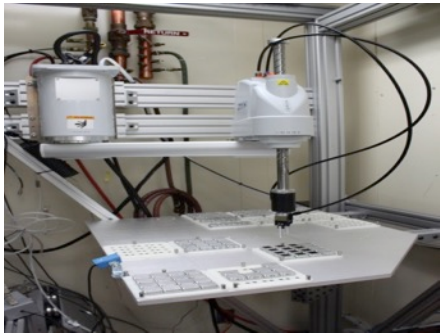
Why Languages Matter
More than Ever

Kathy Yelick
Lawrence Berkeley National Laboratory
and UC Berkeley

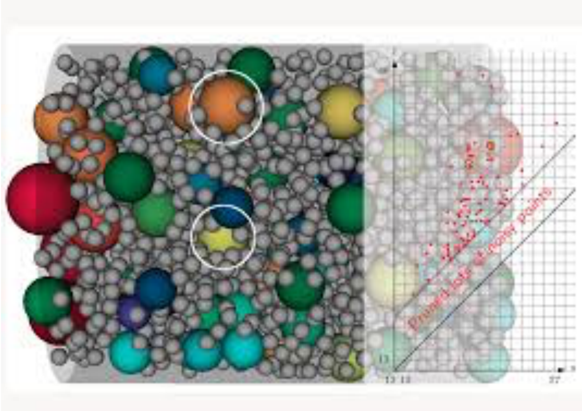
A Focus on Science

The changing nature of scientific discovery

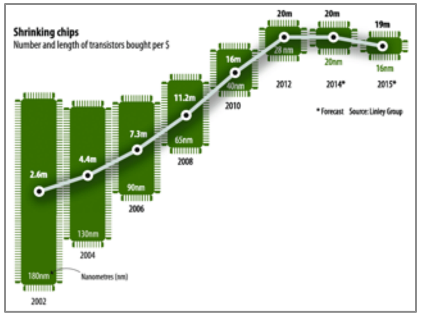
Science at the boundary of simulation and observation



Automation, robotics and new input devices

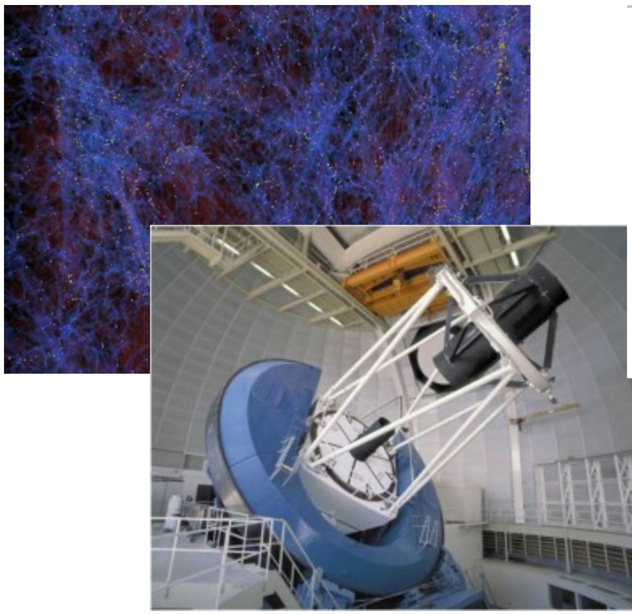


New methods for analyzing and modeling data

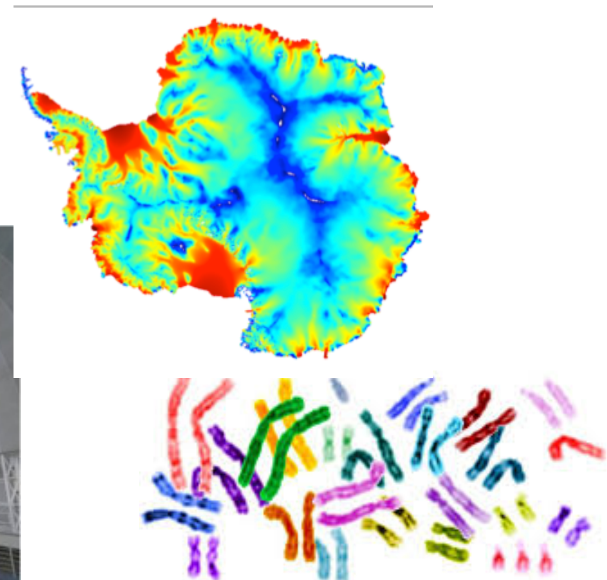


More computing for more complex science questions

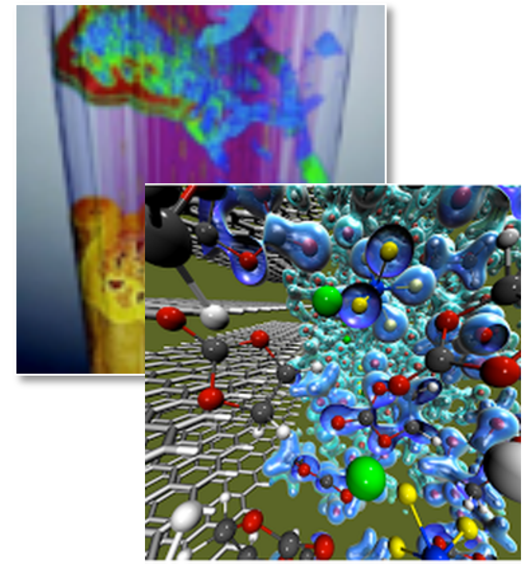
Science at the Boundary of Simulation and Observation



Cosmology



Environment

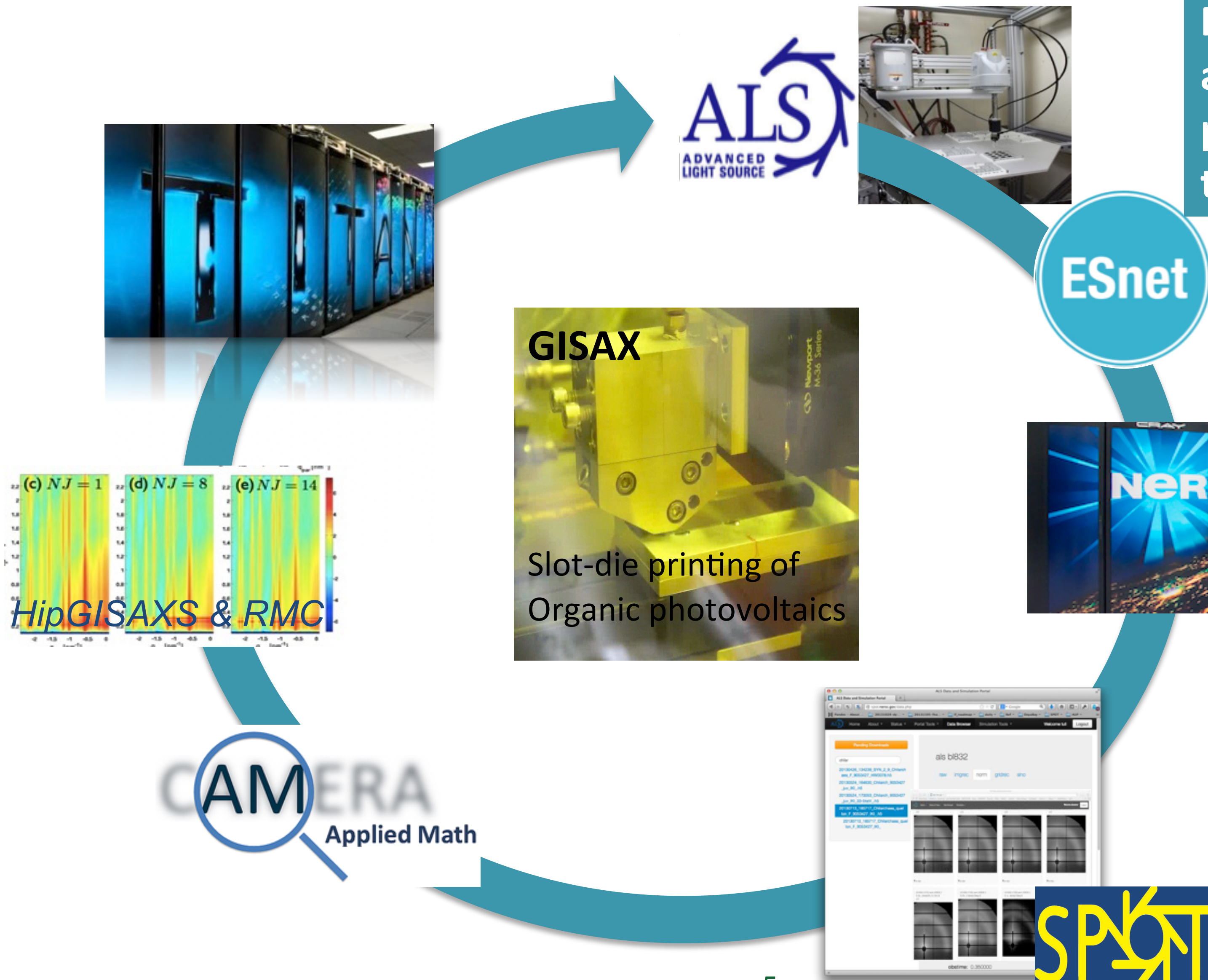


Materials

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

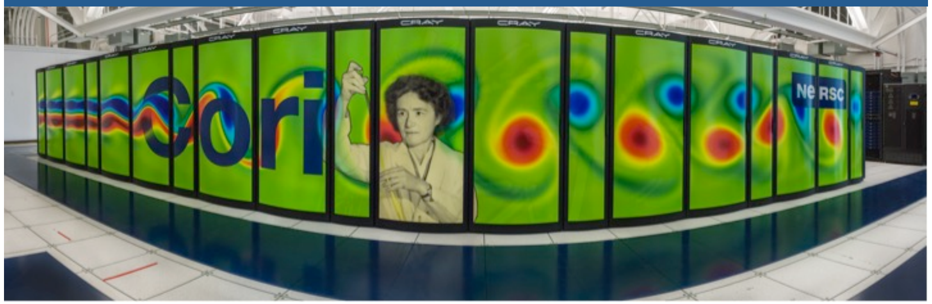
Computing, experiments, networking and expertise in a "Superfacility" for Science

More robotics and specialized processors at the edge



NERSC: Planning Beyond Exascale

Cori at NERSC



- 7,000 users and 2,400 publications in 2017
- Cori production started July 1, 2017

NERSC Future System Sketch

CPUs

Broad workload

GPUs

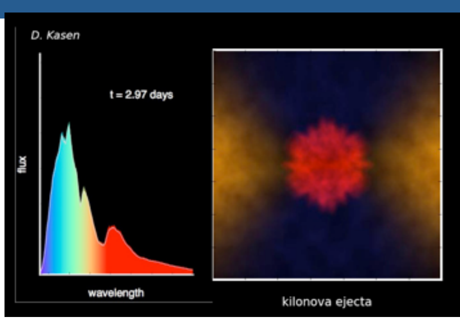
Image Analysis
Deep Learning
Simulations

Flexible Interconnect

Remote data can stream directly into system

Can integrate FPGAs and other accelerators

Simulation

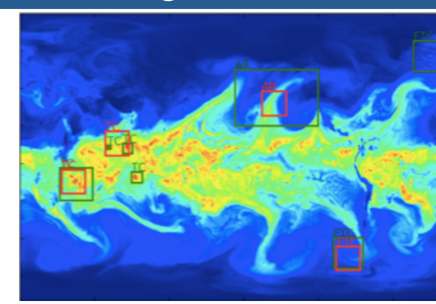


Simulations of neutron star merger shows light spectrum seen in LIGO

Analytics

Clustering of 388M microbial proteins reveals new clusters

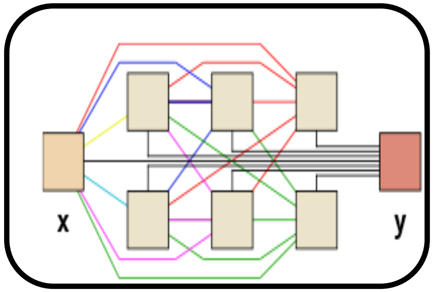
Learning

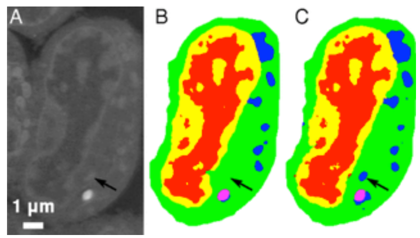


NERSC and Intel have scaled Deep Learning to 15PF on Cori

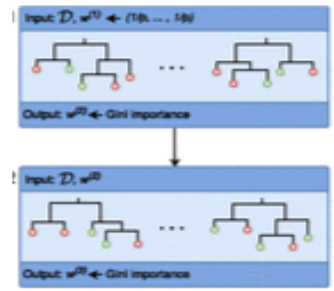
Scalable and Interpretable Machine Learning for Science

Interpretable Algorithms Driven by Breadth of Science



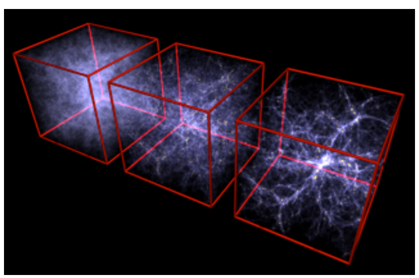


Mixed-scale CNN reduces cost and simplifies model; use on tomographic image segmentation, PNAS 2017

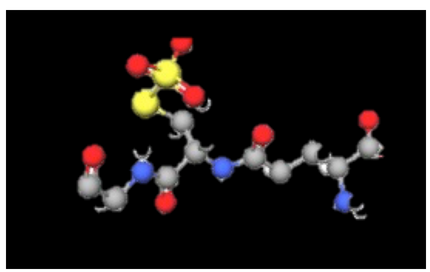


Iterative random forest finds high order interactions for transcriptome regulation in drosophila, PNAS 2018

Current and emerging applications Berkeley Lab



Cosmology:
- Replace simulations with derived models using Generative Adversarial Neural Nets

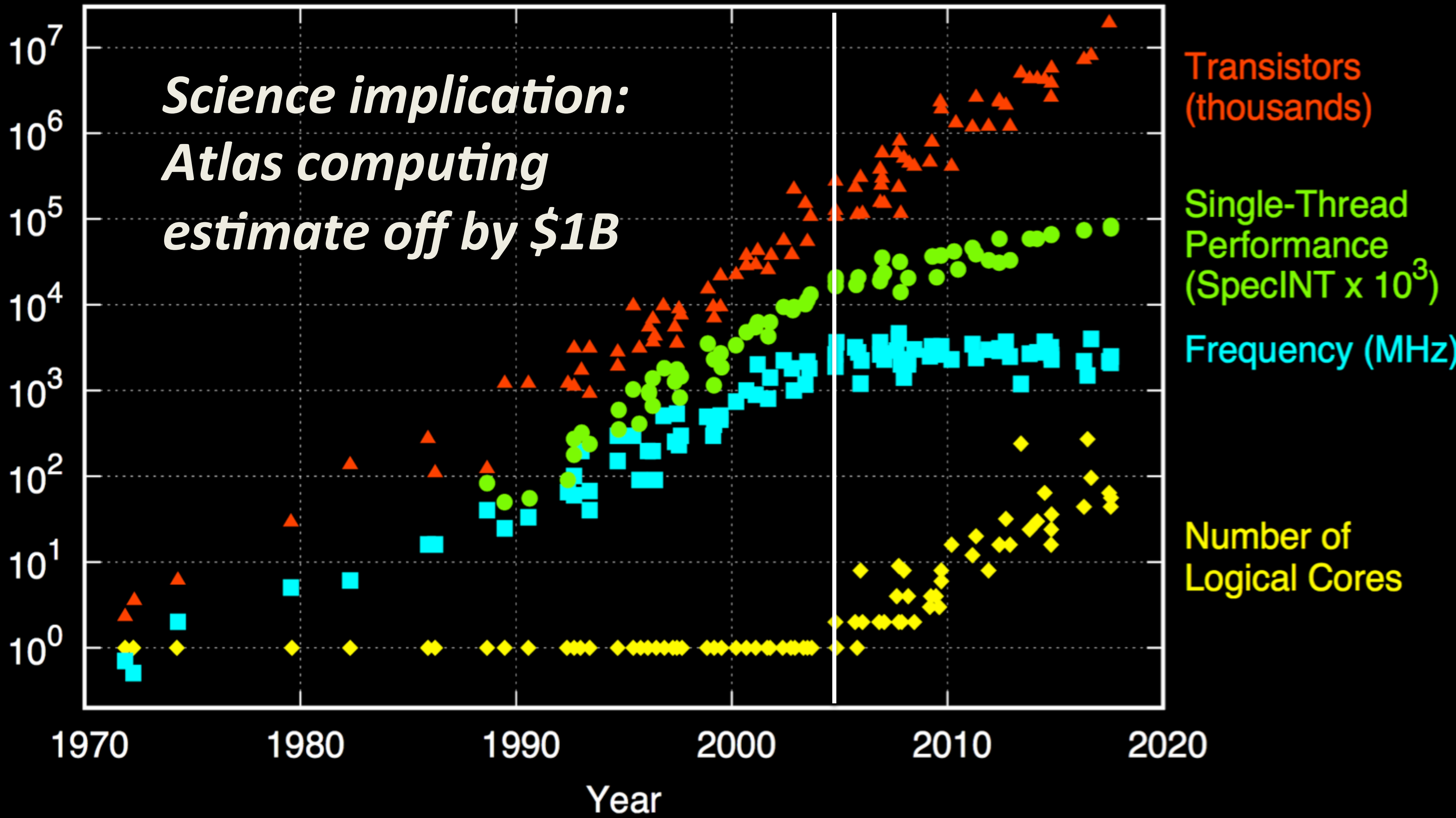


Materials and Chemistry:
- Explore materials universe with CNNs tailored to 3D materials and symmetries

Biology :
- Multi-model analysis of microbiome, images, etc.

Applied Energy:
- Gradient boosting method for building energy use

Dennard Scaling is Dead; Moore's Law Will Follow



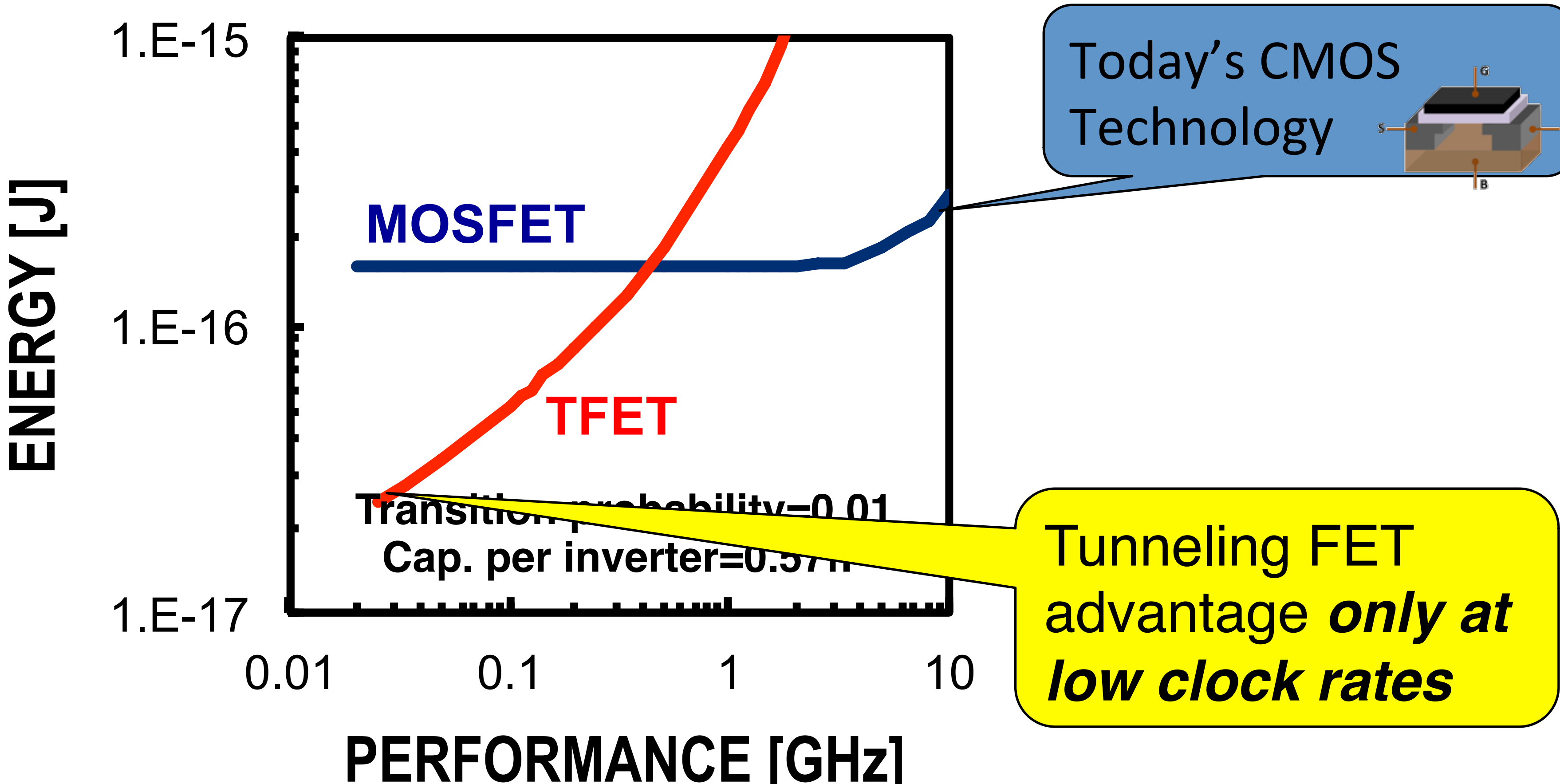
M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, and K. Rupp

Alternatives to Conventional MOS

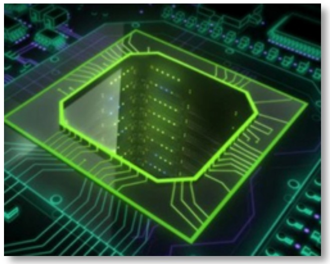
(all require lower clock rate, and much more parallelism)

Energy-Performance Comparison

(30-stage fanout-4 inverter chains)



Specialization: The End Game for Moore's Law

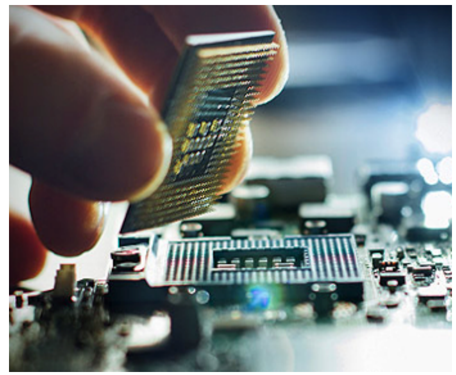


NVIDIA builds deep learning appliance with P100 Tesla's

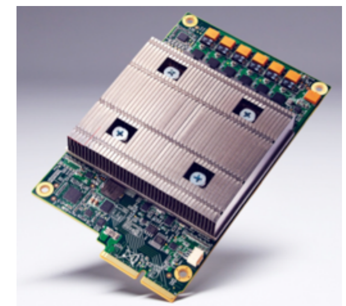
FPGAs in Microsoft cloud

RISC-V is an open hardware platform

RISC-V



Intel buys deep learning startup, Nervana



Google designs its own Tensor Processing Unit (TPU)

Specialization Spectrum

Full Custom · *Open ISA* · *FPGA* · *FPGA + standard ops* · *Old GPU* · *GPGPUs* · *Simple cores* · *High end cores*

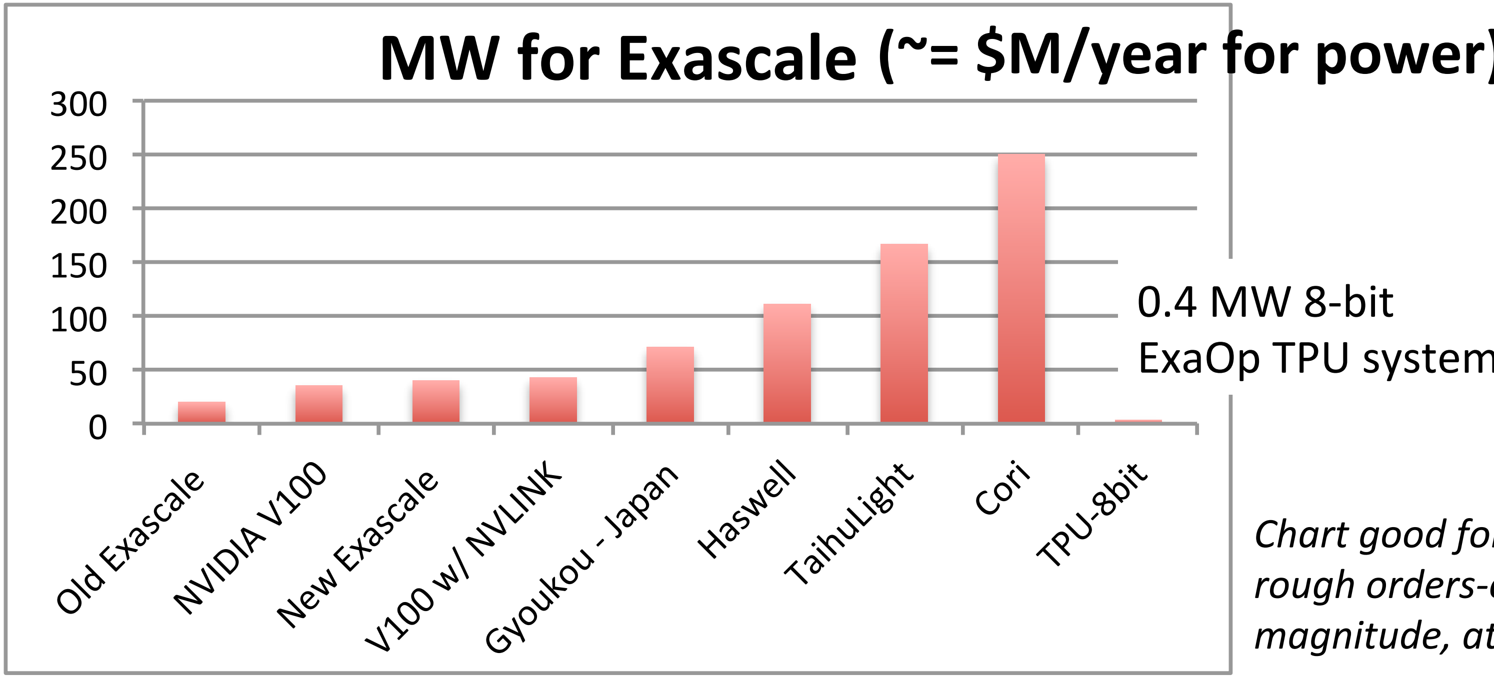
China (Sunway), Japan (ARM), and Europe/Barcelona (RISC-V) are doing this in HPC

Ancient Myths of Specialization

Research needed

- **Outperformed by general purpose processors**
  - Trends for general purpose essentially stopped
- **Too expensive**
  - Reduce cost with open source hardware and tools
  - Equation changed: $600M systems (w/ NRE)
- **Industry won't support them**
  - They already are, when benefits are high enough
- **Too hard to program**
  - Basic compilers provided; DSLs and compilers needed
- **Facilities will not support them**
  - Piloting options at NERSC

Back-of-the-Envelope: Is this interesting?



0.4 MW 8-bit ExaOp TPU system

Chart good for rough orders-of-magnitude, at best

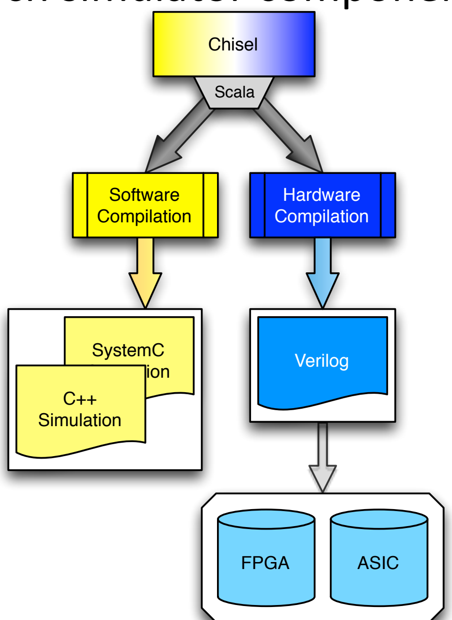
Notional exascale system of TPU-like processors:
2,300 GOPS/W →? 288 GF/W (dp) → a 3.5 MW Exaflop system!

- **Could we use TPU-like ideas for Science?**

Open Hardware (Synthesis & Simulation)

Chisel

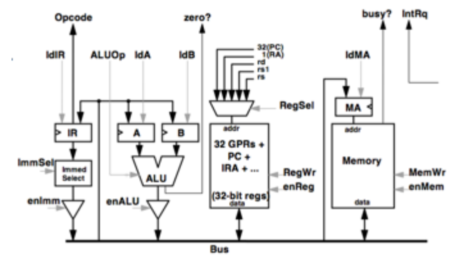
DSL for rapid prototyping of circuits, systems, and arch simulator components

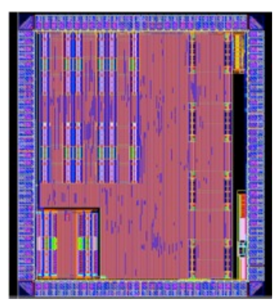


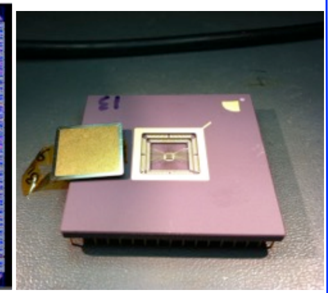
Back-end to synthesize HW with different devices Or new logic families

RISC-V

Open Source Extensible ISA/ Cores



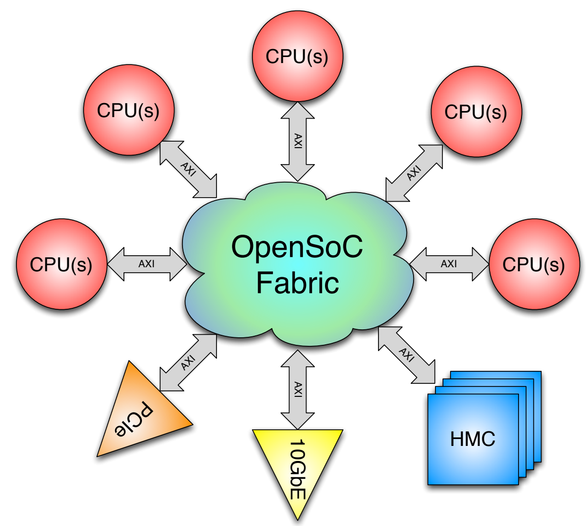




Re-implement processor With different devices or Extend w/accelerators

OpenSOC

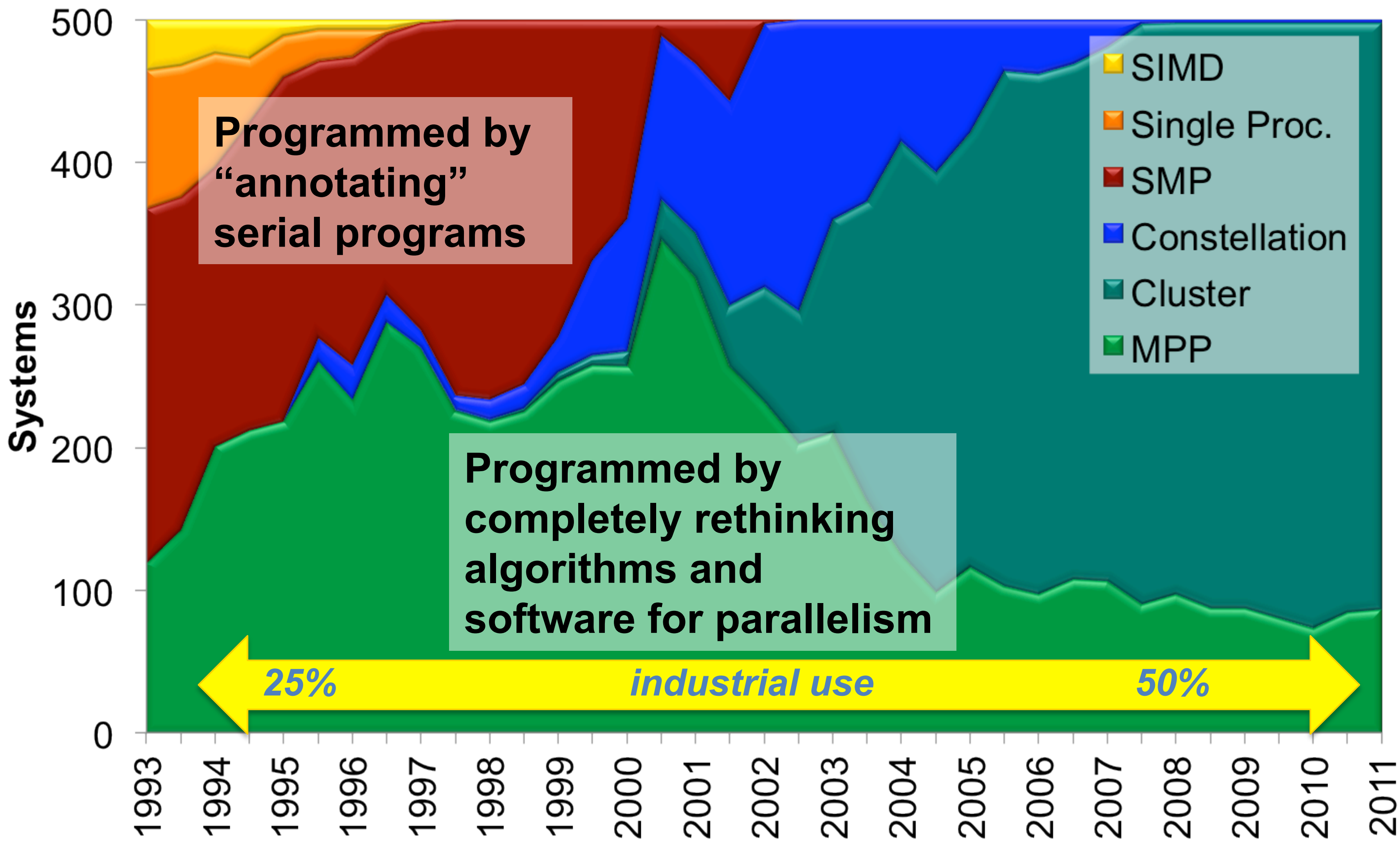
Open Source fabric To integrate accelerators And logic into SOC



Platform for experimentation with specialization to extend Moore's Law

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

HPC: From Vector Supercomputers to Massively Parallel Systems



The end of Relaxed Programming

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

Why Consider New Languages at all?

Syntax

- High level, elegant syntax
- Improve programmer productivity

Semantics

- Static analysis can help with correctness
- We need a compiler (front-end)

Performance

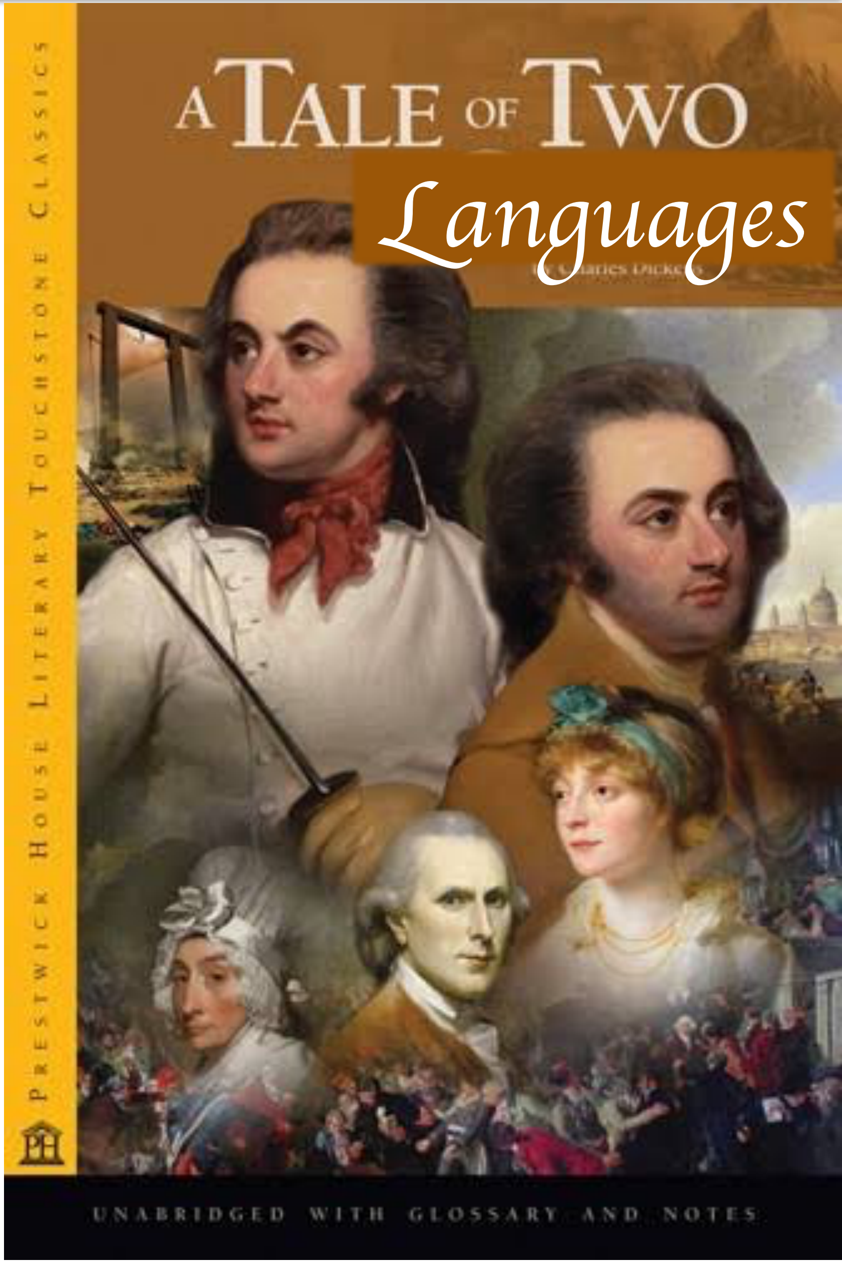
- If optimizations are needed to get performance
- We need a compiler (back-end)

Algorithms

- Language defines what is easy and hard
- Influences algorithmic thinking

Chapel and UPC

- **Partitioned Global Address Space Languages**
  - Communication by remote one-sided access
  - Locality control
  - Remote atomics…eventually in UPC
- **Parallelism**
  - UPC:
    - SPMD, i.e., parallel by default
    - Fixed scale.. eventually with teams
    - Main runs on all threads
  - Chapel:
    - Serial by default
    - Task and Data parallel;
    - main executed on local #0



Berkeley UPC Project Goals of Time

2001-2004: A Portable UPC Compiler

- UPC was (incorrectly) viewed as a language that required shared memory hardware or only ran on Cray machines
- The Berkeley UPC compiler showed it could run on clusters with a lightweight runtime and that source-to-source translation was reasonable

2005-2008: UPC is a High Performance Language

- Conventional wisdom: UPC is more productive than MPI but we should expect it to be slower (maybe by 2x)
- Even on clusters without global address space support, UPC can outperform MPI on some microbenchmarks and apps
- Surprise: bisection bandwidth problems, not just latency-limited

2008-2012: UPC for multicore & hybrid multicore / clusters

- Focus on on-node performance and mixed shared/distributed
- Realization: hierarchical algorithms are necessary even in a single programming model
- Surprise: processes are faster than threads on-node

2013-2016: Killer application(s) for science

- Hummingbird; LU factorization
- Genome assembly

On to UPC++ and UPC++ 2.0

Bringing Users Along: UPC Experience

1991
Active Msgs are fast

1992
First AC (accelerators + split memory)

1992 First Split-C (compiler class)

1993
Split-C funding (DOE)

1997
First UPC Meeting

"best of" AC, Split-C, PCP

2001
First UPC Funding

2002 GASNet Spec

Other GASNet-based languages

2001
gcc-upc at Intrepid

2003 Berkeley Compiler release

2006
UPC in NERSC procurement

2010
Hybrid MPI/UPC

- **Ecosystem:**
  - Users with a need (fine-grained random access)
  - Machines with RDMA (not full hardware GAS)
  - Common runtime; Commercial and free software
  - Sustained funding and Center procurements
- **Success models:**
  - Adoption by users: vectors → MPI, Python and Perl, UPC/CAF
  - Influence traditional models: MPI 1-sided; OpenMP locality control
  - Enable future models: Chapel, X10,…

Why Consider New Languages at all?

Syntax

- High level, elegant syntax
- Improve programmer productivity

Semantics

- Static analysis can help with correctness
- We need a compiler (front-end)

Performance

- If optimizations are needed to get performance
- We need a compiler (back-end)

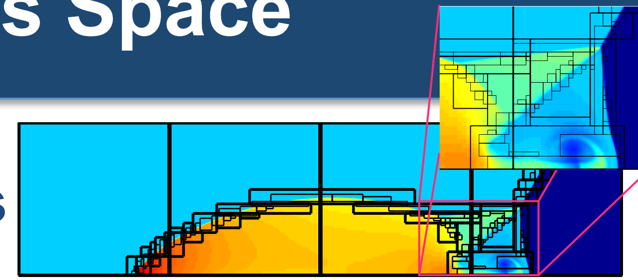
Algorithms

- Language defines what is easy and hard
- Influences algorithmic thinking

Is Chapel High Level Enough?

If not, should you pick a particular domain to support really well?

Arrays in a Global Address Space

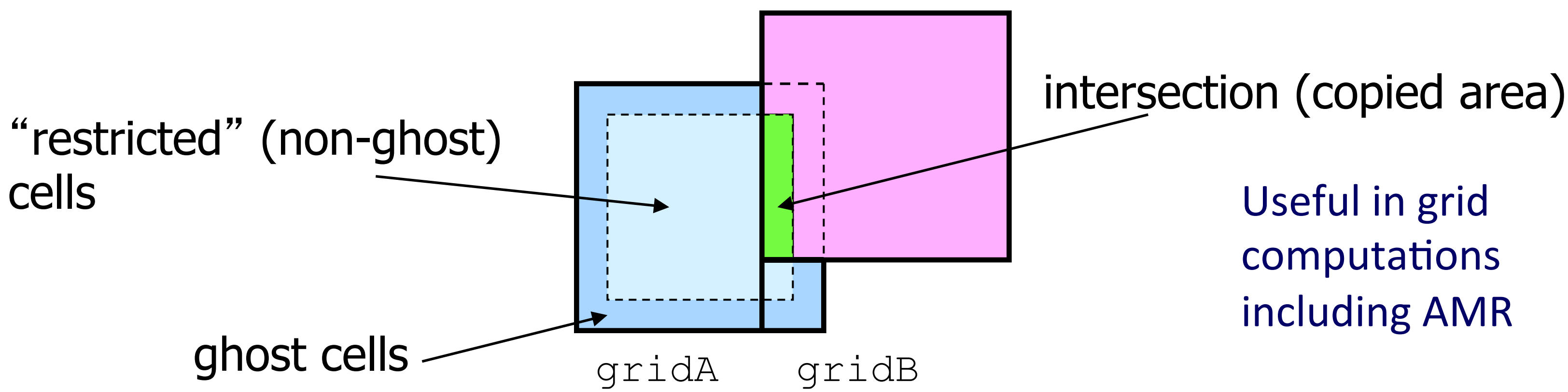


- **UPC++ (1.0) included Titanium Arrays**
- **Key features of Titanium arrays**
  - Generality: indices may start/end and any point
  - Domain calculus allow for slicing, subarray, transpose and other operations without data copies
- **Use domain calculus to identify ghosts and iterate:**

```
foreach (p in gridA.shrink(1).domain()) ...
```

- **Array copies automatically work on intersection**

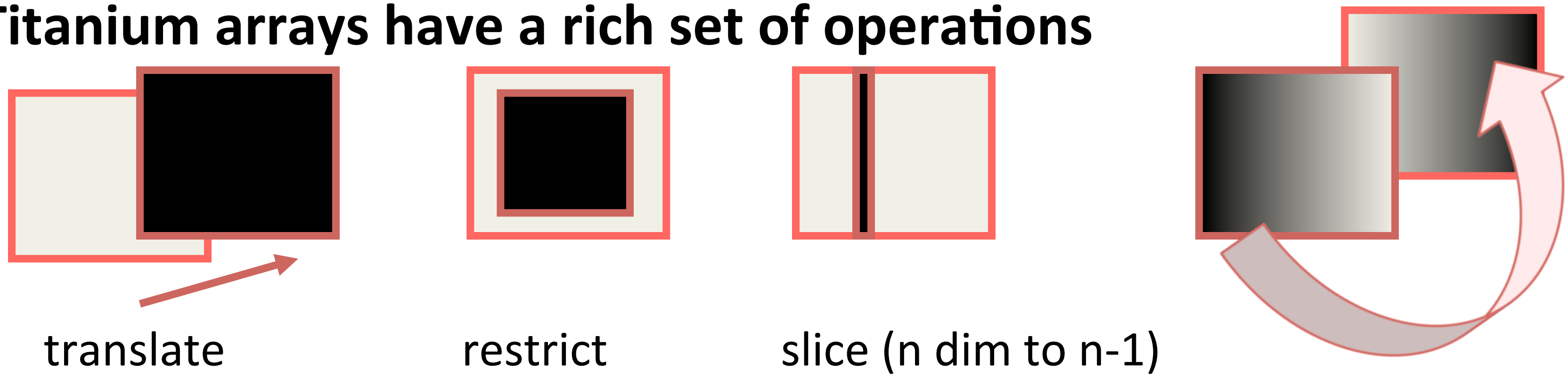
```
gridB.copy(gridA.shrink(1));
```



Useful in grid computations including AMR

Multidimensional Arrays in UPC++ (and Titanium)

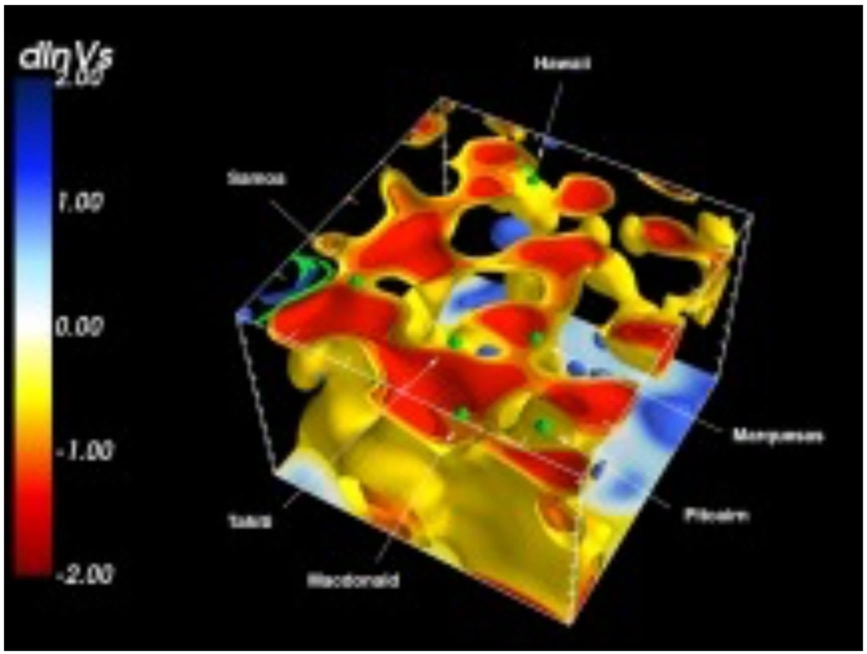
- **Titanium arrays have a rich set of operations**



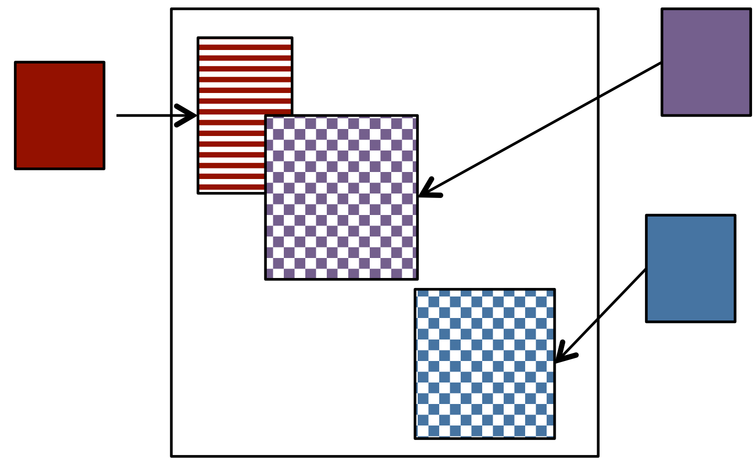
- **None of these modify the original array, they just create another view of the data in that array**
- **You create arrays with a RectDomain and get it back later using A.domain() for array A**
  - A Domain is a set of points in space
  - A RectDomain is a rectangular one
- **Operations on Domains include +, -, * (union, different intersection)**

Data Fusion in UPC++: Can Chapel do this?

- Seismic modeling for energy applications “fuses” observational data into simulation
- With UPC++ “matrix assembly” can solve larger problems



Solving previously unsolvable problems

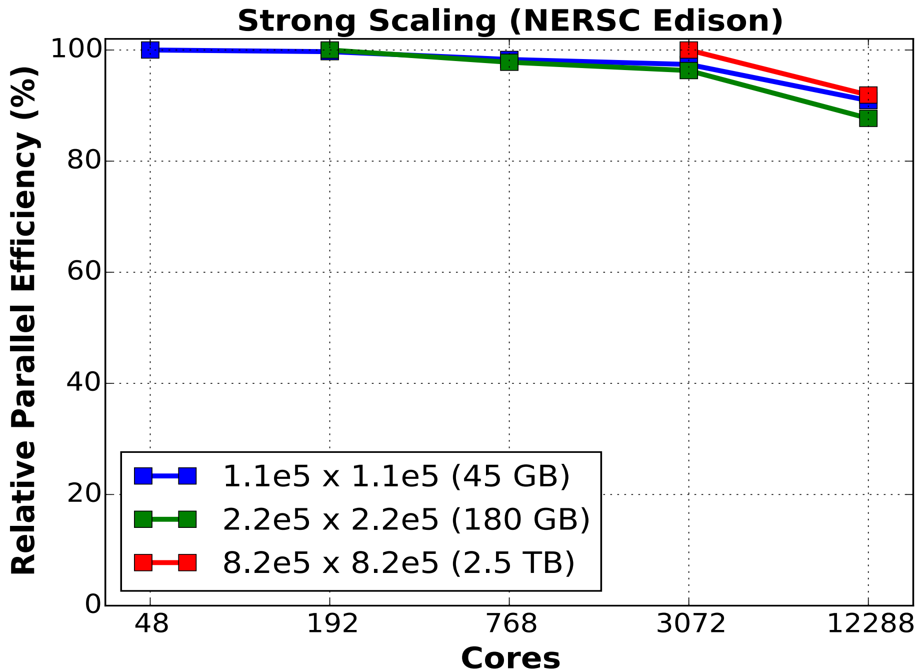


First ever sharp, three-dimensional scan of Earth’s interior that conclusively connects plumes of hot rock rising through the mantle with surface hotspots that generate volcanic island chains like Hawaii, Samoa and Iceland.

DEGAS

French and Romanowicz use code with UPC++ phase to compute *first ever* whole-mantle global tomographic model using numerical seismic wavefield computations (F & R, 2014, GJI, extending F et al., 2013, Science).

Application Challenge: Data Fusion in UPC++



Distributed Matrix Assembly

- Remote asyncs with user-controlled resource management
- Remote memory allocation
- Team idea to divide threads into injectors / updaters
- 6x faster than MPI 3.0 on 1K nodes

→ **Improving UPC++ team support**

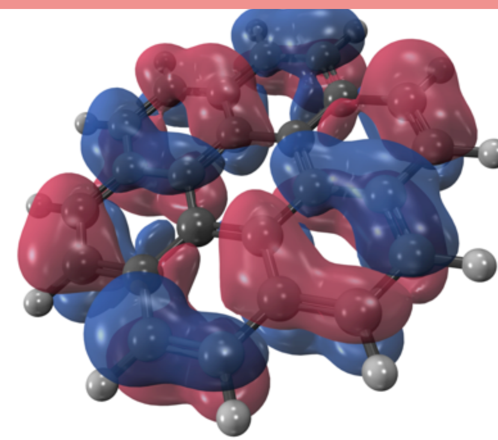
DEGAS

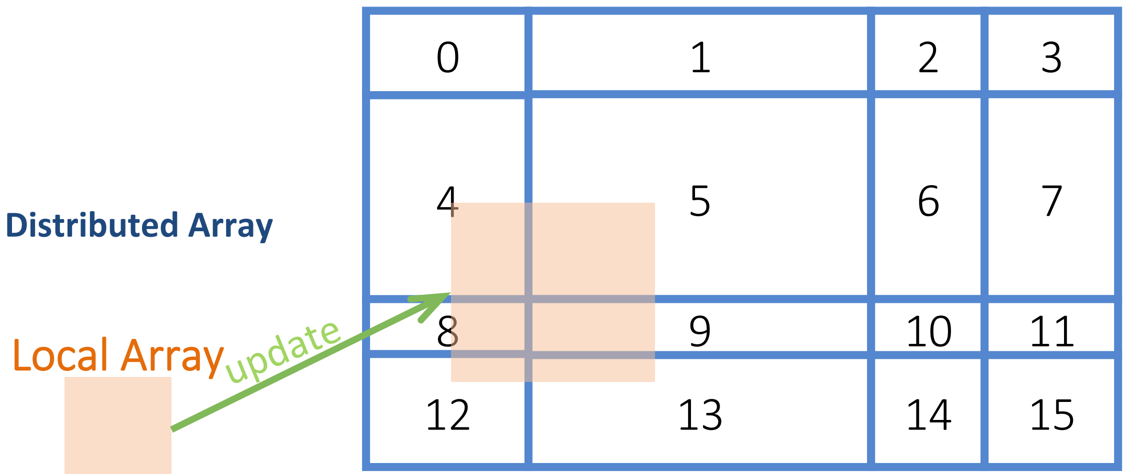
See French et al, IPDPS 2015 for parallelization overview.

Irregular Matrix Transpose: Can Chapel do this?

- Hartree Fock example (e.g., in NWChem)
    - Inherent load imbalance
- UPC++
    - Work stealing and fast atomics
    - Distributed array: easy and fast transpose
- Impact
    - *20% faster than the best existing solution (GTFock with Global Arrays)*

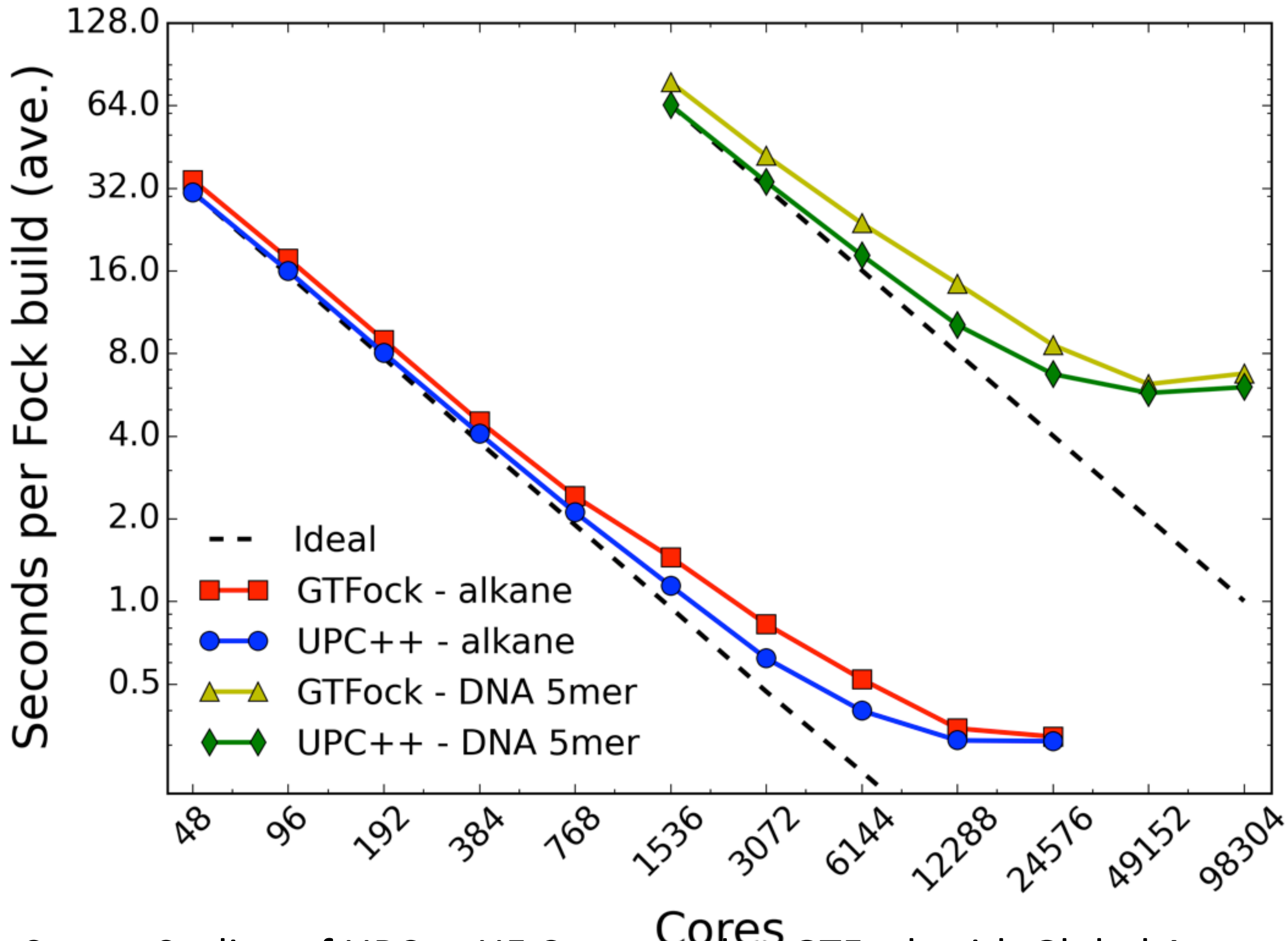
Increase scalability!





upc++
David Ozog , Amir Kamil , Yili Zheng, Paul Hargrove , Jeff R. Hammond, Allen Malony, Wibe de Jong, Katherine Yelick

Hartree Fock Code in UPC++



Strong Scaling of UPC++ HF Compared to GTFock with Global Arrays on NERSC Edison (Cray XC30)

David Ozog , Amir Kamil , Yili Zheng, Paul Hargrove , Jeff R. Hammond, Allen Malony, Wibe de Jong, Katherine Yelick

Why Consider New Languages at all?

Syntax

- High level, elegant syntax
- Improve programmer productivity

Semantics

- Static analysis can help with correctness
- We need a compiler (front-end)

Performance

- If optimizations are needed to get performance
- We need a compiler (back-end)

Algorithms

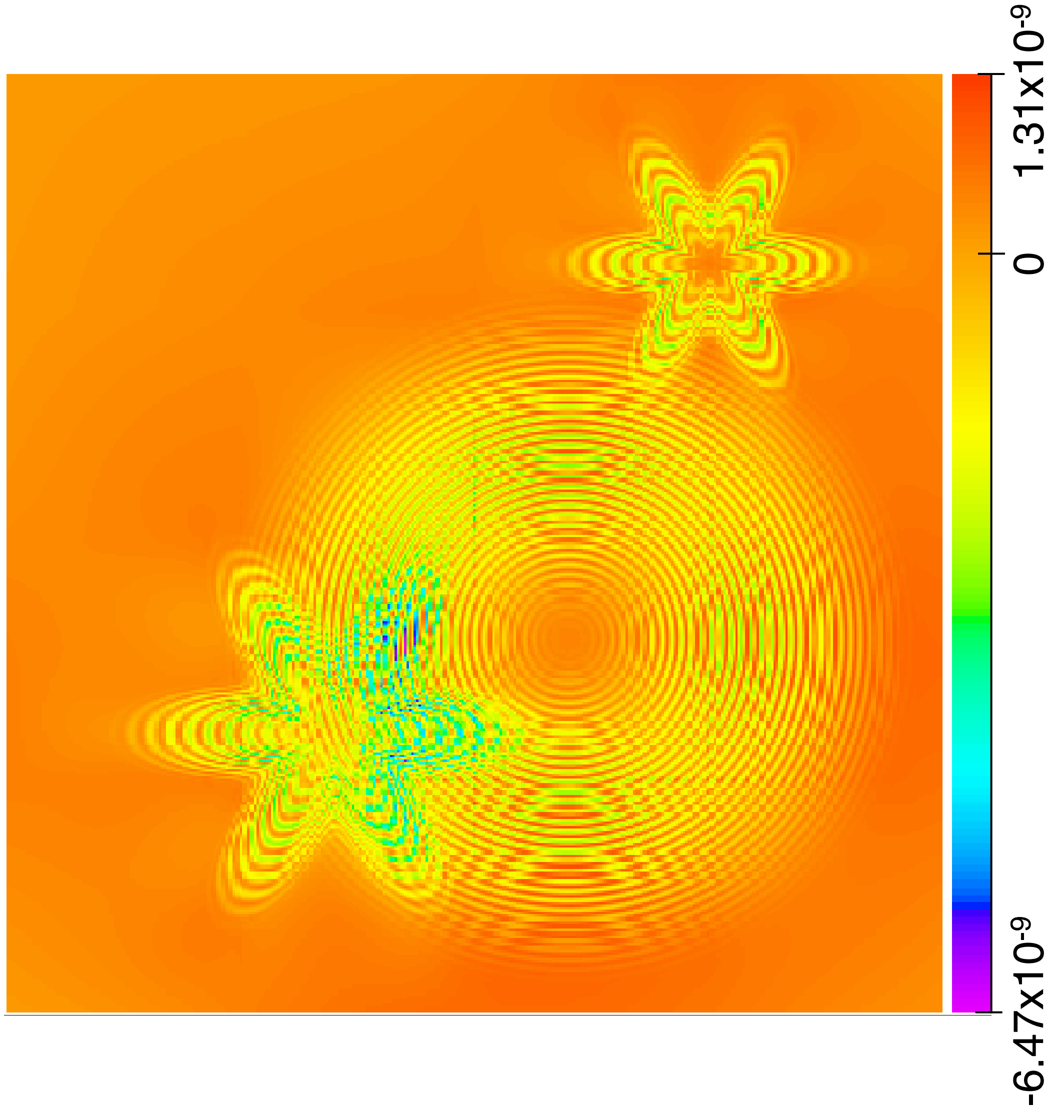
- Language defines what is easy and hard
- Influences algorithmic thinking

What are the big correctness issues in science?

Data races and debugging numerical code

Error on High-Wavenumber Problem

- **Charge is**
  - 1 charge of concentric waves
  - 2 star-shaped charges.
- **Largest error is where the charge is changing rapidly.** Note:
  - discretization error
  - faint decomposition error
- **Run on 16 procs**



Region-Based Memory Management

- **Memory management strategy in Titanium**
  - Need to organize data structures; Allocate set of objects
  - Delete them with a single explicit call (fast)
  - Save in principle; uses B-W collector for everything else
  - Captures references at node boundaries;
  - See David Gay's Ph.D. thesis

```
PrivateRegion r = new PrivateRegion();
for (int j = 0; j < 10; j++) {
   int[] x = new ( r ) int[j + 1];
   work(j, x);
}
try { r.delete(); }
catch (RegionInUse oops) {
    System.out.println("failed to delete");
  }
}
```

Why Consider New Languages at all?

Syntax

- High level, elegant syntax
- Improve programmer productivity

Semantics

- Static analysis can help with correctness
- We need a compiler (front-end)

Performance

- If optimizations are needed to get performance
- We need a compiler (back-end)

Algorithms

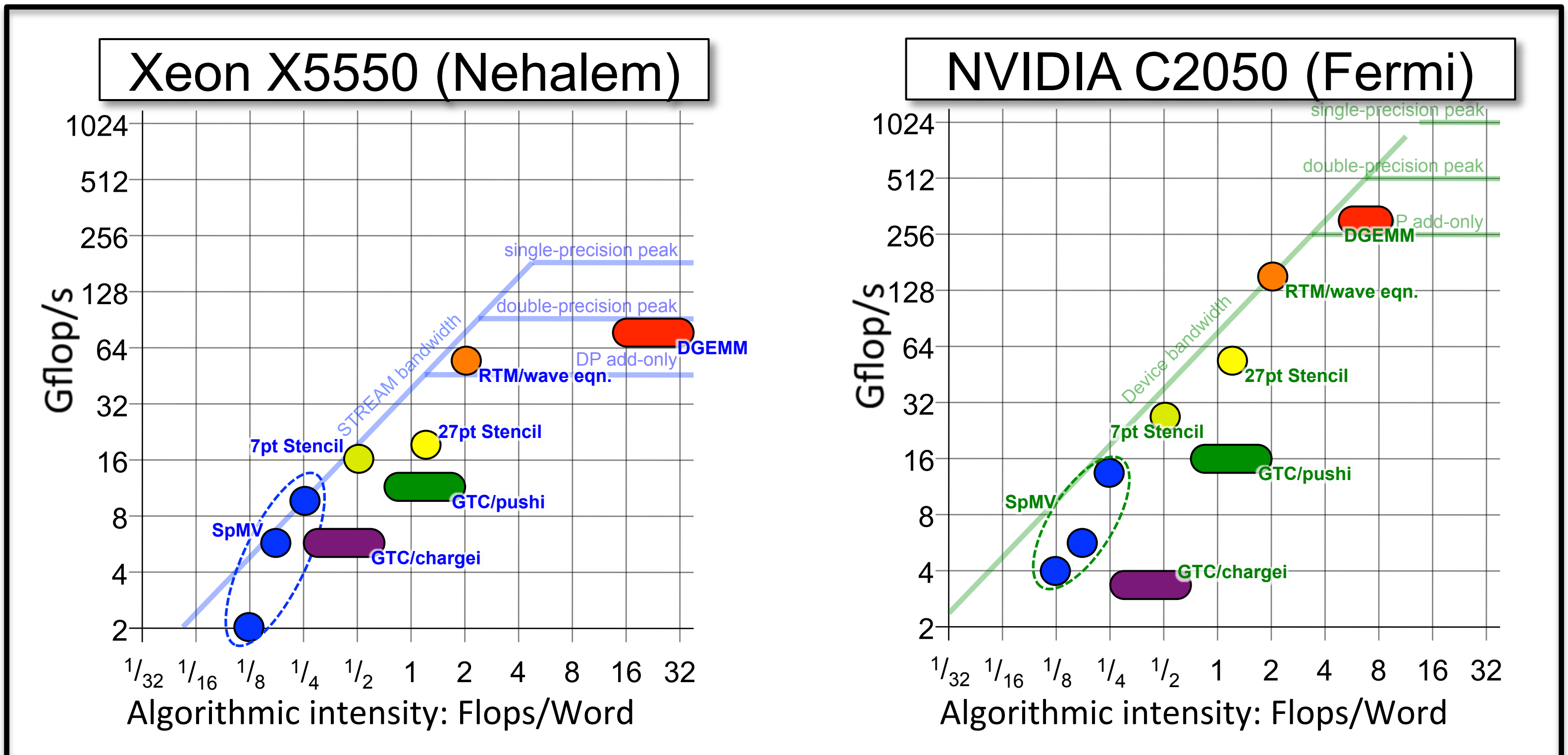
- Language defines what is easy and hard
- Influences algorithmic thinking

Is Chapel High Level Enough?

If not, should you pick a particular domain to support really well?

Autotuning: Write Code Generators

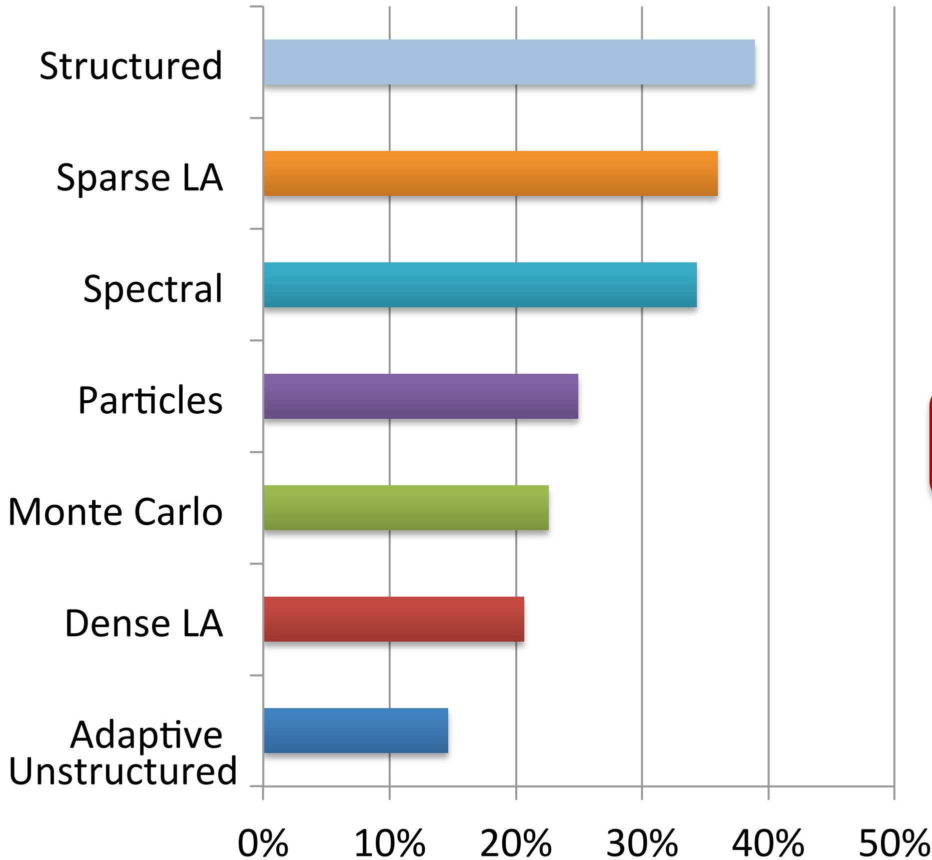
- **Two "unsolved" compiler problems:**
  - dependence analysis and — Domain-Specific Languages help with this
  - ✔ accurate performance models — Autotuning avoids this problem
- **Autotuners are code generators plus search**



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

Libraries vs. DSLs (domain-specific languages)

NERSC survey: what motifs do they use?



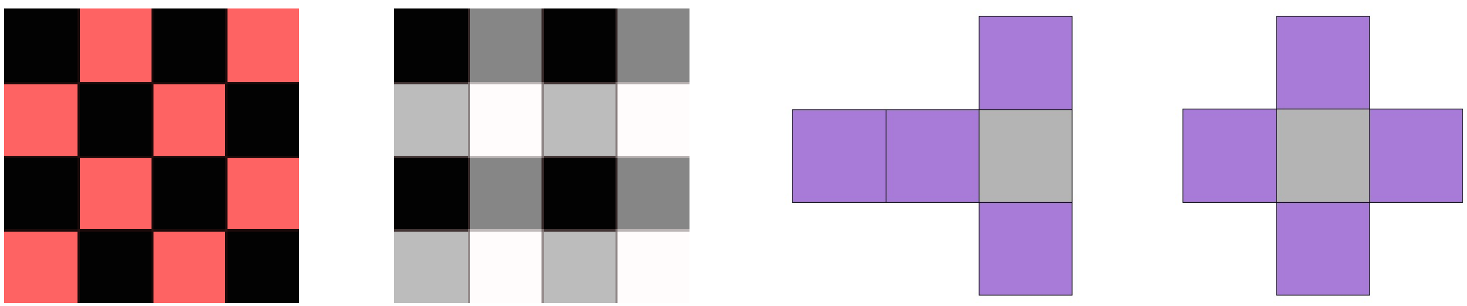
What code generators do we have?

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

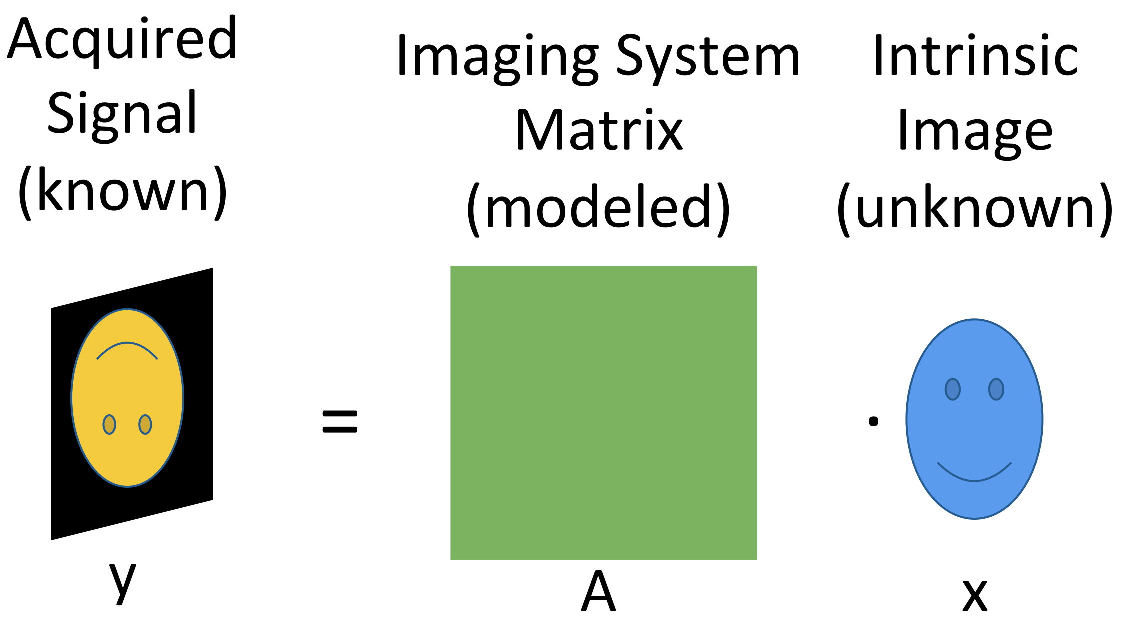
(b) 4-color tiling

(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**

Image Reconstruction as a Linear Inverse Problem



Dominated by linear operator evaluation

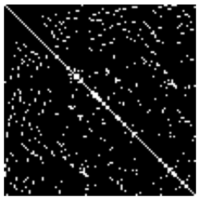
Conjugate gradient: $A^H y = A^H A x$

Convex optimization: minimize $| A^H A x - A^H y | + R(x)$

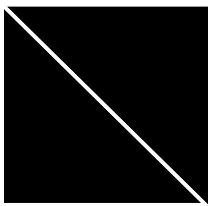
Driscoll et al, IPDPS 2018

Indigo: A DSL for Image Reconstruction

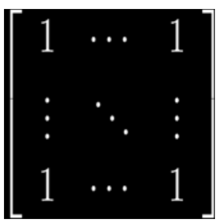
Matrices as building blocks



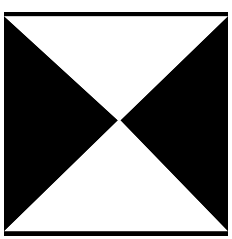
General Matrix



Identity Operator



OneMatrix Operator

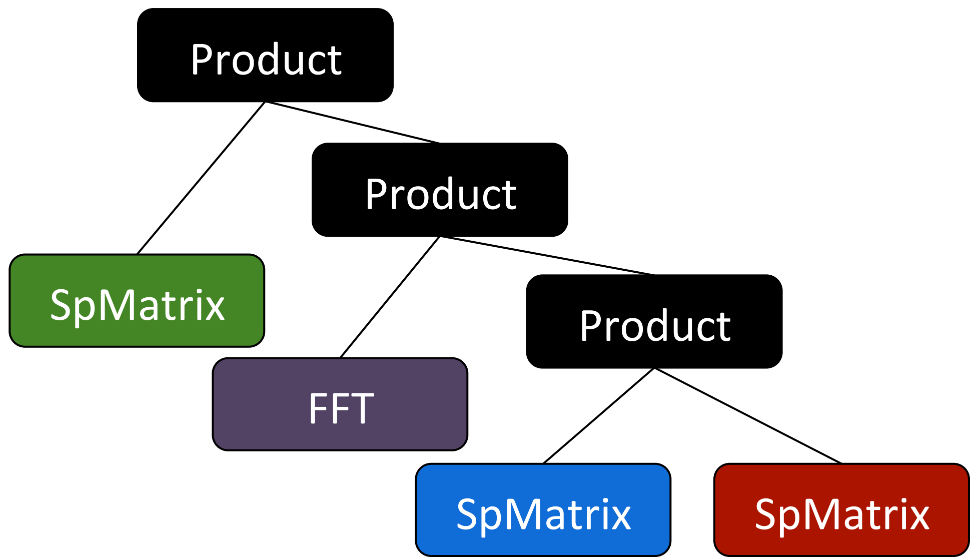


FFT Operator

Driscoll et al, IPDPS 2018

Operators at DGAs of matrix operations

- Arithmetic: Sum, Product, KroneckerProduct, Adjoint, Scale.
- Structural: VerticalStack, HorizontalStack, BlockDiagonal.

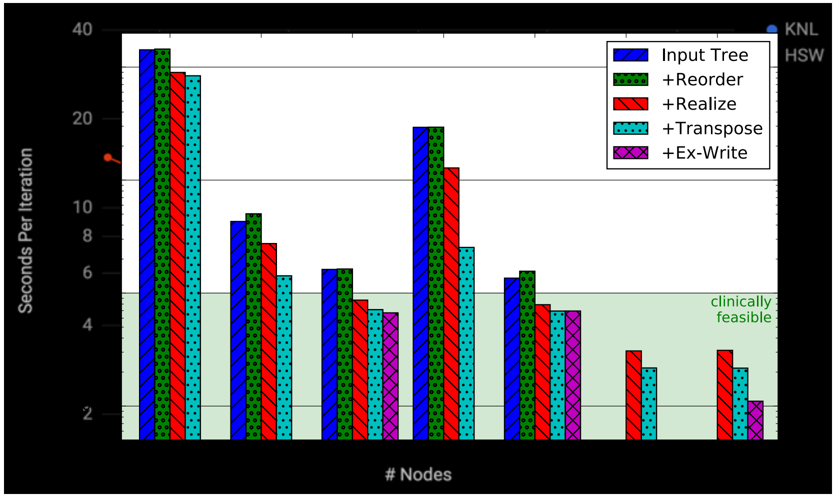


- Derived properties, e.g., 1 nonzero per row
- Transformations use the properties

Indigo Performance on GPUs, GPUs, Manycore

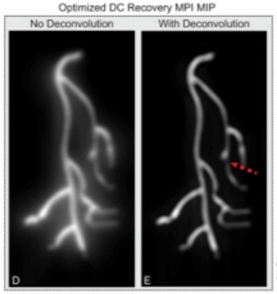
% peaks for for roofline, in this case memory bandwith peak

MRI reconstruction (Jiang, Lustig et al)



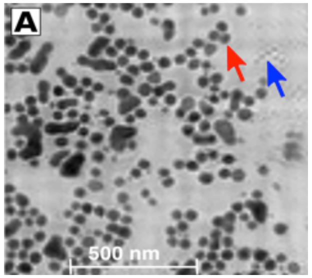
3 min goal (1 sec/iteration)

Magnetic Particle Imaging (Konkle et al 2015)



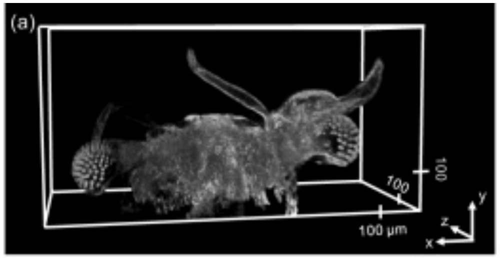
56% CPU peak,
9% KNL,
76% GPU.
258x over Numpy.

Ptychography (Marchesini 2016)



56% CPU peak,
9% KNL,
76% GPU.
258x over Numpy.

Phase-Space Microscopy (Liu et al 2017)



43% Peak CPU,
7% KNL,
46% GPU
186x over Numpy

Driscoll et al, IPDPS 2018

Why Consider New Languages at all?

Syntax

- High level, elegant syntax
- Improve programmer productivity

Semantics

- Static analysis can help with correctness
- We need a compiler (front-end)

Performance

- If optimizations are needed to get performance
- We need a compiler (back-end)

Algorithms

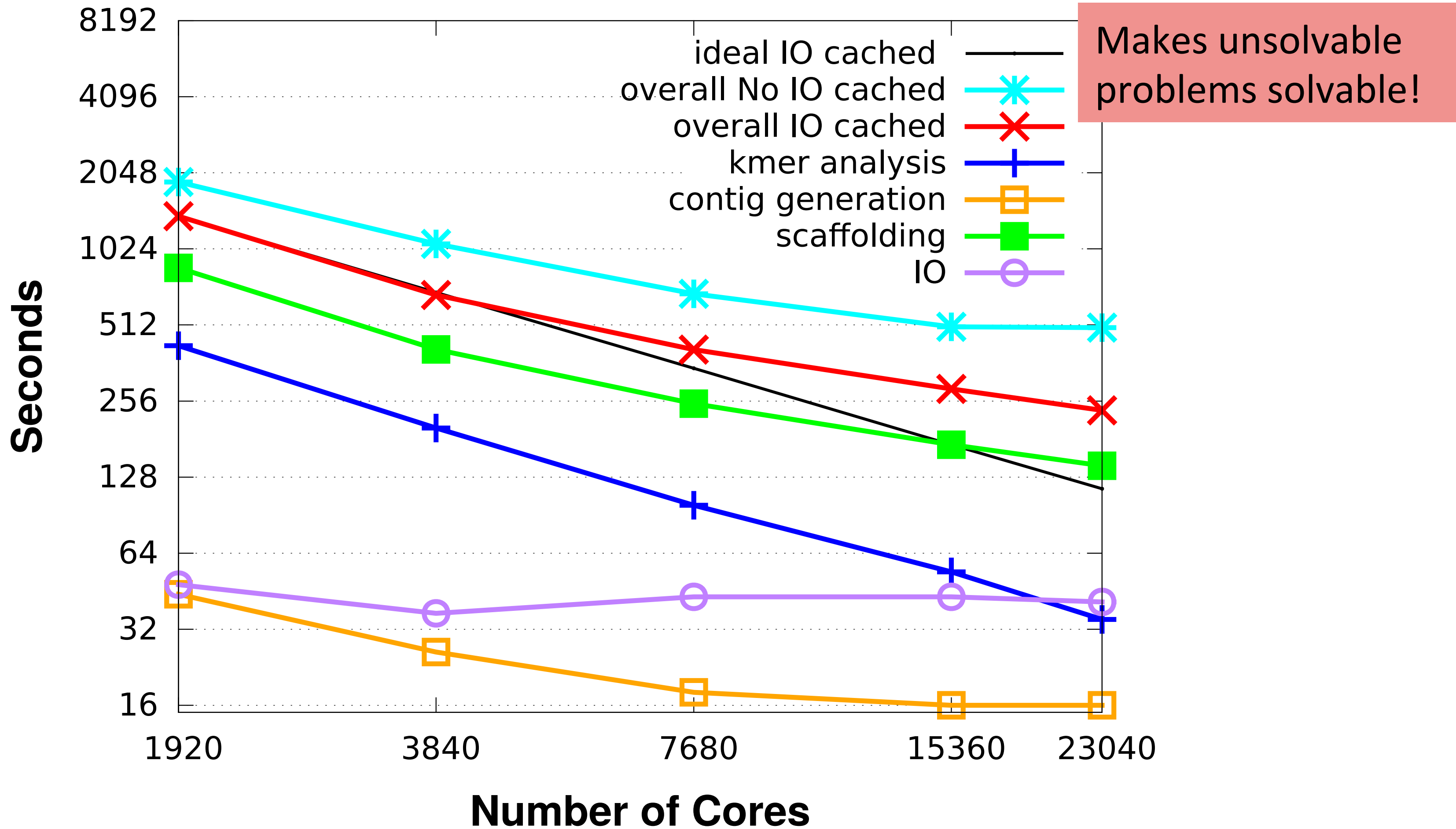
- Language defines what is easy and hard
- Influences algorithmic thinking

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

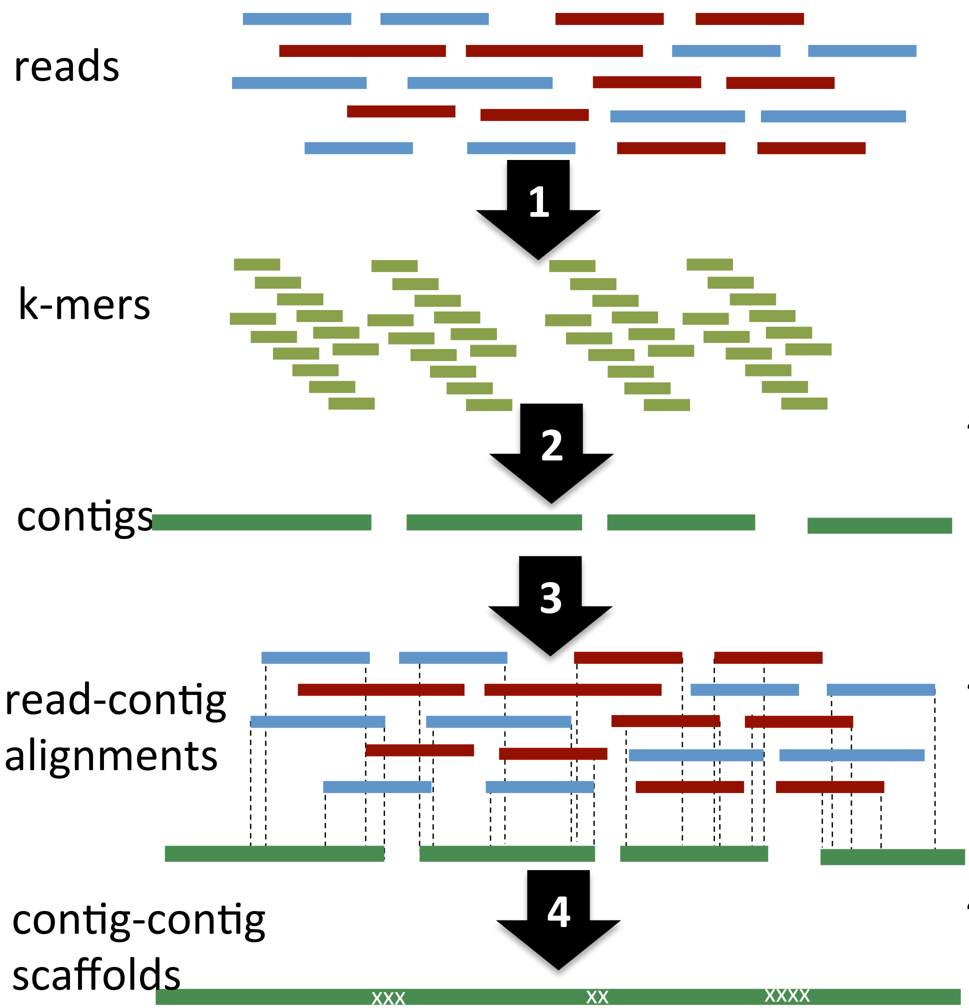
- **Metagenome assembly: 100s-1000s of species mixed together**

Strong scaling (human genome) on Cray XC30



- 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)

The HipMer genome assembly pipeline has 4 phases



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)

Hardware and Programming Requirements

distributed hash tables *all the way down*...

Or at least a global address space

- High injection rate networks
- High bisection bandwidth with modest-sized messages
- Remote (hardware) atomics
- Caching remote values sometimes useful (can be done in software)

Leverages hash table features

- Asynchronous random-access
- Inserts reordered (write-only phase)
- Lookups may involve marking elements (read-only phase)
- Good hash functions for load balance (and locality if genome ~known)

Does Chapel have a Killer App?

Should you?

Summary

- **Many opportunities for languages / compilers**
  - People disenchanted by compilers
  - Blame unrealistic expectations and HPF?
- **Can you get both higher level and superior performance?**
  - For a domain?
  - What parts of programming could be automated?
    - Synthesis, superoptimizers, etc.
- **What are the real pain points for programmers?**
  - Correctness of numerical code? Races?
- **Do you have a killer app or domain?**

Thank you!