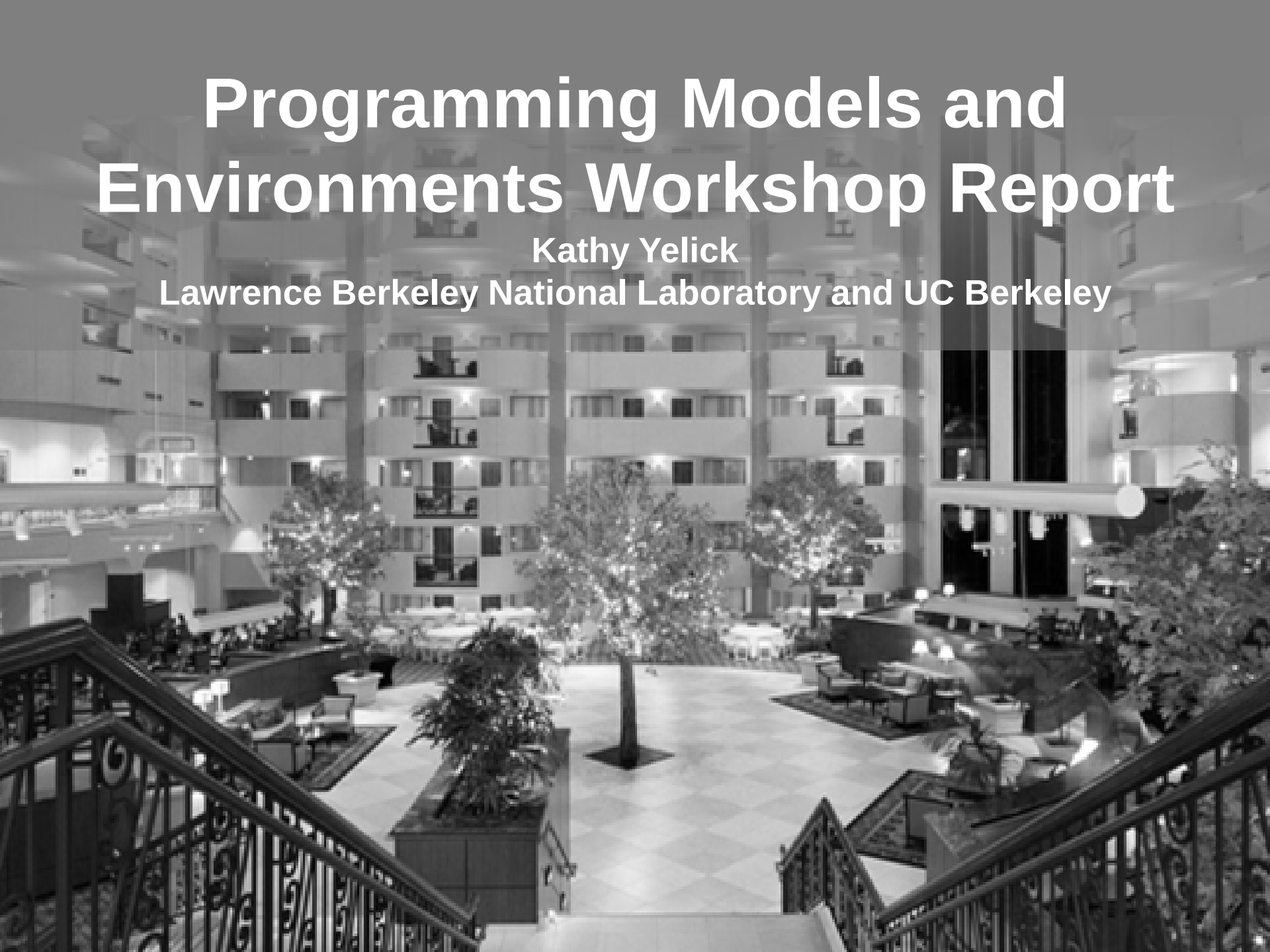
Programming Models and Environments Workshop Report

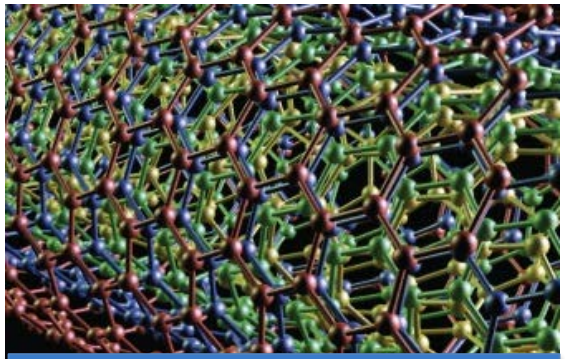
Kathy Yelick
Lawrence Berkeley National Laboratory and UC Berkeley



ASCR Programming Environments Summit Report Summary

Hardware Challenges

- Energy Efficiency
- Node concurrency
- Hierarchy
- Heterogeneity
- Reliability



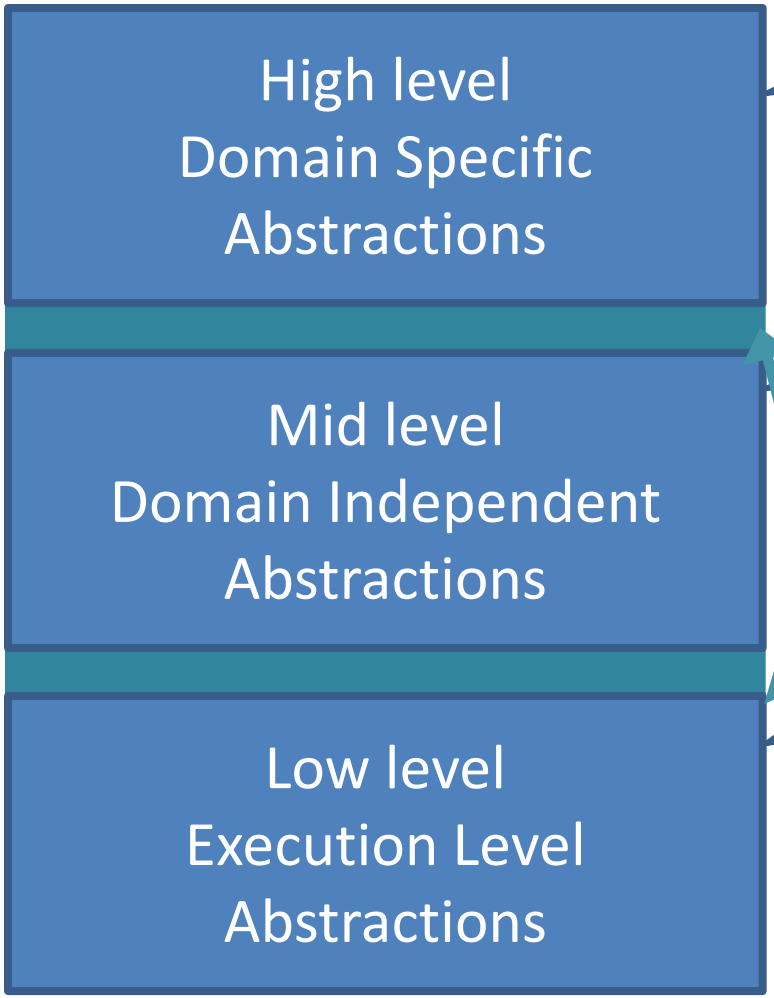
Application Challenges

- Multiscale and multiphysics
- Software size and complexity
- Data-driven computation
- New use models

Ecosystem Issues

- Not all software will be rewritten
- Supercomputing market is small
- Acquiring new skills is hard

Programming Model Stack Overview in Report



Science level:

- Embedded DSLs for important domains
- Support for custom abstractions
- Support for manipulating them

Software level:

- Logical structure of parallelism and locality
- Avoid committing to specific architecture

Platform specific level:

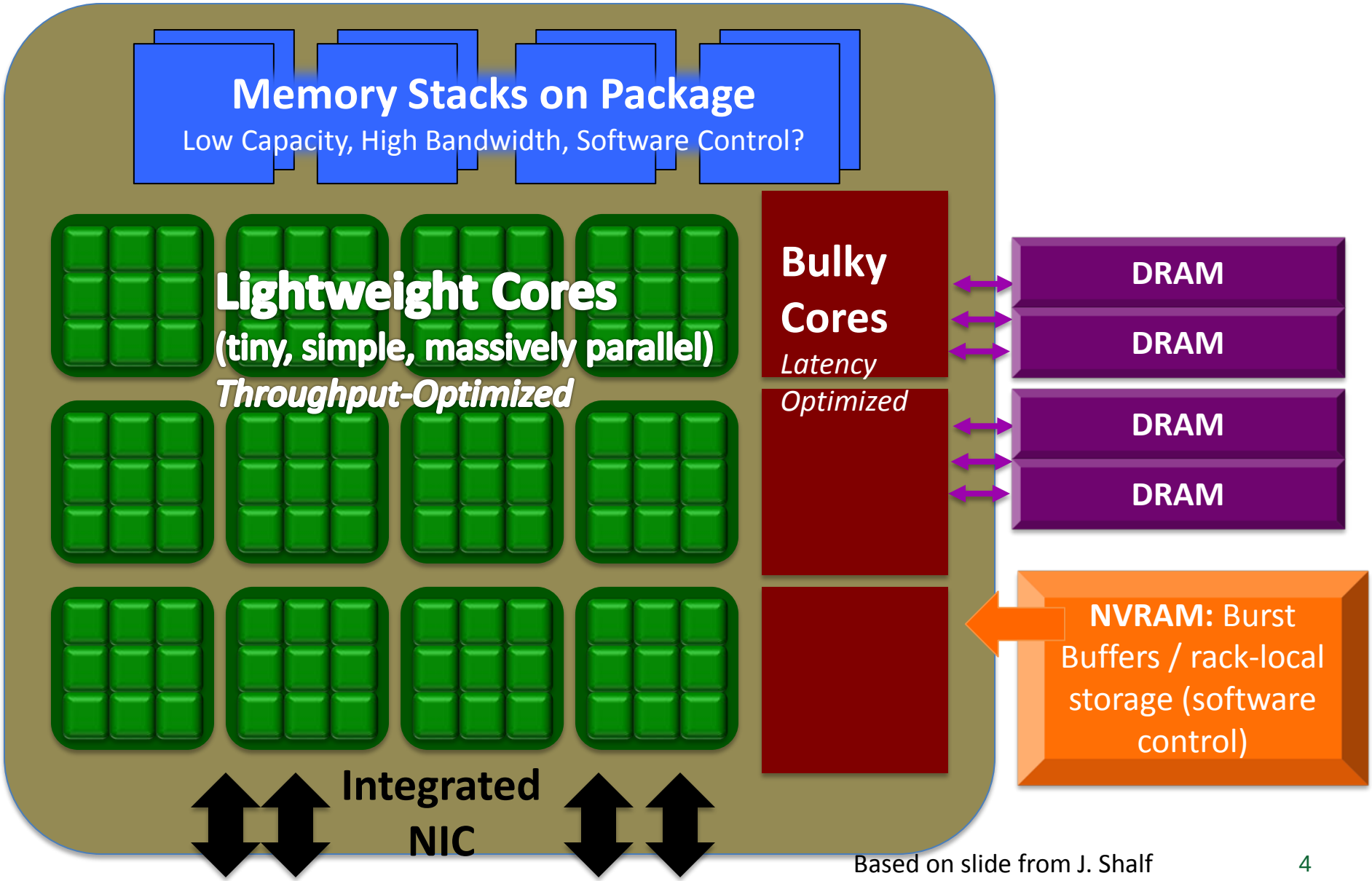
- Explicit interfaces for task creation, data movement, synchronization, etc.
- A lot of programming today is at this level!
- New interfaces for managing power, resilience, and introspection

Mappings

- Automate when possible
- Avoid all-or-nothing mechanisms

Draft report by a dozen researchers from industry, academia and Labs completed in February

Future Generic Node Architecture



Based on slide from J. Shalf

Architecture Challenges and Opportunities

1. **Lightweight cores will have all/most of the system performance**
   - Need fine-grained parallelism; avoid unnecessary synchronization
   - Cores not powerful enough for complex communication protocols ?
2. **On-chip interconnect offers opportunities for performance**
   - New models of communication may be essential
3. **Hardware is heterogeneous: no single ISA**
   - Portability and performance portability are challenging
4. **New levels of memory hierarchy, possibly software-controlled**
   - Locality and communication-avoidance paramount
5. **Performance variability may increase**
   - Software or hardware control clock speeds
6. **Resilience will be paramount at scale**
   - Failures grow with the number of components and connections

OpenMP Loop Parallelism is the Wrong Level

- **OpenMP is popular for its convenient loop parallelism**
- **Loop level parallelism is too coarse and too fine:**
  - Too coarse: Implicit synchronization between loops limits parallelism and adds overhead
  - Too fine: Need to create larger chunks of serial work by combining across loops (fusion) to minimize data movement

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

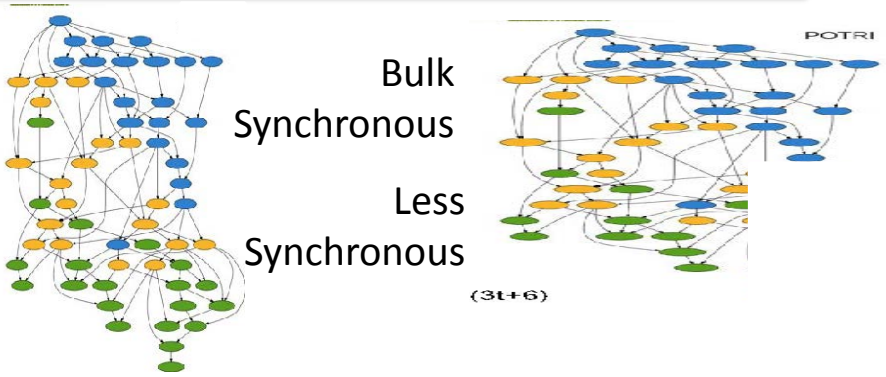
Sources of Unnecessary Synchronization

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

Locality in OpenMP4 is (at Best) Computation-Centric

```
subroutine vec_mult(p, v1, v2, N)
   real    ::  p(N), v1(N), v2(N)
   integer ::  i
   call init(v1, v2, N)
   !$omp target data map(to: v1, v2) map(from: p)
   !$omp target
   !$omp parallel do
      do i=1,N
                        p(i) = v1(i) * v2(i)
      end do
   !$omp end target
   !$omp end target data
   call output(p, N)
```

And you have to do this for every loop!

Based on slide from J. Shalf

Where is Performance Portability?

- **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 absorbed accelerator features (likely), but code still requires a big ifdef

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

Major Programming Model Research Areas

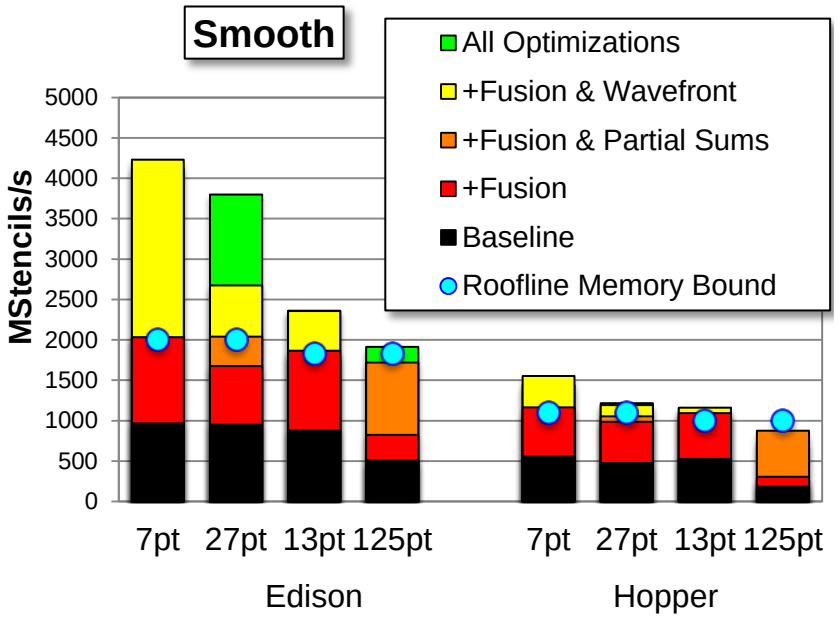
- **Performance Portability through Compilers and Autotuning**
  - Automatically generate GPU and CPU code & automatically tune
  - *E.g., Rose (D-TEC, LLNL), Halide (D-TEC, MIT), CHiLL (X-Tune, Utah), SEJITS (DEGAS, UCB), Legion (ExaCT, Stanford/LANL), SLEEC (Purdue)*
- **Data Locality in Languages and Libraries**
  - Specify location of data (Partitioned Global Address Space)
  - *E.g., UPC/UPC++ (LBNL), CAF (Rice), TiDA (LBNL), RAJA (LLNL), KOKKOS (SNL)*
- **Less Synchronous DAG Execution Models**
  - Static and dynamic DAG construction
  - *Examples: OCR (Intel), HPX (XPRESS), Charm++ (UIUC), Legion (Stanford/LANL), Habanero (Rice)*
- ***Correctness***
  - *Precimonious and OPR (Corvette/UCB)*
- ***Resilience Models and Technology***
  - Use of NVRAM (GVR, UChicago); Containment Domains (DEGAS/UTexas)

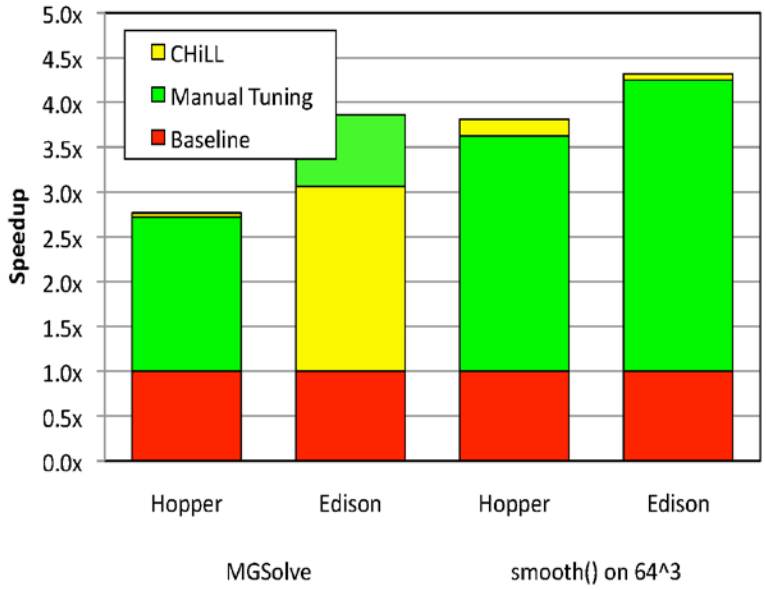
Funded by X-Stack, Co-Design and NNSA

Performance Portability

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

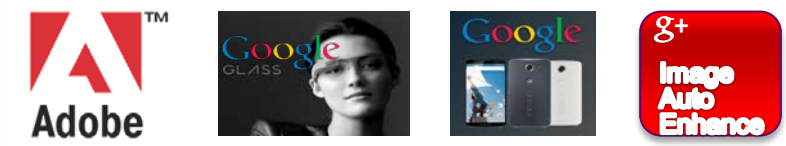




Results on Geometric Multigrid (miniGMG Smoother)

Approach #2: Domain-Specific Languages (but not too specific)

Developed for Image Processing



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

HPGMG (Multigrid on Halide)

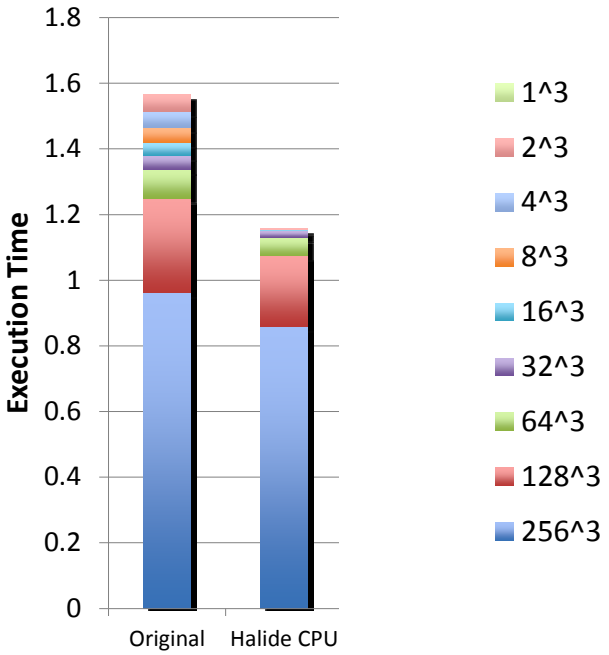
- **Halide Algorithm by domain expert**

```
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



DSLs to Generate Code for Hierarchical Memory

- **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

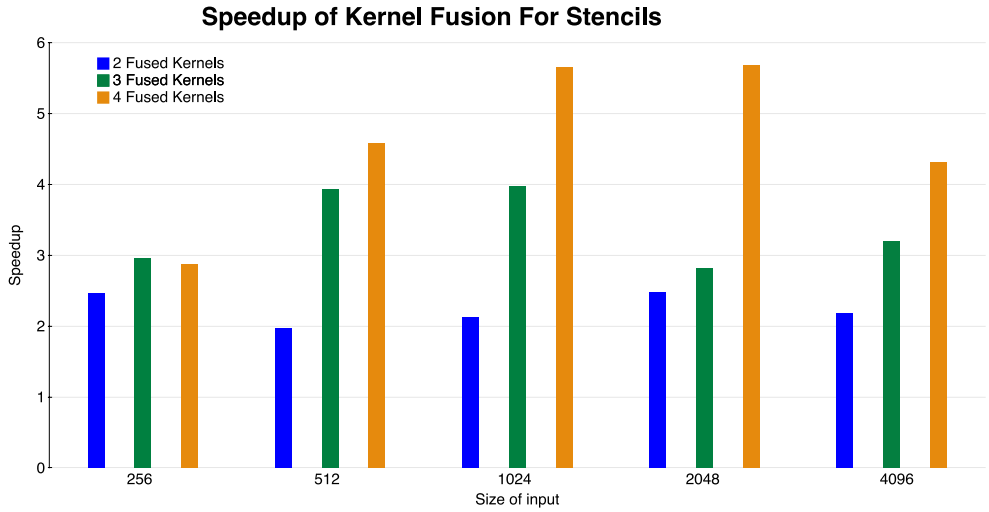
  - Code size of autogenerated code

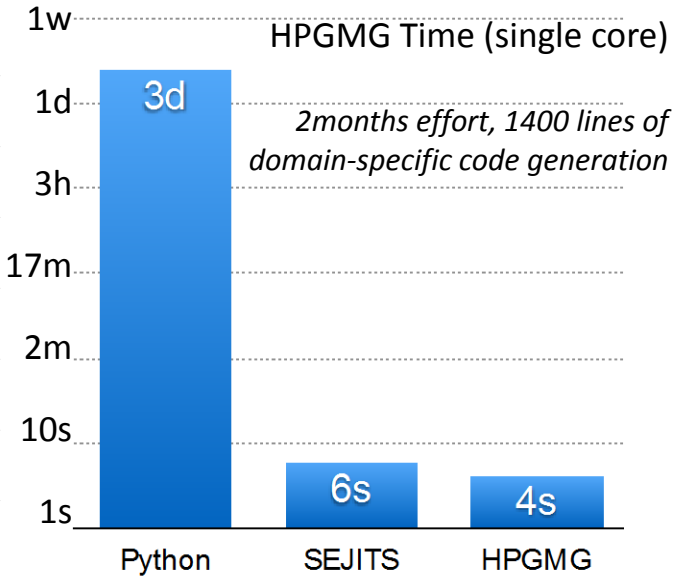
Memory Hierarchy	2 Level	3 Level	4 Level	…	10 level
DSL Code	20				
Auto Generated Code	446	500	553		819

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

Approach #3: Dynamic Specialization

- **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
    - 1400 lines of DSL-specific code; 1 undergrad over <2 months



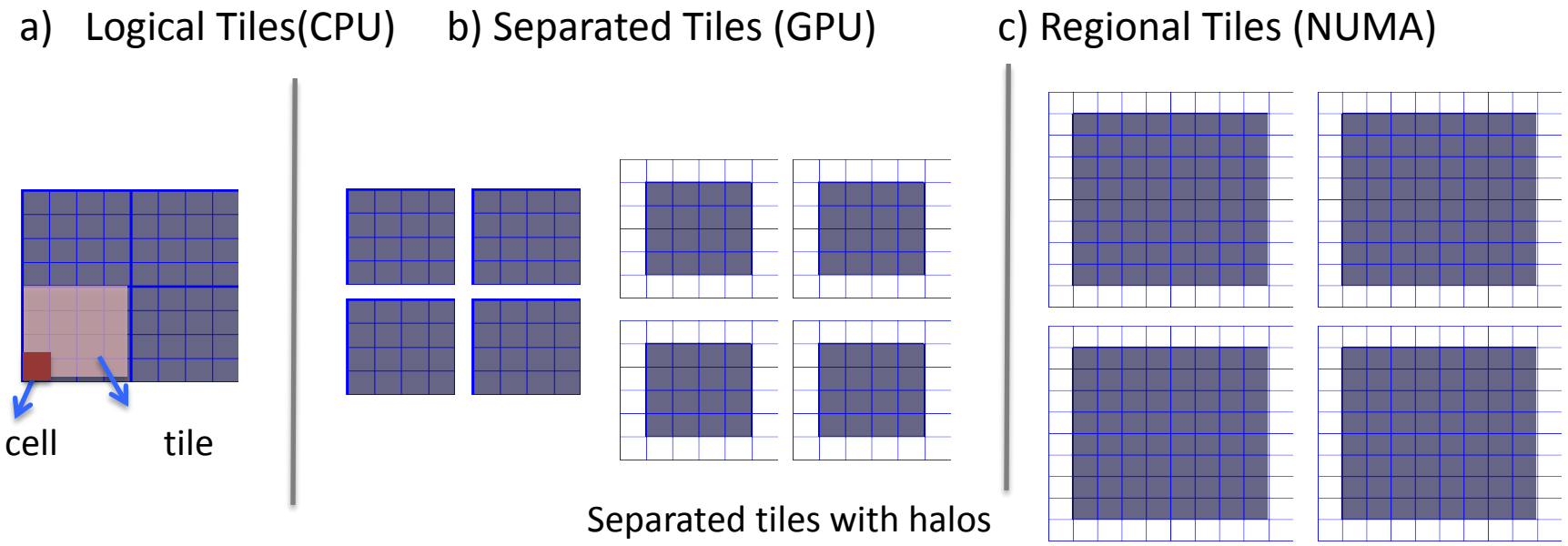


Locality Control

Tiling: Abstraction for Memory Layout

Data layouts can be used to improve locality (and find parallelism), e.g., CAF2, UPC++, Chapel, TiDA, Raja/Kokkos

- OpenMP allows a user to specify any of these layouts
- However, the code is different for GPUs vs CPUs.
- Several approaches pursued here as well

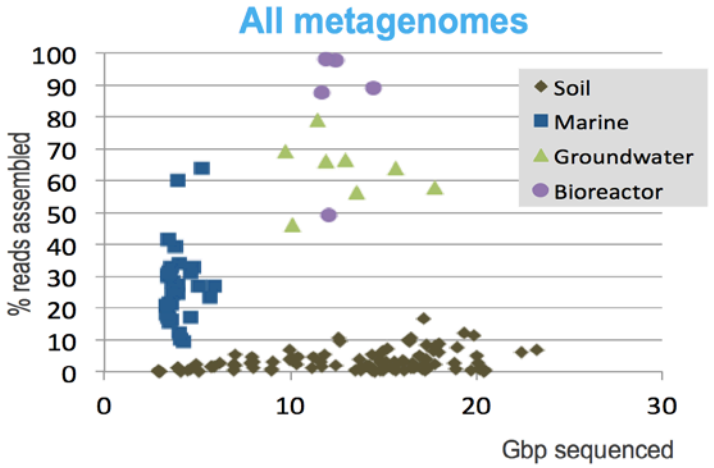


Supporting Applications *without* Locality

Random Access to Large Memory

Meraculous Assembly Pipeline

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

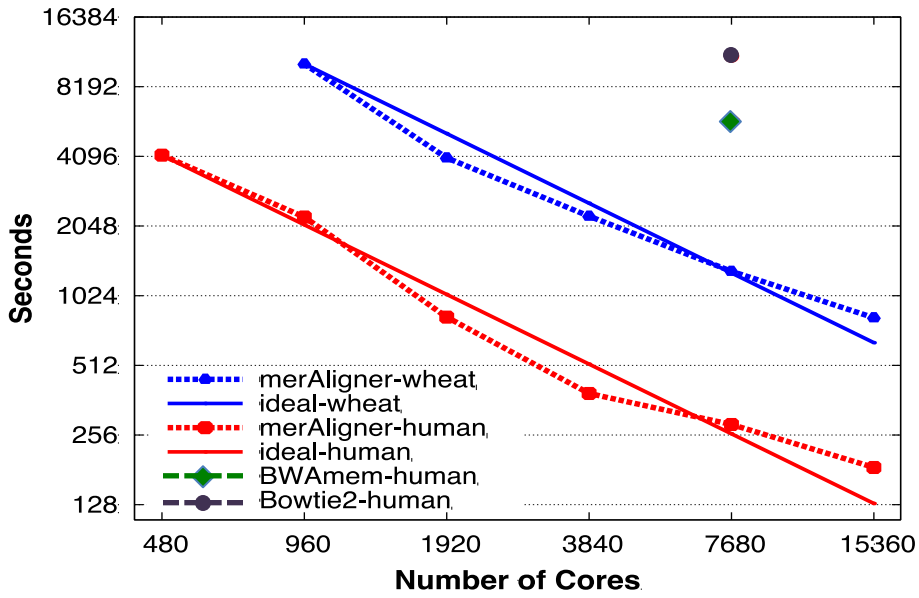


Grand Challenge: Metagenomes

Perl to PGAS: Distributed Hash Tables

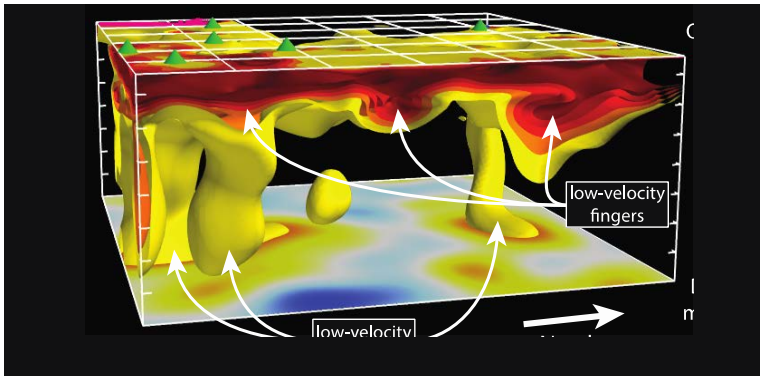
- Remote Atomics
- Dynamic Aggregation
- Software Caching (sometimes)
- Clever algorithms and data structures (bloom filters, locality-aware hashing)

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

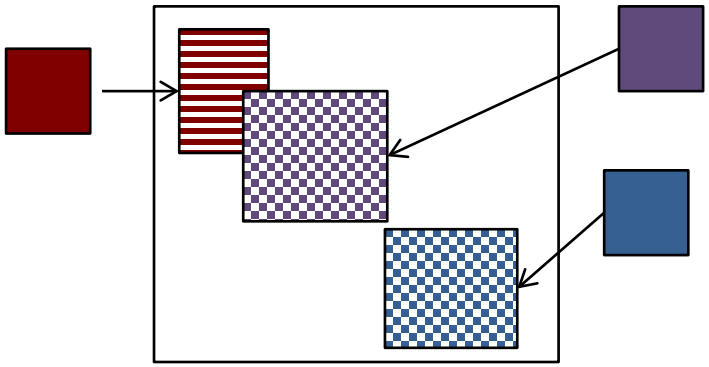


Productivity: Enabling a New Class of Applications?

Data Fusion in UPC++



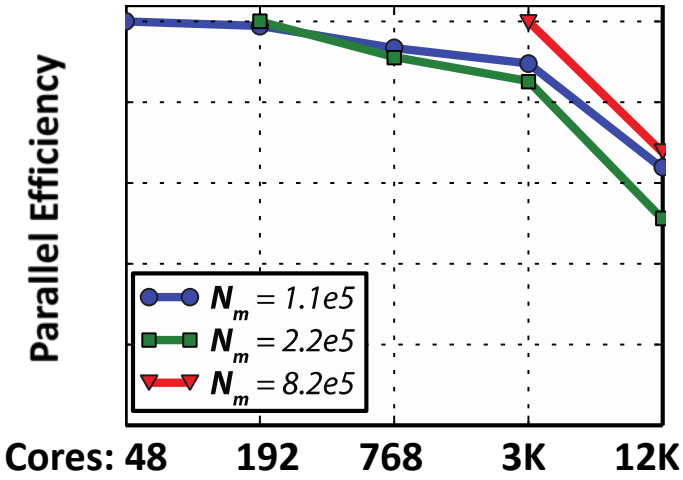
- Seismic modeling for energy applications "fuses" observational data into simulation
- With UPC++, can solve larger problems



Distributed Matrix Assembly

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

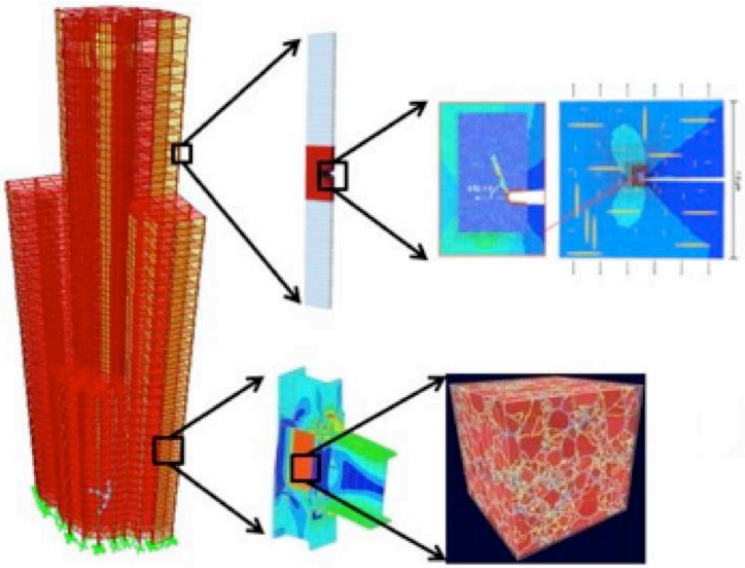
→ **Improving UPC++ team support**

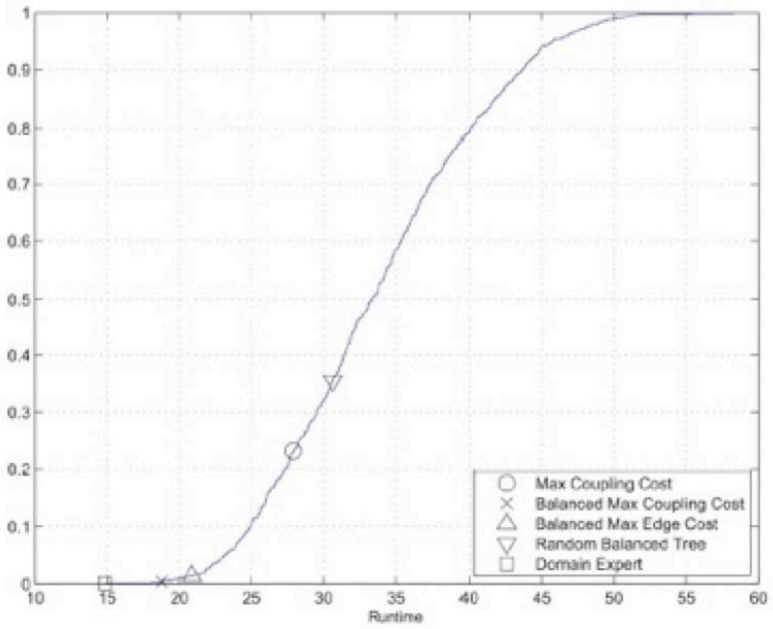


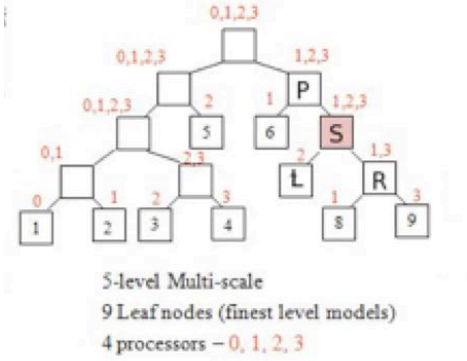
Note scale: >85% efficient in worst case

Similar ideas being use for the Hartree-Fock algorithm as part of NWChem study

Domain Specific Library Interfaces





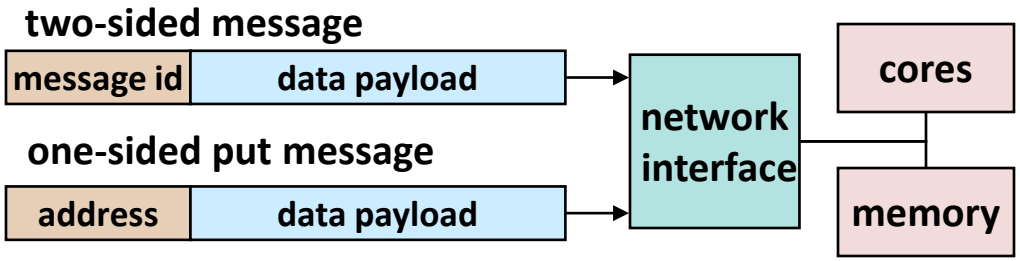


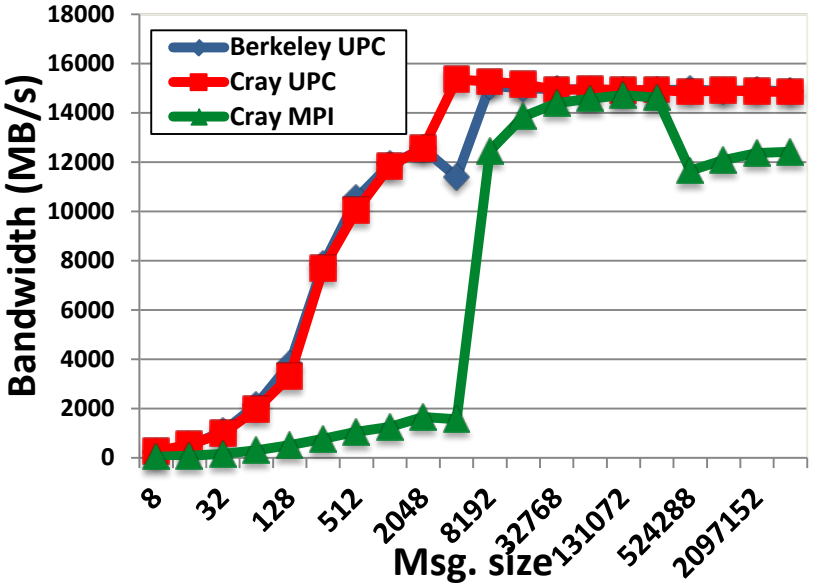
- **SLEEC Project using general-purpose compilers and domain-specific interfaces**
- **Use of Autotuning to align recursive decomposition to machine**

Rethinking Communication

Lowering Overhead for Smaller Messages

Send/Receive





The + in MPI+X

MPI+X today:

- **Communicate on one lightweight core**
- **Reverse offload to heavyweight core**

Want to allow all cores to communicate (but keep the protocol simple!)

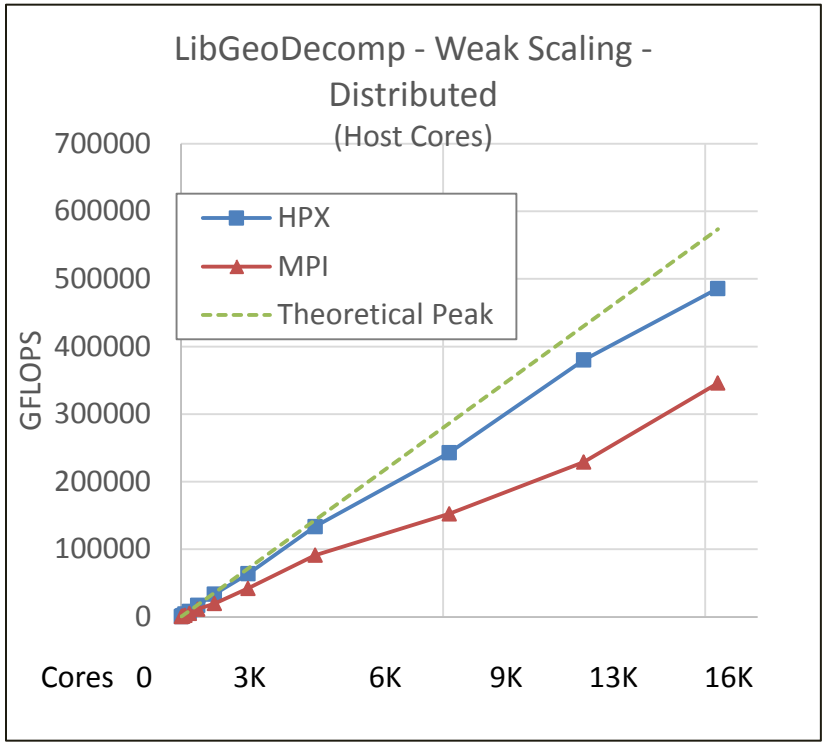
Lightweight communication is more important with lightweight cores

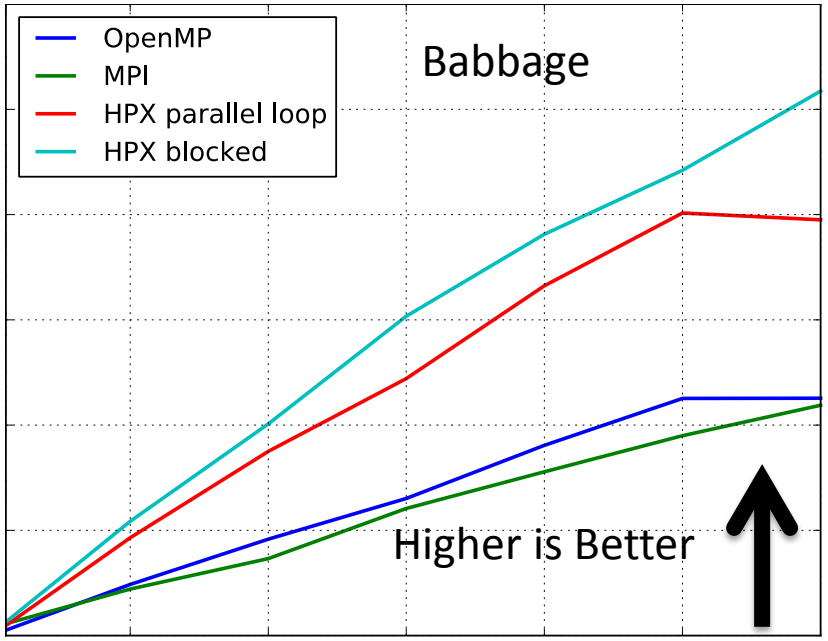
Lightweight Communication for Lightweight Cores

- **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?

Avoiding Synchronization

HPX Asynchronous Runtime Performs on Manycore

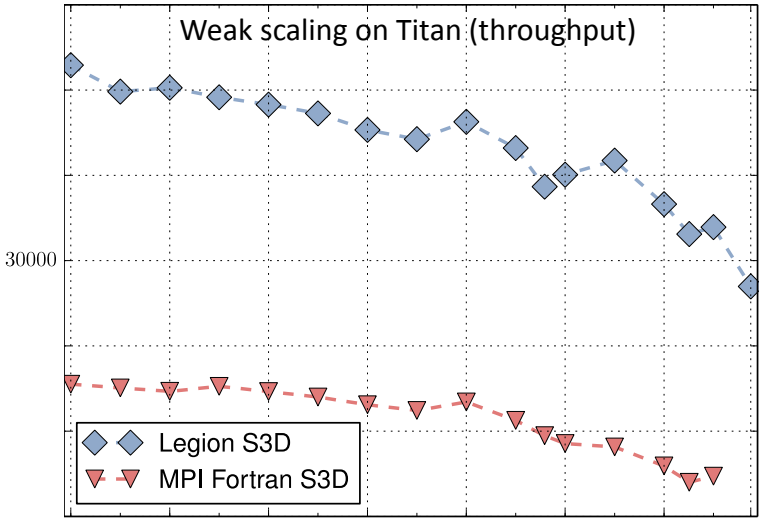


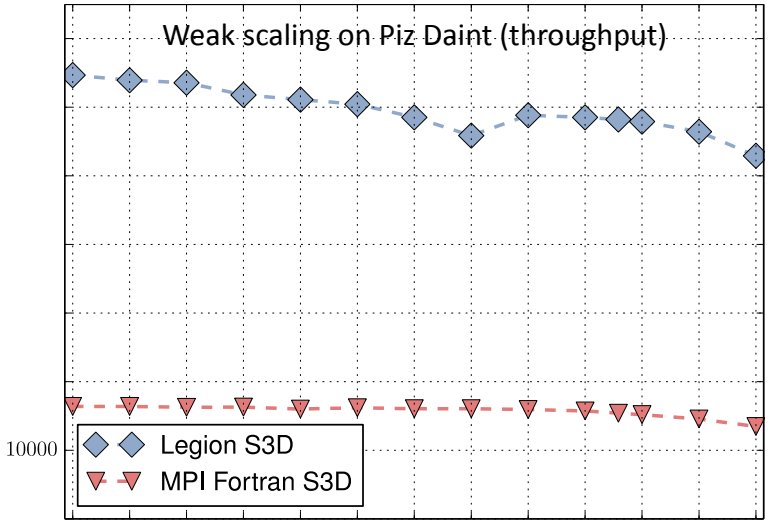


Credit: Harmut Kaiser, LSU and HPX team

Legion Programming Model & Runtime

- **Dynamic task-based**
  - Data-centric – tasks specify what data they access and how they use them (read-only, read-write, exclusive, etc.)
  - Separates task implementation from hardware mapping decisions
  - Latency tolerant
- **Port of S3D complete**
  - Currently programmed at the runtime layer (Realm)
- **Declarative specification of task graph in Legion**
  - Serial program
  - Read/Write effects on regions of data structures
  - Determine maximum parallelism





ExaCT Co-Design Center

Available Proxies and Kernels for OCR

Application	Programming Model
CoMD	Baseline MPI+OpenMP
CoMD	Legacy serial on OCR with newlib
CoMD	MPI-Lite
CoMD	CnC on OCR
CoMD	OCR
HPGMG	Baseline DOE Original in MPI+OpenMP
HPGMG	MPI-Lite
HPGMG	ROCR (R Stream ⇒ OCR)
HPGMG	OCR
LULESH	Baseline MPI+OpenMP
LULESH	Intel CnC
LULESH	Serial C
LULESH	CnC on OCR

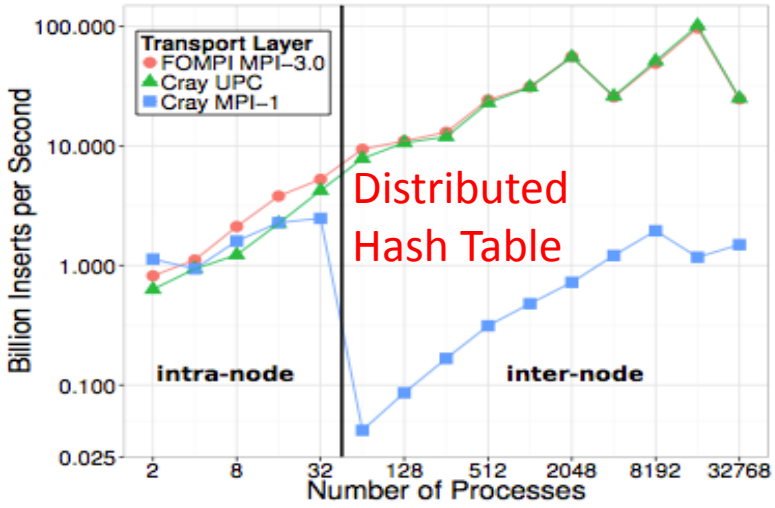
Application	Programming Model
miniAMR	Baseline DOE Original in OpenMP
SNAP	Baseline Translated into C from the DOE Original
SNAP	MPI-Lite
Tempest	Baseline DOE Original in MPI
Tempest	MPI-Lite
RSBench	Baseline in OpenMP
XSBench	Baseline MPI+OpenMP
XSBench	MPI-Lite
XSBench	OCR
Stencil1D	OCR
Stencil1D	OCR
Stencil1D	MPI
Stencil1D	MPI-Lite

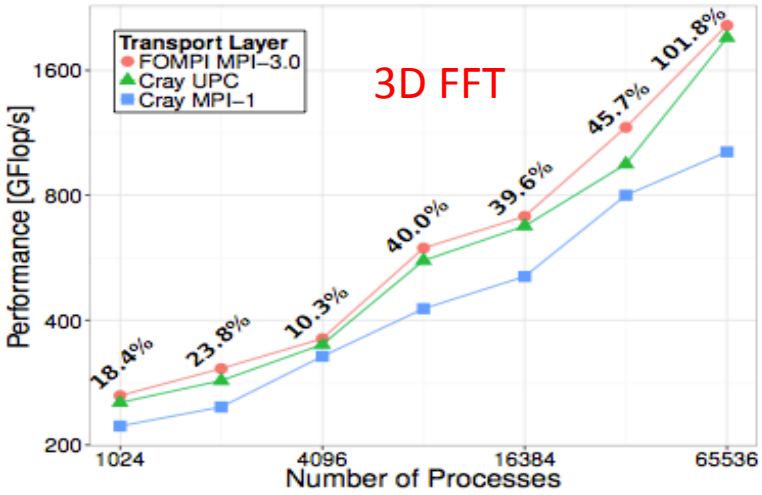
Application	Programming Model
Stencil1D	Serial
Cholesky	OCR
Cholesky	CnC on OCR
Smith Waterman	OCR
Smith Waterman	CnC on OCR
FFT	OCR
Fibonacci	OCR
Synthetic Aperture Radar (SAR)	OCR
Global Sum	OCR
triangle	Serial
triangle	OCR
Synch_p2p	OCR

https://xstack.exascale-tech.com/git/public/xstack.git

OpenMP and MPI Also have Ongoing Research

MPI: Fast implementations and extended interfaces for one-sided communication





Gerstenberger et al (SC13)

OpenMP: Location based on locales, places…

enMP 3.0: privatize data where possible, optimize cache usage

- "First touch" Implicit data layout

- Represent execution environment by collection of "locations" (Chapel/X10)
- **Map data, threads to a location; distribute data across locations**

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

If they come, we will build it!