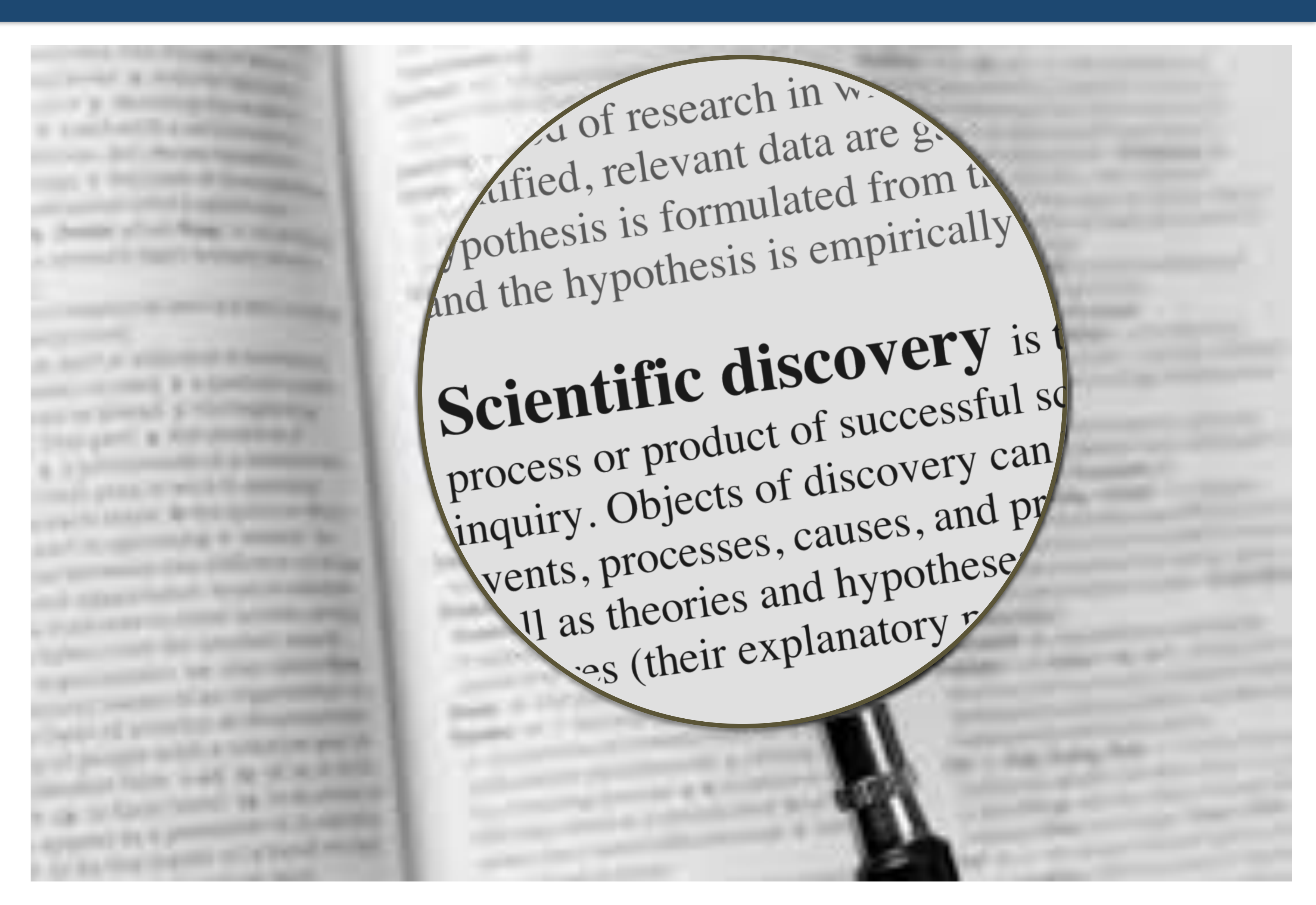
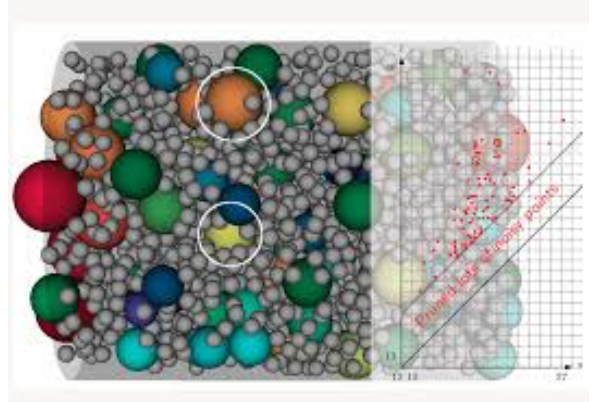
The End Game for Moore's Law

Kathy Yelick
Lawrence Berkeley National Laboratory
and UC Berkeley

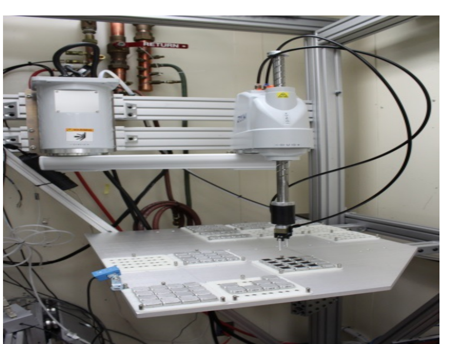
A Focus on Science



The changing nature of scientific discovery

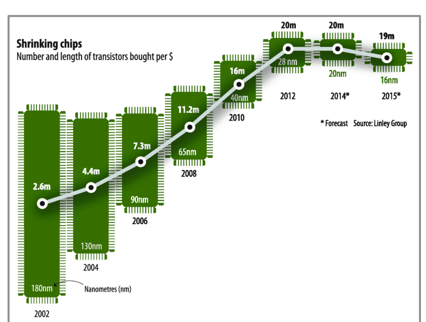


New methods for feature identification and data discovery



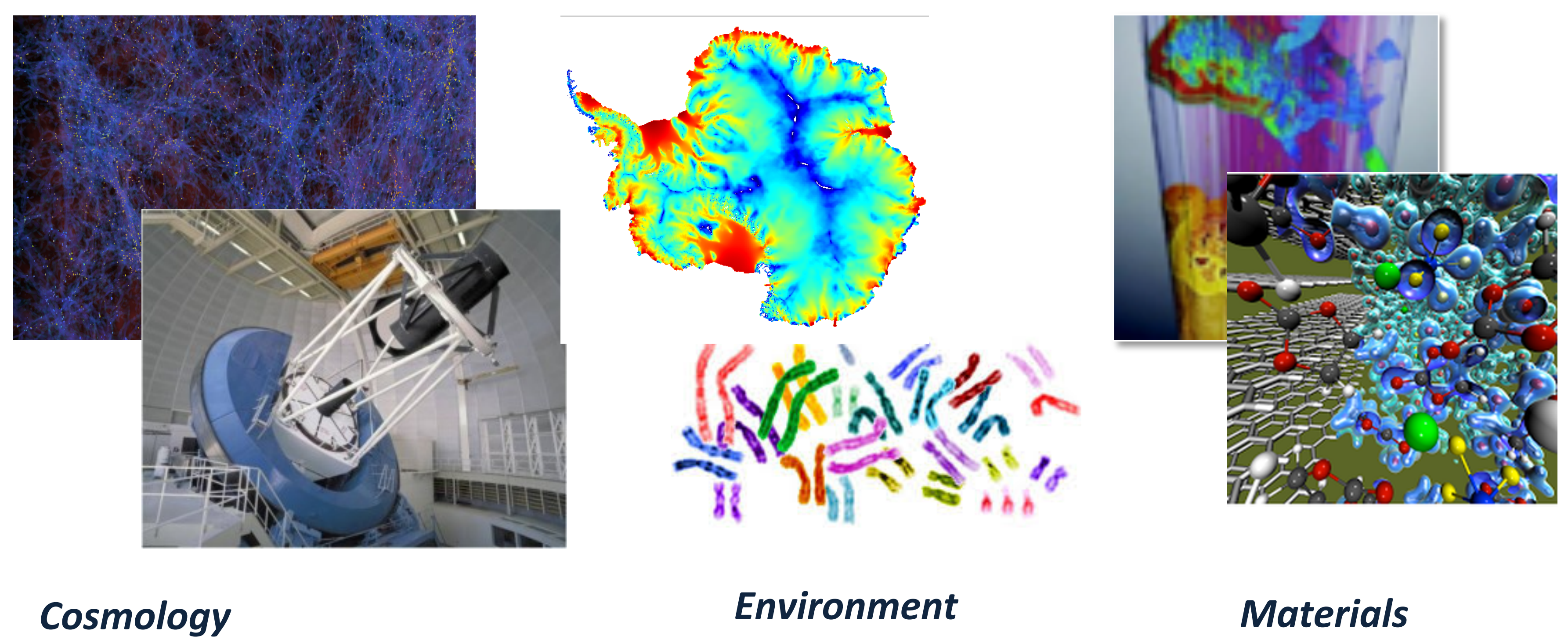
Automation, robotics and new input devices

Science at the boundary of simulation and observation



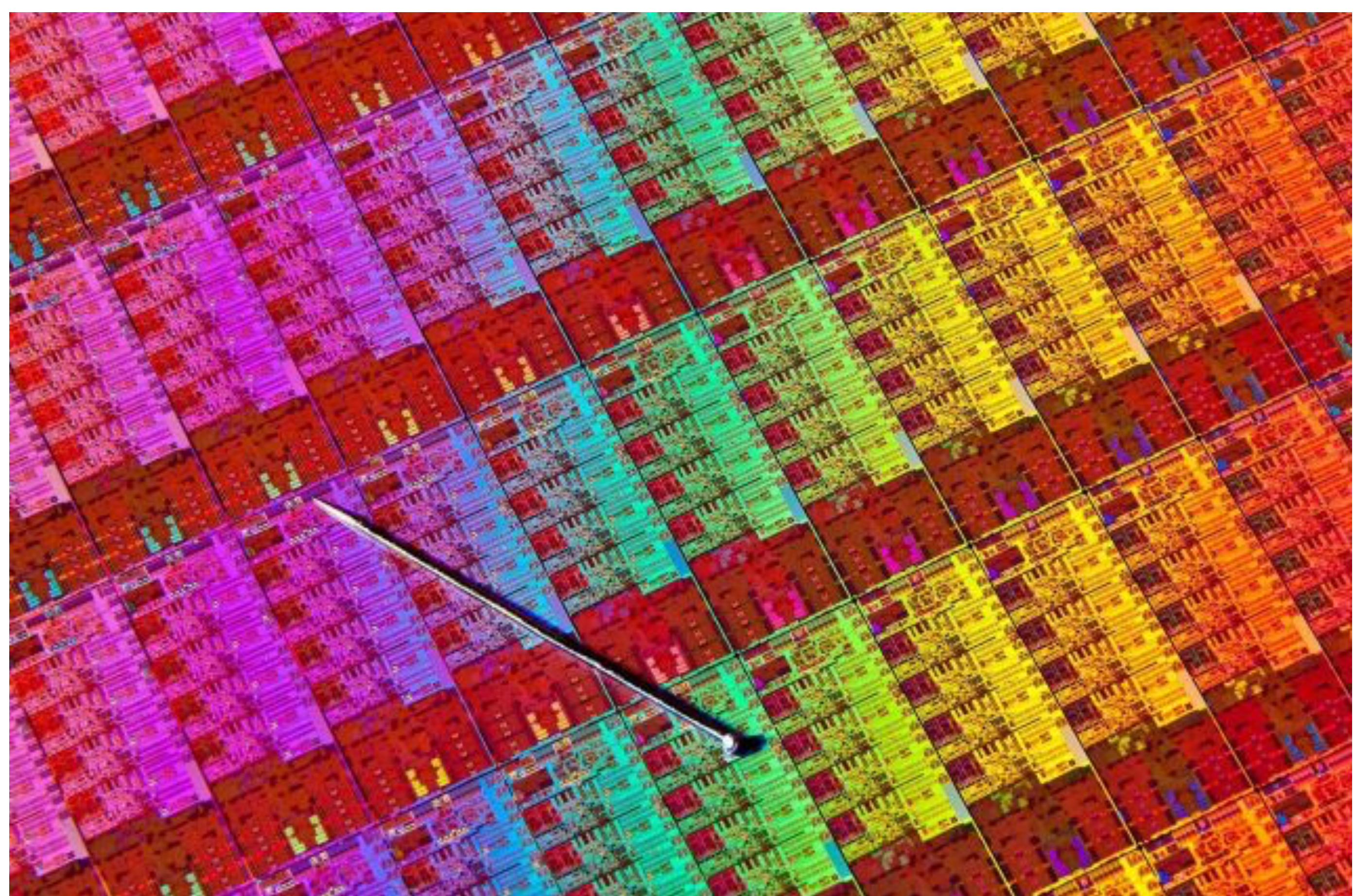
More computing for more complex science questions

Science at the Boundary of Simulation and Observation



In many areas, there are opportunities to combine simulation and observation for new discoveries.

End of Transistor Density Scaling



- **ITRS now sets the end of transistor shrinking to the year 2021**

Technology Scaling Trends

The many ends of "Moore's" Law

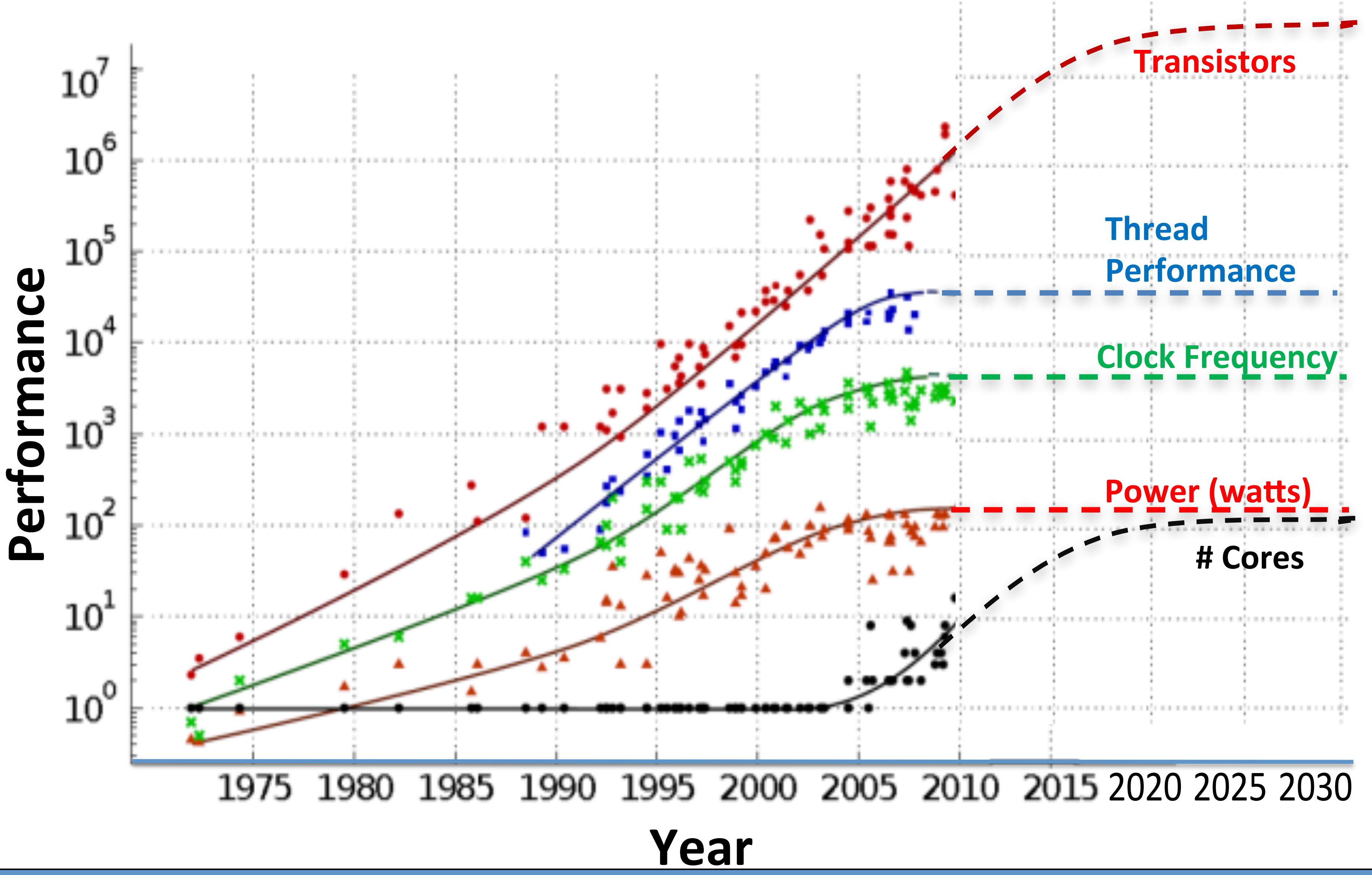


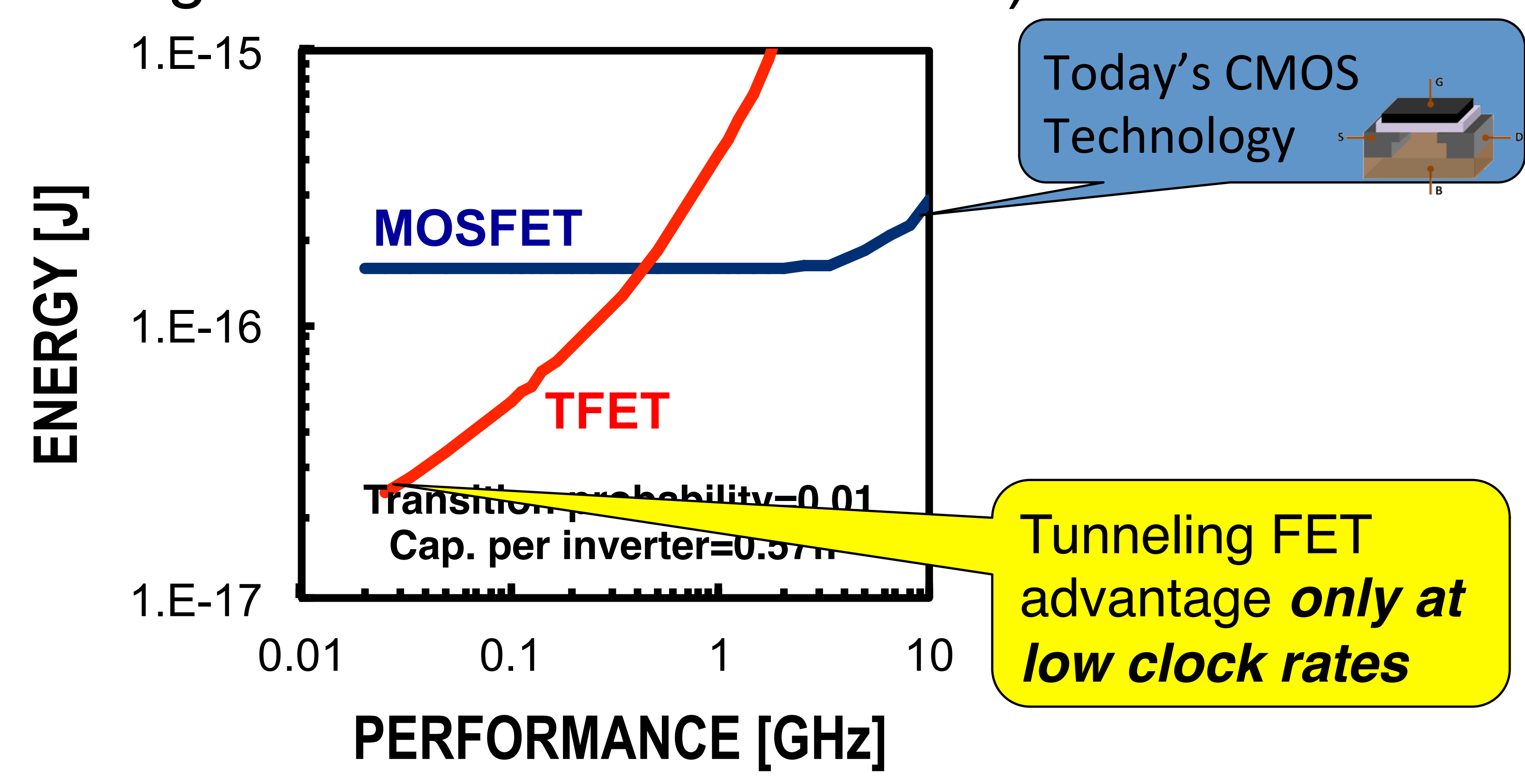
Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

Energy Optimization: Alternatives to Conventional MOS

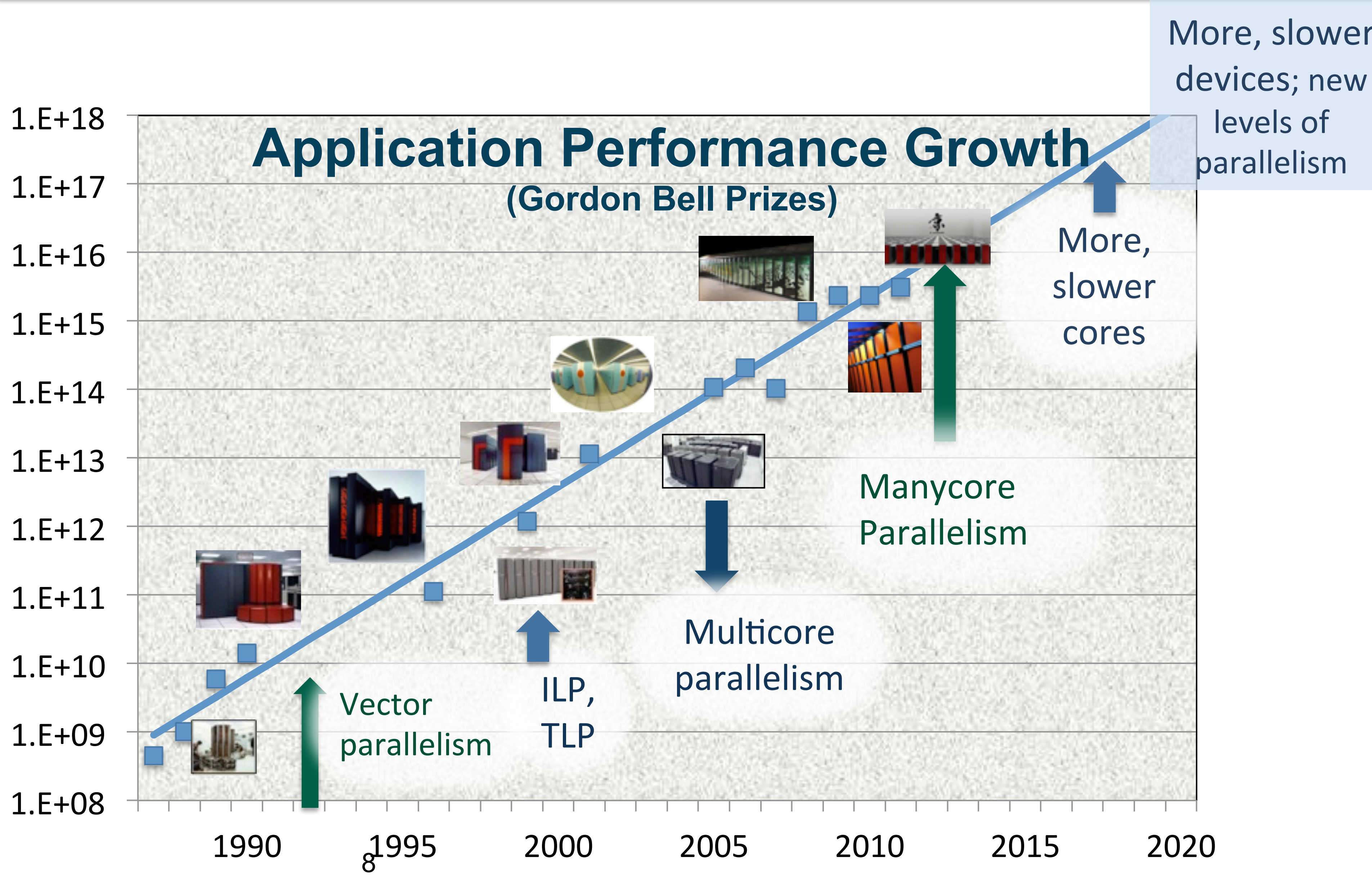
(all require lower clock rate, and much more parallelism)

Energy-Performance Comparison

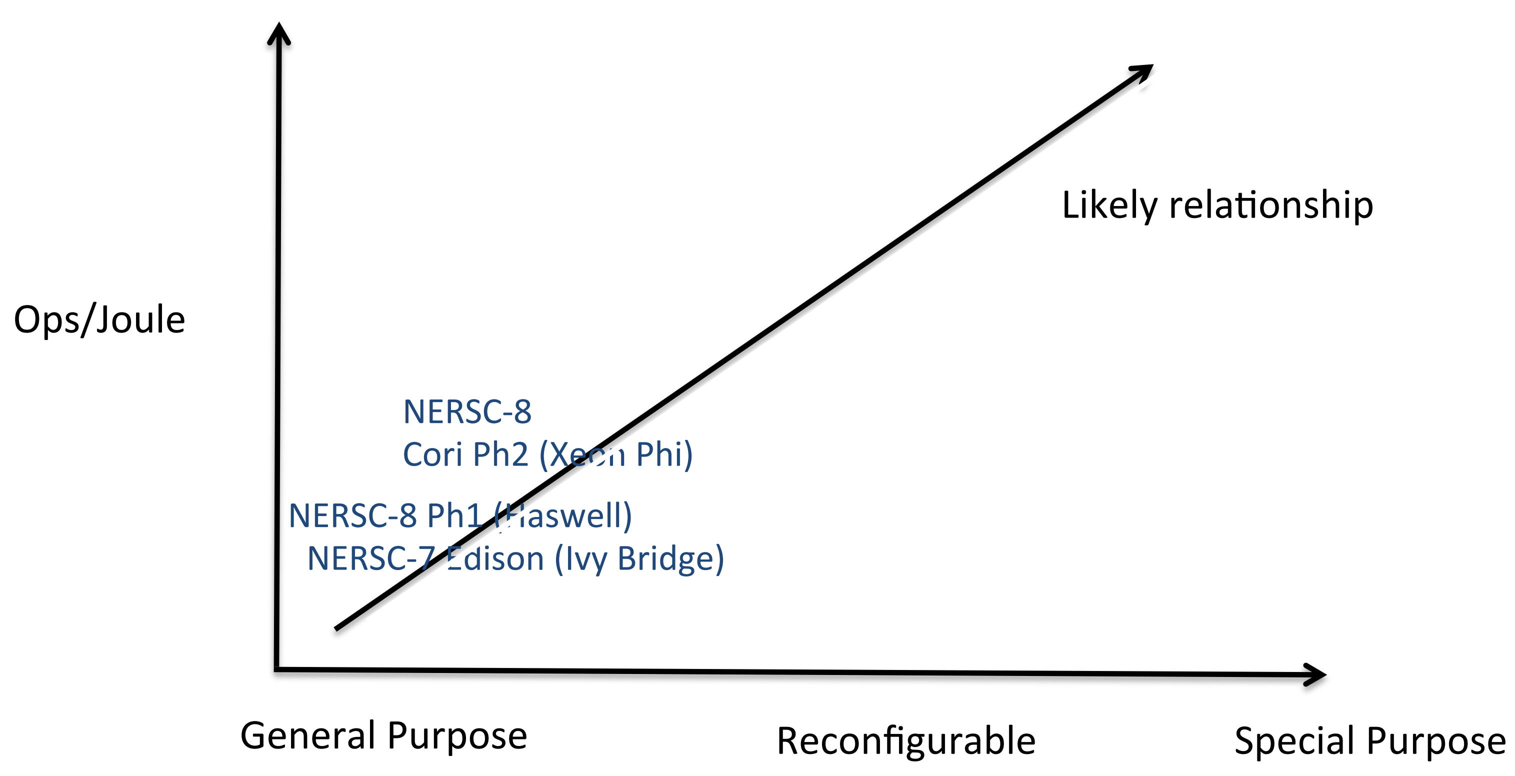
(30-stage fanout-4 inverter chains)



More Parallelism at Lower Levels



Specialization: End Game for Moore's Law

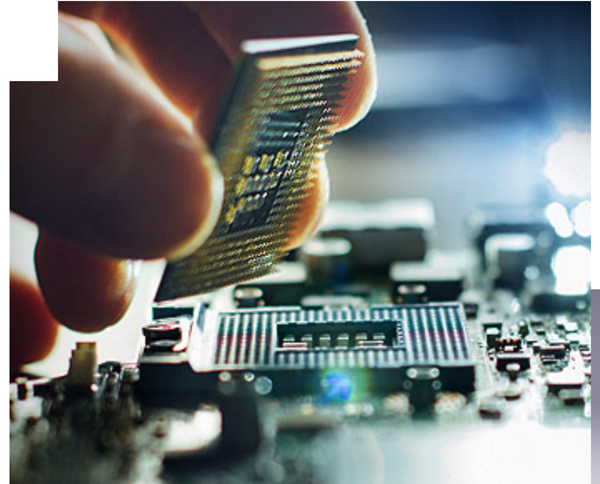


Not just for HPC

Specialization in Deep Learning

Bandwidth, low precision flops, fixed point,…

NVIDIA builds deep learning appliance with P100 Tesla's



Intel buys deep learning startup, Nervana

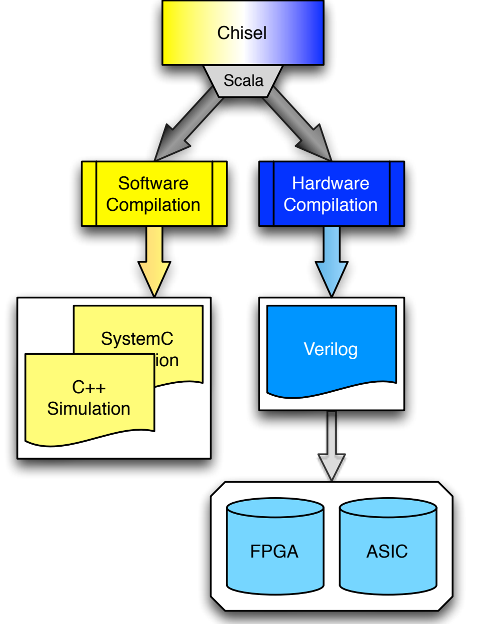
Google designs its own Tensor Processing Unit (TPU)

Can we used their special purpose systems?
Can we design our own?

Open Hardware (Synthesis & Simulation)

Chisel

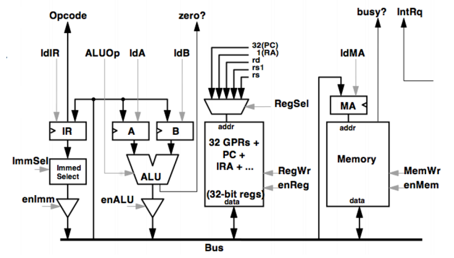
DSL for rapid prototyping of circuits, systems, and arch simulator components

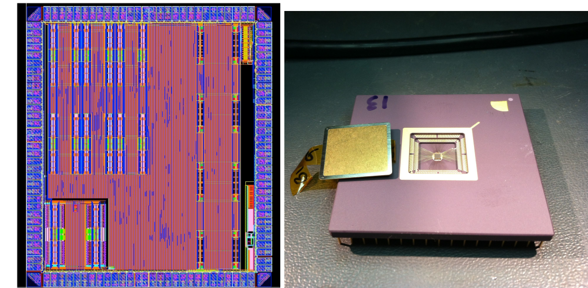


Back-end to synthesize HW with different devices Or new logic families

RISC-V

Open Source Extensible ISA/ Cores

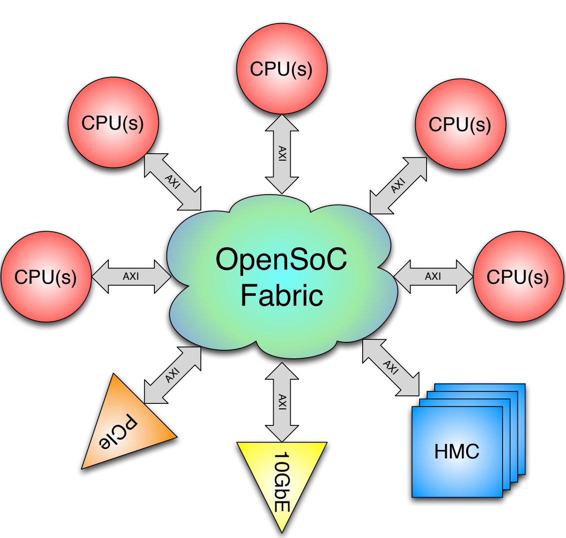




Re-implement processor With different devices or Extend w/accelerators

OpenSOC

Open Source fabric To integrate accelerators And logic into SOC

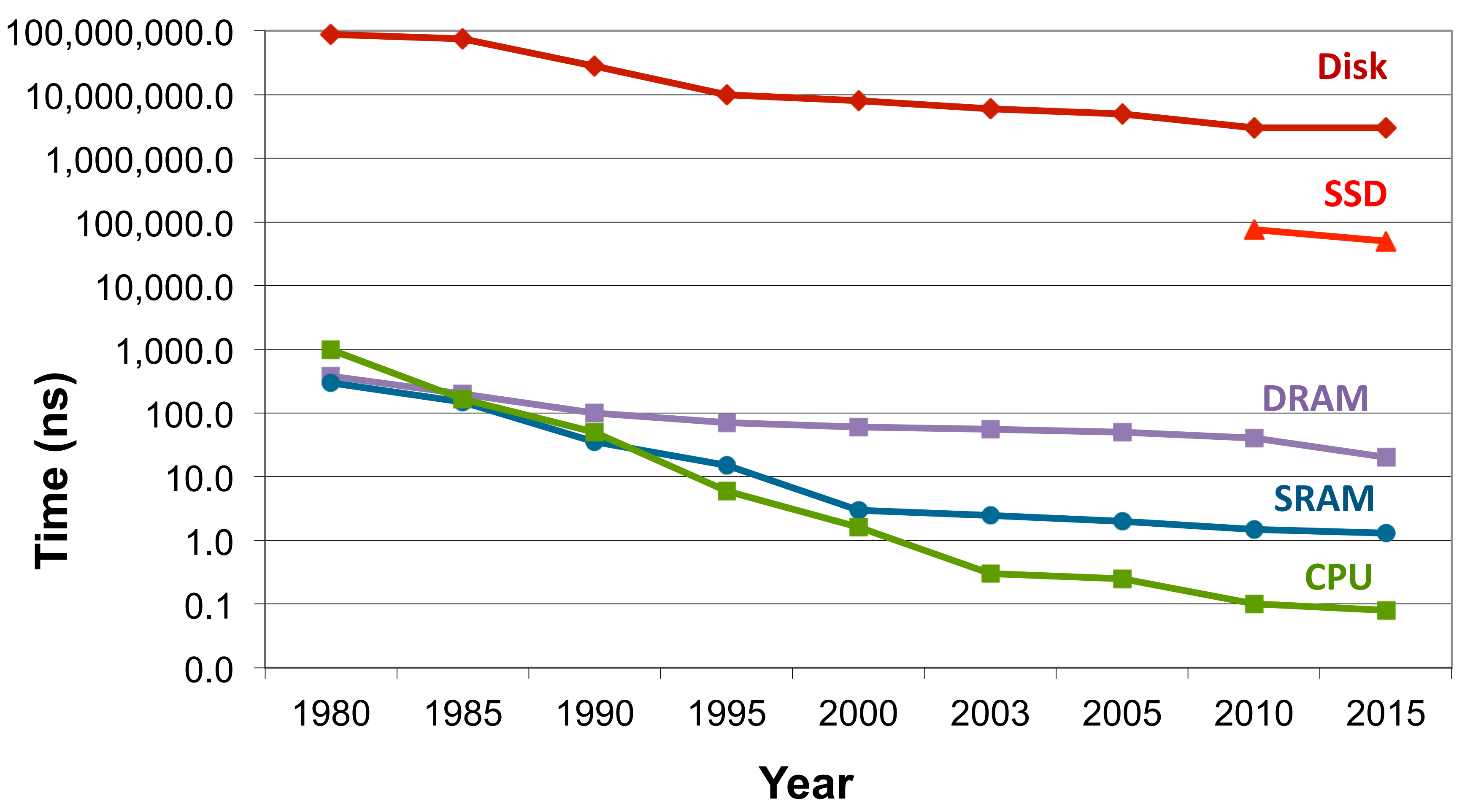


Platform for experimentation with specialization to extend Moore's Law

Shalf, Donofrio, Asasnovic, et al, UCB and LBNL

Data Movement is Expensive

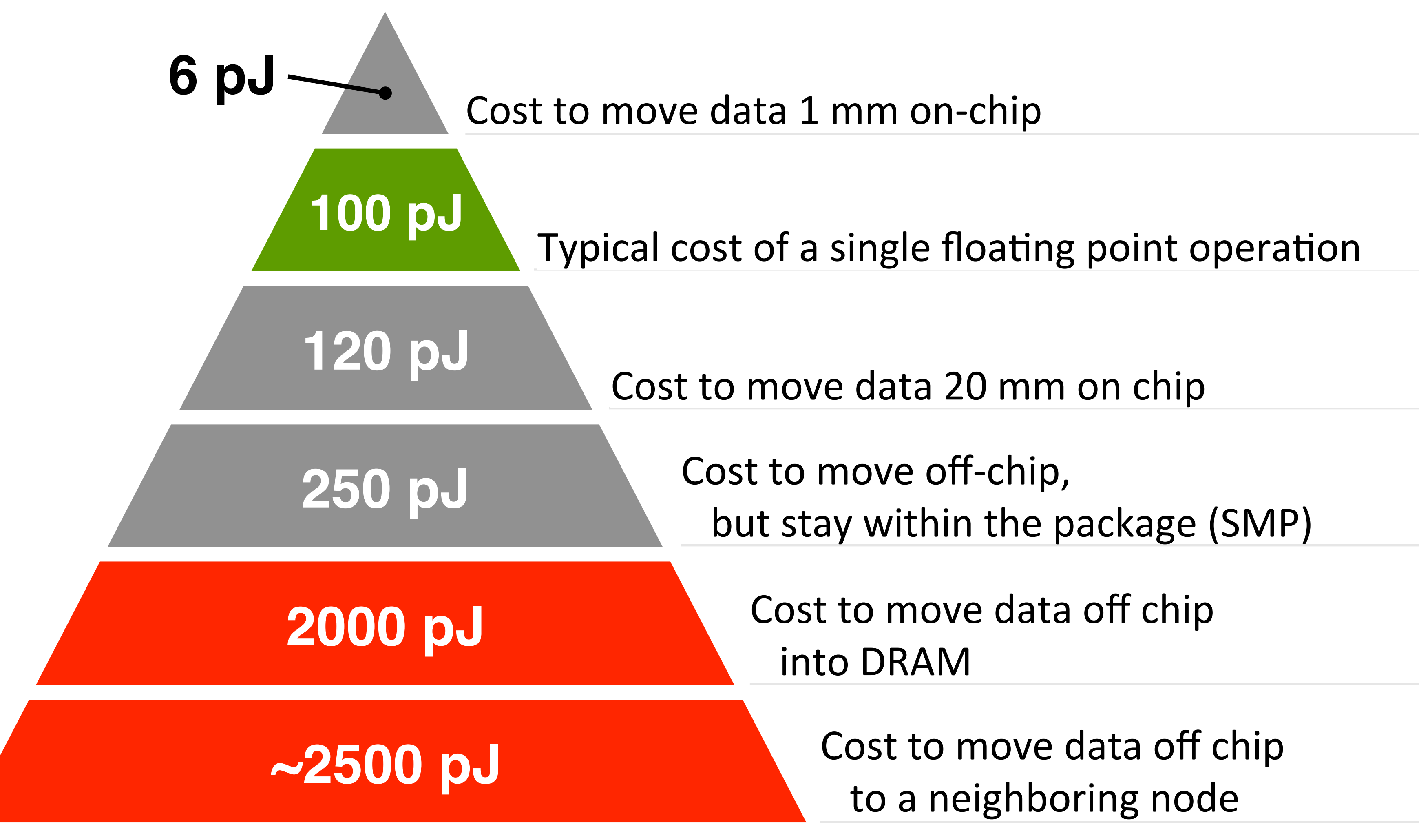
CPU cycle time vs memory access time



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

Data Movement is Expensive

Hierarchical power costs.



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

Summary

- Even more lower level parallelism
- Specialization
- Communication even more expensive (relatively)

The end of Relaxed Programming

- **Moore: The Law that taught performance programmers to relax**

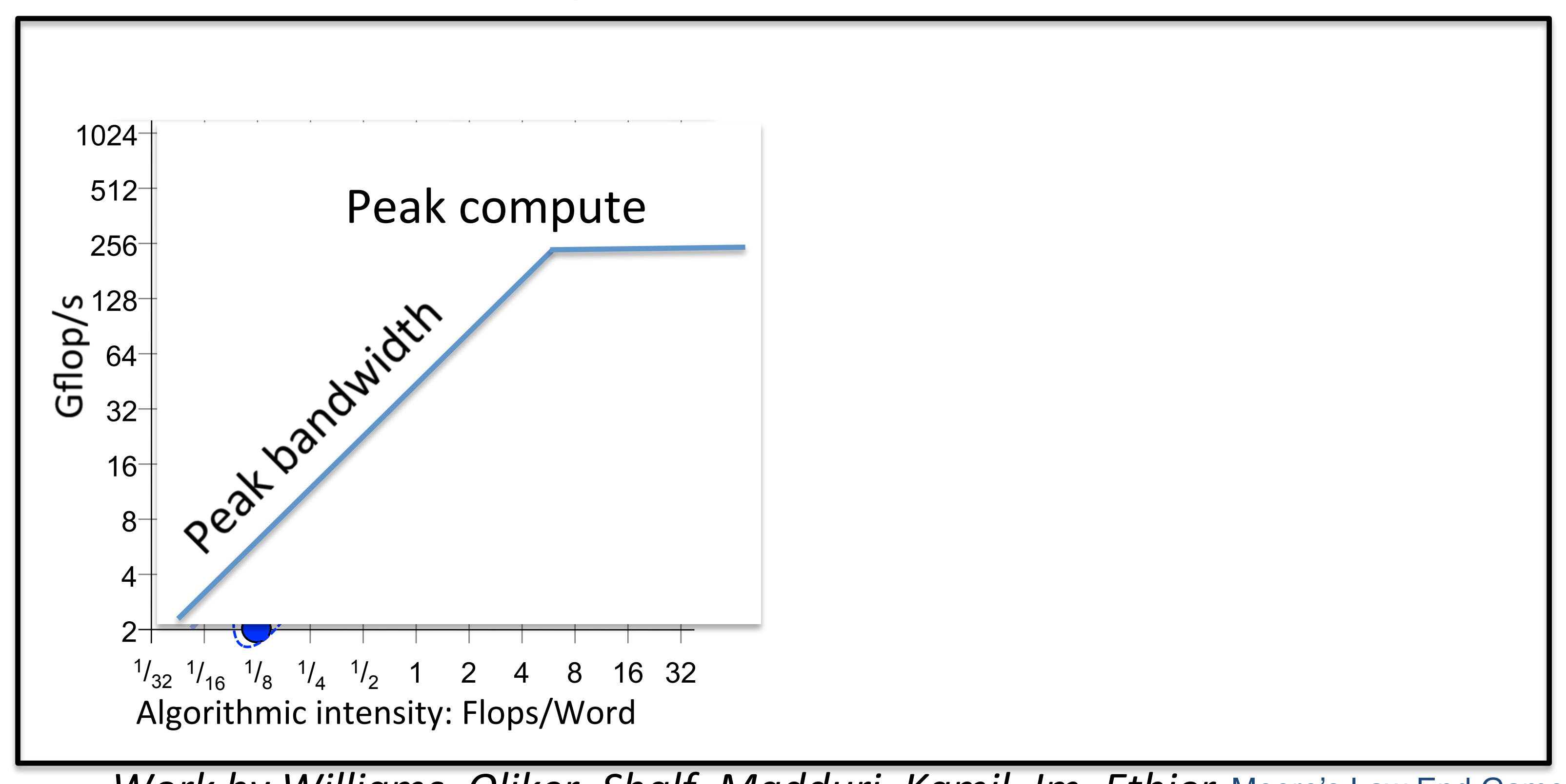
Don’t Fear the Compiler

Who needs compilers?

- **Scientific computing relies heavily on libraries**
  - E.g., LAPACK and FFTW are widely used
- **Languages and compilers are still useful**
  - Higher level syntax is needed for productivity
    - We need a language
  - Static analysis is helps with correctness
    - We need a compiler (front-end)
  - Optimizations are needed to get performance
    - We need a compiler (back-end)

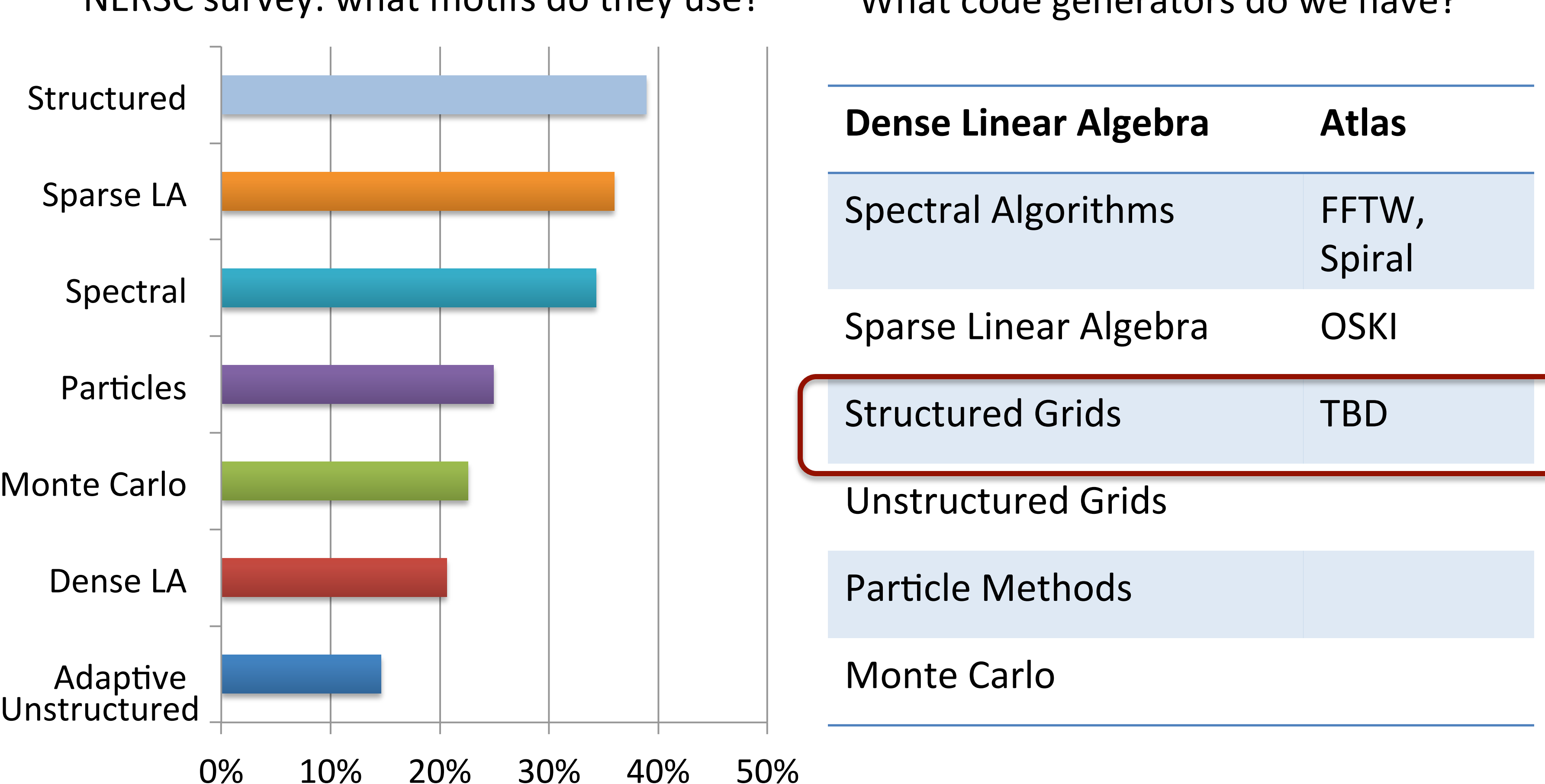
Autotuning: Write Code Generators

- **Two "unsolved" compiler problems:**
  - dependence analysis and
  - accurate performance models
- **Autotuners are code generators plus search**



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

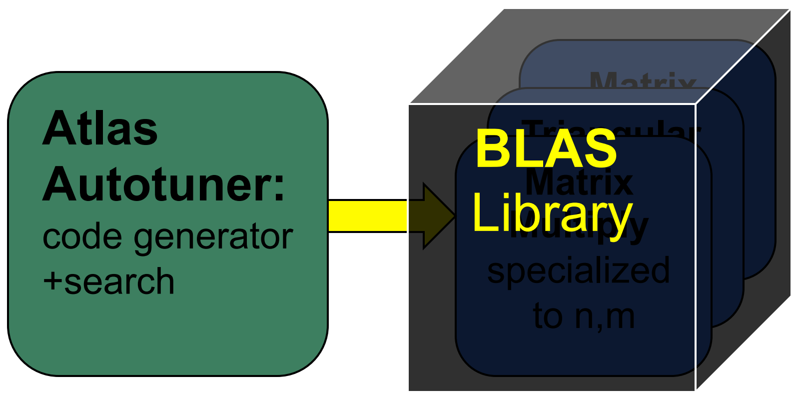
What we have and what we need



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

Approaches to Autotuning



Approximate categorization!

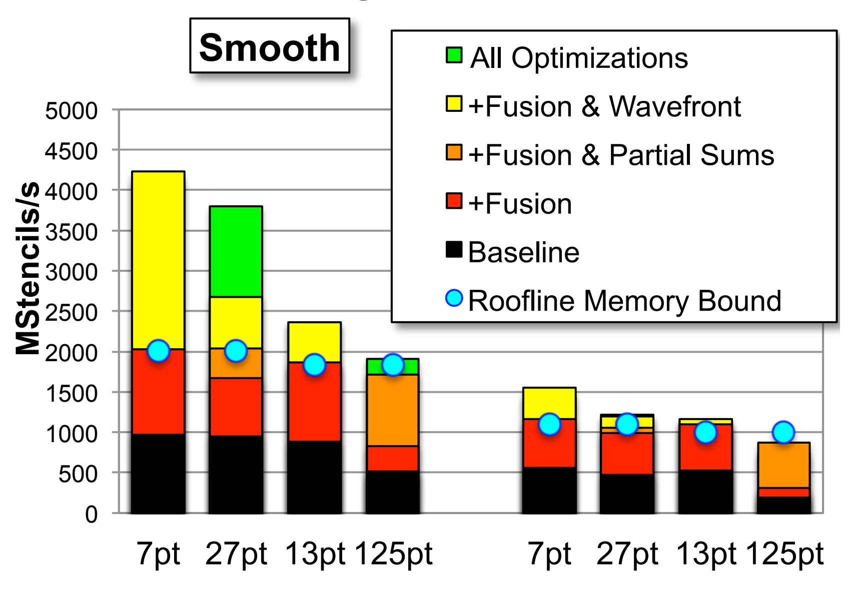
How do we produce all of these (correct) versions?

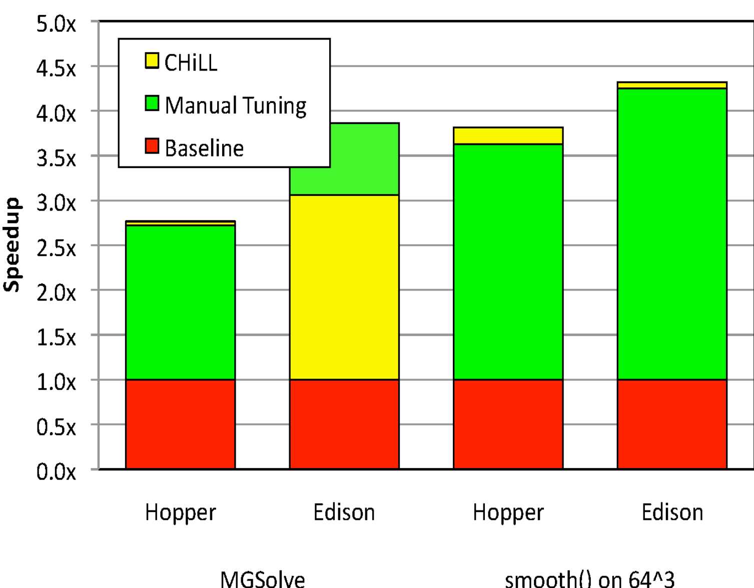
- Using scripts (Python, perl, ML, C,..)
- Compiling annotated general-purpose language (X-Tune,…)
- Use preprocessor to generator code (Raja, Kokkos,TiDA)
- Compile a domain-specific language (D-TEC, Halide)
- Domain-specific compiler for domain-specific language (SEJITS)

Several Projects and PIs: Sam Williams, Mary Hall, Dan Quinlan, Armando Fox, Saman Amarsinghe, Armando Solar-Lezama, Jack Dongarra,

Approach #1: Compiler-Directed Autotuning

- **Two hard compiler problems**
  - Analyzing the code to determine legal transformations
  - Selecting the best (or close) optimized version
- **Approach #1: General-purpose compilers (+ annotations)**
  - Use ***communication-avoiding optimizations*** to reduce memory bandwidth
  - Apply ***CHiLL compiler*** technology with general polyhedral optimizations
  - Use autotuning to select optimized version

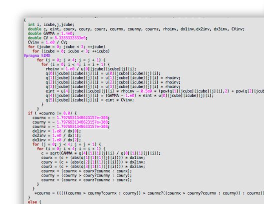


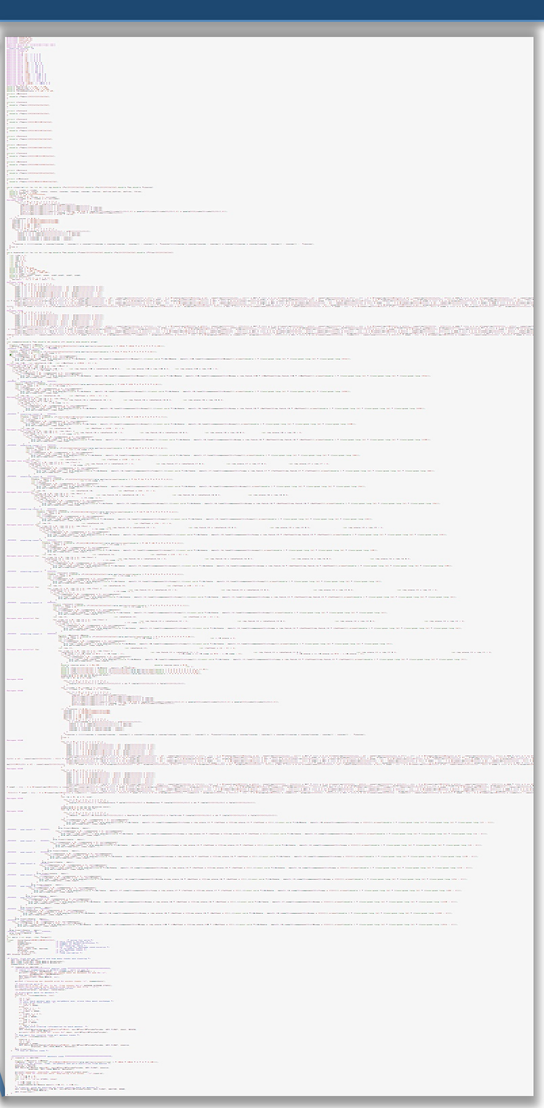


Results on Geometric Multigrid (miniGMG Smoother)

Approach #2: DSLs with General Purpose Compiler

- **Generation of Complex Code for 10 Levels of Memory Hierarchy with SW managed cache**
  - 4th order stencil computation from CNS Co-Design Proxy-App
  - Same DSL code can generate to 2, 3, 4, … levels too
  - Code size of autogenerated code



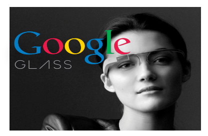


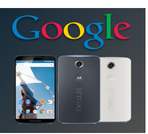
Memory Hierarchy	2 Level	3 Level	4 Level	…	10 level
DSL Code	20				
Auto Generated Code	446	500	553		819

Use of Rose/PolyOpt to apply DSLs to large applications and collaboration on AMR

Approach #3: Domain-Specific (but not too specific) Languages used by other markets

Developed for Image Processing





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

HPGMG (Multigrid on Halide)

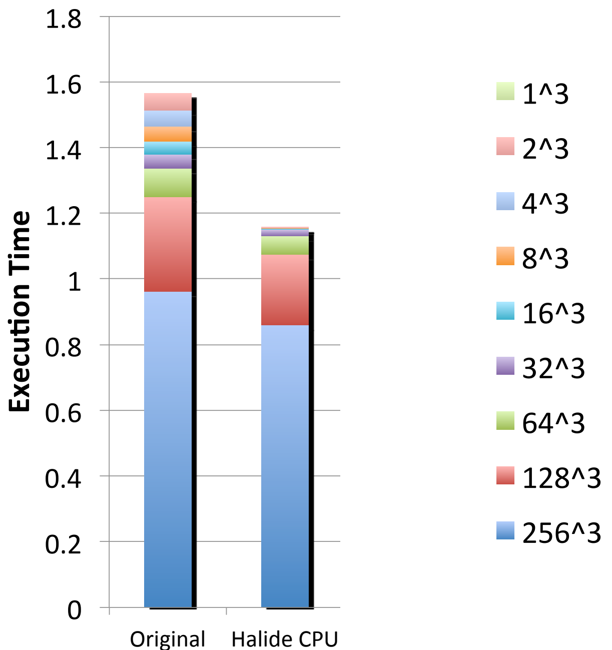
- **Halide Algorithm by domain expert**

```
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

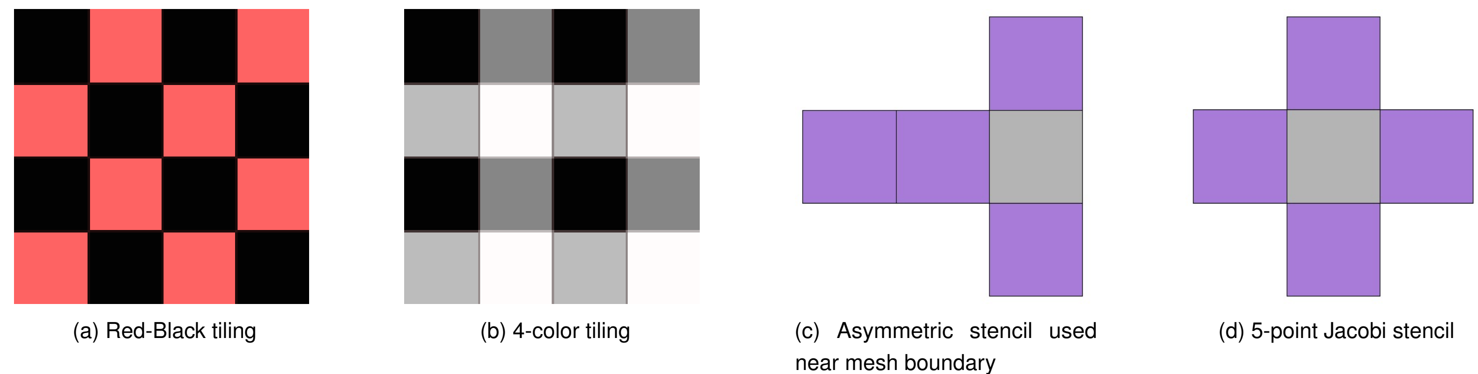
Halide performance

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



Approach #4: Small Compiler for Small Language

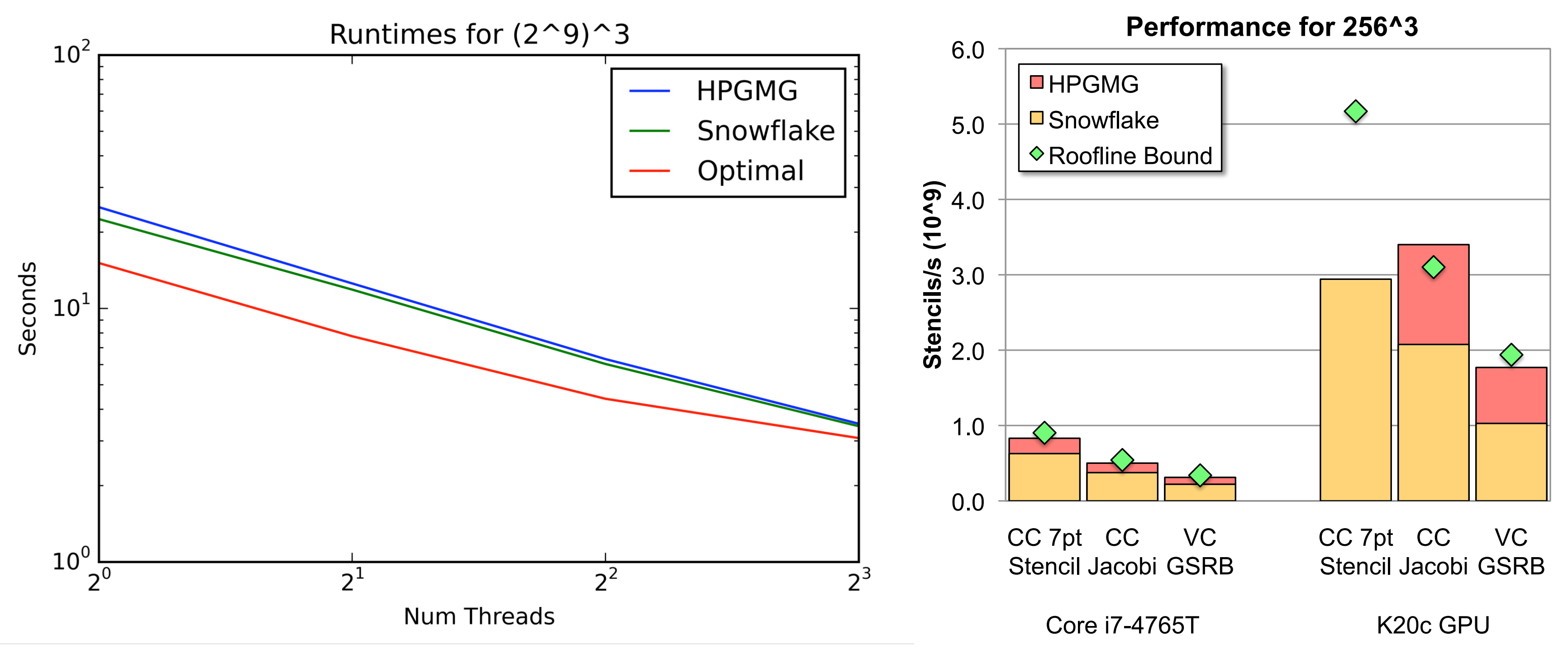
- **Snowflake: A DSL for Science Stencils**
  - Domain calculus inspired by Titanium, UPC++, and AMR in general



(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**

Algorithms for the Hardware

Beyond Domain Decomposition

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

Surprises:

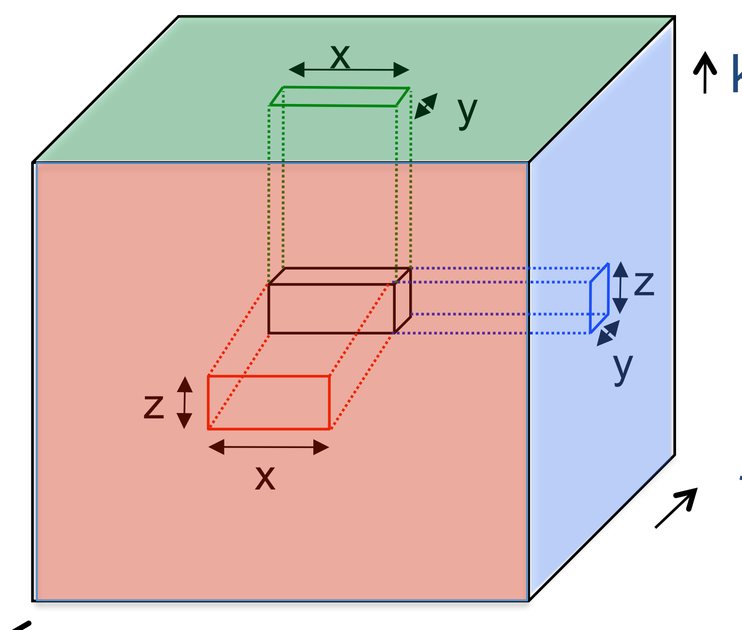
- Even Matrix Multiply had room for improvement
- Idea: make copies of C matrix (as in prior 3D algorithm, but not as many)
- Result is provably optimal in communication

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]  …
```

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 = $\Omega(\#iterations/M^e)$

  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**

Communication Lower Bounds

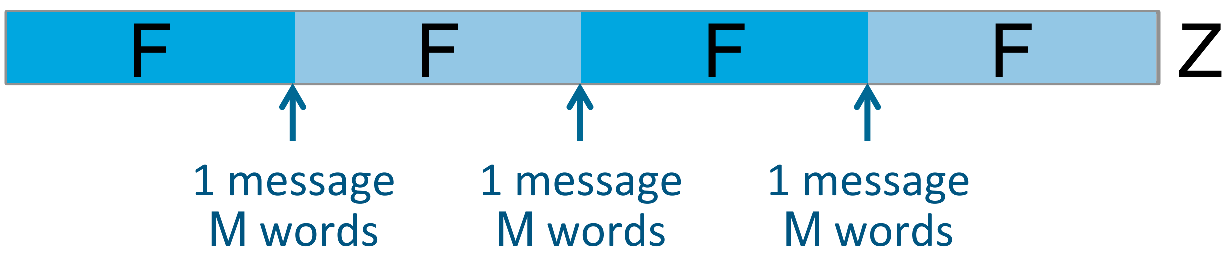
- **For loop nests with arrays**
  - **M** words of data in fast memory, i.e., **n/p**.

$$\#\text{msgs} = \frac{\#\text{flops each processor has to do } \mathbf{(Z)}}{\text{max } \#\text{useful flops with } \mathbf{M} \text{ words } \mathbf{(F)}}$$

$$\#\text{words} = \#\text{msgs} \cdot \mathbf{M}$$

$\mathbf{M^2}$ for N-body

$\mathbf{M^{3/2}}$ for matmul



Implications for Compilers

x += …

x += …

x += …

x += …

Using x for C[i,j] here

- **Much of the work on compilers is based on owner-computes**
  - 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

Communication Avoiding Version (using a "1.5D" decomposition)

p/c →

c ↓

- **Divide p into c groups. Replicate particles within group.**
  - First row responsible for updating all by orange, second all by green,…
- **Algorithm: shift copy of n/(p*c) particles to the left**
  - Combine with previous data before passing further level (log steps)
- **Reduce across c to produce final value for each particle**
- Total Computation: $O(n^2/p)$;
- Total Communication: $O(\log(p/c) + \log c)$ messages,

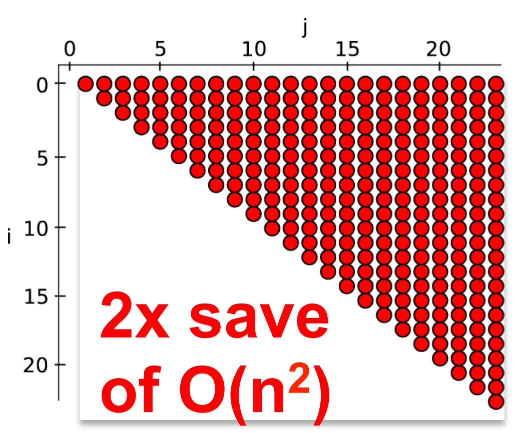
  $O(n*(c/p+1/c))$ words

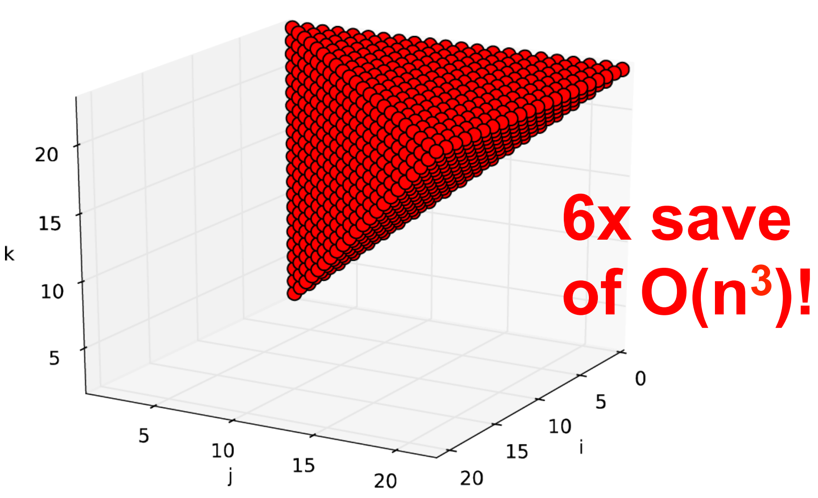
Limit: $c \leq p^{1/2}$

Driscoll, Georganas, Koanantakool, Solomonik, Yelick

Challenge: Symmetry & Load Balance

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

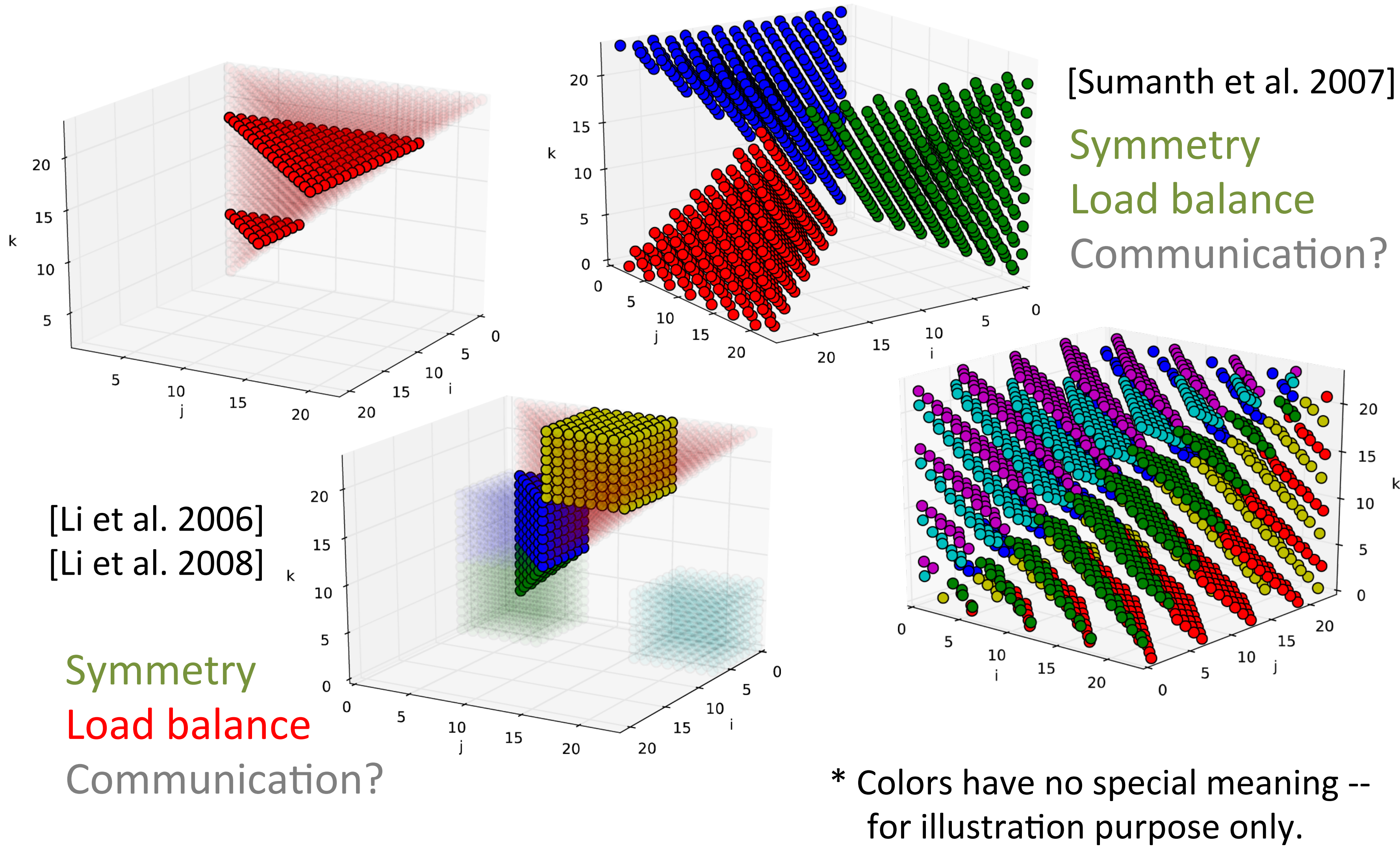




- How to divide work equally?

Koanantakool & Yelick

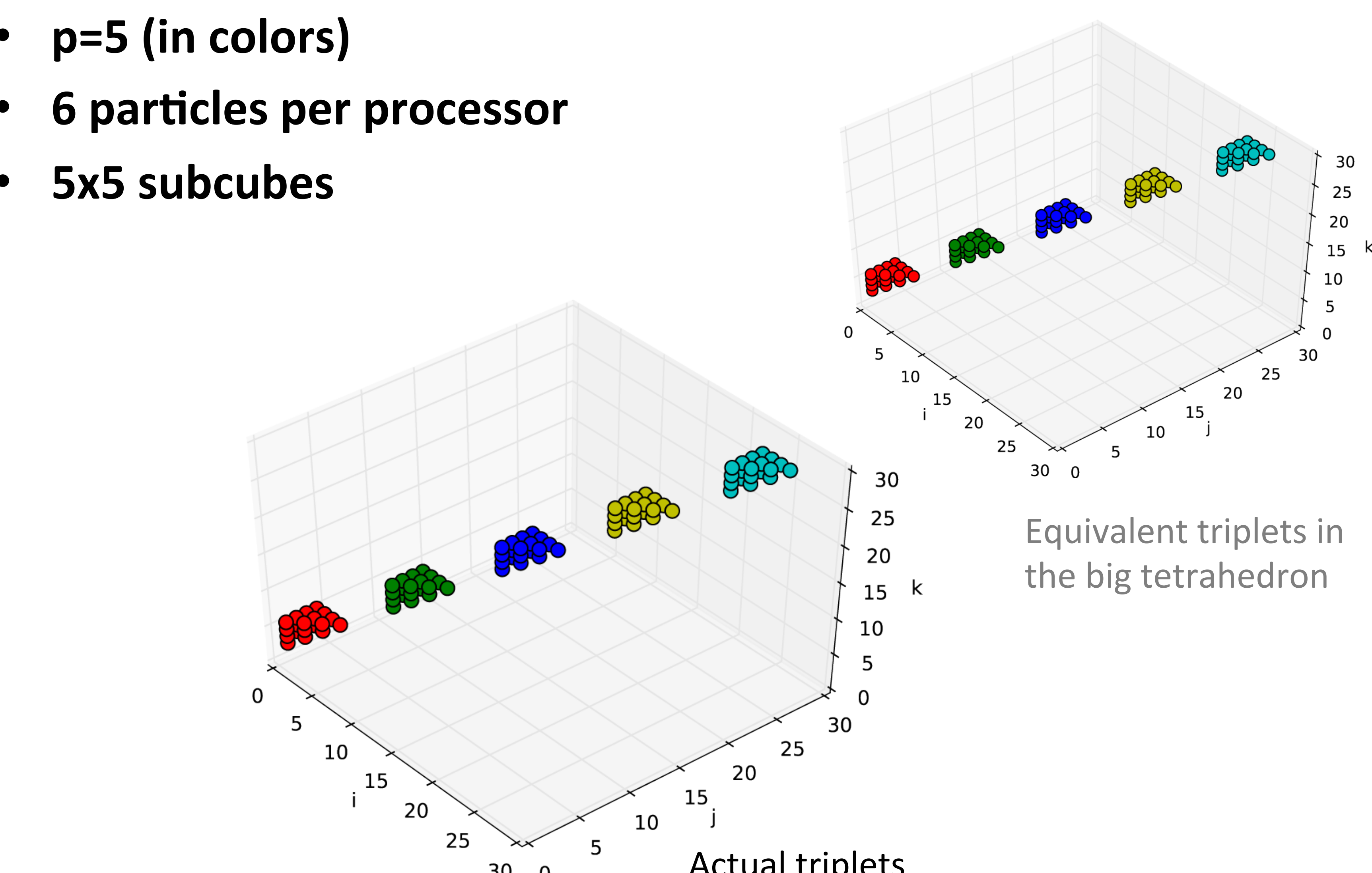
All-triplets 3-body: Challenges



CA 3-body

[Koanantakool and Yelick 2014]

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



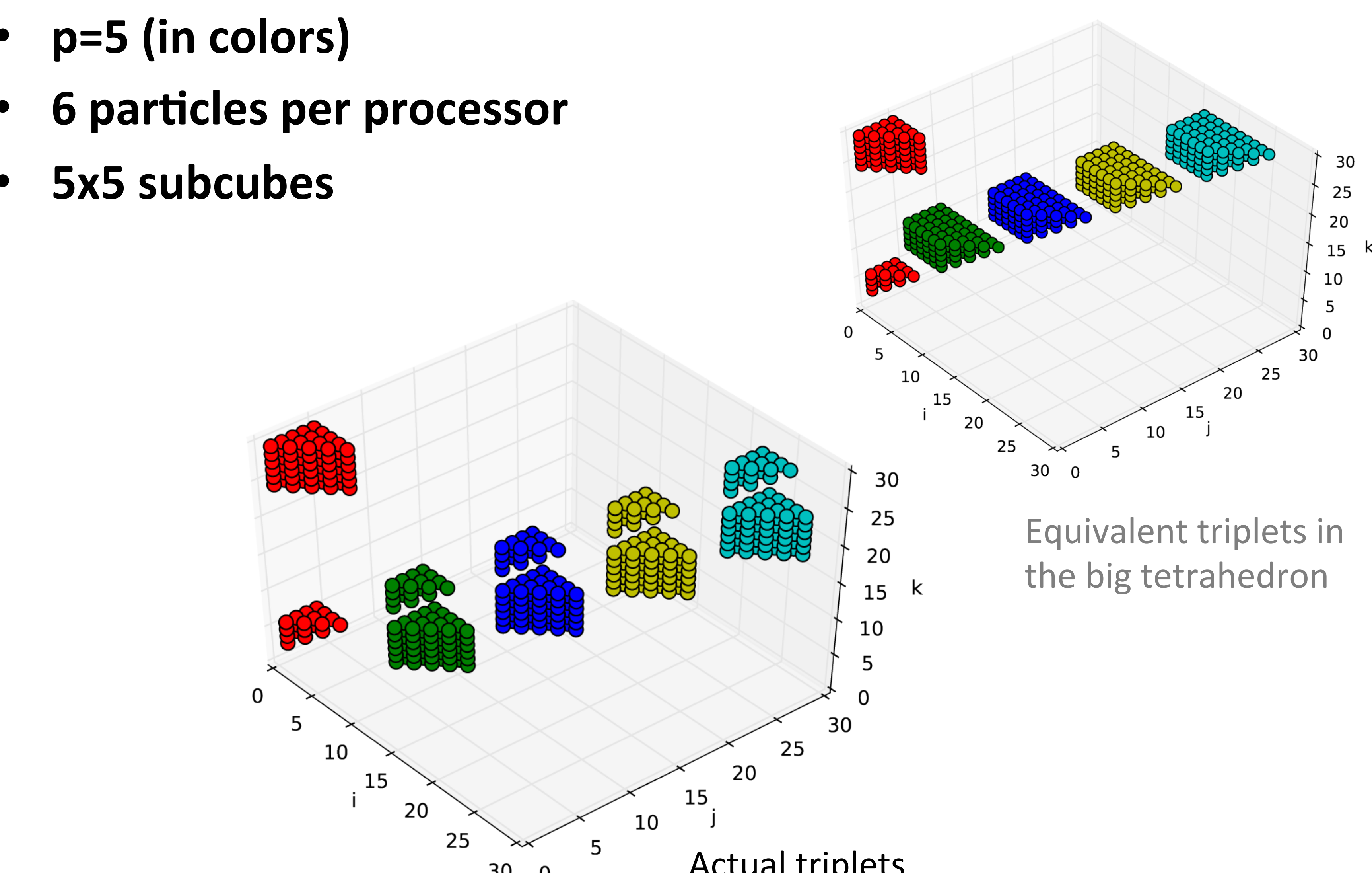
Equivalent triplets in the big tetrahedron

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

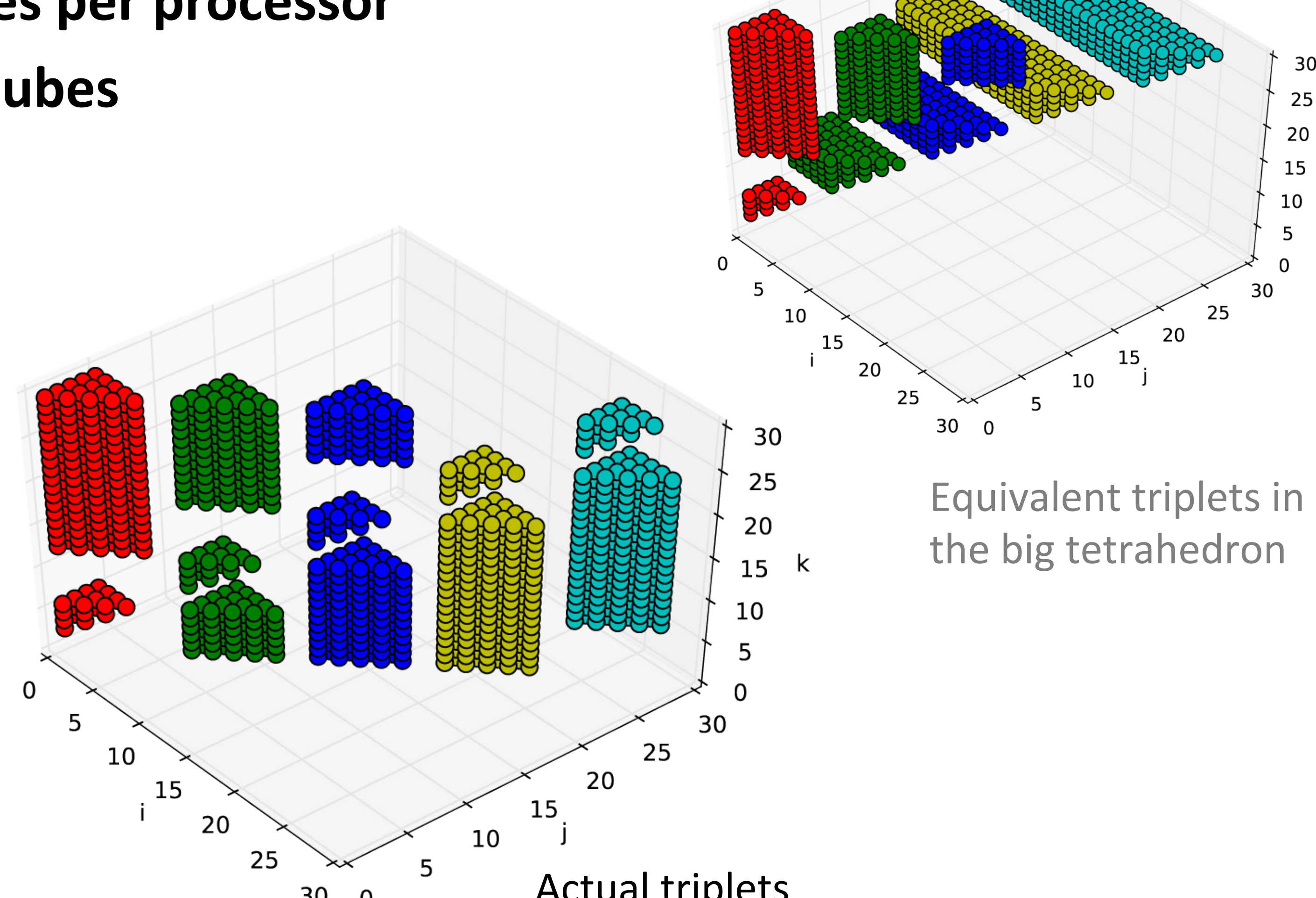
Equivalent triplets in the big tetrahedron

Actual triplets

CA 3-body

[Koanantakool and Yelick 2014]

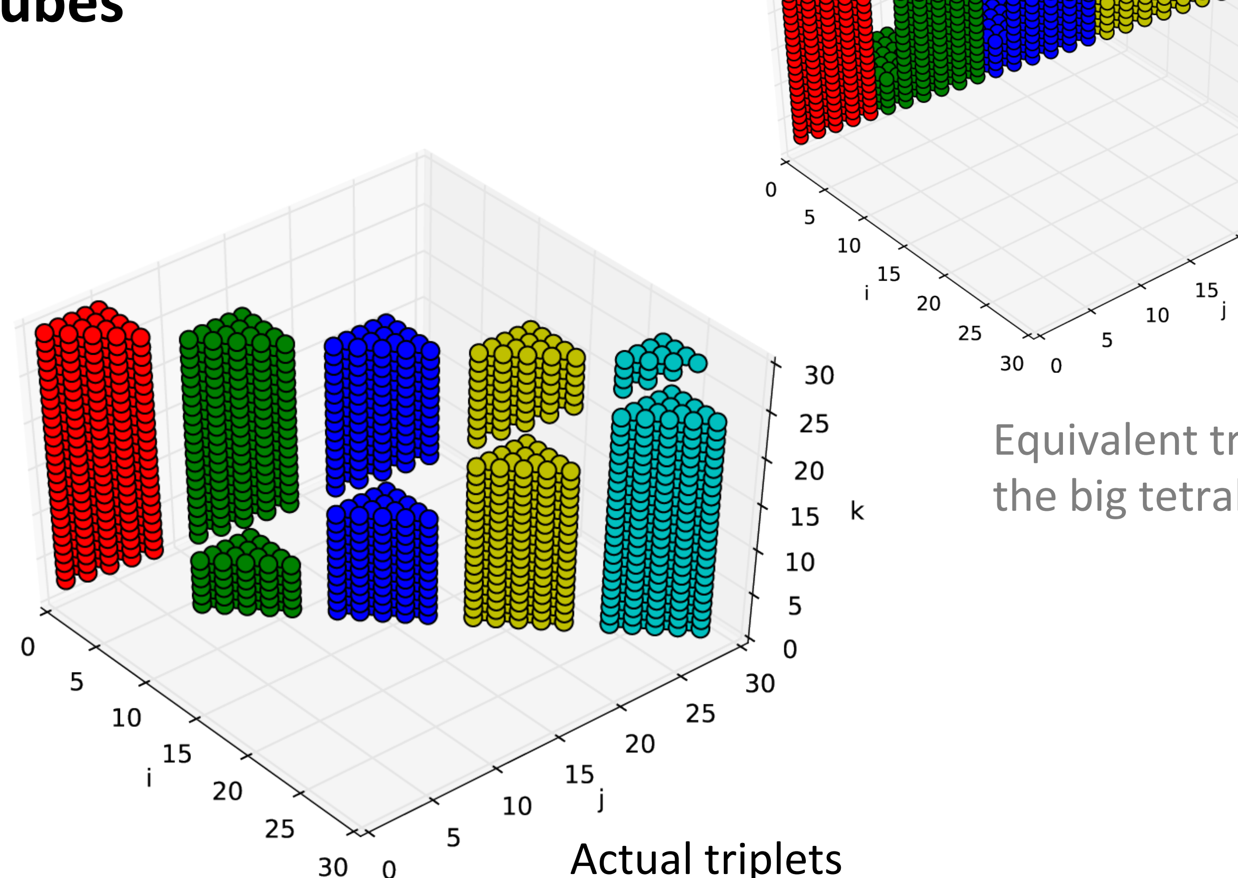
- **p=5 (in colors)**
- **6 particles per processor**
- **5x5 subcubes**



CA 3-body

[Koanantakool and Yelick 2014]

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



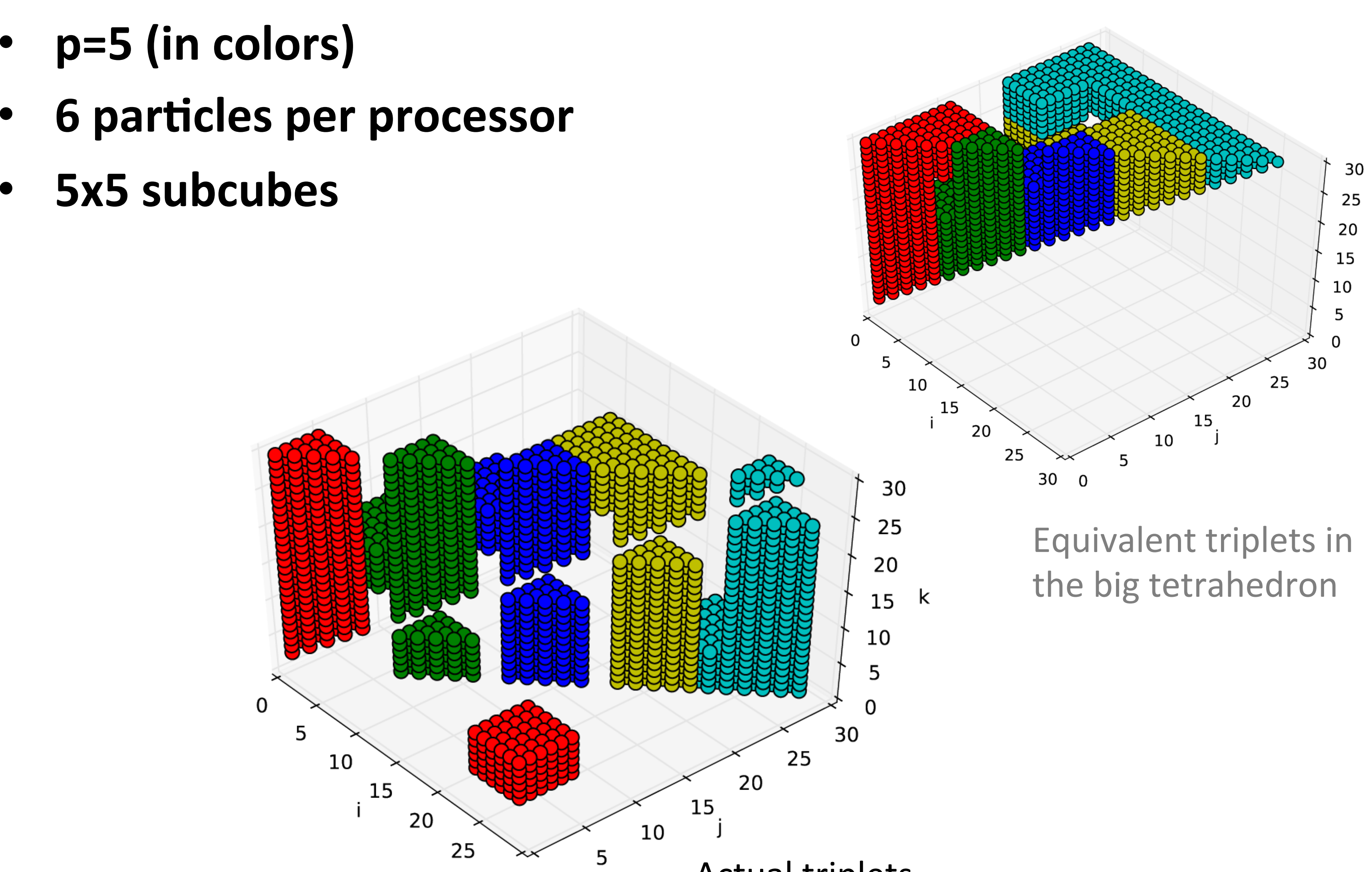
Equivalent triplets in the big tetrahedron

Actual triplets

CA 3-body

[Koanantakool and Yelick 2014]

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



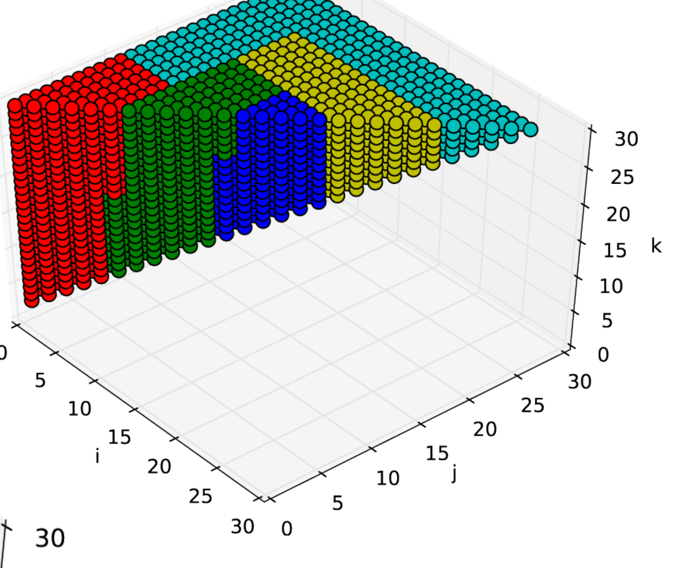
Equivalent triplets in the big tetrahedron

Actual triplets

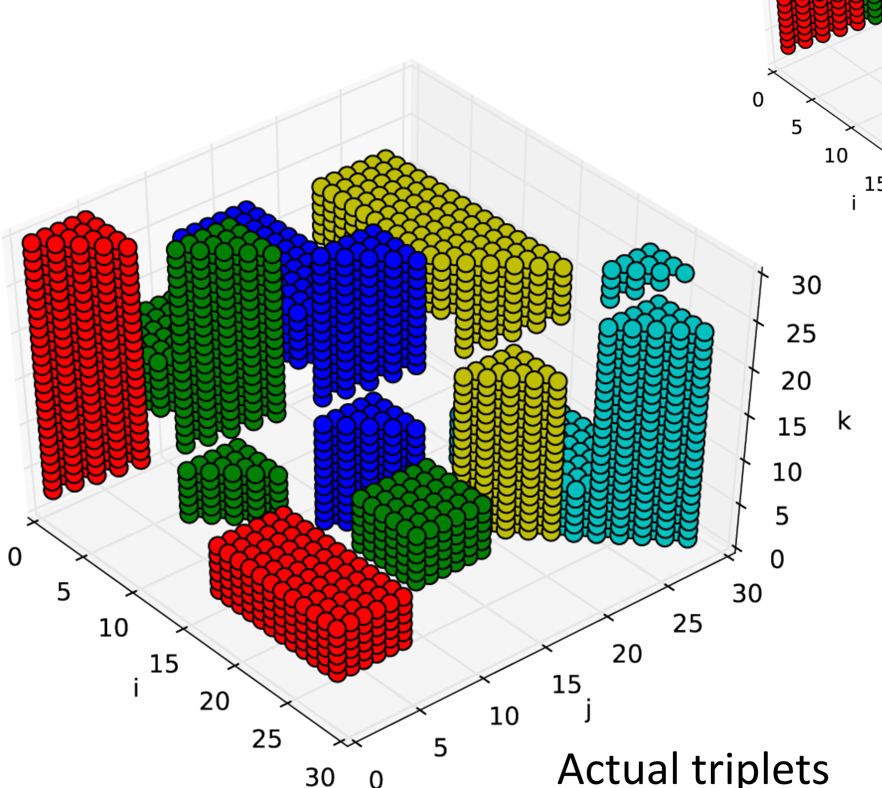
CA 3-body

[Koanantakool and Yelick 2014]

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



Equivalent triplets in the big tetrahedron

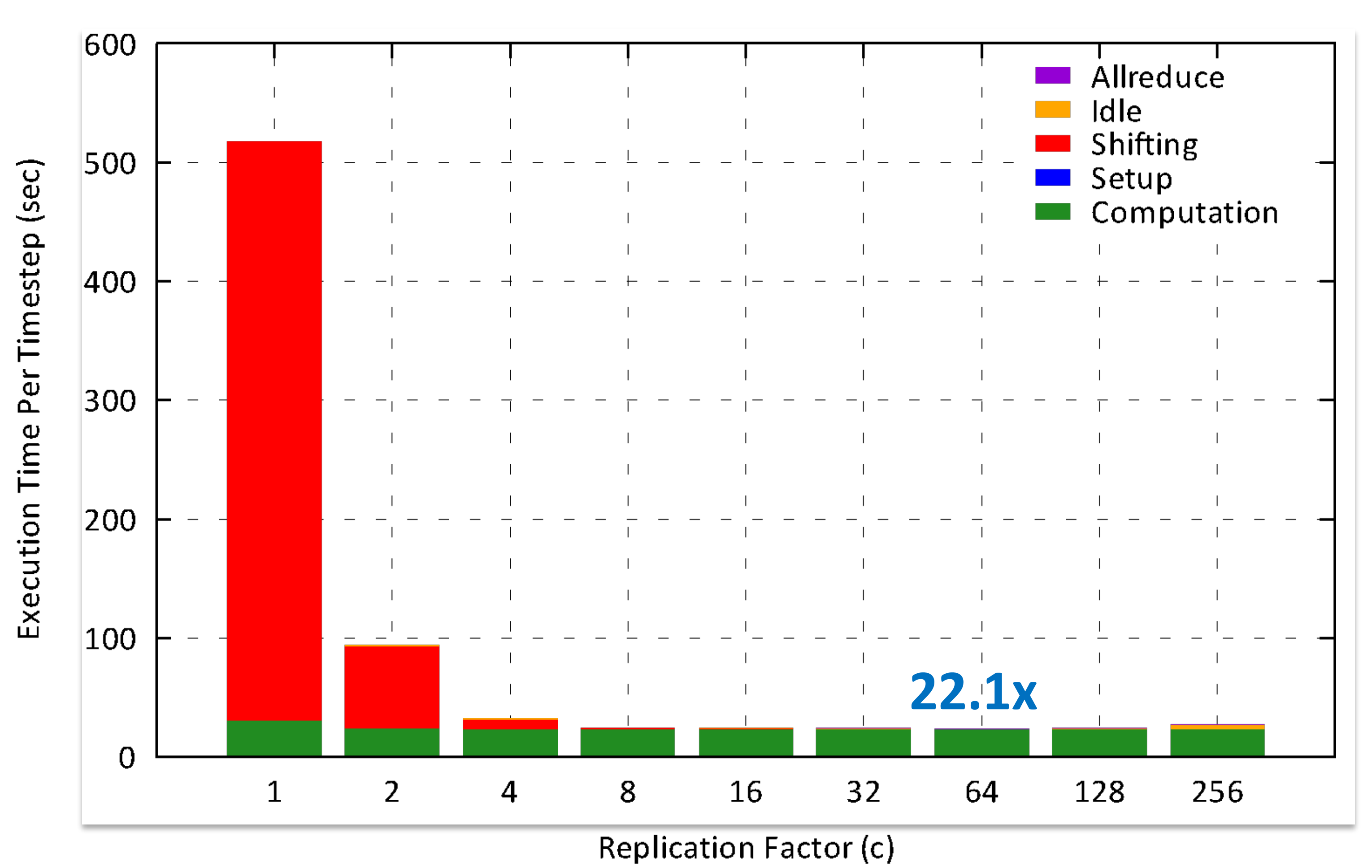


Actual triplets

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

3-Way N-Body Speedup

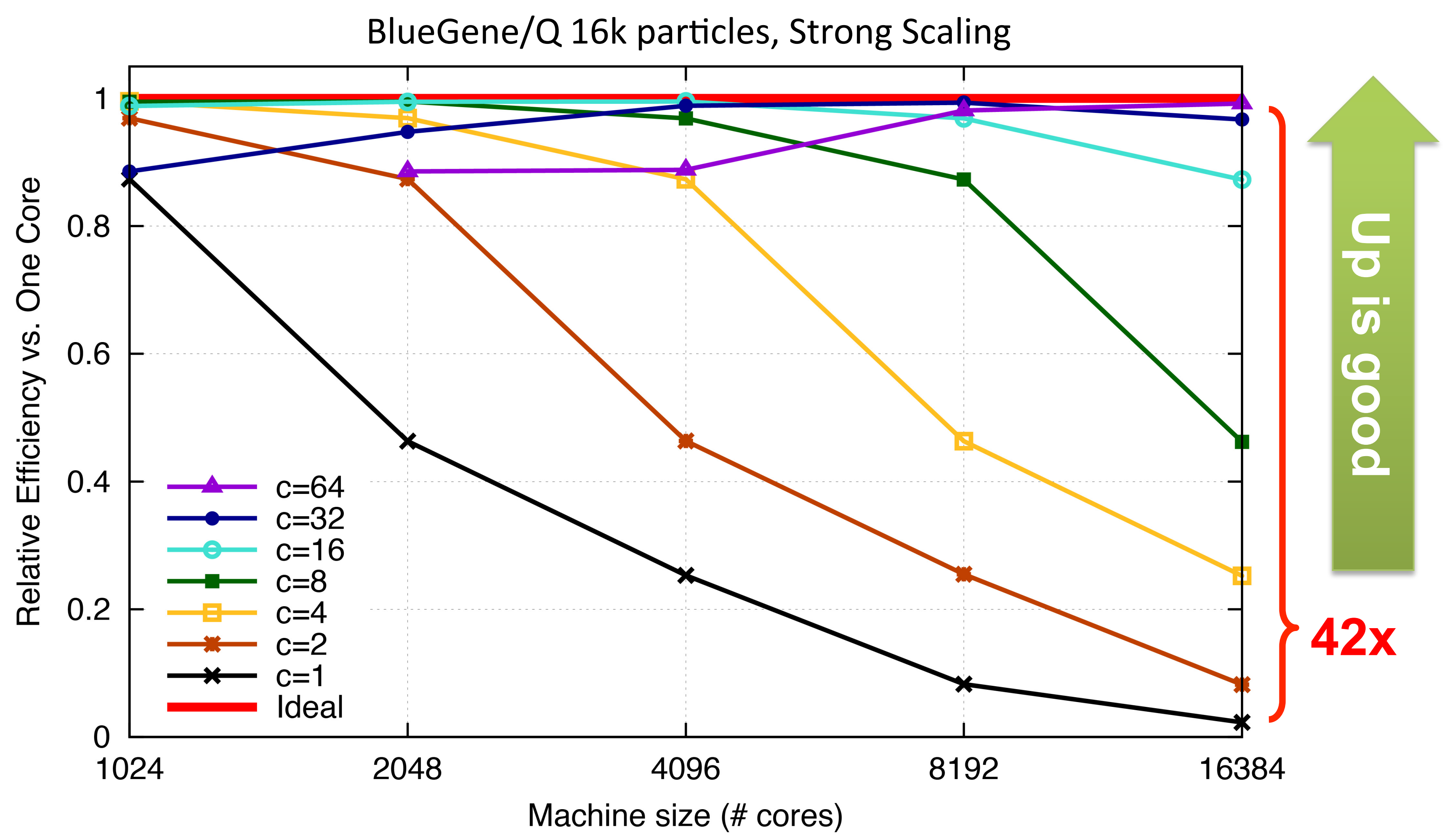
- **Cray XC30, 24k cores, 24k particles**



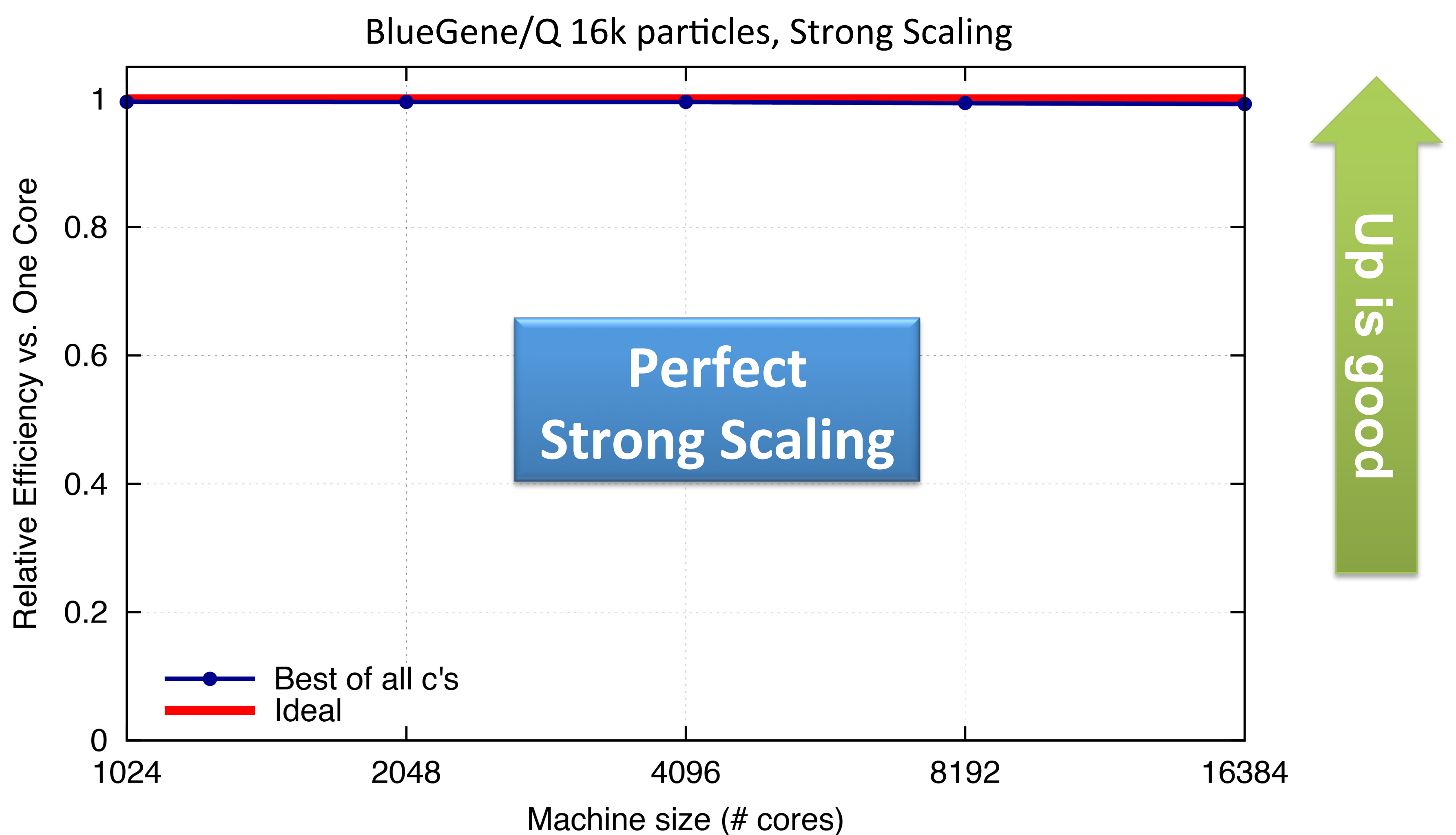
Down is good

Koanantakool & Yelick

Perfect Strong Scaling



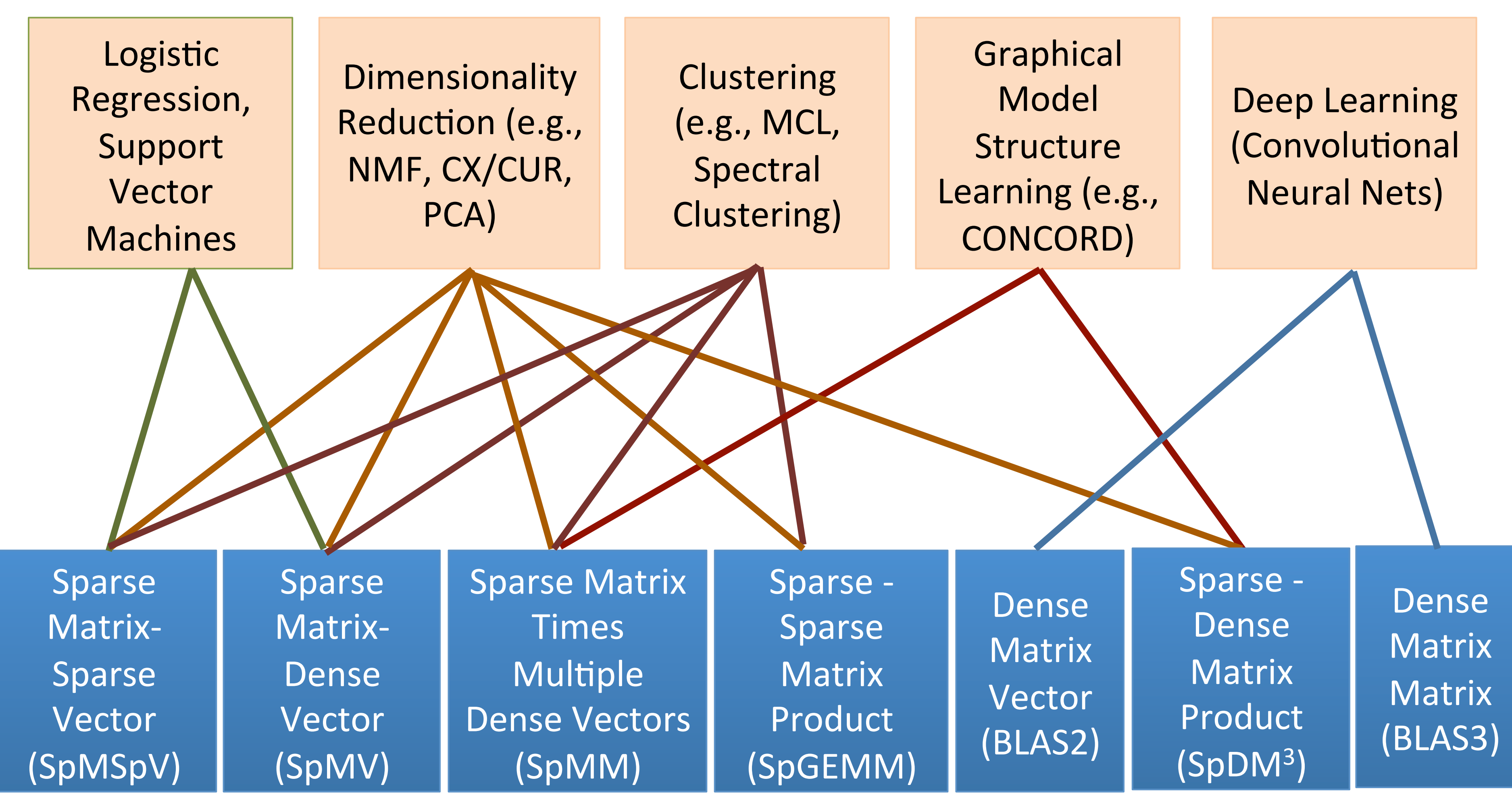
Perfect Strong Scaling



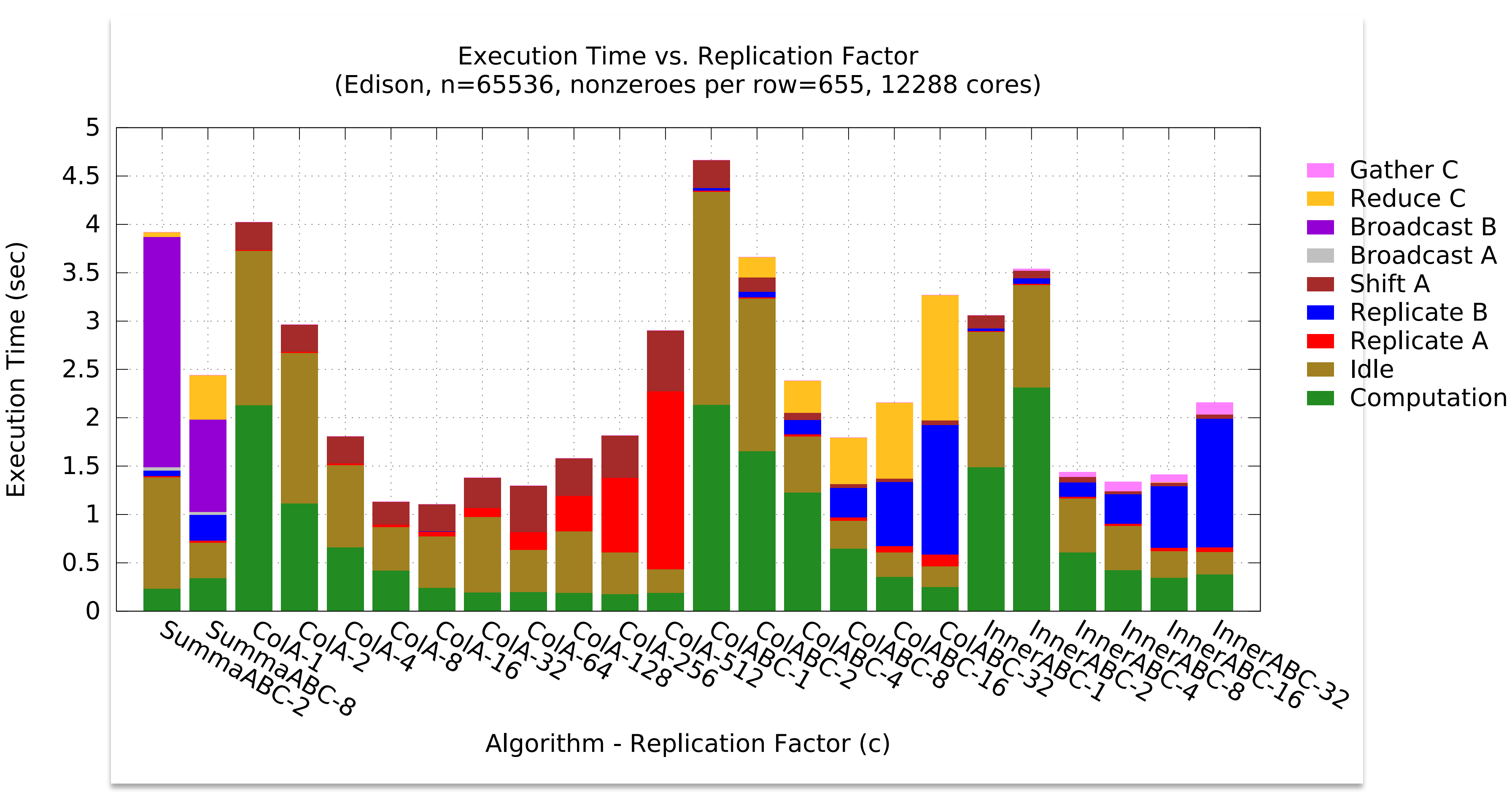
Analytics vs. Simulation Kernels:

7 Giants of Data	7 Dwarfs of Simulation
Basic statistics	Monte Carlo methods
Generalized N-Body	Particle methods
Graph-theory	Unstructured meshes
Linear algebra	Dense Linear Algebra
Optimizations	Sparse Linear Algebra
Integrations	Spectral methods
Alignment	Structured Meshes

Machine Learning Mapping to Linear Algebra



Sparse-Dense Matrix Multiply Too!

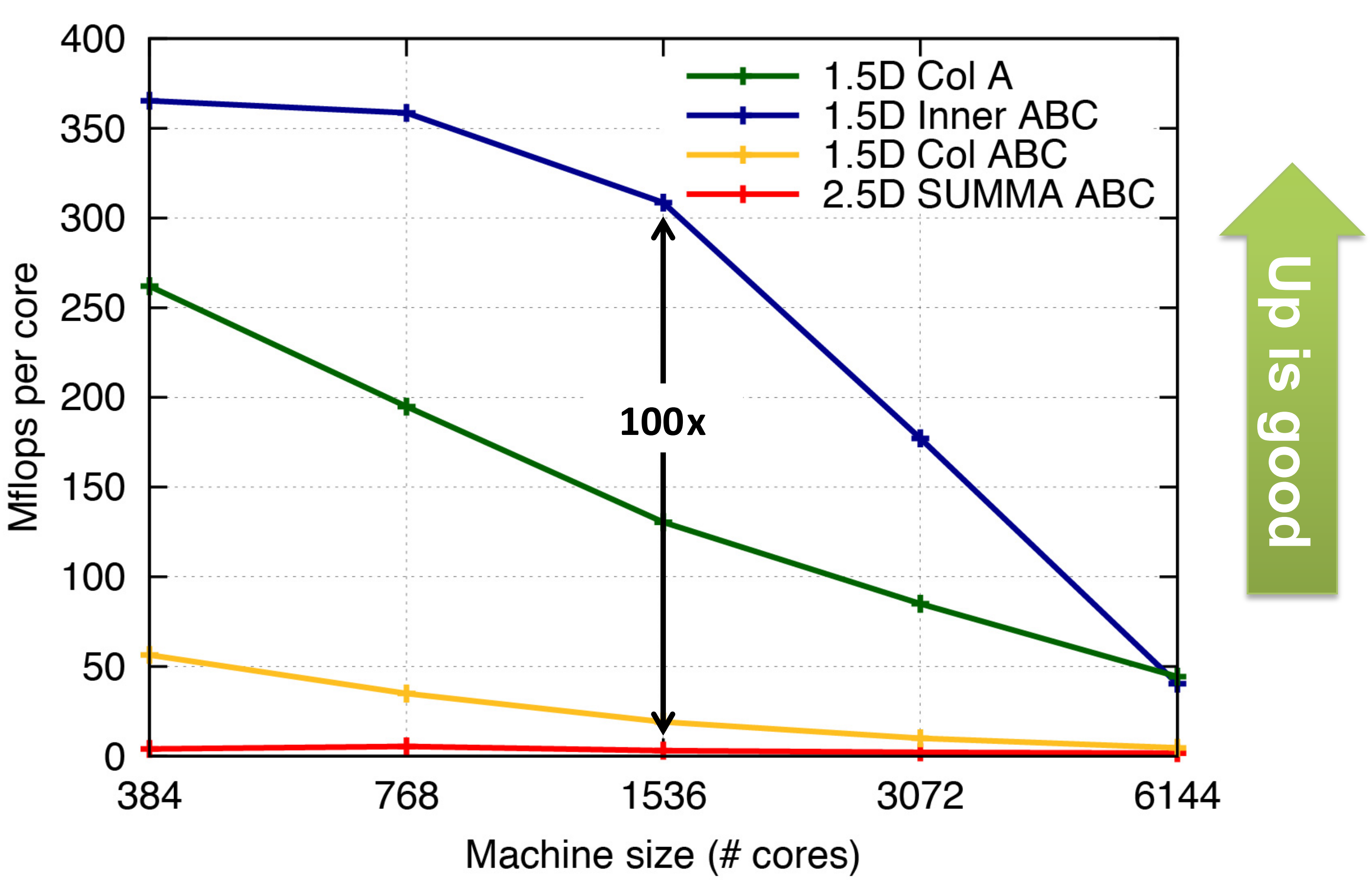


- **Variety of algorithms that divide in or 2 dimensions**

Koanantakool & Yelick

100x Improvement

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



Communication-Avoiding Algorithm Sample Speedups

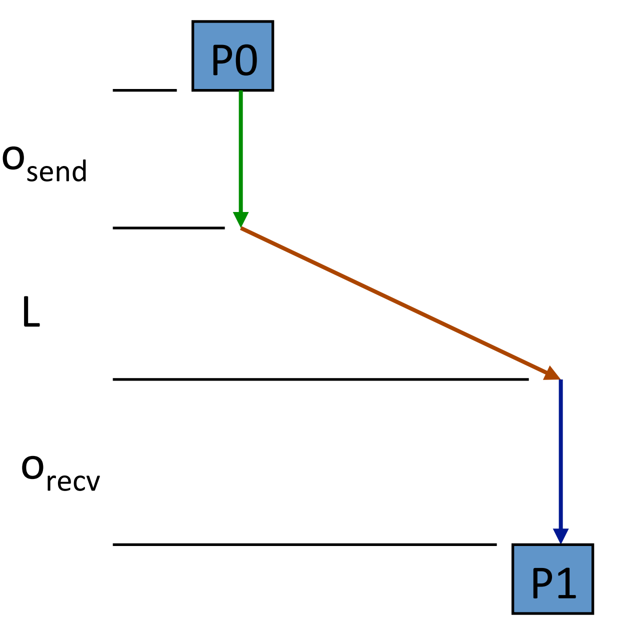
- **Up to 11.8x faster for direct N-body on 32K core IBM BG/P**
- **Up to 100x faster for sparse-dense matmul on Cray XC30**
- **Up to 12x faster for 2.5D matmul on 64K core IBM BG/P**
- **Up to 3x faster for tensor contractions on 2K core Cray XE/6**
- **Up to 6.2x faster for APSP on 24K core Cray CE6**
- **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 et al

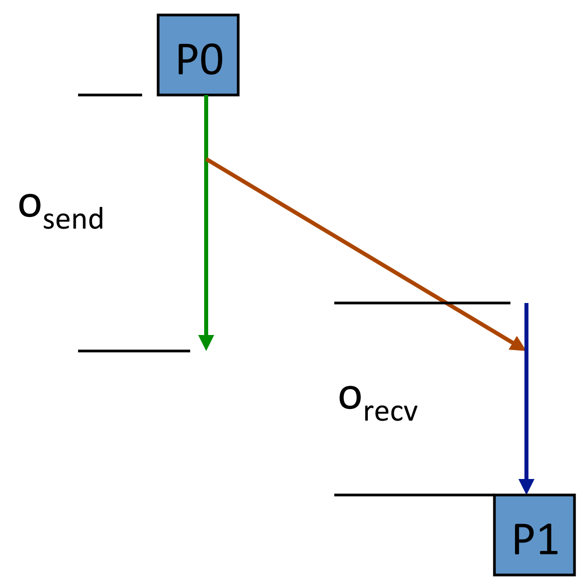
Overhead Can’t be Tolerated

Modified LogGP Model

- **LogGP: no overlap**

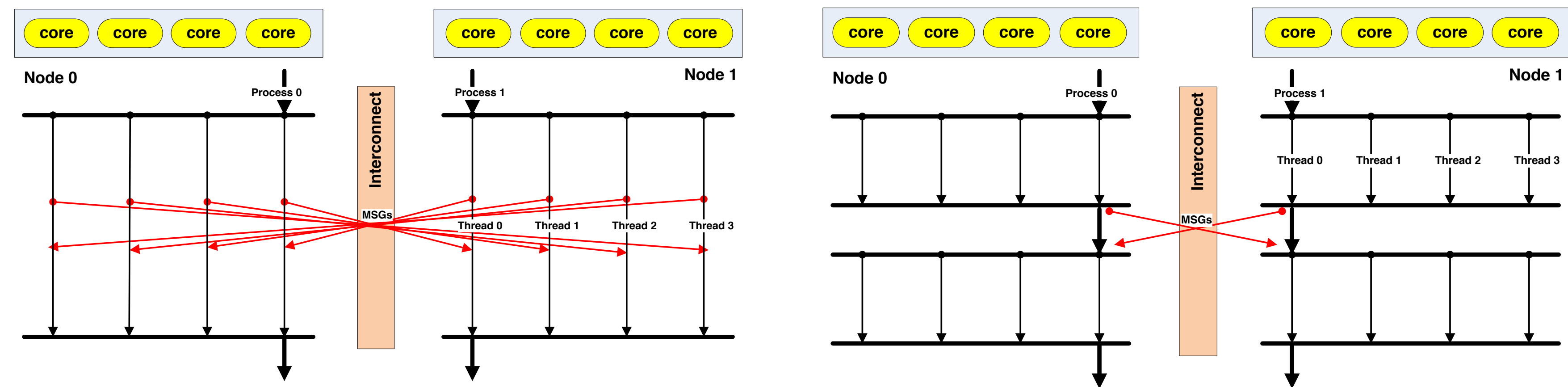


- **Observed: overheads can overlap: L can be negative**



EEL: end to end latency (instead of transport latency L)
g: minimum time between small message sends
G: additional gap per byte for larger messages

Communication and Manycore: the problem is the "+"

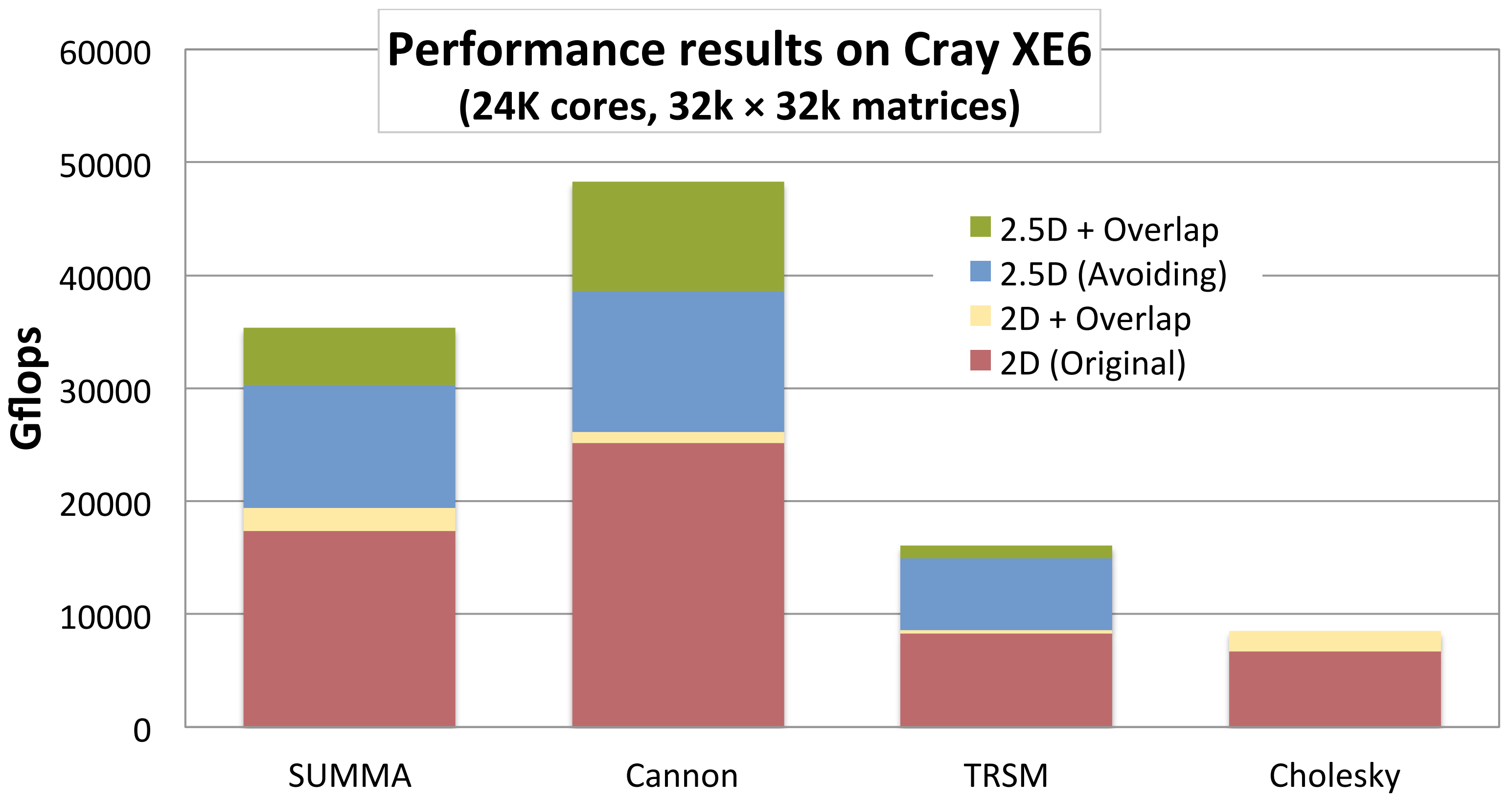


Ideal hybrid programming *Default hybrid programming*

- **MPI + X today:**
  - Communicate on one lightweight core
  - Reverse offload to heavyweight core
- **MPI stack may not run well on lightweight cores**
- **Issues preventing efficient interoperability:**
  - Addressability: can't name remote threads?
  - Separability: How to manage communication resources for independent paths
- **More feasible for 1-sided than 2-sided**

Khaled Ibrahim, ICS 2014

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)

Avoid Unnecessary Synchronization

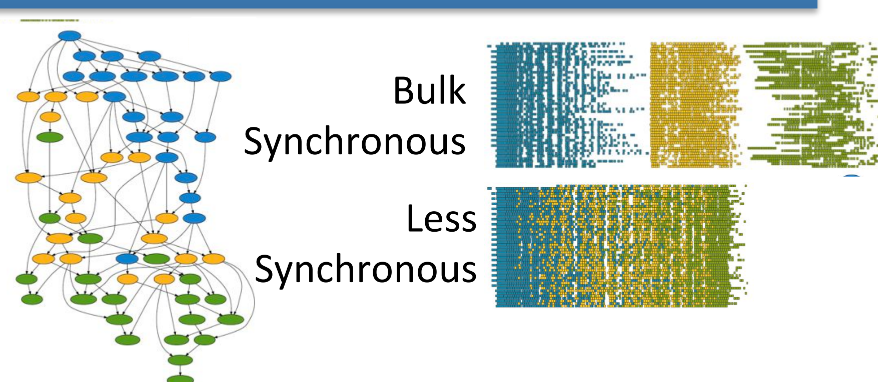
Sources of Unnecessary Synchronization

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



LAPACK: removing barriers ~2x faster (PLASMA)

Libraries

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

NWChem: most of barriers are unnecessary (Corvette)

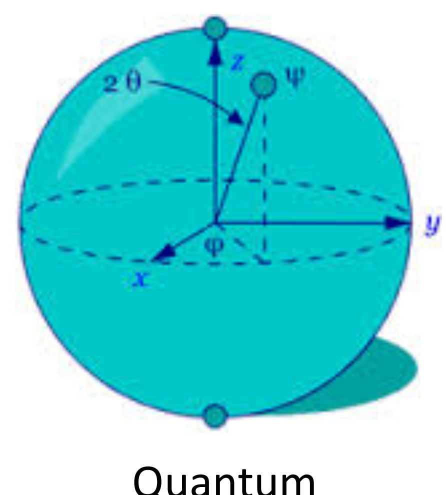
Accelerator Offload

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$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)
```

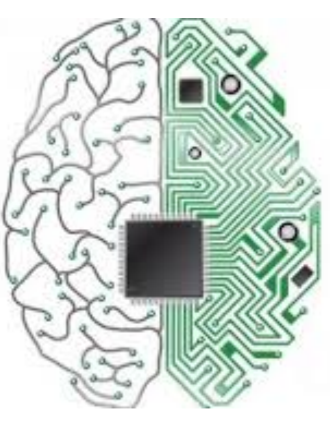
The transfer between host and GPU can be slow and cumbersome, and may (if not careful) get synchronized

Beyond Moore

Beyond Digital Computing Law

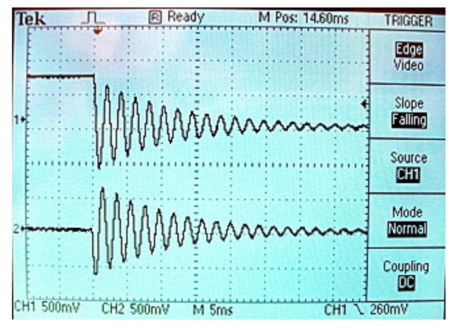


Quantum



Neuromorphic

Is there a new model of computing that is useful for science?

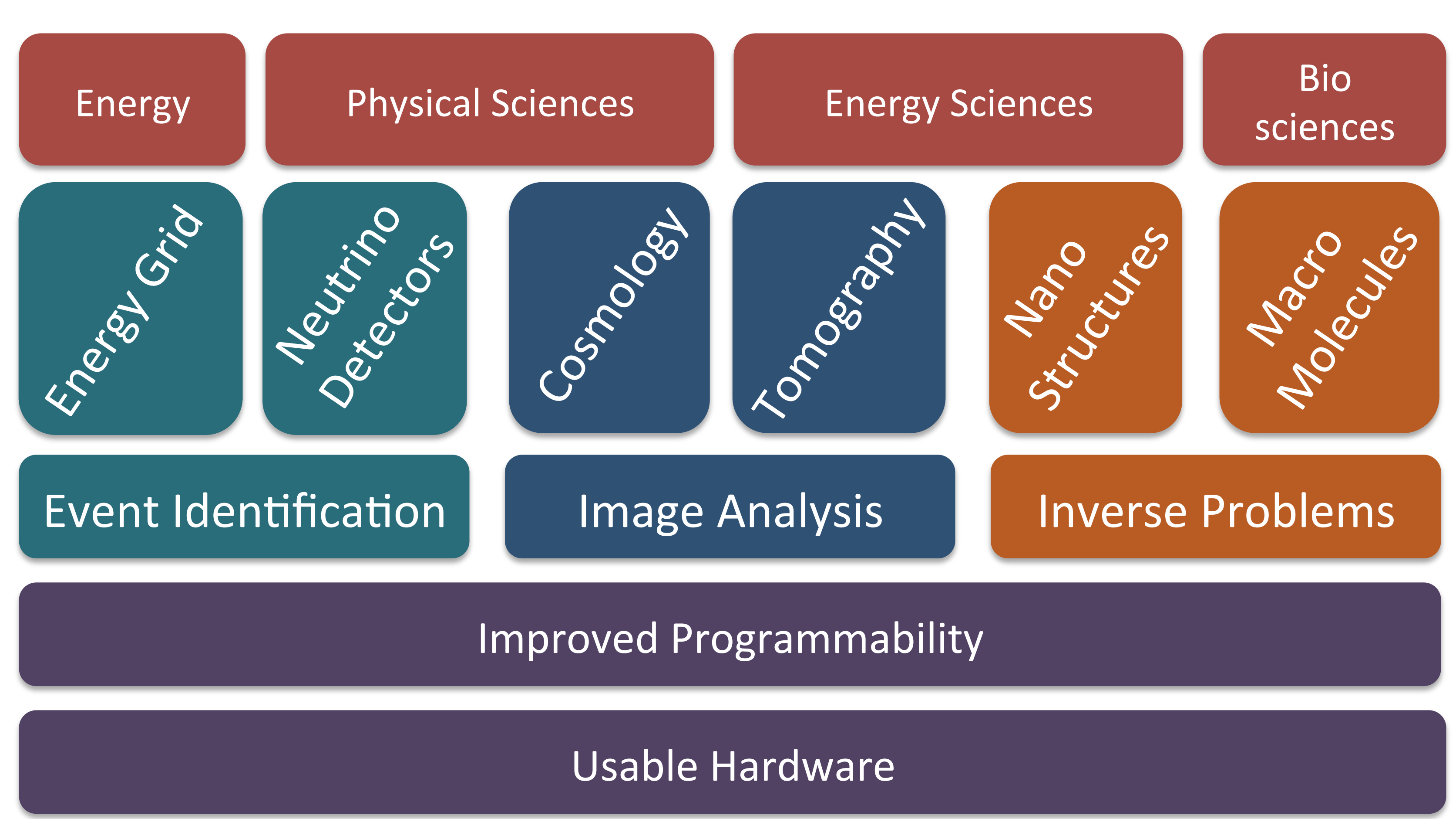


Analog

Are there ways of storing, transferring and computing on information that significantly reduce power?

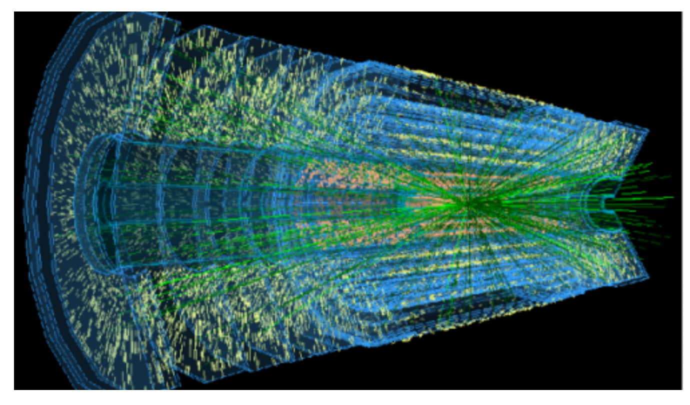
FPGAs

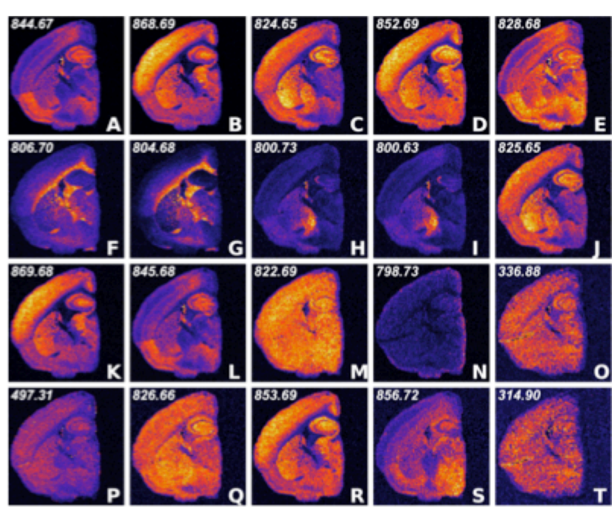
Science Applications of Neuromorphic Computing



Data processing with special purpose hardware

- General trend towards specialization for continued performance growth
- Data processing (on raw data) will be first in science





Particle Tracking with Neuromorphic chips

Deep learning processors for image analysis

Computing in Detectors

FPGAS for genome analysis

And can we also use these for simulation?

QUANTUM INFORMATION SCIENCE (QIS): AN APPLICATIONS ORIENTED VIEW

QIS

Computing (~100-100,000)

- Gate Based: Shor, Grover,…
- Adiabatic Quantum Computing
- Quantum Annealing

Simulations (~1-100)

- Chemical & Materials Science
- Theoretical Physics: Cosmology,...

Communication (~1, flying)

- Quantum Key Distribution
- Quantum Commitment

Metrology (~1-10)

- Precision measurements, squeezing
- Sensors (Magnetic, Charge, Light)

Irfan Siddiqi 2015

Electronic Structure Methods for Chemistry

Method	Scaling for N electrons
DFT	$O(N^2)$-$O(N^3)$
HF	$O(N^2)$-$O(N^4)$
MP2	$O(N^5)$
CISD, CCSD	$O(N^6)$
CCSD(T)	$O(N^7)$
CCSDT	$O(N^8)$
FCI	O(exp(N))

Full configuration interaction

Improving Quantum Algorithms for Quantum Chemistry

M. B. Hastings,[1,2] Dave Wecker,[2] Bela Bauer,[1] and Matthias Troyer[3]
[1] *Station Q, Microsoft Research, Santa Barbara, CA 93106-6105, USA*
[2] *Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052,*
[3] *Theoretische Physik, ETH Zurich, 8093 Zurich, Switzerland*

We present several improvements to the standard Trotter-Suzuki based algorithms used i
simulation of quantum chemistry on a quantum computer. First, we modify how Jordan-W
transformations are implemented to reduce their cost from linear or logarithmic in the nu
of orbitals to a constant. Our modification does not require additional ancilla qubits. Then
demonstrate how many operations c … r decrease i
parallel depth of the circuit, at the … number of q
required. Thirdly, we modify the t … ion, significa
reducing the error at given Trotter-S … the Hamilto
to reduce errors introduced by the … se technique
validated using numerical simulation … ic molecules

Scaling for N electrons on Quantum Device	
FC #1	$O(N^9)$
FC #2	$O(N^7)$
FC #3	$O(N^{5.5})$

Quantum Simulation:

“What quantum computers do in their sleep” [Scott Aaronson]

Open questions in theory and practice

End Game for Moore's Law

More parallelism

More specialization

(hardware and programming models)

Less communication

Understand your applications!

Thank you!
BERKELEY LAB
U.S. DEPARTMENT OF
ENERGY
Office of
Science