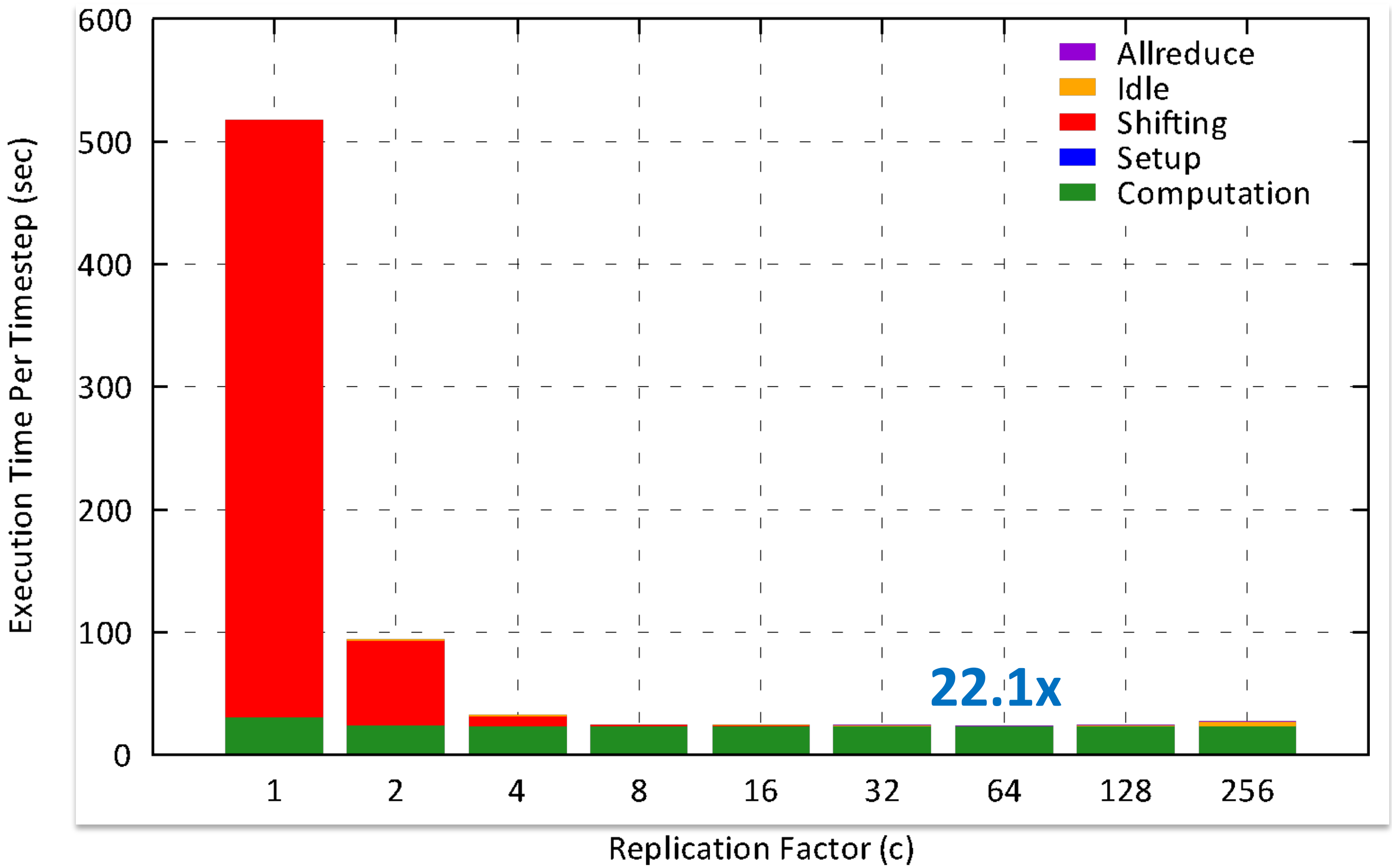


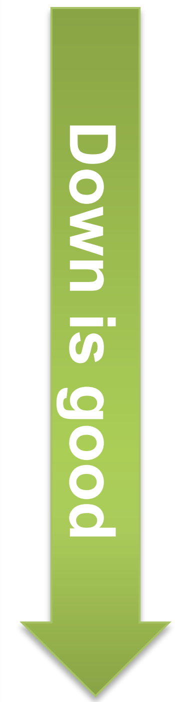
Actual triplets

Communication optimal. Replication decreases #msgs and #words by factors of c^3 and c^2.

3-Way N-Body Speedup

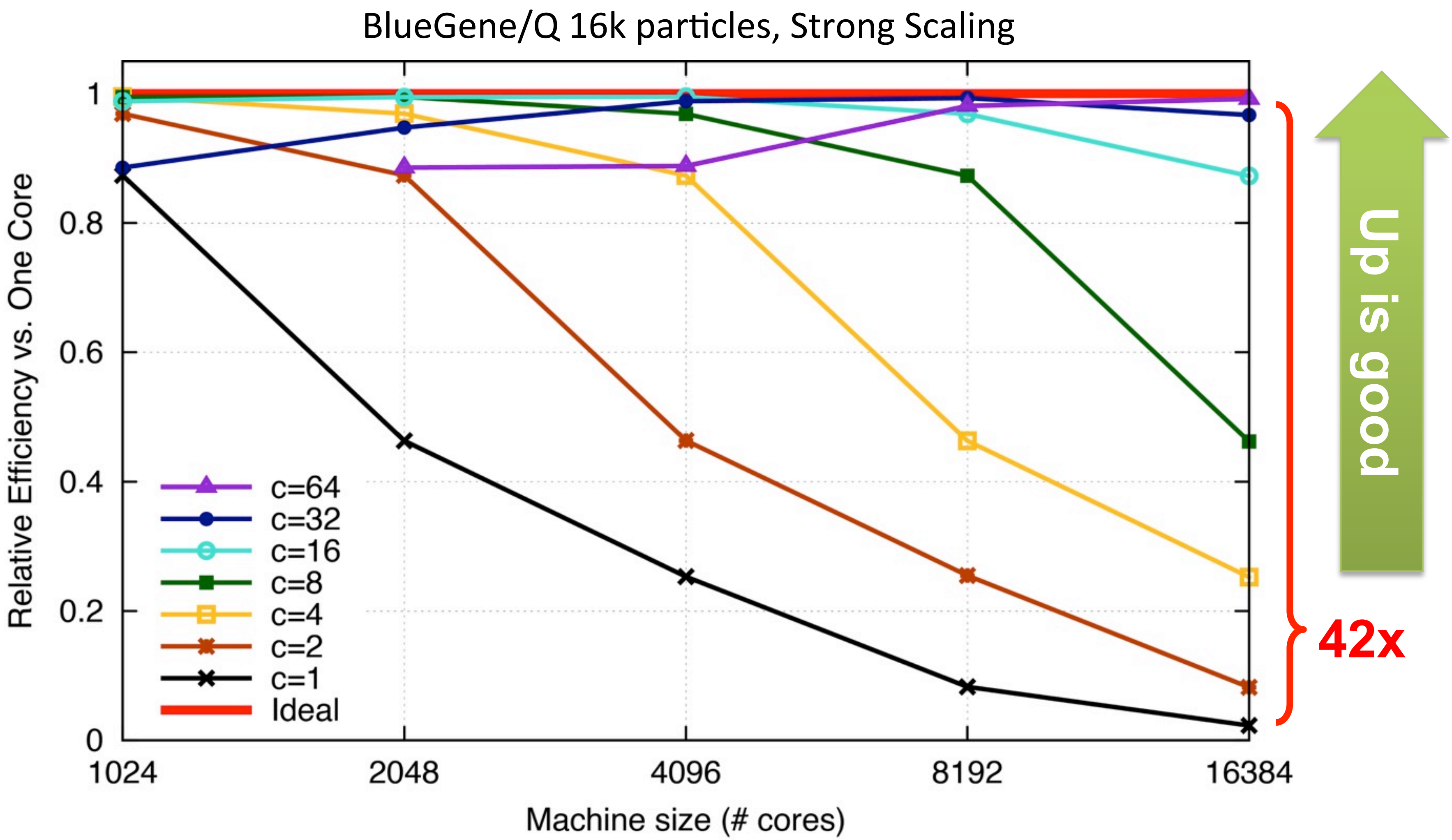
- **Cray XC30, 24k cores, 24k particles**





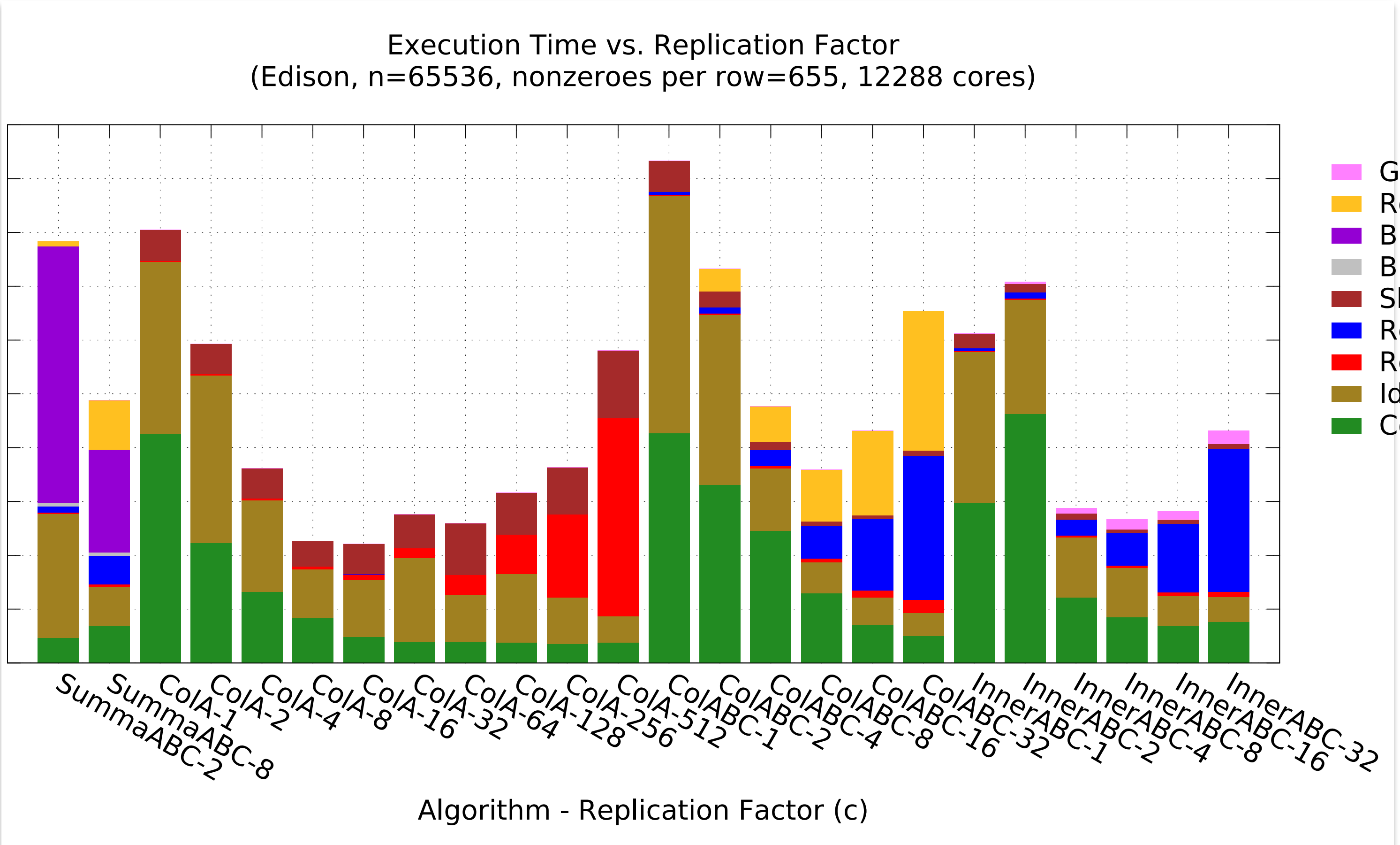
Koanantakool & Yelick

Perfect Strong Scaling



Koanantakool & Yelick

Sparse-Dense Matrix Multiply Too!

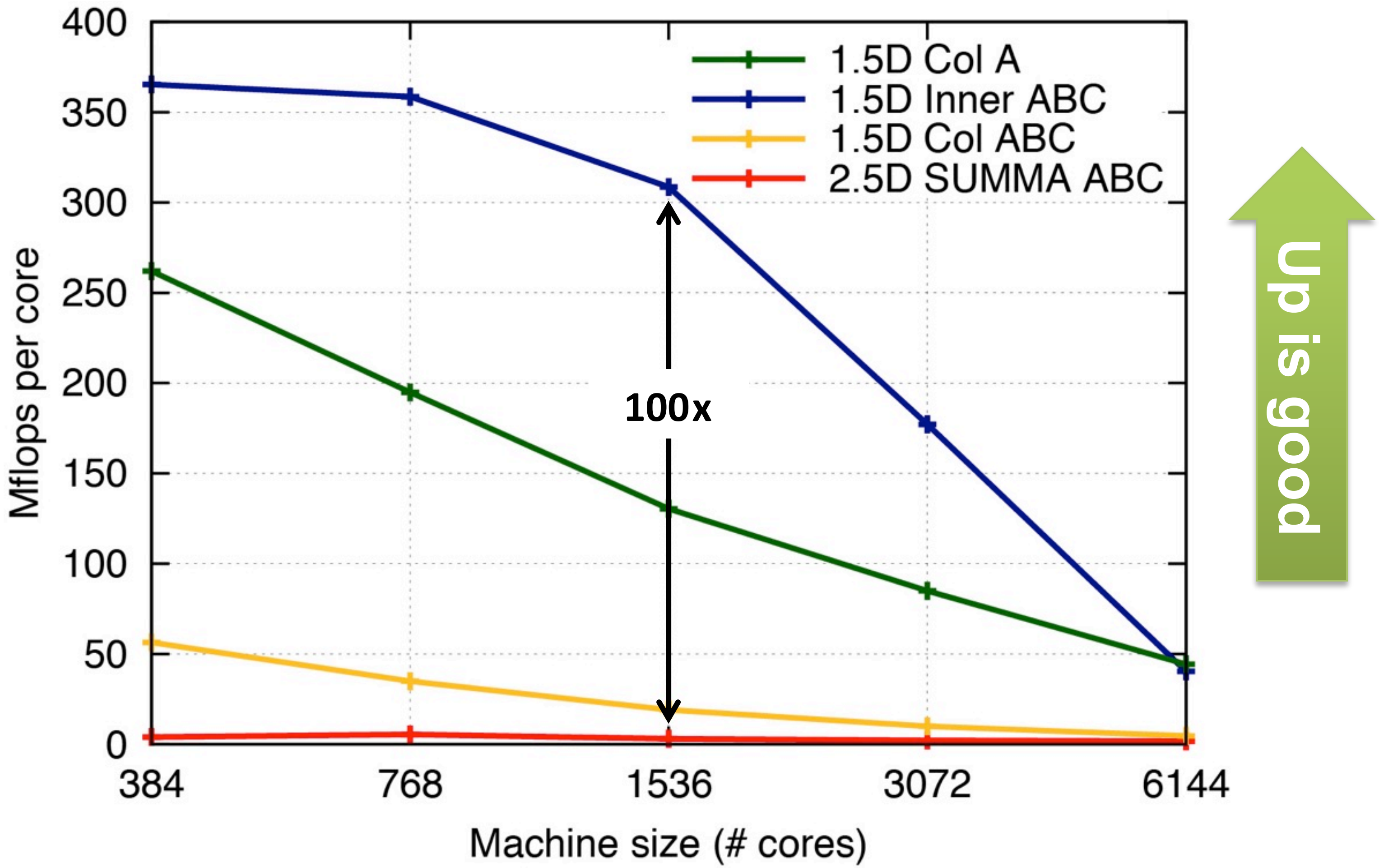


- **Variety of algorithms that divide in or 2 dimensions**

Koanantakool et al

100x Improvement

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



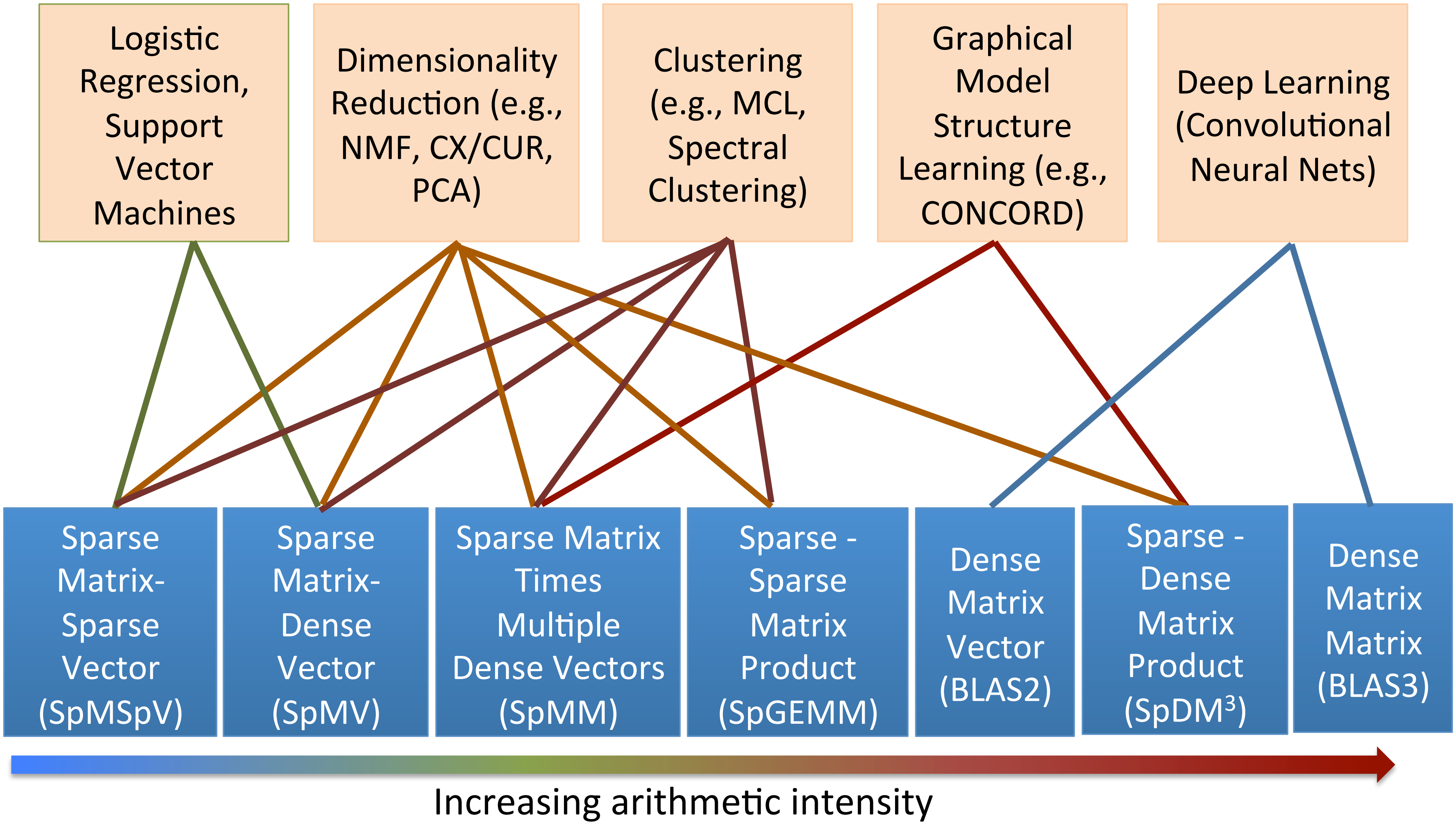
Koanantakool & Yelick

Communication-Avoiding Algorithm Sample Speedups

- **Up to 11.8x faster for direct N-body on 32K core IBM BG/P**
- **Up to 100x faster for sparse-dense matmul on Cray XC30**
- **Up to 12x faster for 2.5D matmul on 64K core IBM BG/P**
- **Up to 3x faster for tensor contractions on 2K core Cray XE/6**
- **Up to 6.2x faster for APSP on 24K core Cray CE6**
- **Up to 2.1x faster for 2.5D LU on 64K core IBM BG/P**
- **Up to 13x faster for TSQR on Tesla C2050 Fermi NVIDIA GPU**
- **Up to 6.7x faster for symeig (band A) on 10 core Intel Westmere**
- **Up to 2x faster for 2.5D Strassen on 38K core Cray XT4**
- **Up to 4.2x faster for MiniGMG benchmark bottom solver, using CA-BiCGStab (2.5x for overall solve)**

Dense Linear algebra results by Demmel, Ballard, Solomonik, Grigori, et al

Linear Algebra is important to Machine Learning too!



Aydin Buluc, Sang Oh, John Gilbert, Kathy Yelick

Overhead Can’t be Tolerated

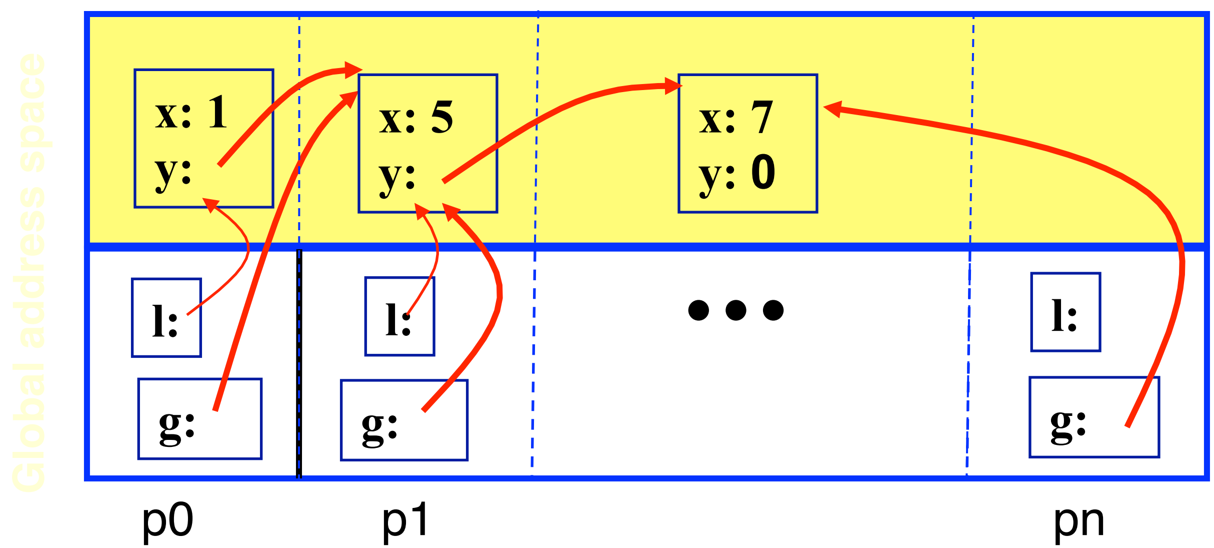
PGAS: A programming model for exascale

- ***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

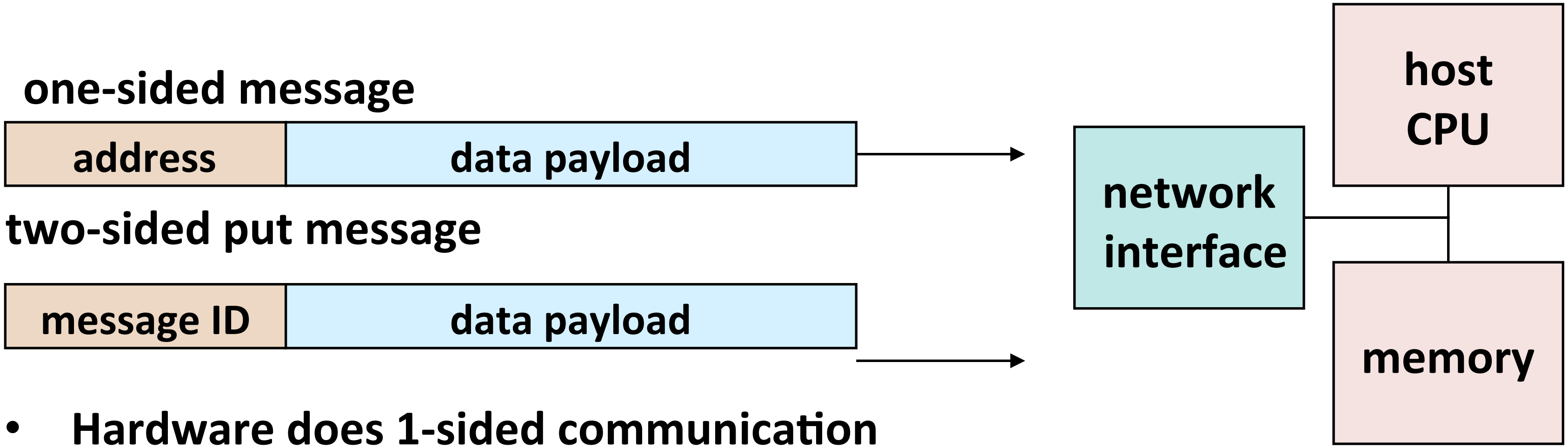
    shared int [ ] ga;   and   upc_malloc (…)



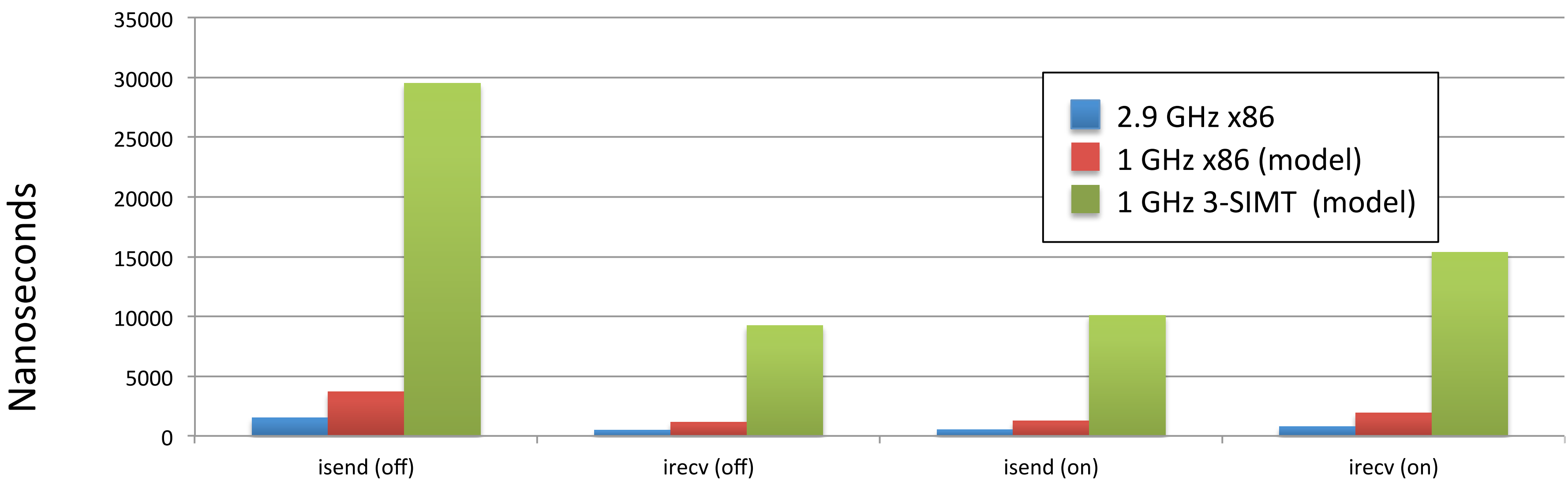
Examples:
UPC
UPC++

A programming model can influence how programmers think

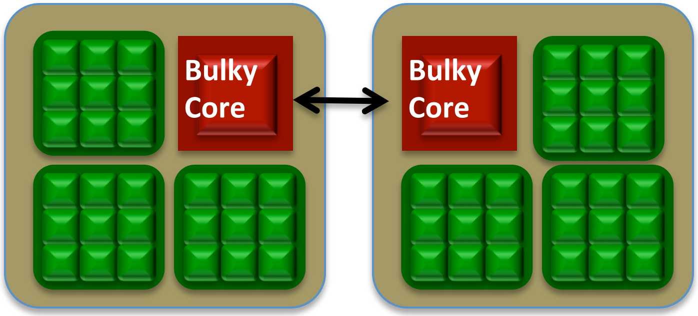
One-Sided Communication is Closer to Hardware



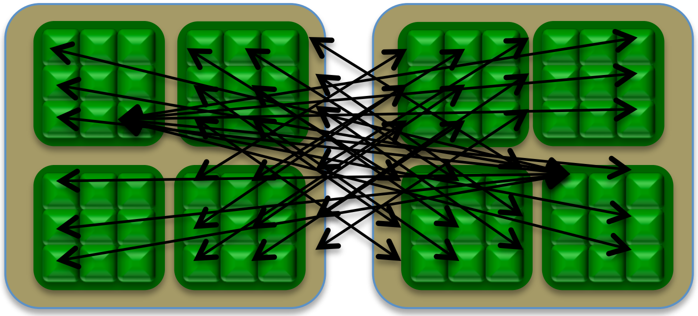
- Hardware does 1-sided communication
- Overhead for send/receive messaging is worse at exascale



MPI+OpenMP: The Problem is the "+"

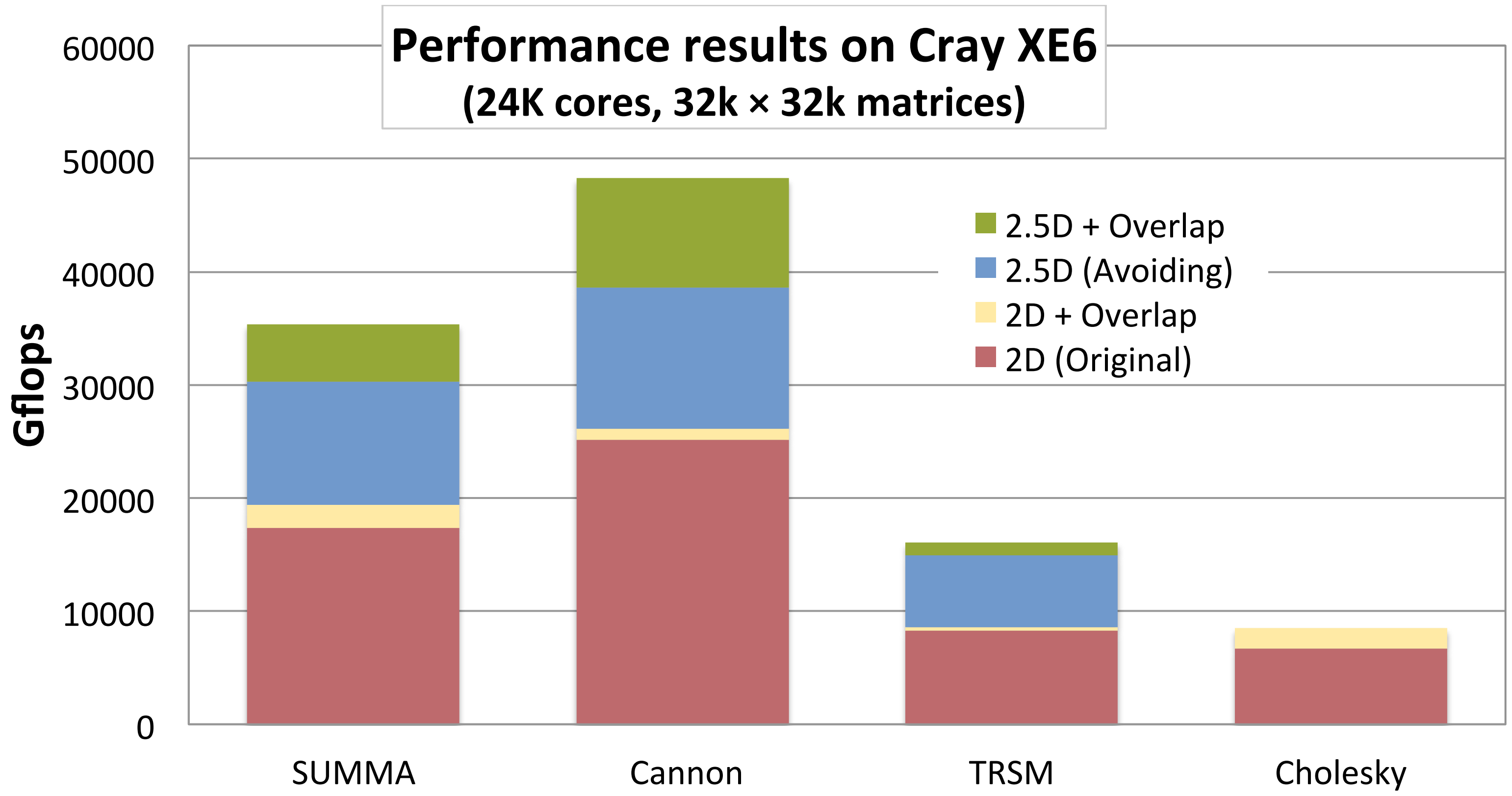


Funneled model does not scale



Addresses are better than tags

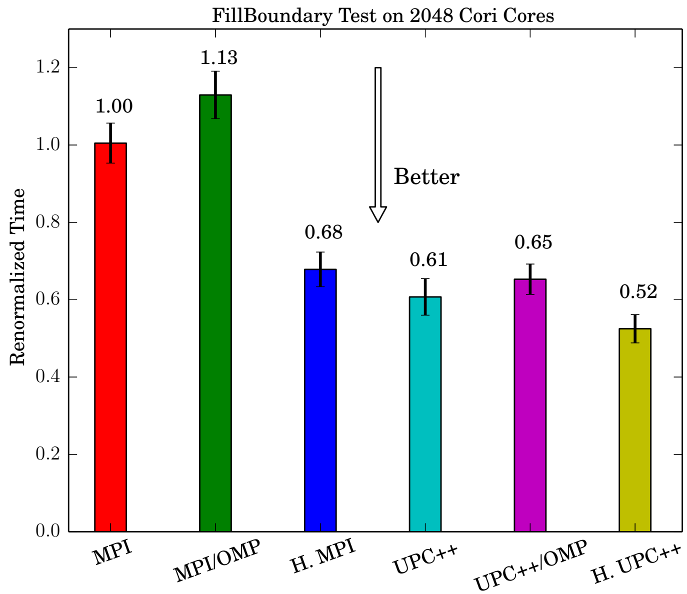
- **Avoid congestion at node interface:**
  – allow all cores to communicate
- **Coordinate on-node and between-node communication**
  – Performance is often lost between two software layers
- **Need small memory footprint per core**
  – Memory access, not heavyweight communication layer per core

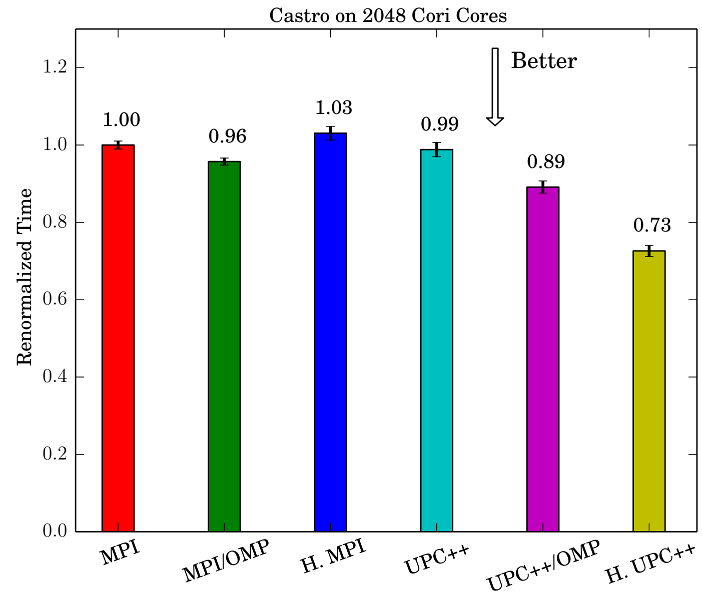


Even with communication-optimal algorithms (minimized bandwidth) there are still benefits to overlap and other things that speed up networks

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

One-sided PGAS (UPC++) in AMR





- **Adaptive Mesh Refinement (AMR) using UPC++**
  - Metadata costs make flat MPI impractical
  - Replaced communication (retained most code)
  - Hierarchical algorithms (UPC++/UPC++ or MPI/MPI best)

Weiquin Zhang, Y. Zheng

Avoid Unnecessary Synchronization

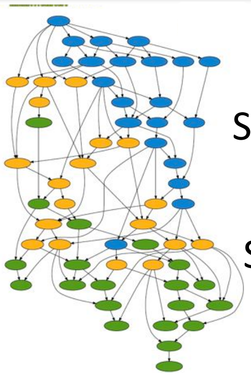
Sources of Unnecessary Synchronization

Loop Parallelism

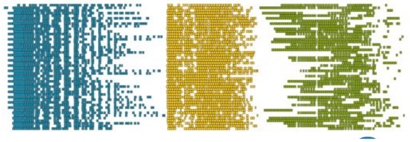
```
!$OMP PARALLEL DO
   DO I=2,N
     B(I) = (A(I) + A(I-1)) / 2.0
   ENDDO
!$OMP END PARALLEL DO
```

"Simple" OpenMP parallelism implicitly synchronized between loops

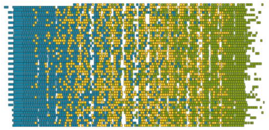
Abstraction



Bulk Synchronous



Less Synchronous



LAPACK: removing barriers ~2x faster (PLASMA)

Libraries

Analysis	% barriers	Speedup
Auto	42%	**13%**
Guided	63%	**14%**

NWChem: most of barriers are unnecessary (Corvette)

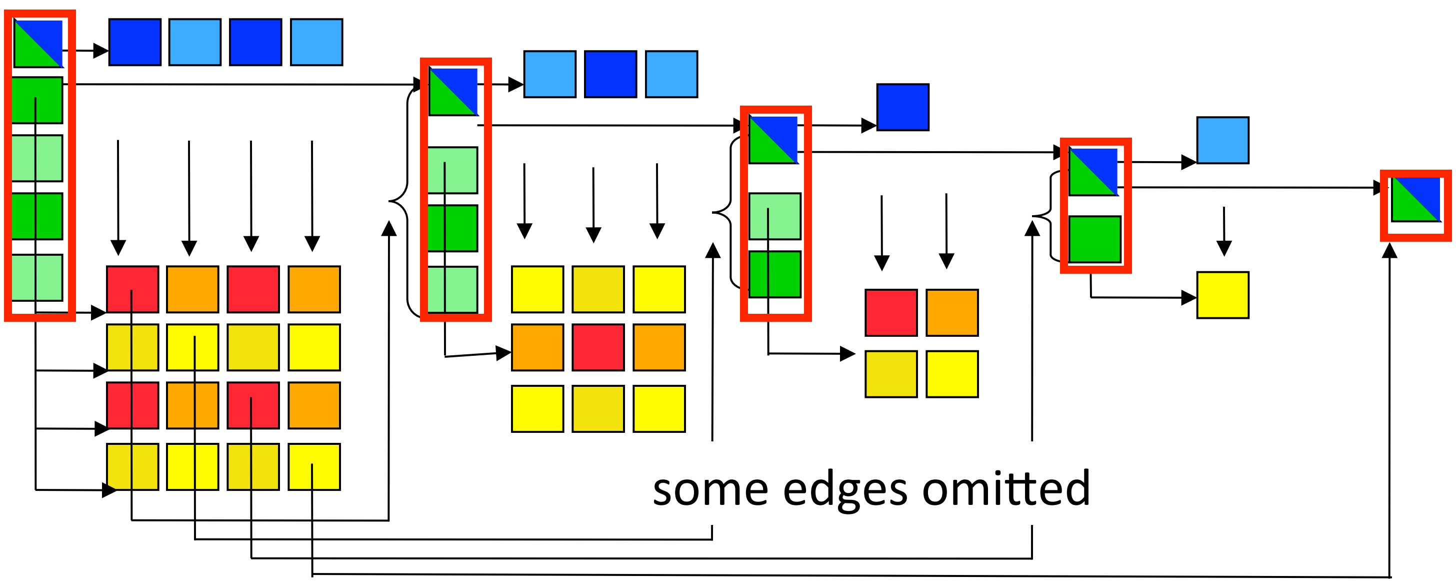
Accelerator Offload

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc&    streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc&    streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc&    streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc&    streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc&    streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc&    streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc&    streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc&    streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc&    streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc&    streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc&    send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
```

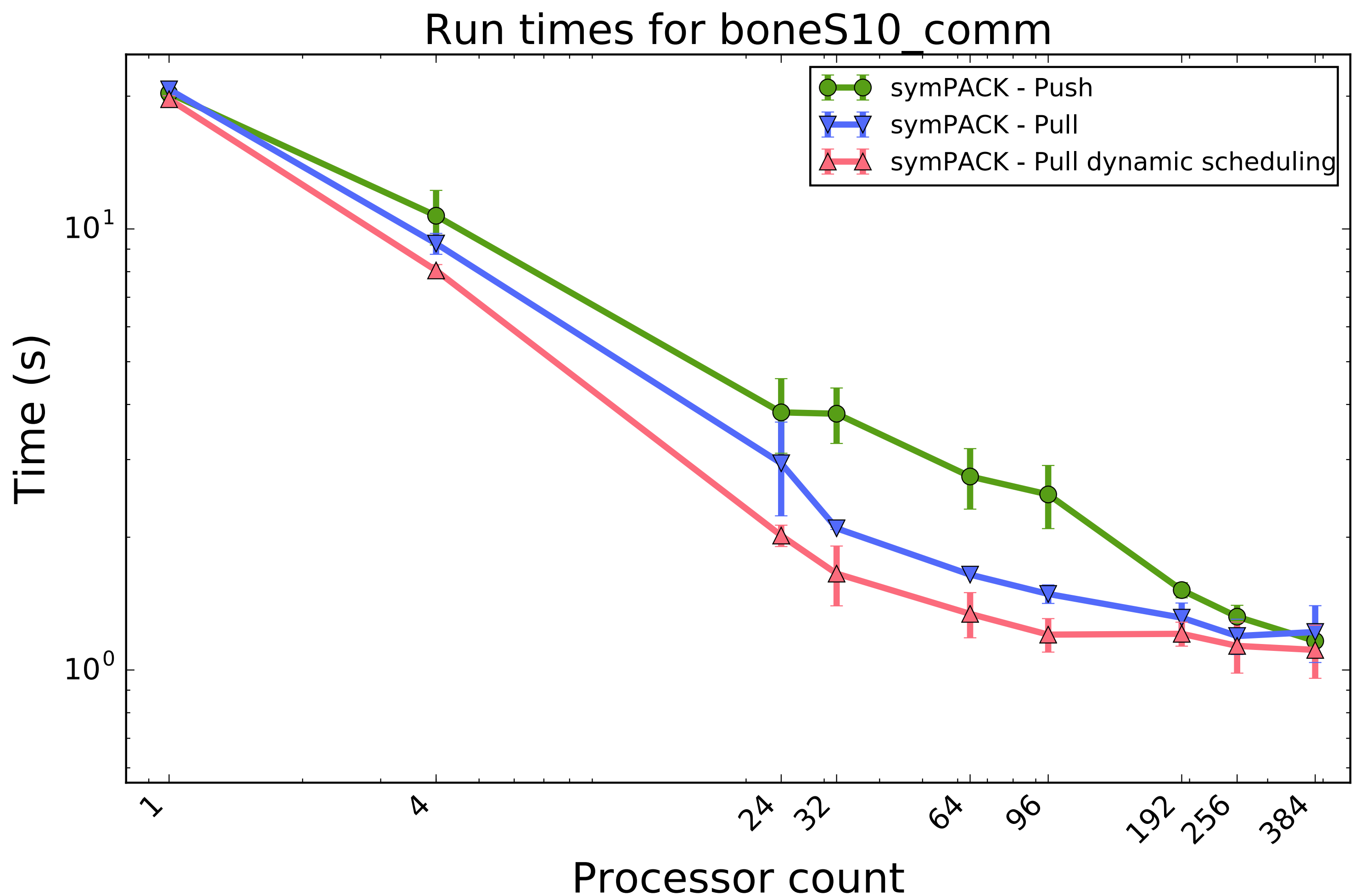
The transfer between host and GPU can be slow and cumbersome, and may (if not careful) get synchronized

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



Asynchronous Sparse Cholesky in UPC++

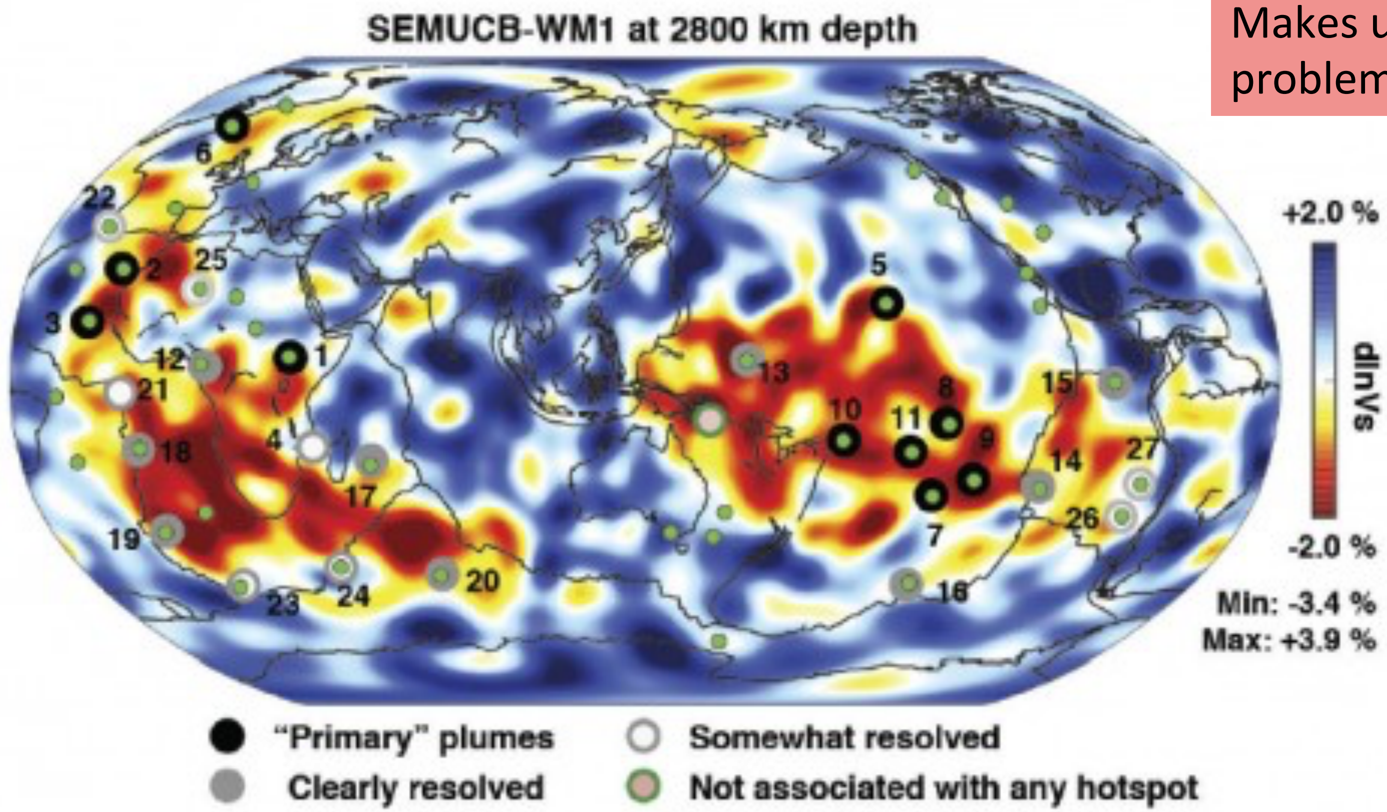


- **Fan-both algorithm by Jacquelin & Ng, in UPC++**

Whole-Mantle Seismic Model Using UPC++

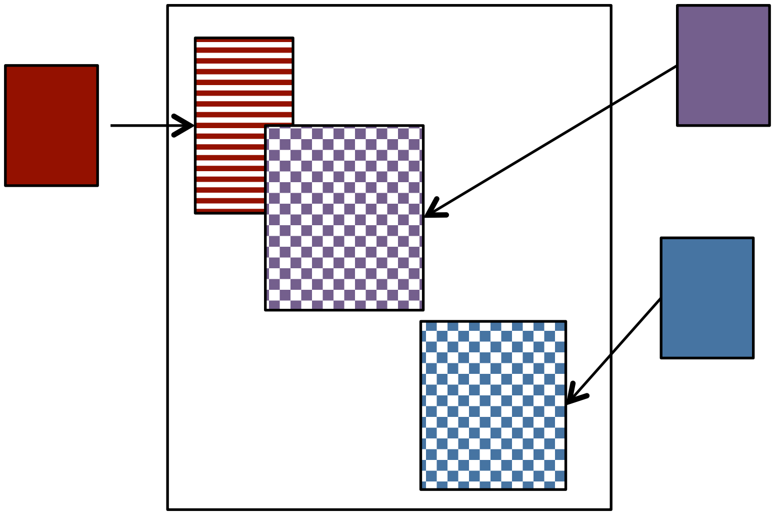
- *First-ever whole-mantle seismic model from numerical waveform tomography*
- *Finding: Most* volcanic hotspots are linked to two spots on the boundary between the metal core and rocky mantle 1,800 miles below Earth's surface.

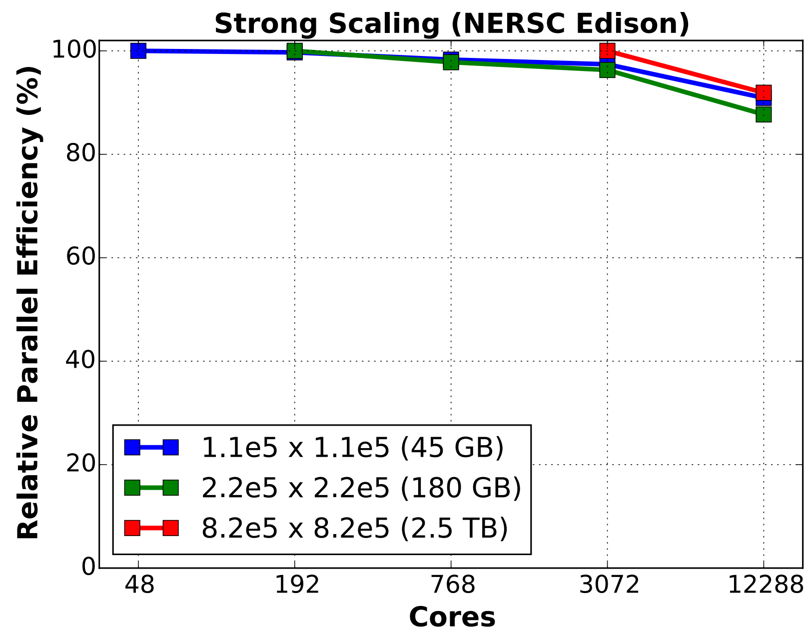
Makes unsolvable problems solvable!



Scott French, Barbara Romanowicz, *"Broad plumes rooted at the base of the Earth's mantle beneath major hotspots"*, **Nature, 2015**

Data Fusion for Observation with Simulation





- **Unaligned data from observation**
- **One-sided strided updates**
- **Could MPI-3.0 one-sided do this? Yes, but not well so far**

Scott French, Y. Zheng, B. Romanowicz, K. Yelick

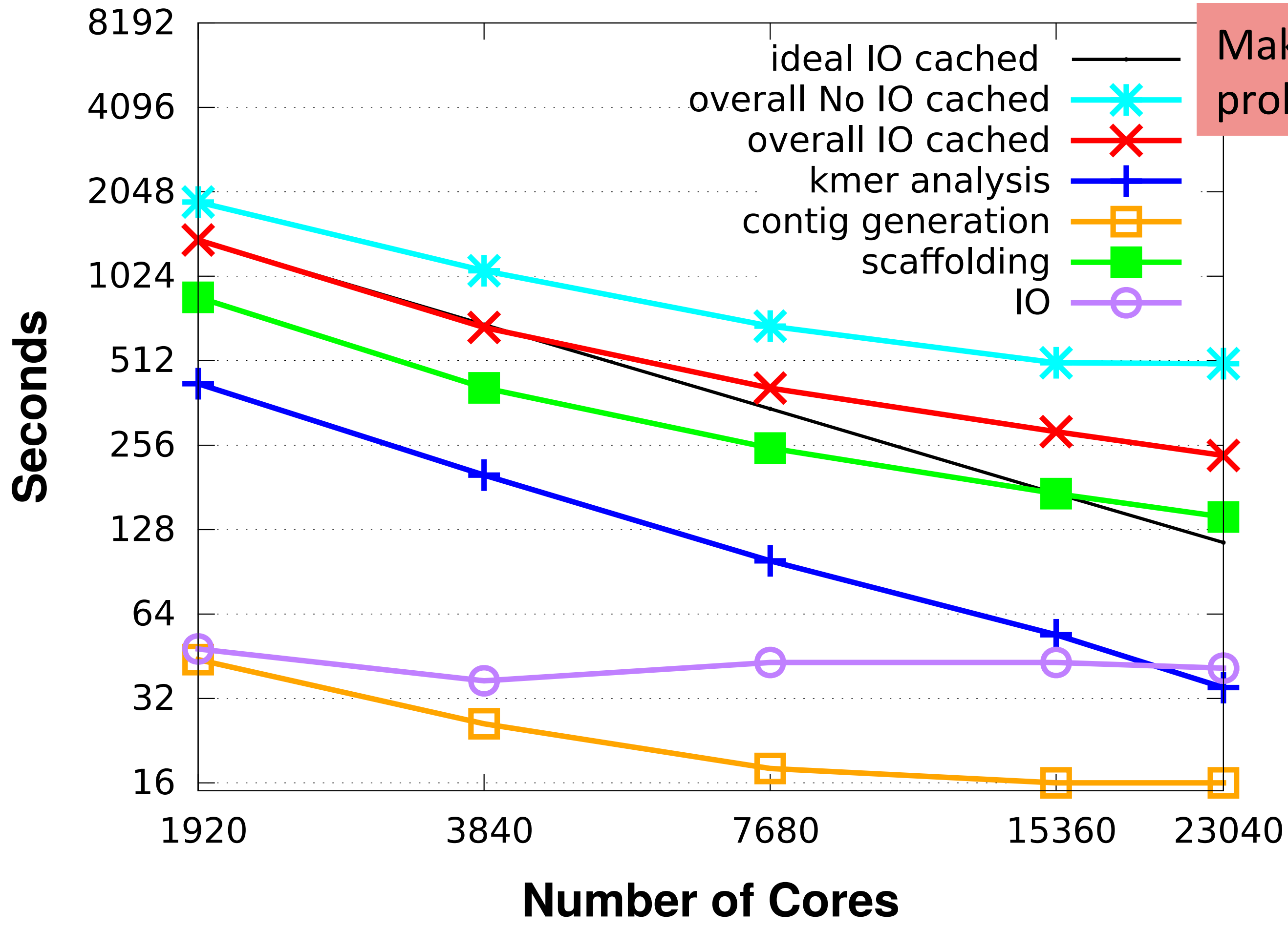
Unstructured, Graph-based, Data analytics problem: *De novo* Genome Assembly

- **DNA sequence consists of 4 bases: A/C/G/T**

- **Read: short fragment of DNA sequence that can be read by a DNA sequencing technology – can't read whole DNA at once.**

- **De novo genome assembly: Reconstruct an unknown genome from a collection of short reads.**
  - Constructing a jigsaw puzzle without having the picture on the box

Strong scaling (human genome) on Cray XC30



Makes unsolvable problems solvable!

- Complete assembly of human genome in **4 minutes using 23K cores.**
- **700x speedup over** original Meraculous (took **2,880 minutes** on large shared memory with some Perl code); Some problems (wheat, squid, only run on HipMer version)

Summary

- **Communication is the most expensive thing computers do**
  - Memory
  - Network
- **Compilers**
  - Domain specific languages simplify program analysis
  - Autotuning helps identify optimizations
- **PGAS and one-sided communication can help lower costs**
  - Better match to RDMA hardware
  - Also supports applications with irregular accesses
- **Algorithms can avoid communication**
  - Both data volume (bandwidth) and number of messages (latency)
  - Probably optimal and faster (more scalable) in practice

Thank you!