Programming Models for SOCs in HPC
(A Play in 3 Acts)

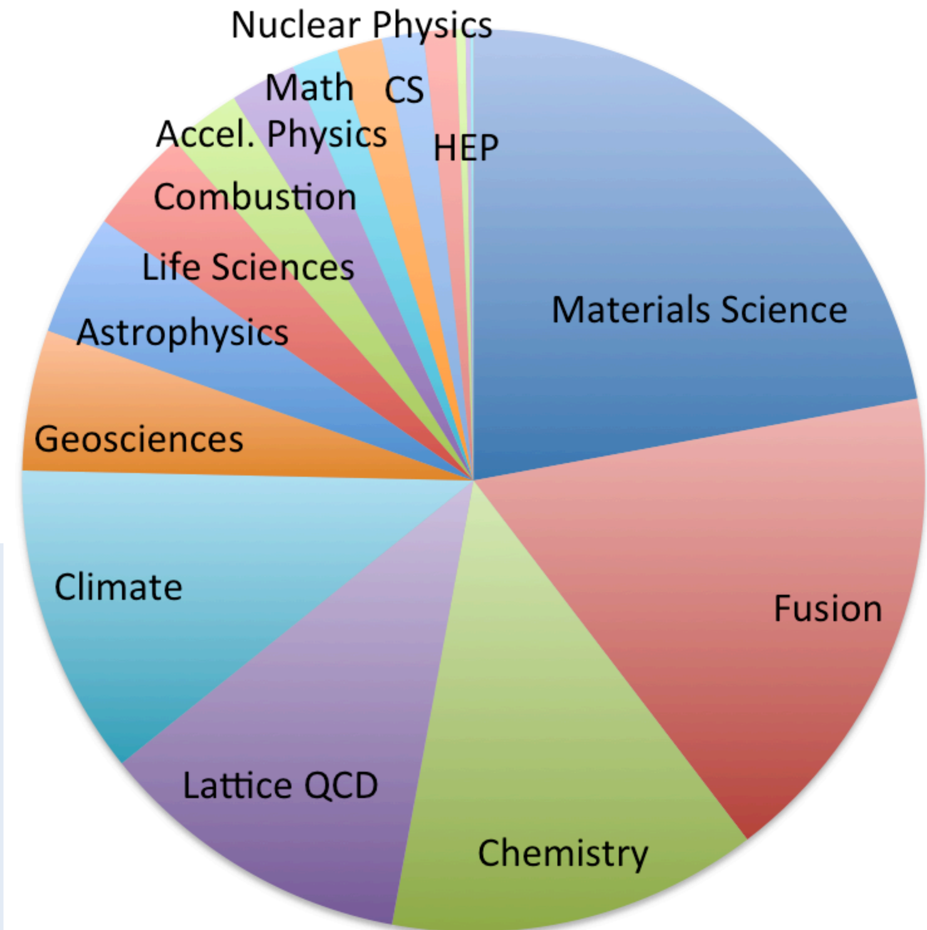
Kathy Yelick

Lawrence Berkeley National Laboratory and UC Berkeley

Act I:
SOCs in a Strange Land

NERSC is the Primary High Performance Computing Center for Office of Science Use

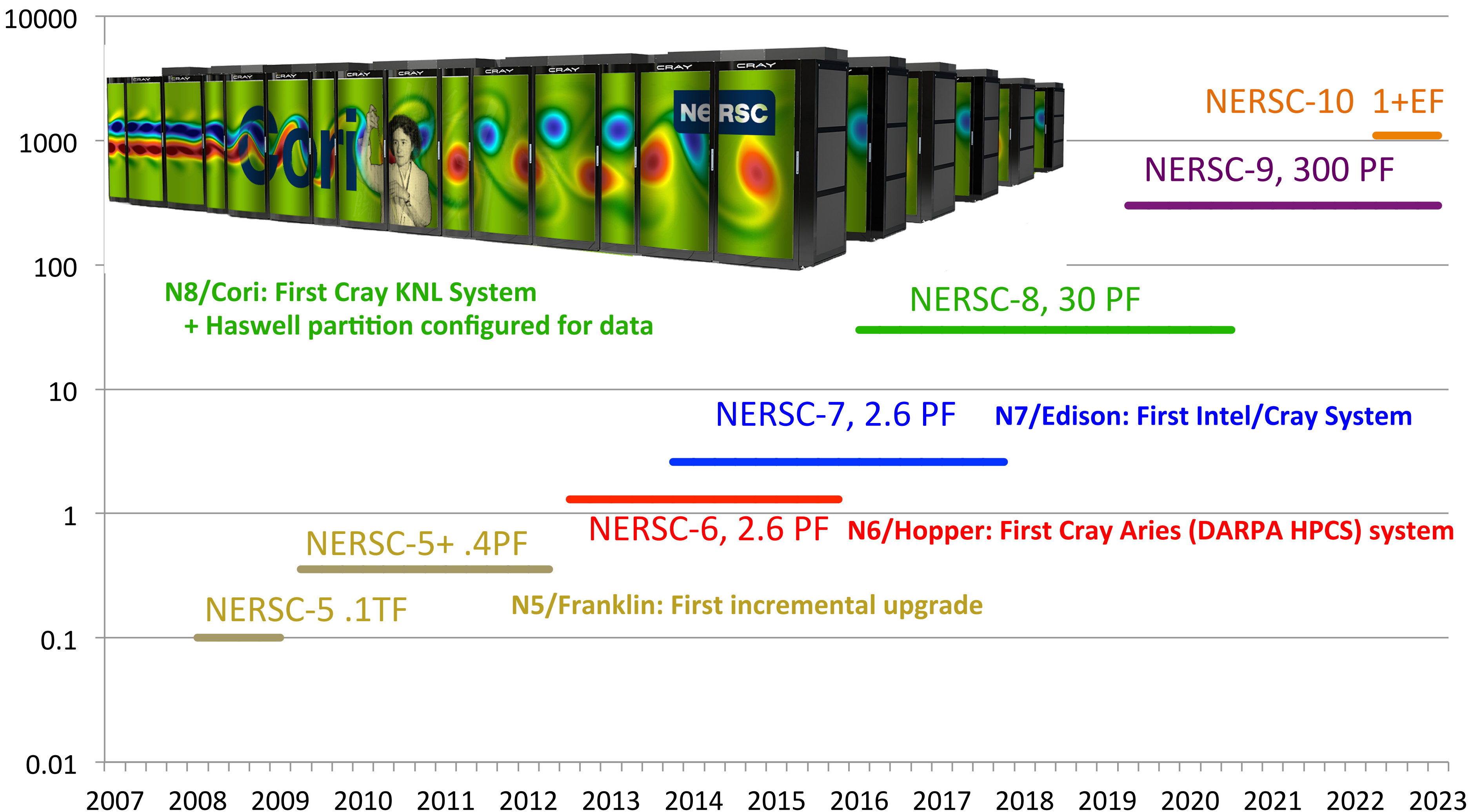
- **DOE/SC allocates NERSC resources for their mission**
- **Over 5000 users and 700 projects run at NERSC**
- **They write about 2000 publications per year**
- **2 Petascale systems today**
  - NERSC-7: Hopper
  - NERSC-8: Edison



The workload is diverse and increasingly complex due to science workflows, integration of data, and demand for higher resolution and scale

U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC technology leadership includes a path to Exascale



Two Steps toward Exascale at NERSC

Cori: energy-efficient architecture on the exascale roadmap

- Over 9,300 Knights Landing compute nodes
- Self-hosted, up to 72 cores, 16 GB high bandwidth memory
- 1,600 Haswell nodes in data partition
- Cray Aries Interconnect

Wang Hall: New computing facility

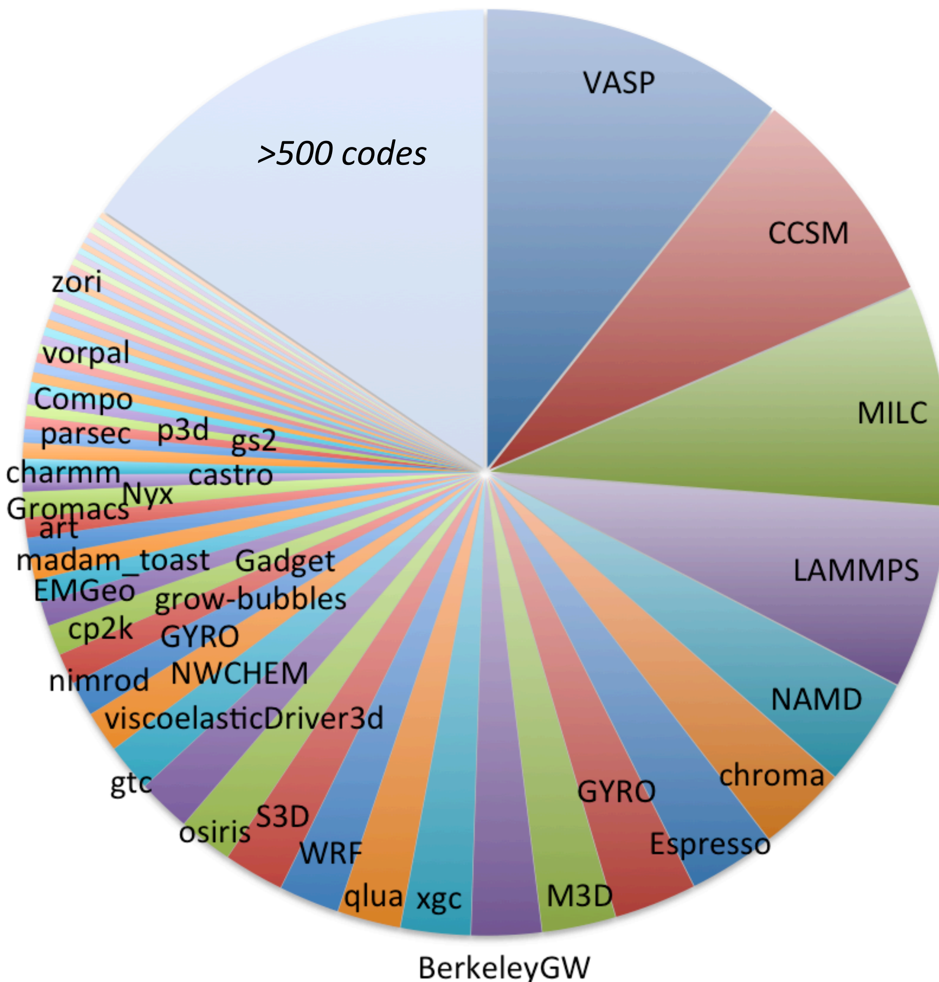
- 12.5 MW initial capacity
- Expandable to 42MW
- Energy efficient design (PUE < 1.1)

Short walk from Berkeley Campus

Dedication on Nov 12

Materials and Chemistry are a Significant Fraction of the DOE/SC Computing Workload

Breakdown of Application Hours

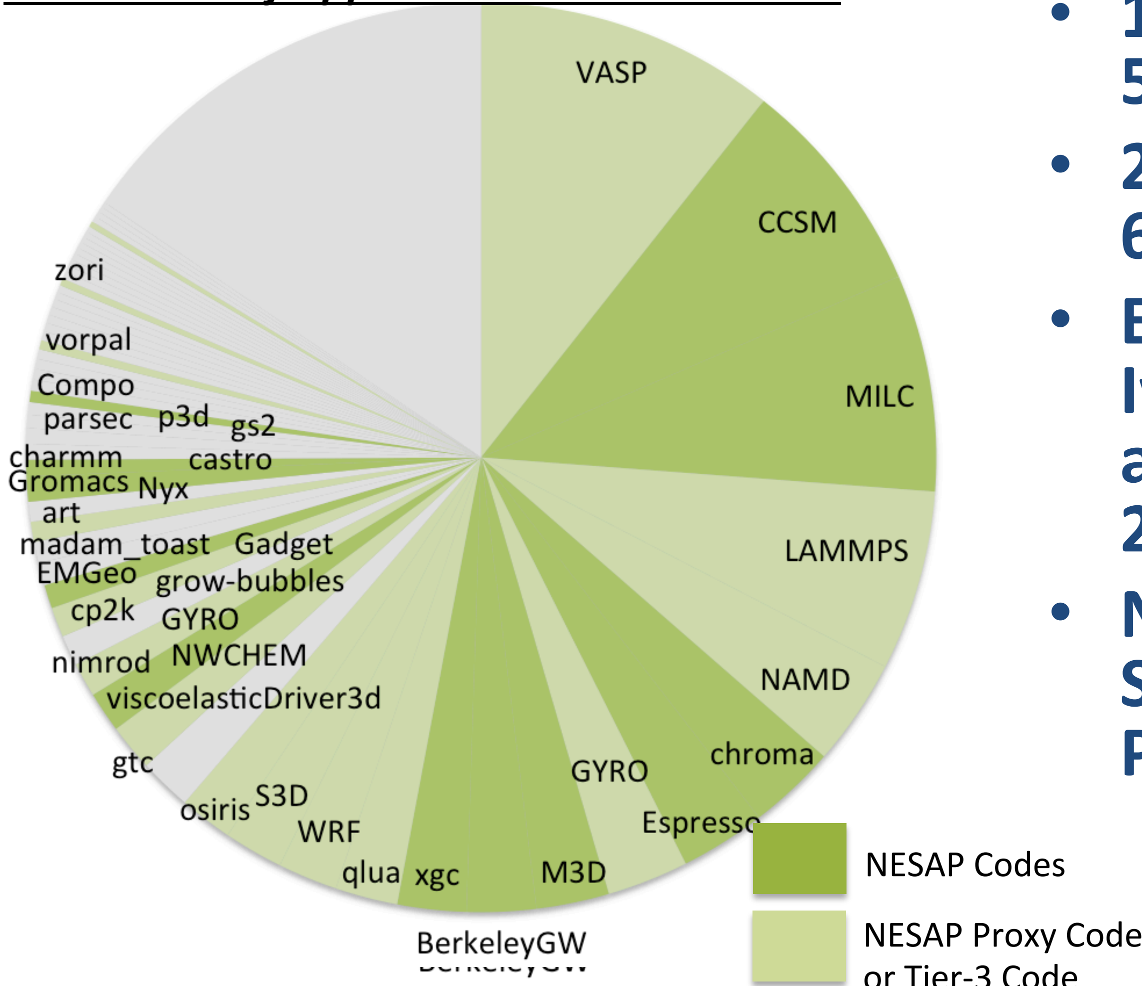


- **10 codes make up 50% of the workload**
- **25 codes make up 66% of the workload**
- **Edison (Cray with Intel IvyBridge) will be available until 2019/2020**

Materials and Chemistry are a Significant Fraction of the DOE/SC Computing Workload

Breakdown of Application Hours

Breakdown of Application Hours at NERSC



- **10 codes make up 50% of the workload**
- **25 codes make up 66% of the workload**
- **Edison (Cray with Intel IvyBridge) will be available until 2019/2020**
- **NERSC Exascale Science Applications Program (NESAP)**
  - New staff, training and partnerships with Intel for KNL

Performance Portability is a Goal Across DOE

- **Titan, Mira and Edison represent 3 distinct architectures in SC**
  - Not performance portable across systems
- **APEX 2016 and CORAL @ ANL**
  - Xeon Phi, no accelerator
- **CORAL 2017**
  - IBM + NVIDIA

Two different version of the code

Best case #1: OpenMP4 absorbs accelerator features (likely), but code still requires a big ifdef

Best case #2: Architectures “converge” by 2023, perhaps with co-design help

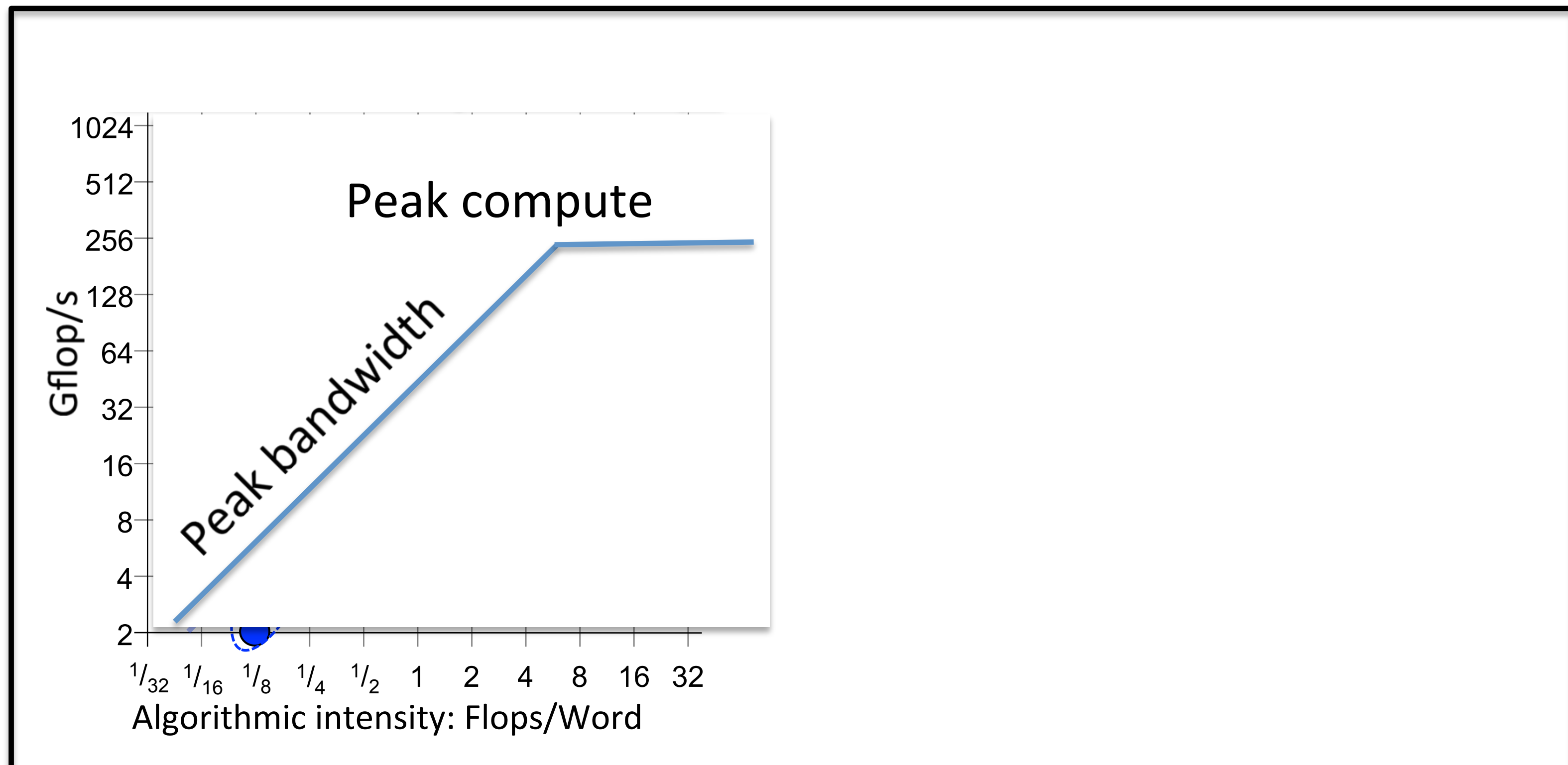
Act II:
Don’t Fear the Compiler

A Compiler is Just a Translator

- **Scientific computing relies heavily on libraries**
  - LAPACK and FFTW are widely used at NERSC
- **People use languages for their libraries**
- **Do we need a language? And a compiler?**
  - If higher level syntax is needed for productivity
    - We need a language
  - If static analysis is needed to help with correctness
    - We need a compiler (front-end)
  - If static optimizations are needed to get performance
    - We need a compiler (back-end)

Autotuning: Write Code Generators

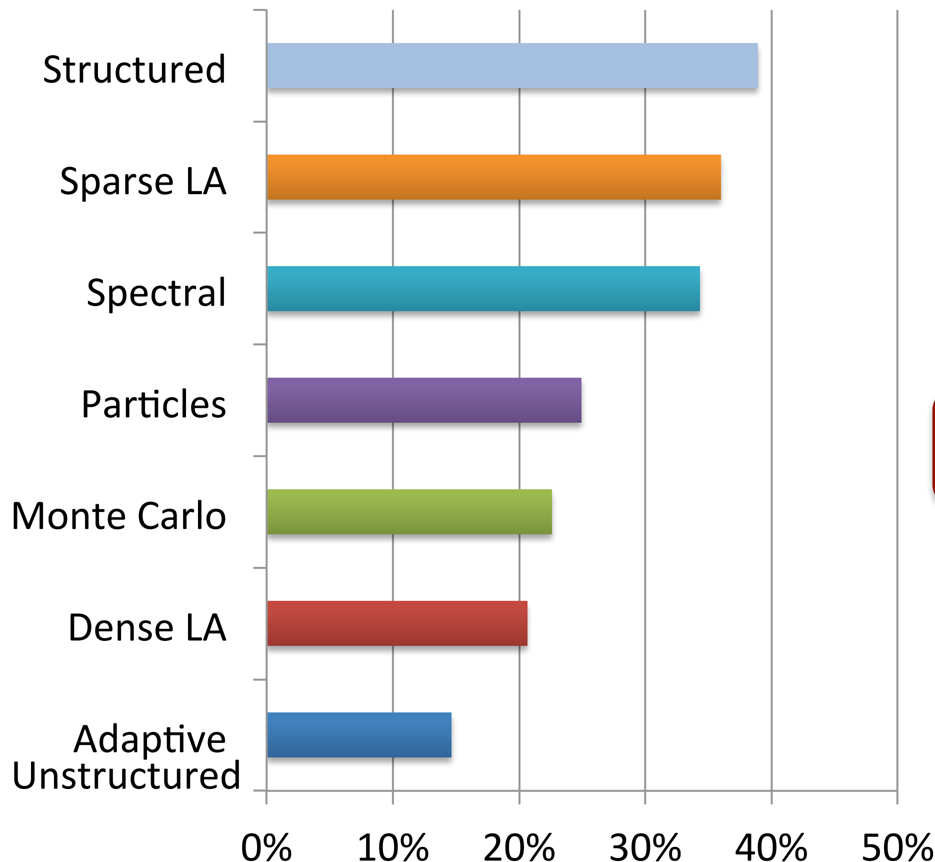
- **Two unsolved compiler problems:**
  - dependence analysis and
  - accurate performance models
- **Autotuners are code generators plus search**



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

What we have and what we need

NERSC survey: what motifs do they use?

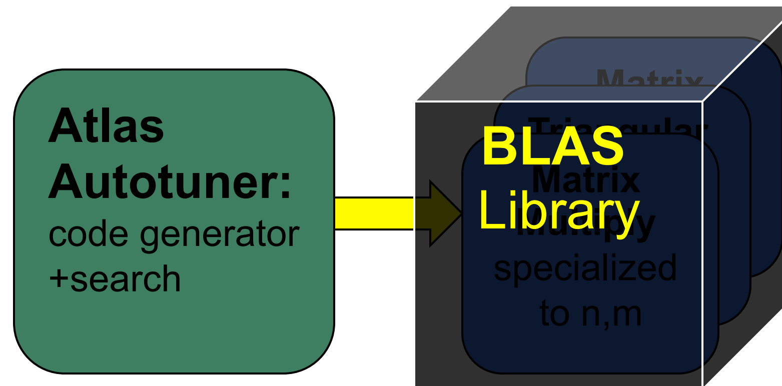


What code generators do we have?

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

Approaches to Autotuning



Approximate categorization!

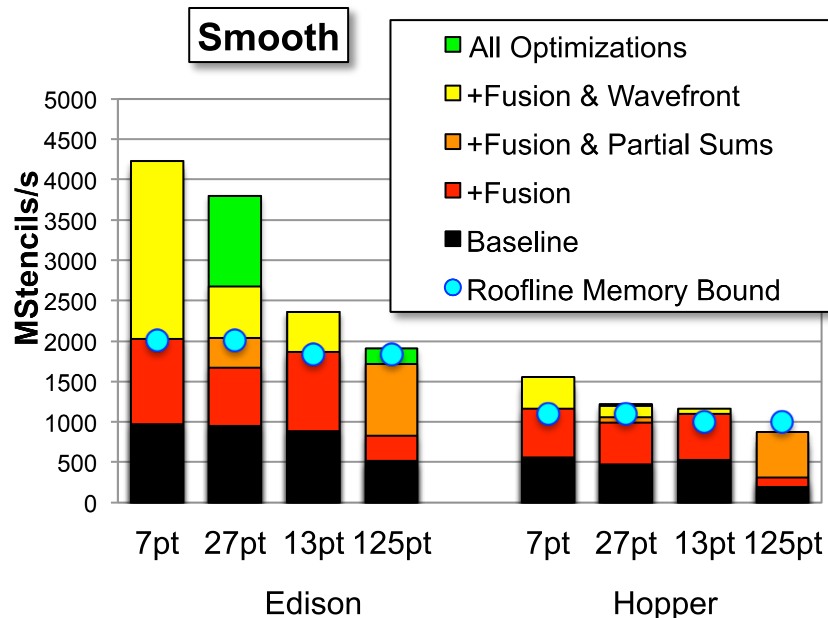
How do we produce all of these (correct) versions?

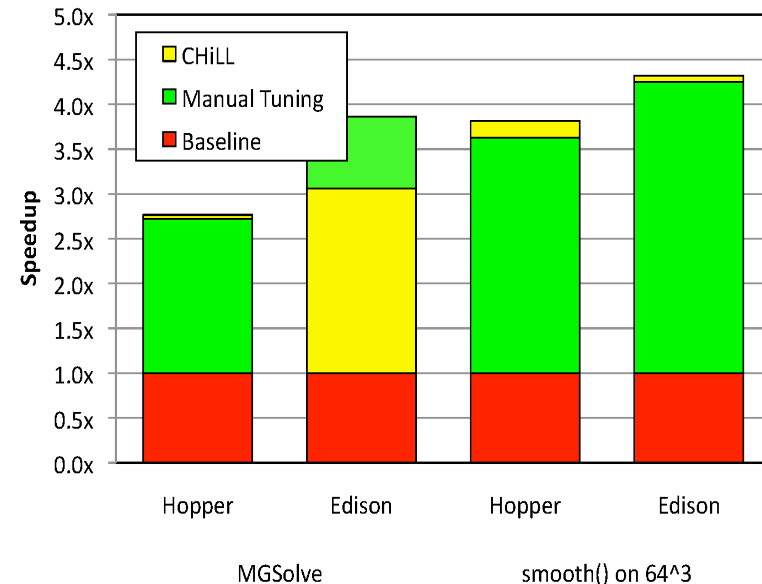
- Using scripts (Python, perl, ML, C,..)
- Compiling annotated general-purpose language (X-Tune,...)
- Use preprocessor to generator code (Raja, Kokkos,TiDA)
- Compile a domain-specific language (D-TEC, Halide)
- Domain-specific compiler for domain-specific language (SEJITS)

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

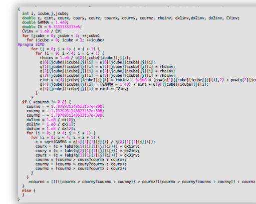


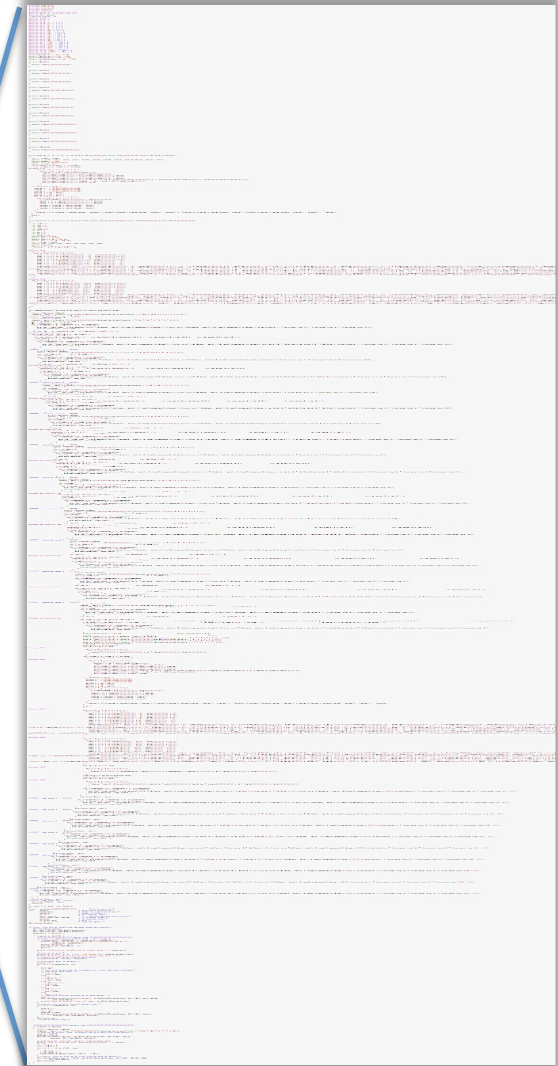


Results on Geometric Multigrid (miniGMG Smoother)

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





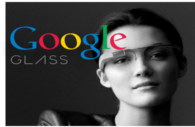
  - Code size of autogenerated code

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)

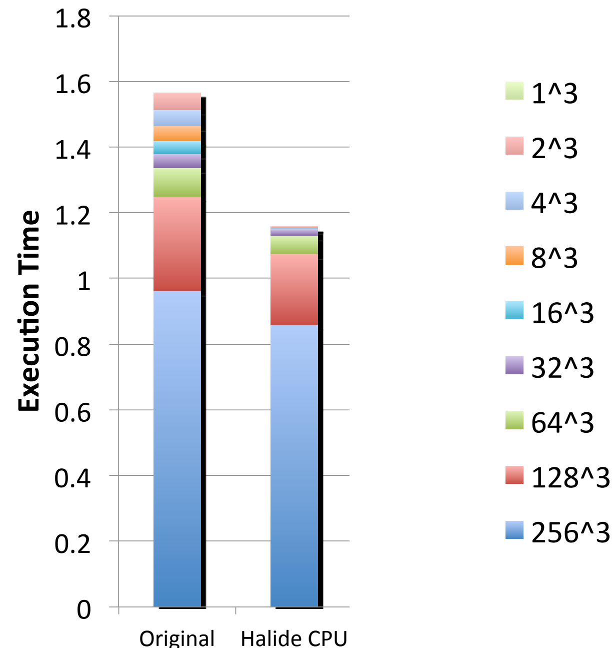
- **Halide Algorithm by domain expert**

```
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

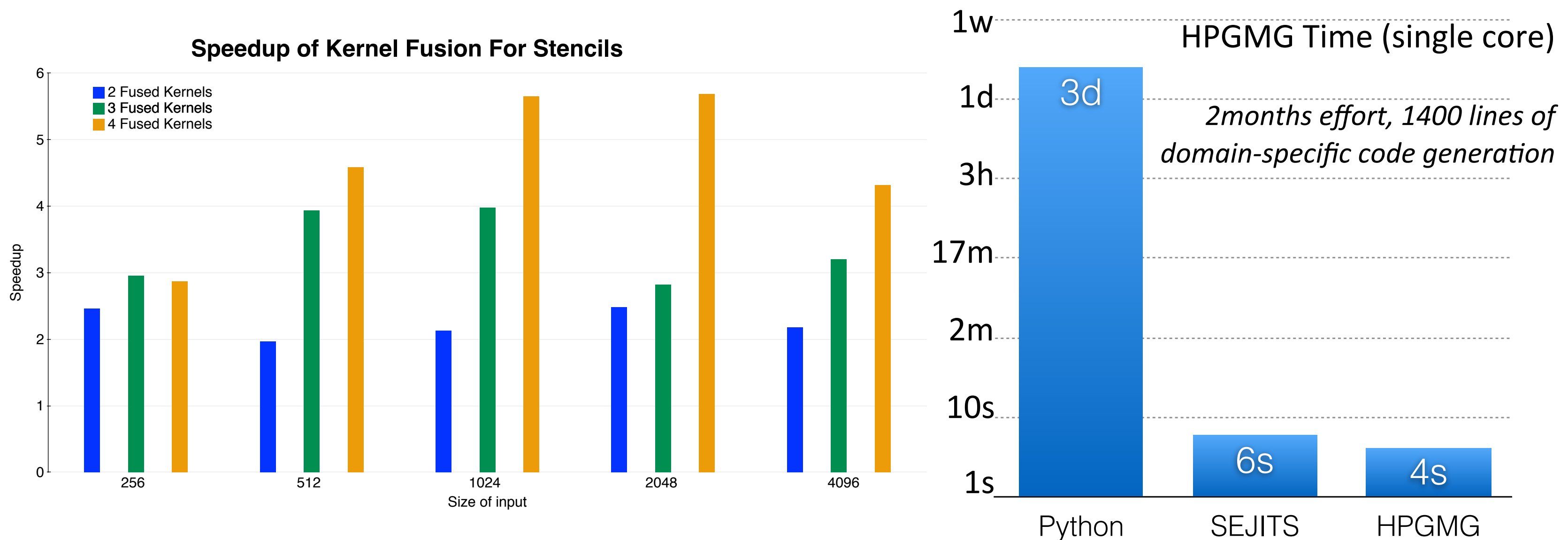
Halide performance

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



Approach #4: Small Compiler for Small Language

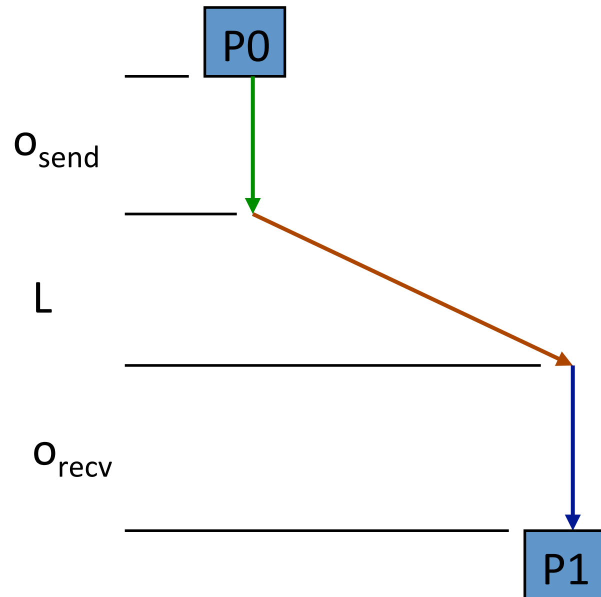
- **SEJITS: Selected Embedded Just-In-Time Specialiation:**
  - General optimization framework (Ctree)
  - Currently implemented part of HPGMG benchmark in stencil DSL
    - Within 50% of hand-optimized code
    - ~1000 lines of DSL-specific code; 1 undergrad over <2 months



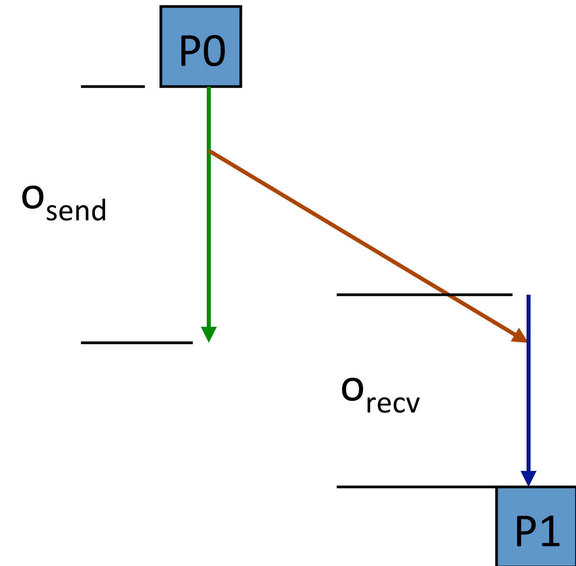
Act III:
Overhead Can’t be Tolerated

Modified LogGP Model

- **LogGP: no overlap**

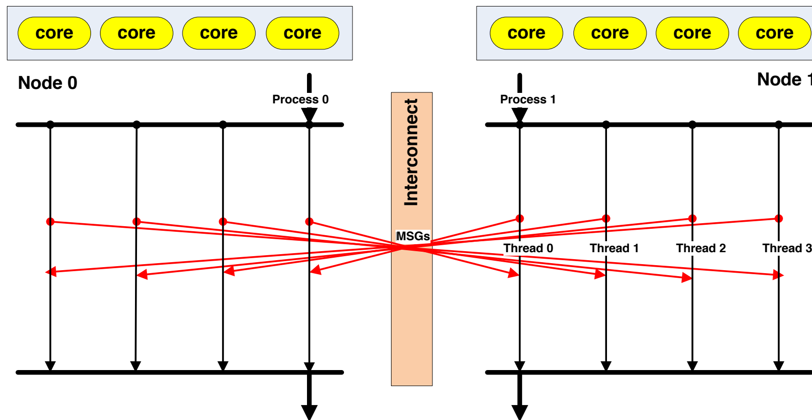


- **Observed: overheads can overlap: L can be negative**

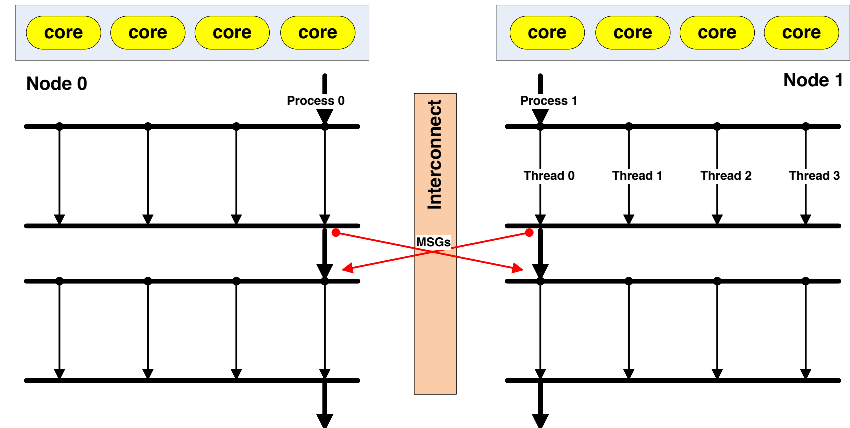


EEL: end to end latency (instead of transport latency L)
g: minimum time between small message sends
G: additional gap per byte for larger messages

Communication and Manycore: the problem is the “+”



Ideal hybrid programming

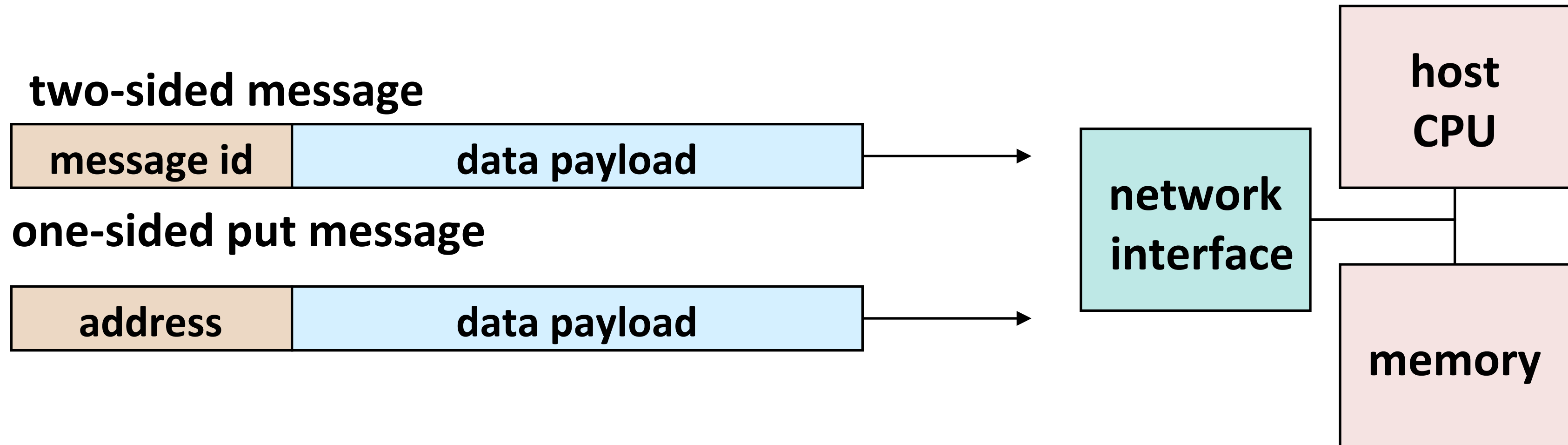


Default hybrid programming

- **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**

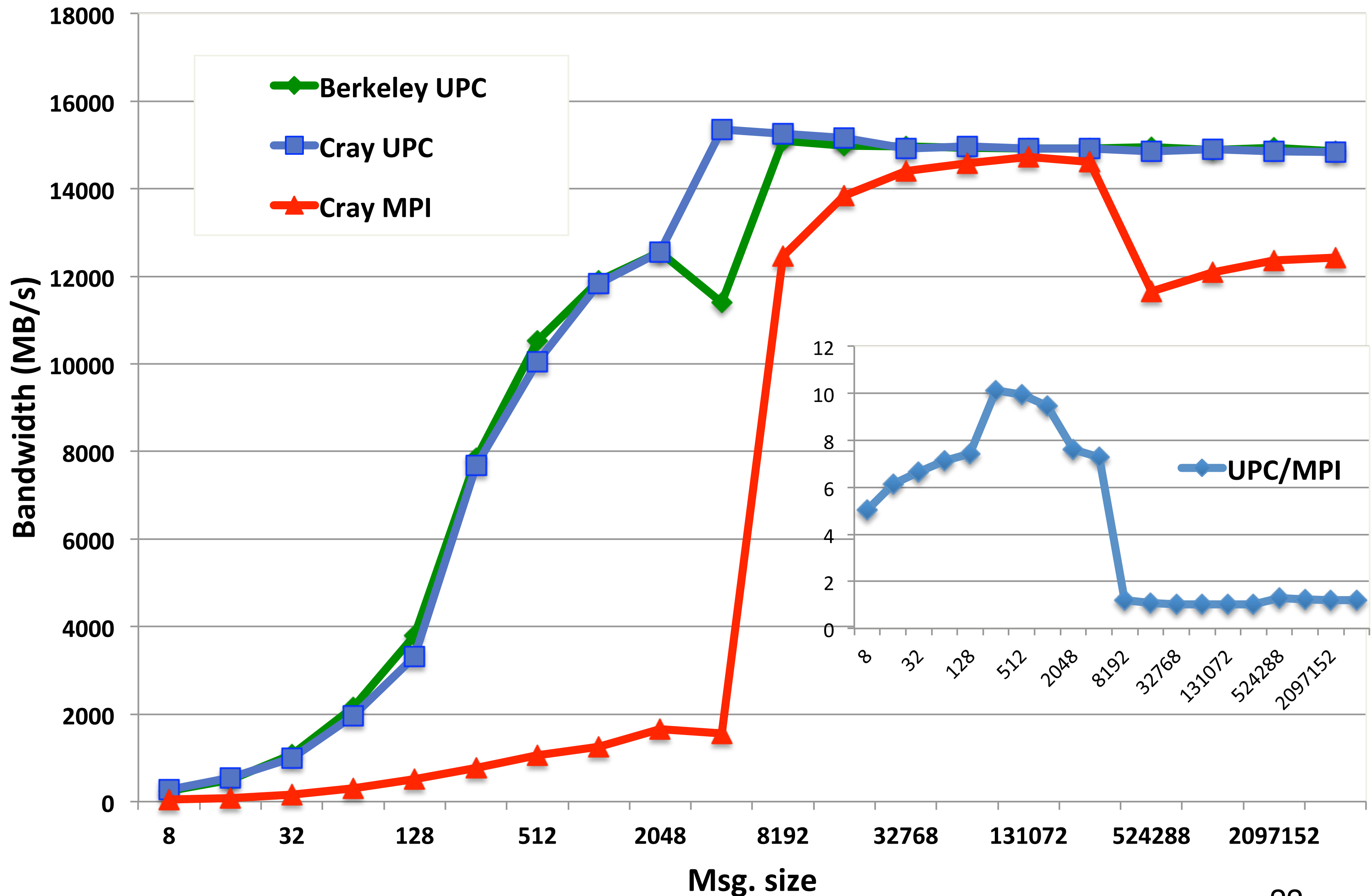
Khaled Ibrahim, ICS 2014

Avoid Latency and Implicit Synchronization



- **Two-sided message passing (e.g., send/receive in MPI) requires matching a send with a receive to identify memory address to put data**
  - Couples data transfer with synchronization, which is sometimes what you want
- **Using global address space decouples synchronization**
  - Pay for what you need!

Bandwidths on Cray XE6 (Hopper)



Lightweight Communication for Lightweight Cores

- **DMA (Put/Get)**
  - Blocking and non-blocking (completion signaled on initiator)
  - Single word or Bulk
  - Strided (multi-dimensional), Index (sparse matrix)
- **Signaling Store**
  - All of the above, but with completion on receiver
  - What type of “signal”?
    - Set a bit (index into fixed set of bits ☹)
    - Set a bit (second address sent 😐)
    - Increment a counter (index into fixed set of counters ☹)
    - Increment a counter (second address for counter 😐)
    - Universal primitives: compare-and-swap (2nd address + value), fetch-and-add handy but not sufficient for multi/reader-writers ☺
- **Remote atomic (see above) – should allow for remote enqueue**
- **Remote invocation**
  - Requires resources to run: use dedicated set of threads?

Technology Transfer Paths

- **Languages**
  - Adoption into popular programming models
    - One-sided into MPI (again)
    - Locality control into OpenMP
  - Adoption by a compiler community (Chemistry DSL)
- **Compilers**
  - Leverage mainstream compilers (LLVM)
  - Leverage another existing “domain-specific” language
  - Small compilers for small languages
- **Next phase**
  - Focus on application partnerships
  - Partnerships with library and frame work deveopers
  - Collaborate with vendors on hardware desires and constraints

Thank you!

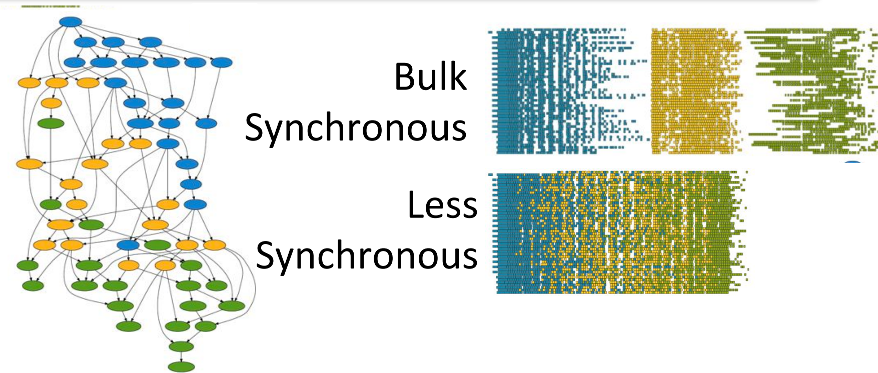
Sources of Unnecessary Synchronization

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

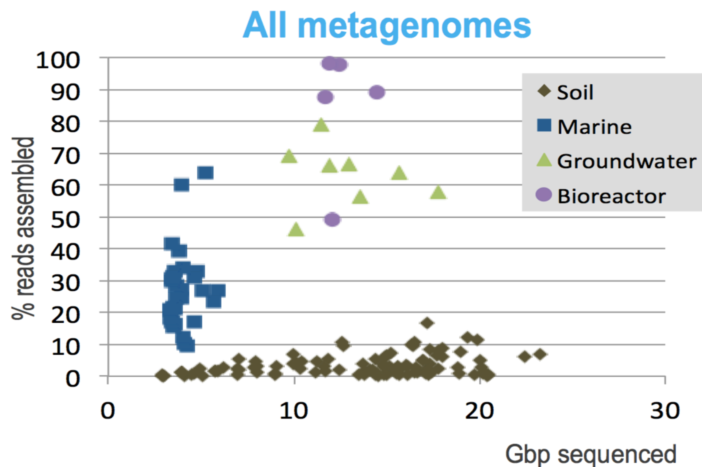
```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc&    streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc&    streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$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

Random Access to Large Memory

Meraculous Assembly Pipeline

Human: 44 hours to 20 secs
Wheat: "doesn't run" to 32 secs

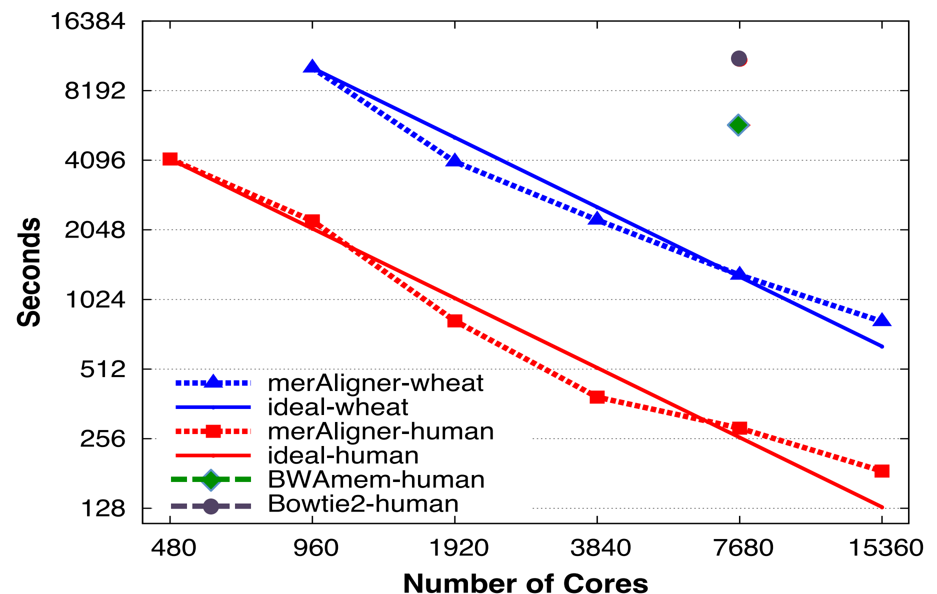


Grand Challenge: Metagenomes

Perl to PGAS: Distributed Hash Tables

- Remote Atomics
- Dynamic Aggregation
- Software Caching (sometimes)
- Clever algorithms and data structures (bloom filters, locality-aware hashing)

→ UPC++ Hash Table with "tunable" runtime optimizations



Productivity: Enabling a New Class of Applications?