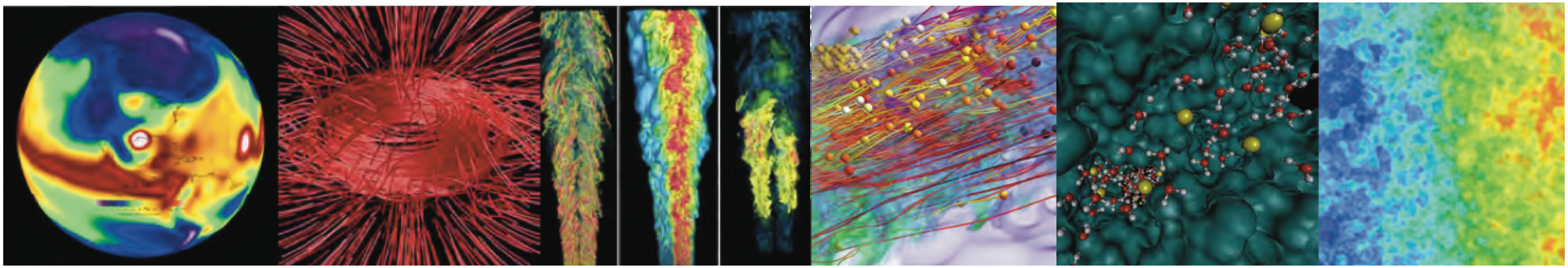


DEGAS:
Dynamic Exascale Global Address Space

Katherine Yelick, LBNL PI

Vivek Sarkar & John Mellor-Crummey, Rice

James Demmel, Krste Asanoviç & Armando Fox, UC Berkeley

Mattan Erez, UT Austin

Dan Quinlan, LLNL

Surendra Byna, Paul Hargrove, Steven Hofmeyr, Costin Iancu, Khaled Ibrahim, Leonid Oliker, Eric Roman, John Shalf, David Skinner, Erich Strohmaier, Samuel Williams, Yili Zheng, LBNL

Introductions


Vivek Sarkar


Kathy Yelick


John MC

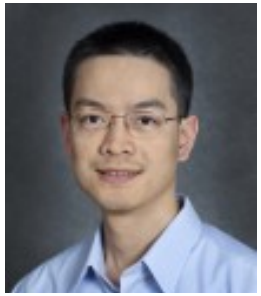

Costin Iancu


Paul Hargrove

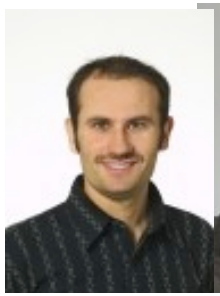

John Shalf


Dan Quinlan


Brian VS


Yili Zheng


Mattan Erez

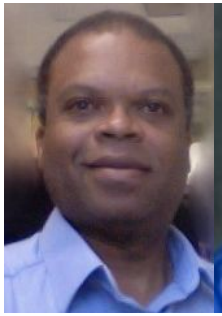

Lenny Oliker


Jim Demmel


Krste Asanovic

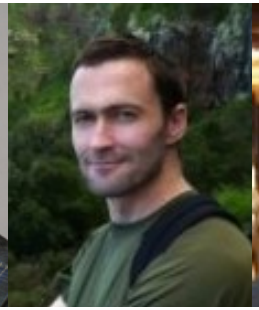

Eric Roman


Khaled Ibraham


Tony Drummond


Erich Strohmaier


Armando Fox


Steve Hofmeyer


Surendra Bayna

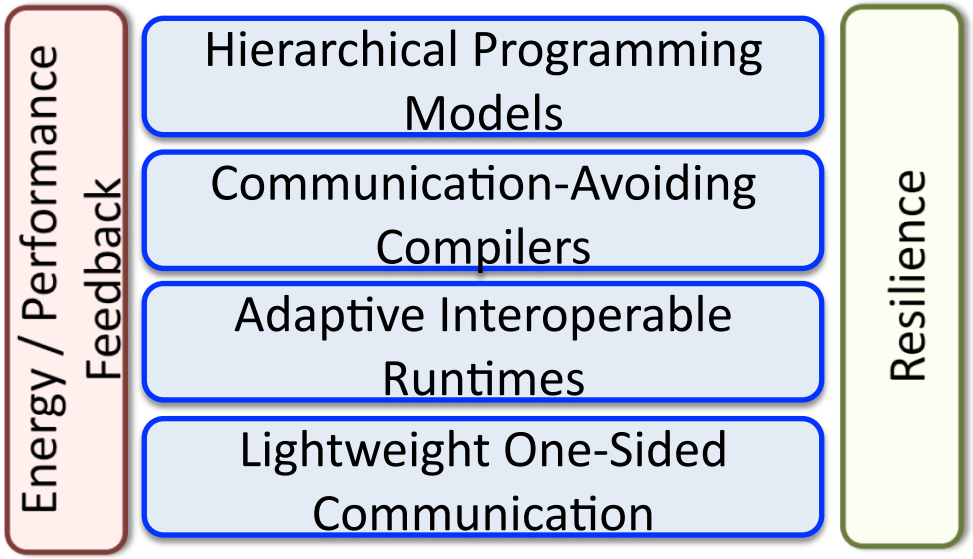

David Skinner


Frank Mueller


Sam Williams

DEGAS Mission

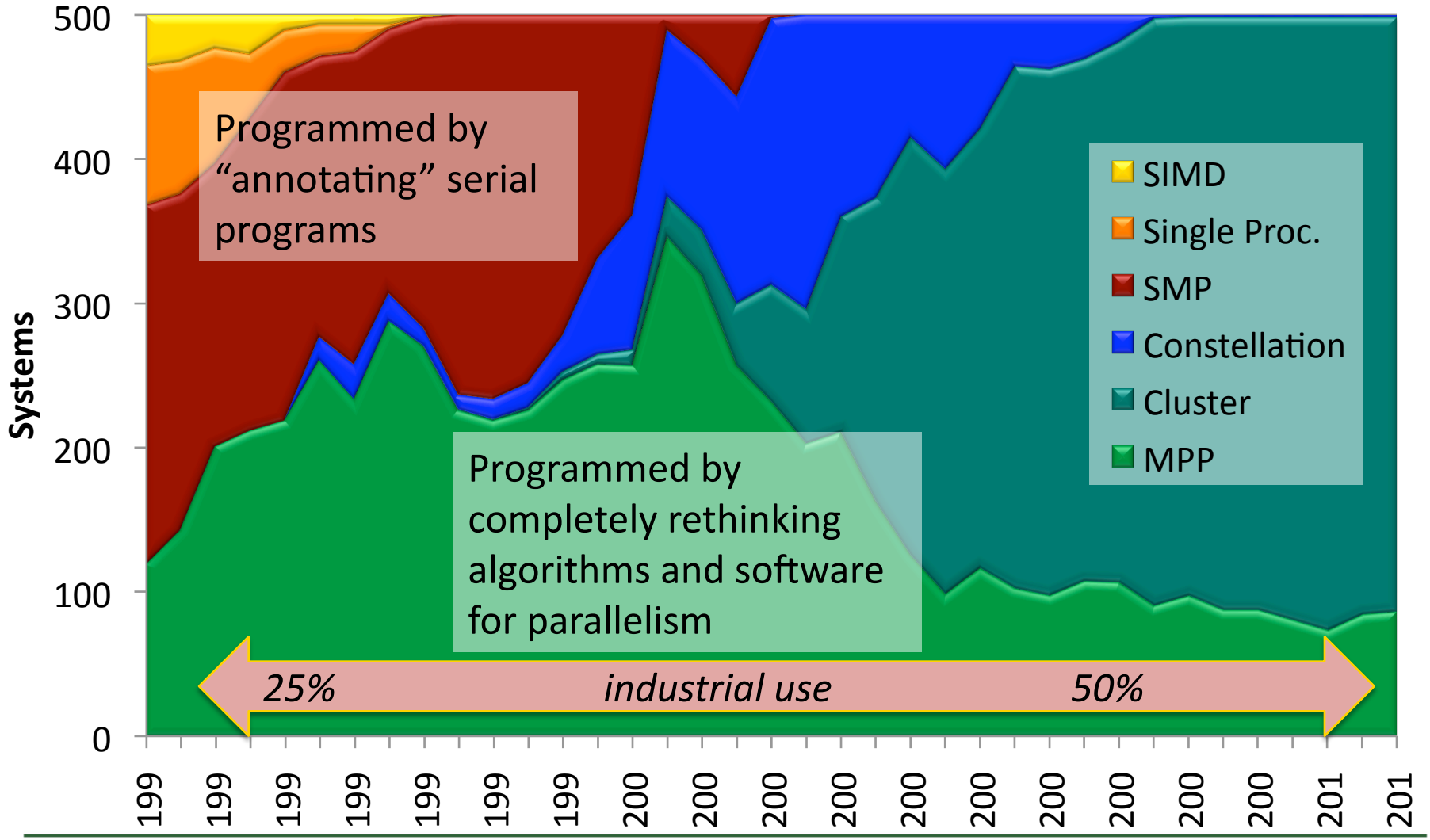
Mission Statement: To ensure the broad success of Exascale systems through a unified programming model that is productive, scalable, portable, and interoperable, and meets the unique Exascale demands of energy efficiency and resilience



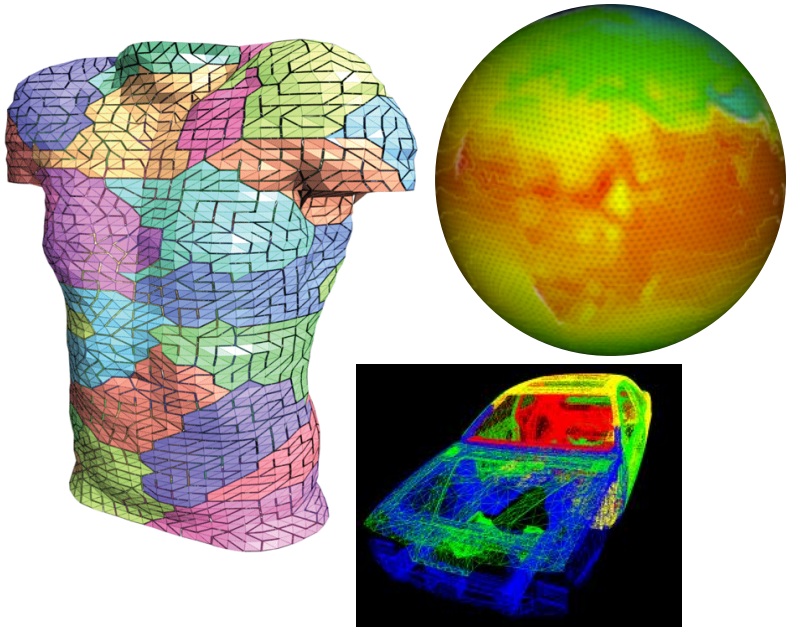
DEGAS Proposal: Goals and Objectives

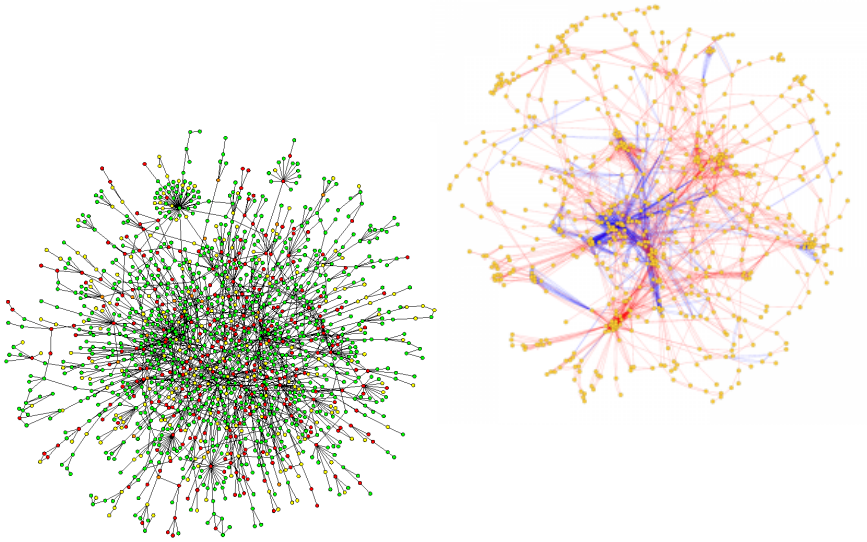
- **Scalability:**
  - Billion-way concurrency, thousand-way on chip with new architectures
- **Programmability:**
  - Convenient programming through a global address space and high-level abstractions for parallelism, data movement and resilience
- **Performance Portability:**
  - Ensure applications can be moved across diverse machines using implicit (automatic) compiler optimizations and runtime adaptation
- **Resilience:**
  - Integrated language support for capturing state and recovering from faults
- **Energy Efficiency:**
  - Avoid communication, which will dominate energy costs, and adapt to performance heterogeneity due to system-level energy management
- **Interoperability:**
  - Encourage use of languages and features through incremental adoption

Systems Drive Programming Models



Applications Drive Programming Models





Message Passing Programming

Divide up domain in pieces

Compute one piece and exchange

MPI, and many libraries

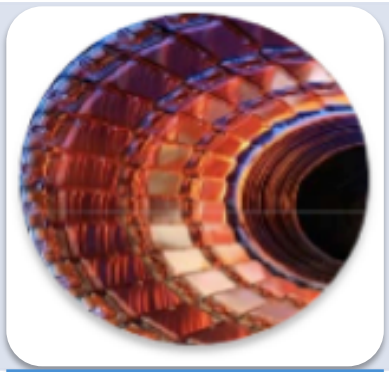
Global Address Space Programming

Each start computing

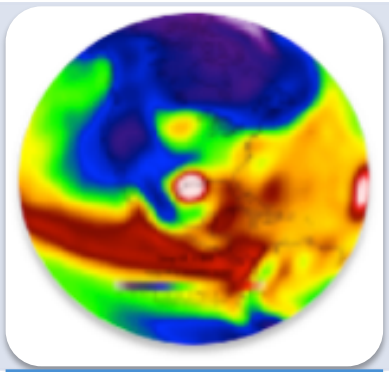
Grab whatever / whenever

UPC, CAF, X10, Chapel, Fortress, Titanium, GlobalArrays

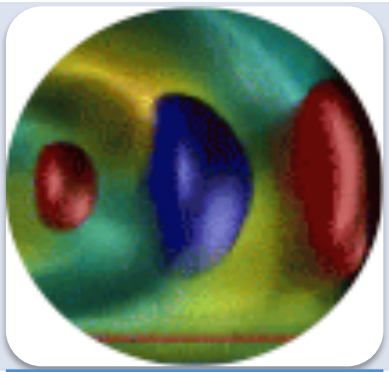
Science Problems Fit Across the "Irregularity" Spectrum



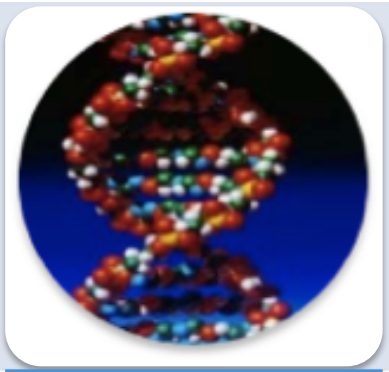
Massive Independent Jobs for Analysis and Simulations



Nearest Neighbor Simulations



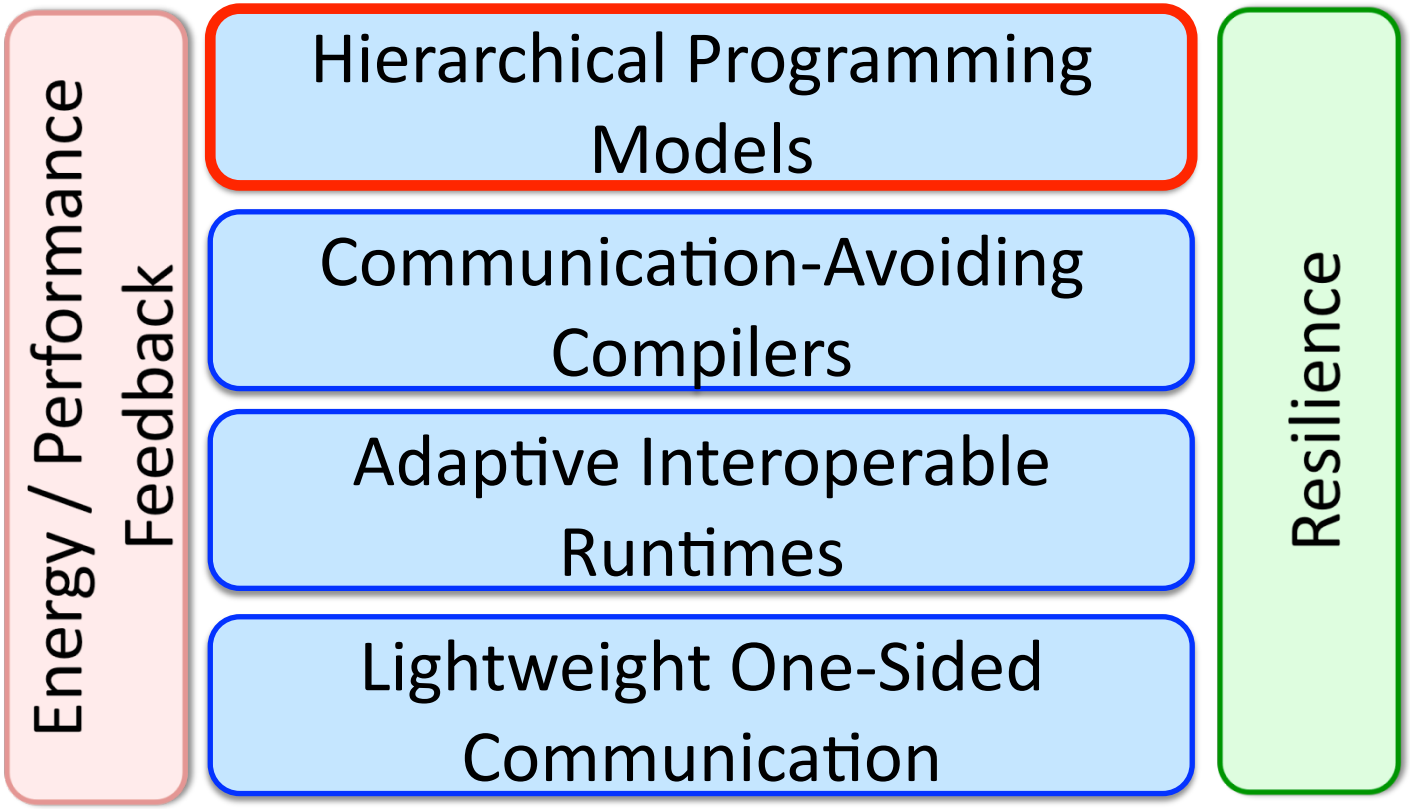
All-to-All Simulations



Random access, large data Analysis

… often they fit in multiple categories

DEGAS: Dynamic Exascale Global Address Space

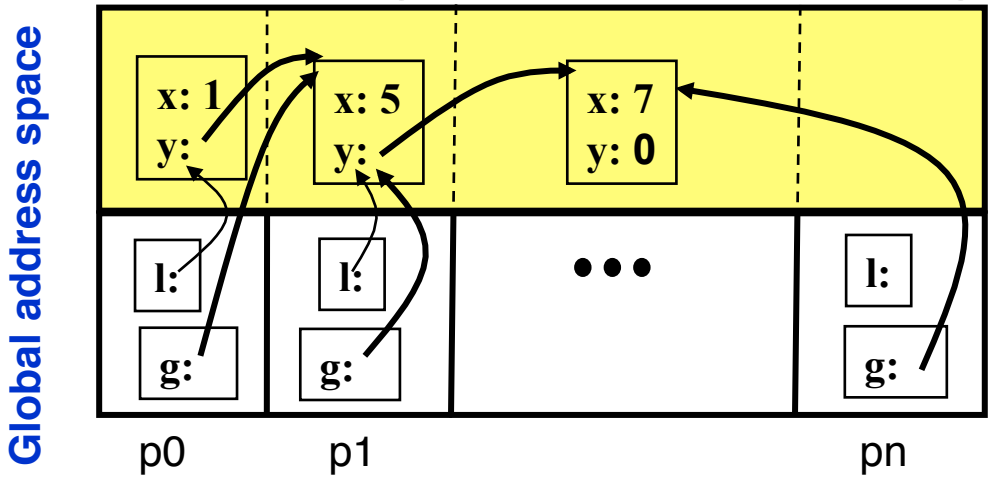


UPC, Co-Array Fortran (CAF), Habanero-C, and libraries!

Partitioned Global Address Space PGAS

Global address space: **directly read/write remote data**

Partitioned: **data is designated as local or global**



What's important:

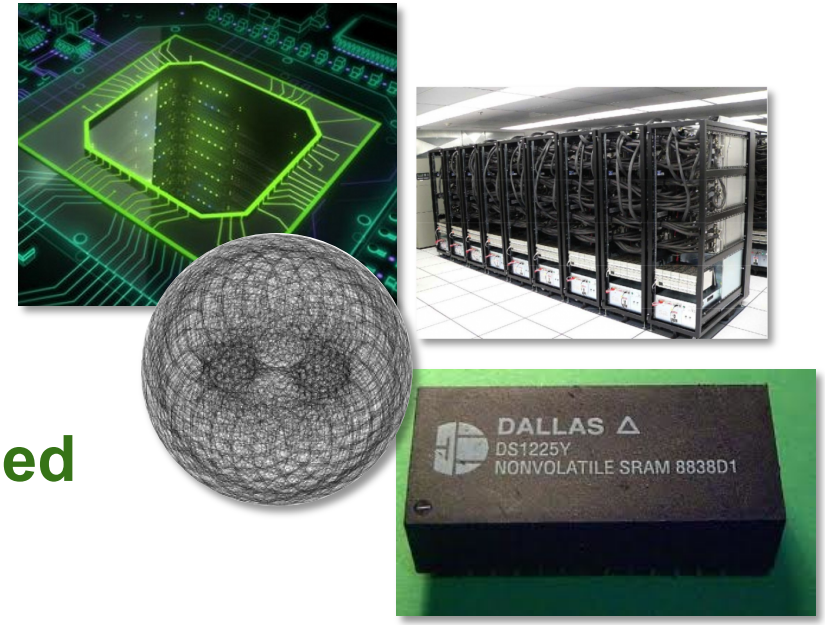
- **Global address space ≠ cache coherent shared memory, i.e., don't cache remote data**
- **Affinity control through partitioning ➔ scalability**
- **Improve bandwidth utilization through overlap**

One-sided communication works everywhere

PGAS programming model

```
*p1 = *p2 + 1;
A[i] = B[i];

upc_memput(A,B,64);
```



It is implemented using one-sided communication: put/get

Support for one-sided communication (DMA) appears in:

- Fast one-sided network communication (RDMA, Remote DMA)
- Move data to/from accelerators
- Move data to/from I/O system (Flash, disks,..)
- Movement of data in/out of local-store (scratchpad) memory

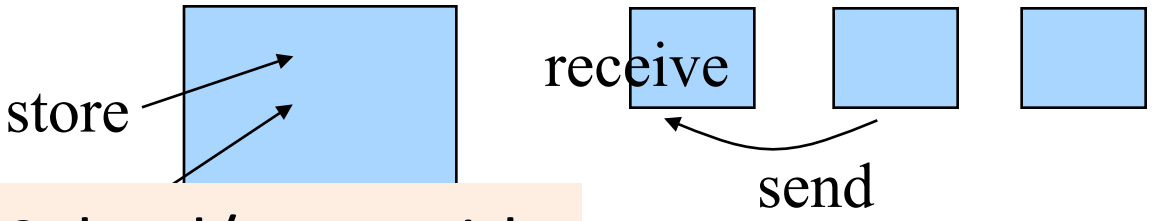
Two Distinct Parallel Programming Questions

- **What is the parallel control model?**

DEGAS: All three? With SPMD "default" plus data

data parallel (singe thread of control) **dynamic threads** **single program multiple data (SPMD)**

- **What is the model for sharing/communication?**

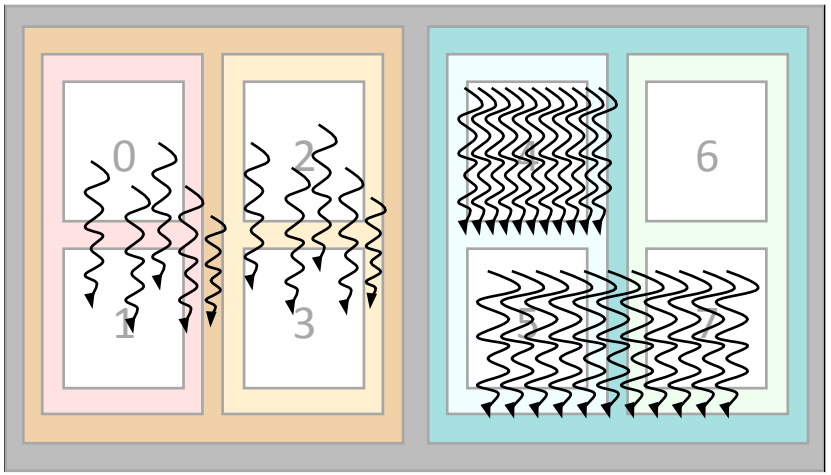


DEGAS: load/store with partitioning for locality

message passing

ed (implicit) or separate (explicit)

Hierarchical PGAS (HPGAS) hierarchical memory & control



Beyond (Single Program Multiple Data, SPMD)

- Hierarchical locality model for network and node hierarchies
- Hierarchical control model for applications (e.g., multiphysics)

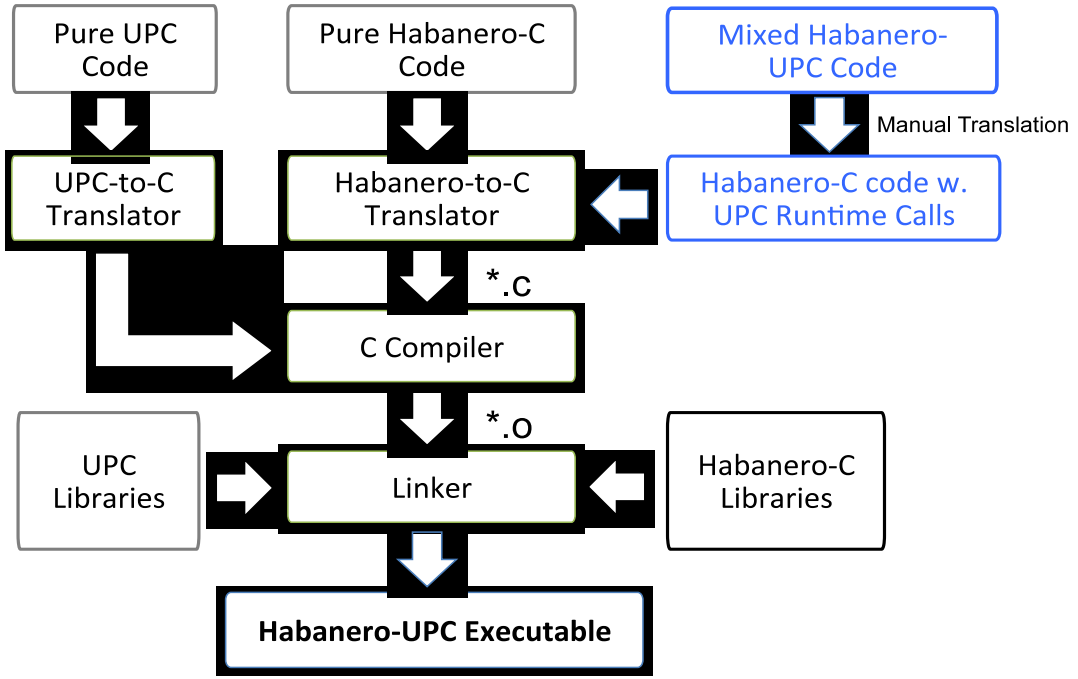
- **Option 1: Dynamic parallelism creation**
  - Recursively divide until… you run out of work (or hardware)
- **Option 2: Hierarchical SPMD with "Mix-ins"**
  - Hardware threads can be grouped into units hierarchically
  - Add dynamic parallelism with voluntary tasking on a group
  - Add data parallelism with collectives on a group

Two approaches: collecting vs spreading threads

Scalability of UPC with Dynamism of Habanero-C

Combined UPC and Habanero-C demonstrated

Two possible approaches going forward: single compiler or library with overloading



- **Habanero-C offers asynchronous tasks scheduled dynamically**
  - Extended to work with MPI (Asynchronous Partitioned Global Name Space, has core/node for communication) [Chatterjee et al, IPDPS 2013]
- **UPC has static (SPMD) threading**
- **Phalanx (based on C++) uses PGAS ideas globally with GPU support**
  - Provides a UPC++ like library using overloading and UPC runtime

DEGAS: Hierarchical Programming Model

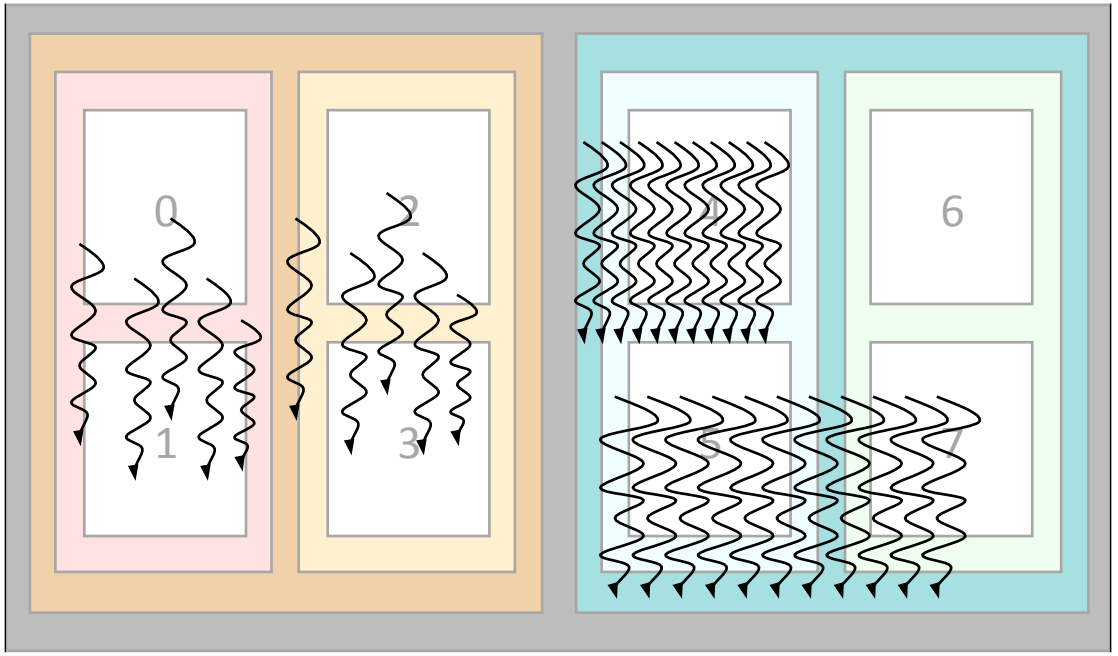
Goal: Programmability of exascale applications while providing scalability, locality, energy efficiency, resilience, and portability

- **Implicit constructs:** parallel multidimensional loops, global distributed data structures, adaptation for performance heterogeneity
- **Explicit constructs:** asynchronous tasks, phaser synchronization, locality

Built on scalability, performance, and asynchrony of PGAS models

- Language experience from UPC, Habanero-C, Co-Array Fortran, Titanium

Both intra and inter-node; focus is on node model



DEGAS: Hierarchical Programming Models

Languages demonstrate DEGAS programming model

- **Habanero-UPC:** Habanero's intra-node model with UPC's inter-node model
- **Hierarchical Co-Array Fortran (CAF):** CAF for on-chip scaling and more
- **Exploration of high level languages:** E.g., Python extended with H-PGAS

Language-independent H-PGAS Features:

- Hierarchical distributed arrays, asynchronous tasks, and compiler specialization for hybrid (task/loop) parallelism and heterogeneity
- Semantic guarantees for deadlock avoidance, determinism, etc.
- Asynchronous collectives, function shipping, and hierarchical places
- End-to-end support for asynchrony (messaging, tasking, bandwidth utilization through concurrency)
- Early concept exploration for applications and benchmarks

DEGAS: Hierarchical Programming Models

Language-independent H-PGAS features

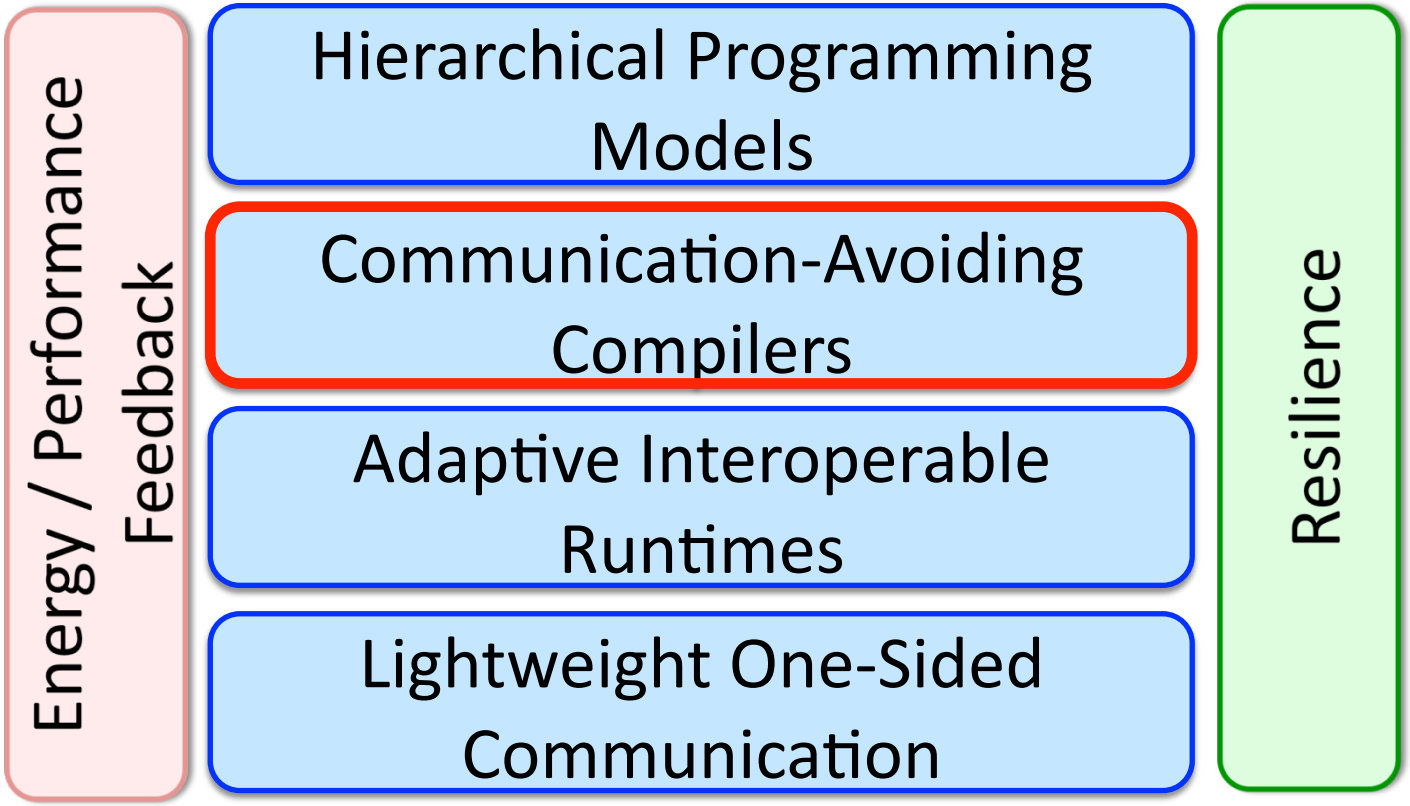
Languages demonstrate DEGAS programming model

- **Habanero-UPC:** Habanero's intra-node model with UPC's inter-node model
- **Hierarchical Co-Array Fortran (CAF):** CAF for on-chip scaling and more
- **Exploration of high level languages:** E.g., Python extended with H-PGAS

First year: UPC+Habanero C (possibly with C++ library version)

```
Using namespace phalanx::gasnet;
// A static "int" type global variable shared by all threads
global_var_t<int> gv = 0;
global_lock_t gv_lock;
// All threads execute the update function
void update() {
  // shared data accesses with race condition
  for (int i=0; i<100; i++)
    gv = gv + 1;
  barrier();
  // shared data accesses with lock protection
  for (int i=0; i<100; i++) {
    gv_lock.lock(); gv = gv + 1; gv_lock.unlock();
  }
}
```

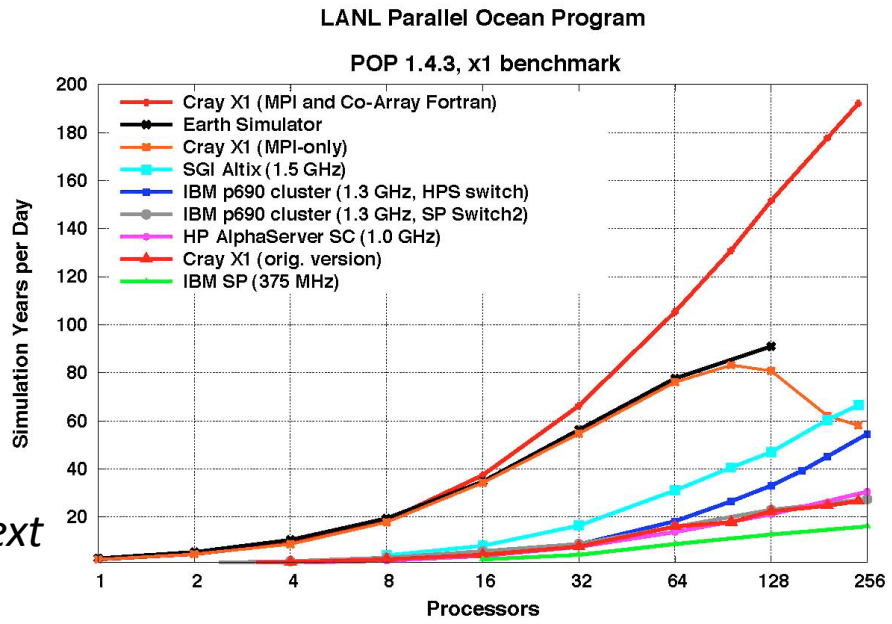
DEGAS: Dynamic Exascale Global Address Space



Communication-avoiding algorithms generalized to compilers, and communication optimizations in PGAS

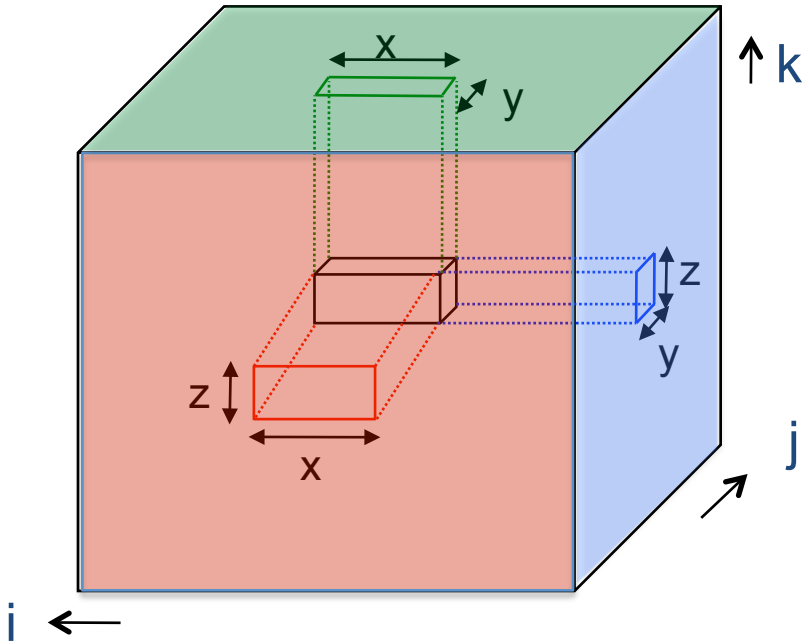
Co-Array Fortran (CAF) Demonstrates Efficiency of Overlap from One-Sided Communication

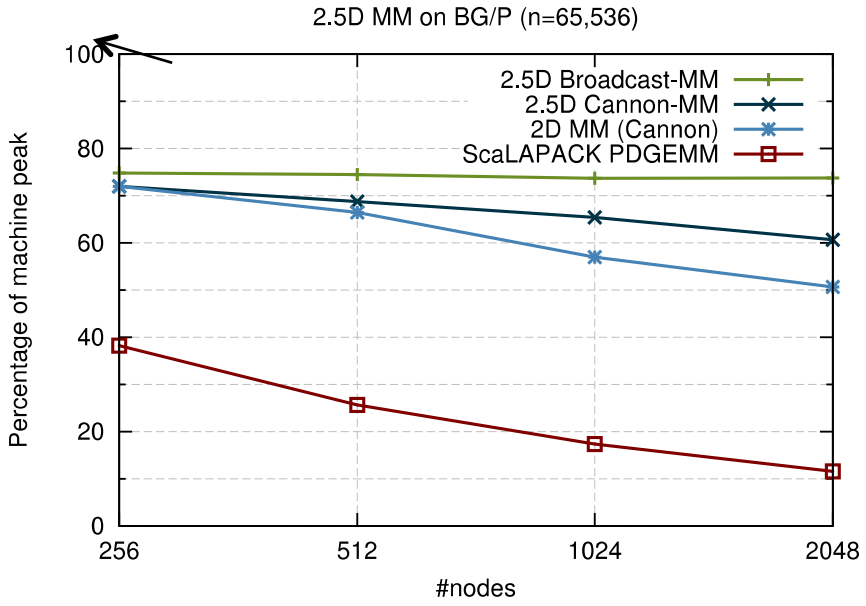
- **POP Ocean model has Co-Array version**
  - Communication-intensive (reductions and halo ghost exchanges)
  - Historical: CAF faster than MPI on Cray X1
- **CAF 2.0 provides more programming flexibility than original CAF**
- **CGPOP mini-App in CAF 2.0**
  - Using HPCToolkit for tuning
  - Limited by serial code (multi-dimensional array pointers) and parallel I/O (netCDF)



Worley et al results shown for historical context
See also Lumsdaine et al for Graph500, SC12

Towards Communication-Avoiding Compilers: Deconstructing 2.5D Matrix Multiply





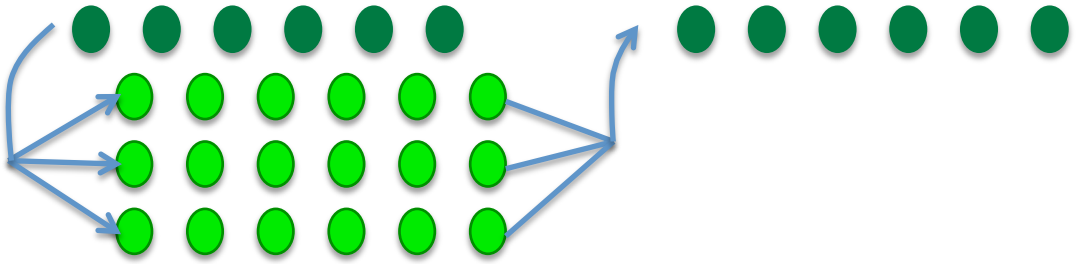
Matrix Multiplication code has a 3D iteration space
Each point in the space is a constant computation (*/+)

for i, for j, for k C[i,j] … A[i,k] … B[k,j] …

These are not just "avoiding," they are "communication-optimal"

Generalizing Communication Optimal Transformations to Arbitrary Loop Nests

1.5D N-Body: Replicate and Reduce



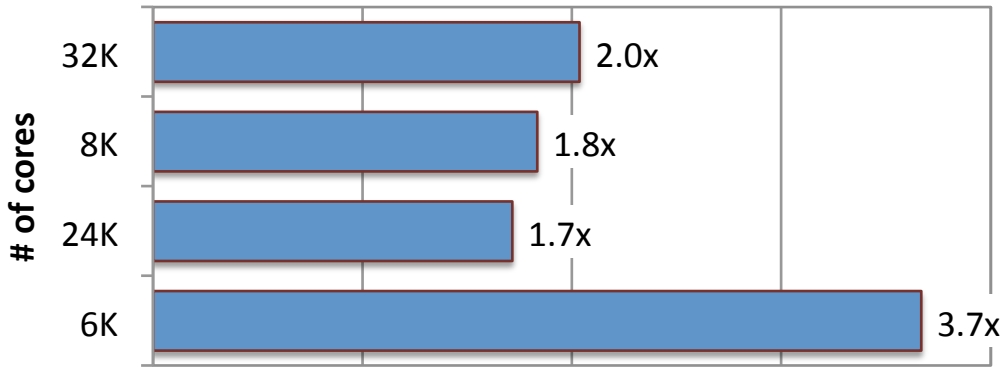
The same idea (replicate and reduce) can be used on (direct) N-Body code:

1D decomposition → "1.5D"

Does this work in general?

- *Yes, for certain loops and array expressions*
- *Relies on basic result in group theory*
- *Compiler work TBD*

Speedup of 1.5D N-Body over 1D

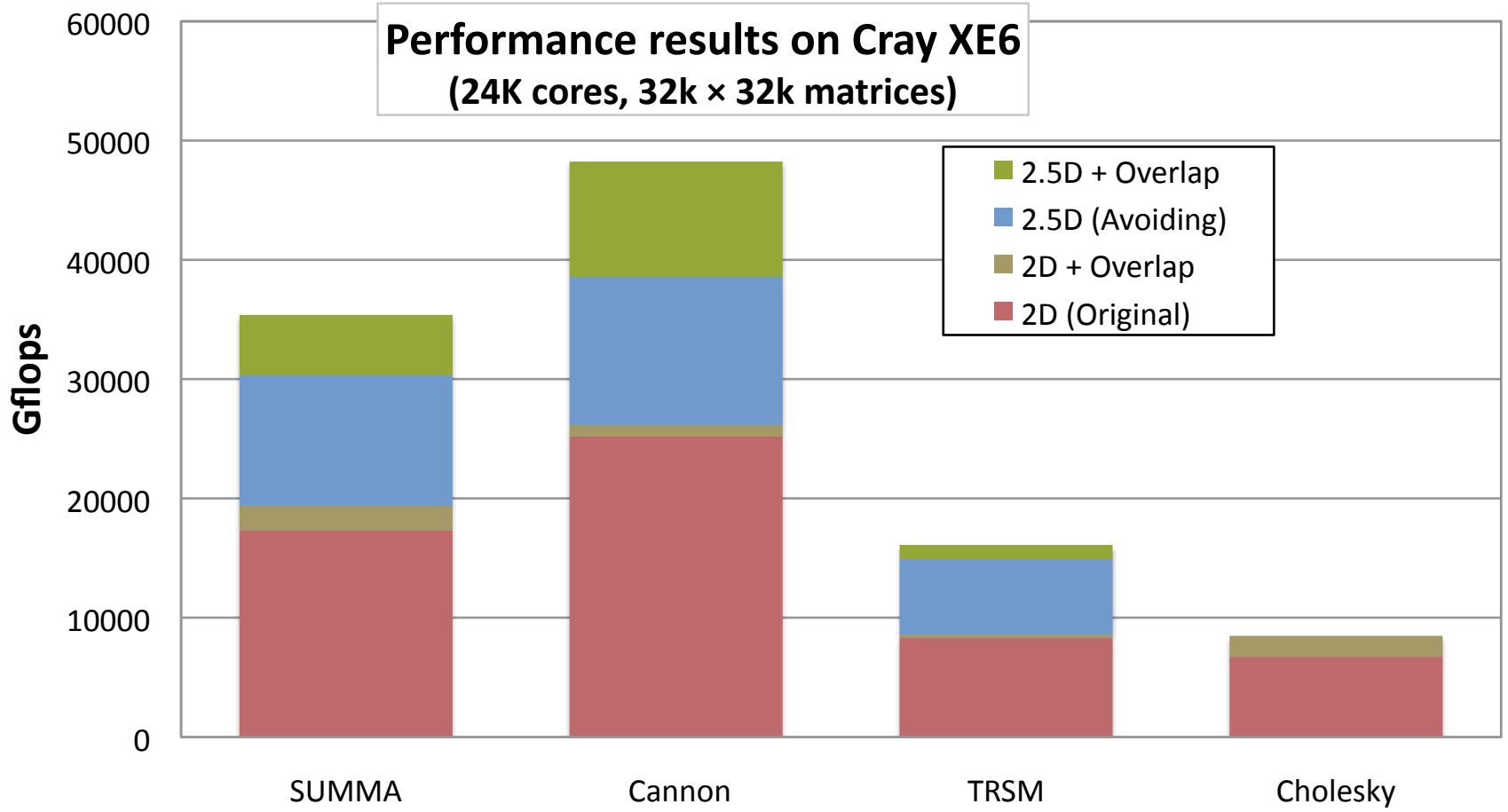


A Communication-Optimal N-Body Algorithm for Direct Interactions, *Driscoll et al, IPDPS'13*

Generalizing Communication Lower Bounds and Optimal Algorithms

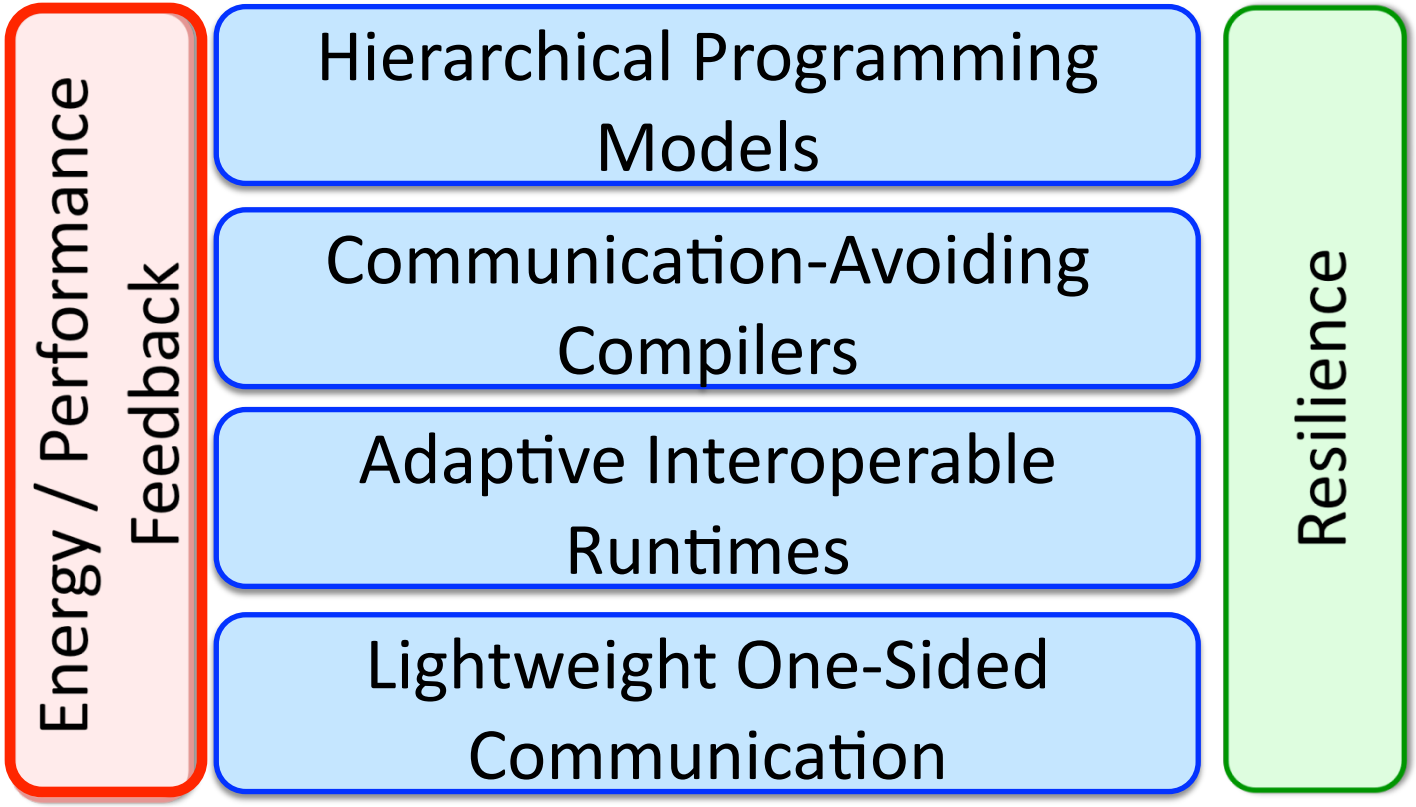
- **For serial matmul, we know #words_moved = $\Omega(n^3/M^{1/2})$, attained by tile sizes $M^{1/2} \times M^{1/2}$**
  - Where do all the ½'s come from?
- **Thm (Christ,Demmel,Knight,Scanlon,Yelick): For any program that "smells like" nested loops, accessing arrays with subscripts that are linear functions of the loop indices, #words_moved = $\Omega(\#\text{iterations}/M^{e})$, for some e we can determine**
- **Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes**
- **Long term goal: All compilers should generate communication optimal code from nested loops**

Communication Overlap Complements Avoidance



- **Even with communication-optimal algorithms (minimized bandwidth) there are still benefits to overlap and other things that speed up networks**
- ***Communication Avoiding and Overlapping for Numerical Linear Algebra*, Georganas et al, SC12**

DEGAS: Dynamic Exascale Global Address Space

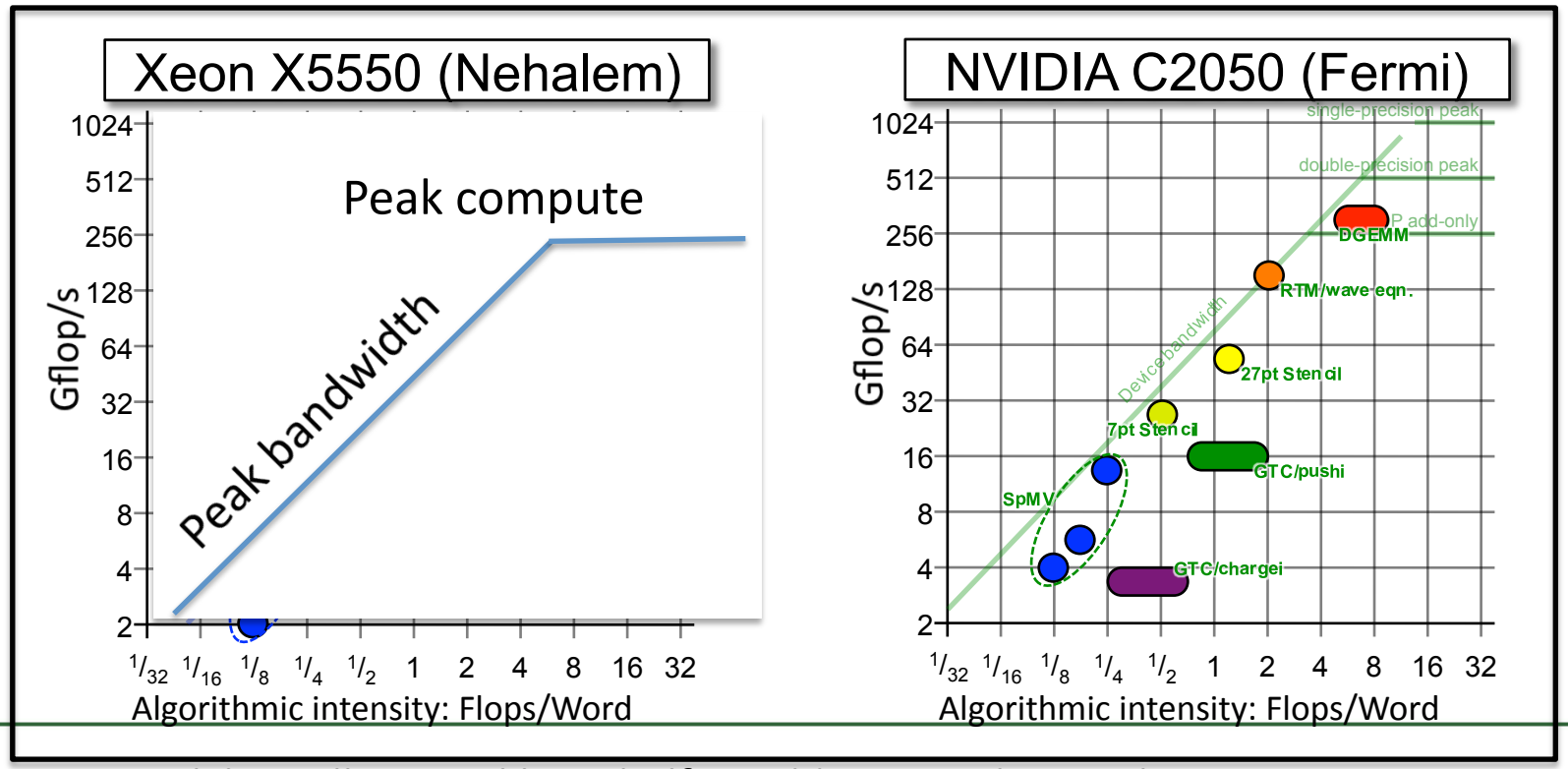


Integrated stack extends PGAS to be:

- **Hierarchical for machines and applications**
- **Communication-avoiding for performance and energy**

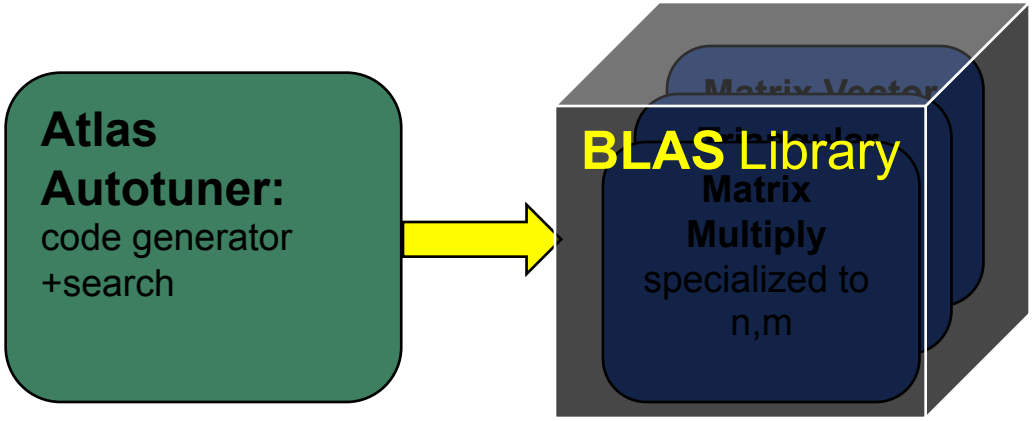
Autotuning: Write Code Generators

- **Major Exascale issues are in the compute node (heterogeneity, scratchpad memory, NVRAM,…)**
- **Autotuning avoids two hard compiler problems: 1) dependence analysis and 2) accurate performance models**
- **Popular in libraries: Atlas, FFTW, OSKI,…**



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

Approaches to Autotuning



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

- DEGAS: node runtime for NUMA, Scratchpad,… SIMD?
- **Transform high level representation (FFTW, Spiral)**
- **Compile a domain-specific language (D-TEC)**
- **Compile a general-purpose language + annotations (X-Tune)**
- **Dynamic compilation of a domain-specific (SEJITS)**
- DEGAS: interoperability

Correctness Tools for Concurrency and Numerics

- **Active Testing**
  - **Phase 1:** Static or dynamic analysis to find potential concurrency bug patterns (data races, deadlocks, etc.)
  - **Phase 2:** "Direct" testing (or model checking) based on the bug patterns obtained from

Correctness tools for different programming models:
PGAS, MPI, dynamic parallelism

- Identify source of non-determinism in executions
- Concurrency bugs include data races, atomicity violations, non-reproducible floating-point results, etc.

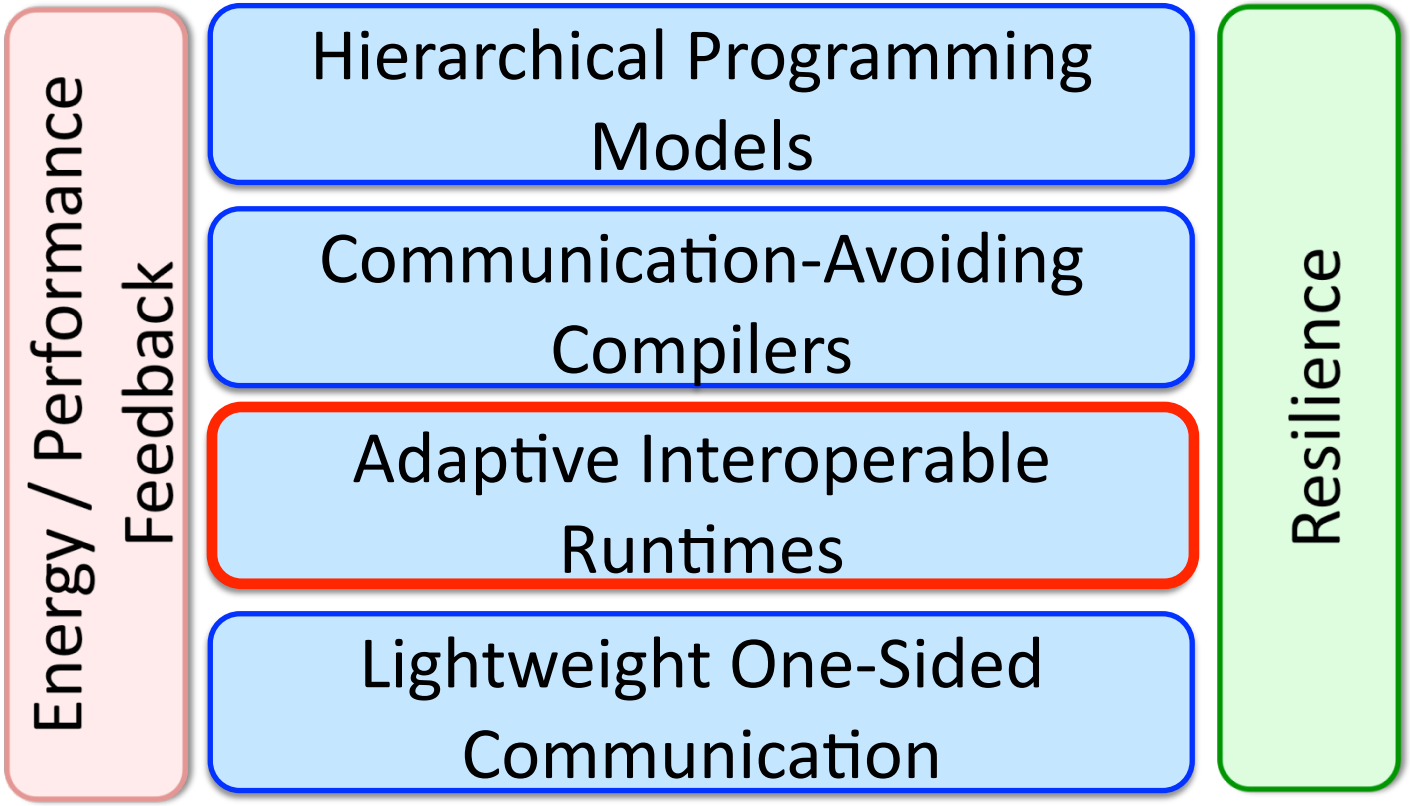
Concolic Testing (concrete + symbolic execution)

- Synthesize inputs and simplified executions able to reproduce bugs using the minimal amount of concurrency

Delta Debugging for Floating Point code:

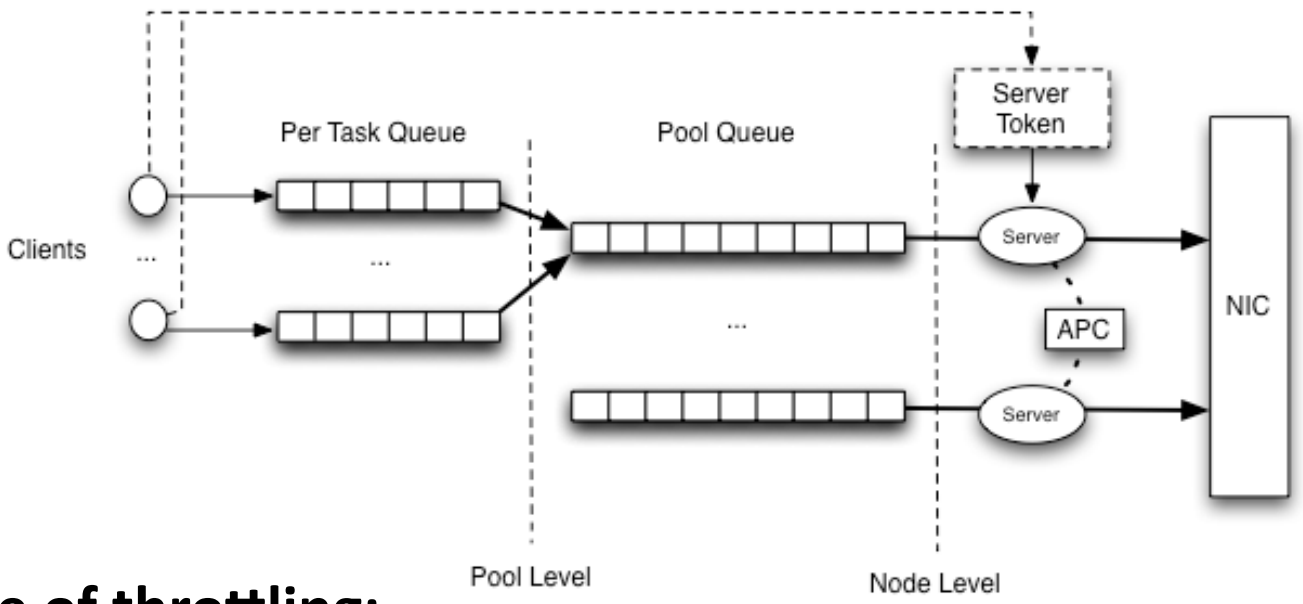
- How much precision does each variable need?
- Approach: automatically rewrite and test sensitivity

DEGAS: Dynamic Exascale Global Address Space



LITHE, JUGGLE: adaptive and efficient runtime

Resource management will require adaptive runtime systems

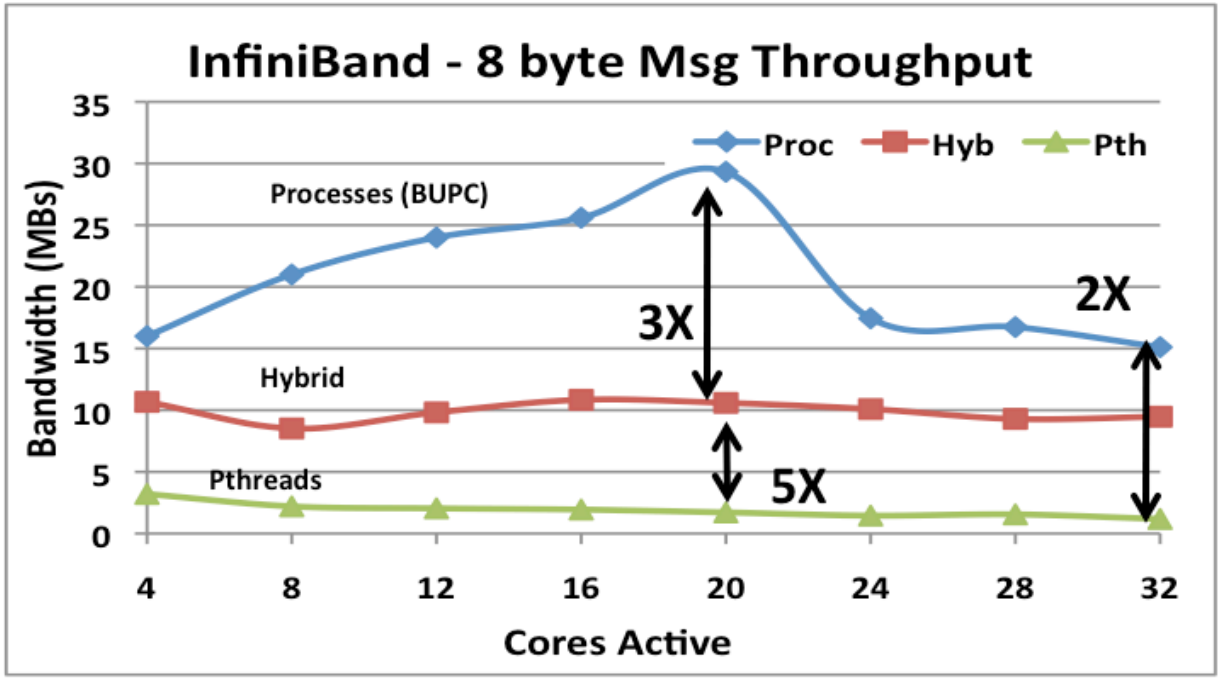


- **The value of throttling:**
- **the number of messages in flight per core provides up to 4X performance improvements**
- **the number of active cores per node can provide additional 40% performance improvement for**
- **Developing a**daptation based on history and (user-supplied) intent

THOR: Throughput Oriented Runtime to Manage Resources

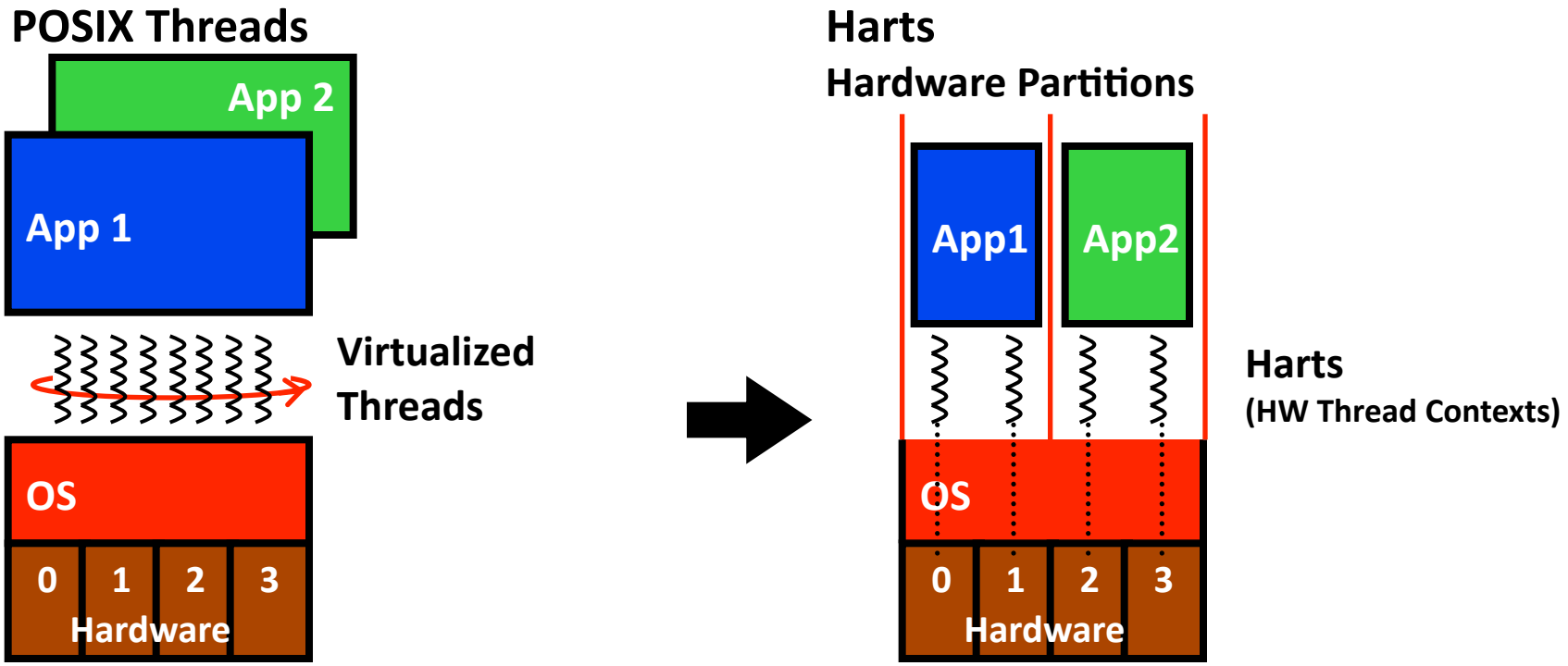
Juggle: Management of critical resources is increasingly important:

- **Memory and network bandwidth limited** by cost and energy
- **Capacity limited at many levels:** network buffers at interfaces, internal network congestion are real and growing problems



Having more than 4 submitting processes can negatively impact performance by up to 4x

Lithe Scheduling Abstraction: "Harts": Hardware Threads



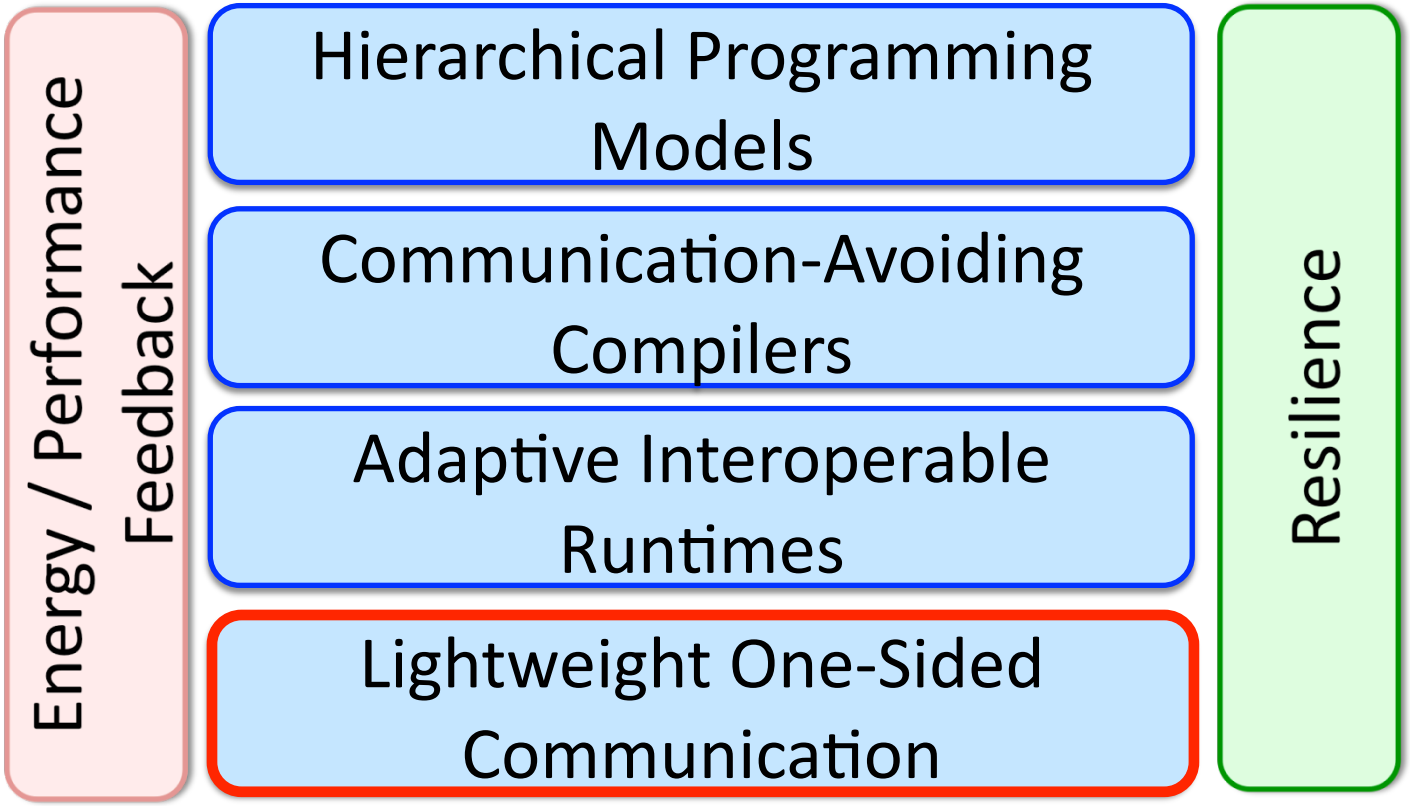
- *Merged* resource and computation abstraction.

- More accurate resource abstraction.

Release planned for this spring with substantial rewrite

1) Separation of Lithe's API from OS functionality
2) Restructuring to support future preemption work.
3) Updated OpenMP and TBB ports.
4) Documentation: lithe.eecs.berkeley.edu

DEGAS: Dynamic Exascale Global Address Space

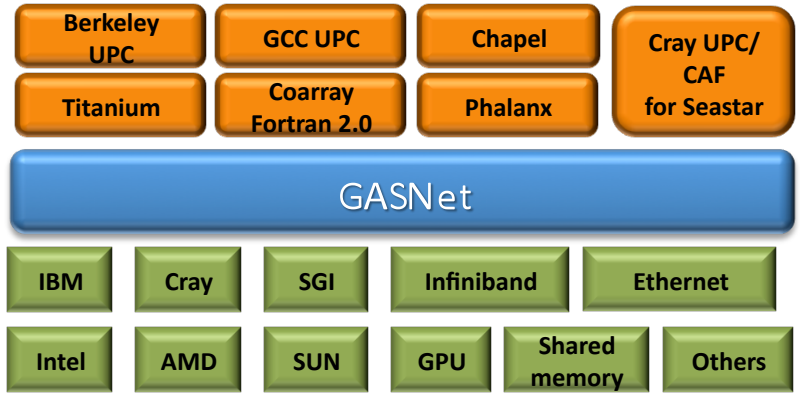


Next Generation GASNet

DEGAS: Lightweight Communication (GASNet-EX)

GASNet-EX plans:

- **Congestion management:** for 1-sided communication with ARTS
- **Hierarchical:** communication management for H-PGAS
- **Resilience:** globally consist states and fine-grained fault recovery
- **Progress:** new models for scalability and interoperatbility



Leverage GASNet (redesigned)

- Major changes for on-chip interconnects
- Each network has unique opportunities

- **Interface under design: "Speak now or…"**
  - https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/.

New Systems with New Networks

- **Upgrades in the DOE computing landscape**
  - XK7 (Gemini interconnect but combined with GPUs)
  - BG/Q (IBM custom interconnect, PAMI interface)
  - Cascade (Cray Aries interconnect)

TOP500
November 2012

Titan at ORNL
(#1, 17+ PF)


Sequoia at LLNL
(#2, 16+ PF)

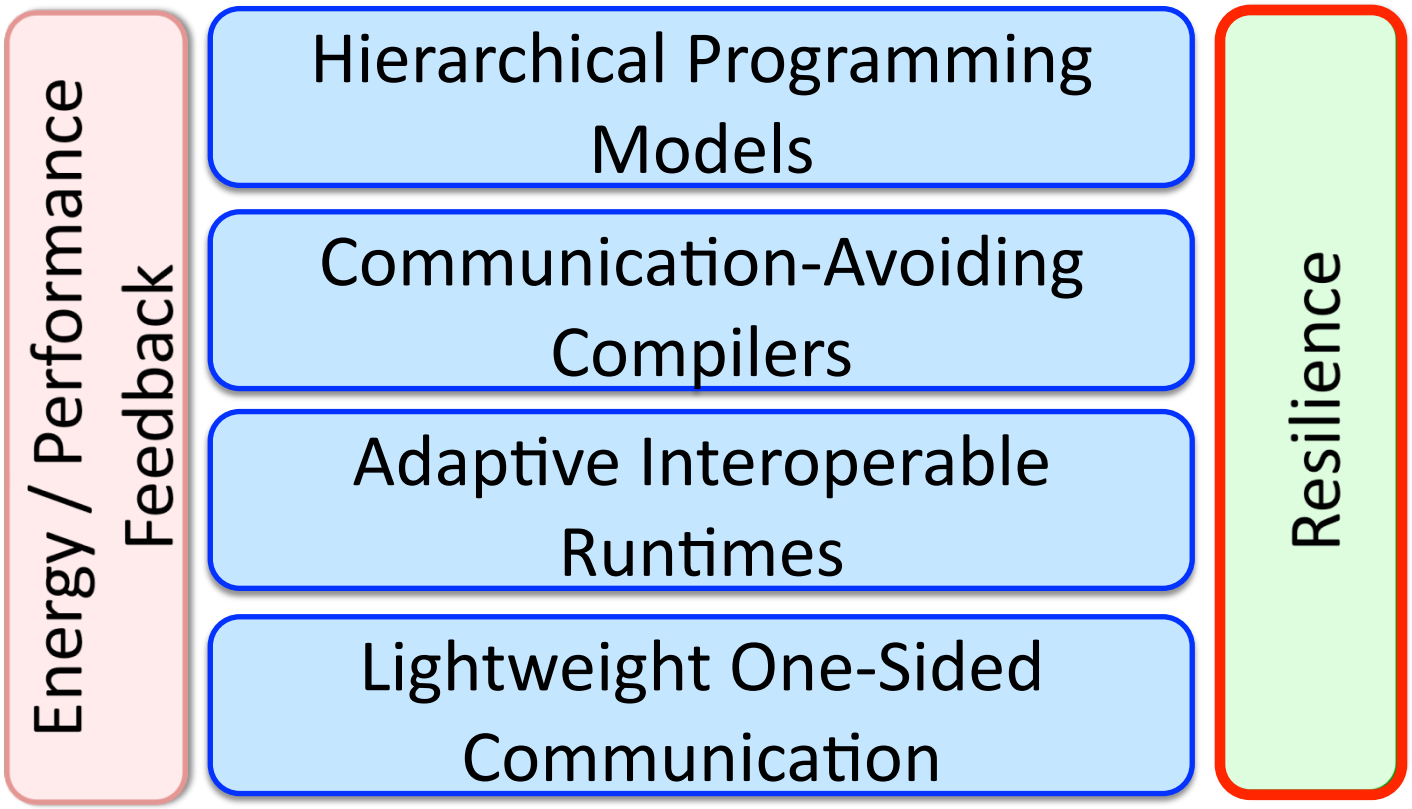
Mira at ANL
(#4, 8+ PF)


Cielo at LANL/SNL
(#18, 1+PF)

Hopper at LBNL
(#19, 1+PF)

Performance increase on Gemini: 20% for CG, 40% for GUPPIE

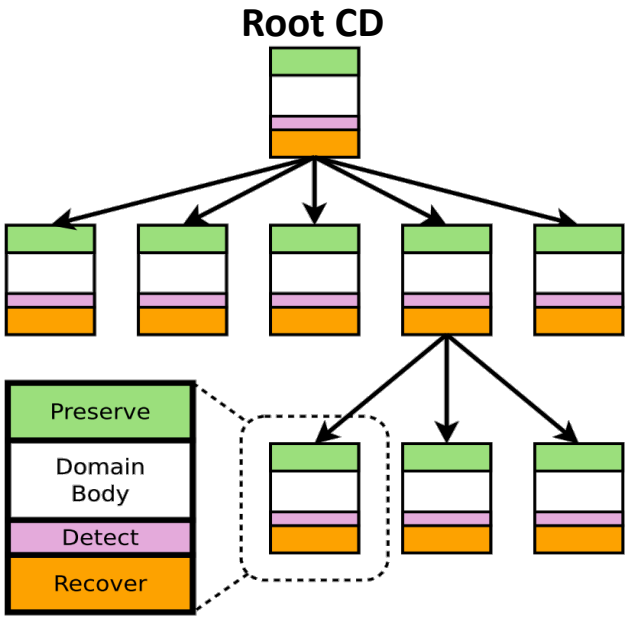
DEGAS: Dynamic Exascale Global Address Space



Containment Domans and BLCR (Berkeley Lab Checkpoint Restart)

Resilience Approaches

- **Containment Domains (CDs) for trees**
  - Flexible resilience techniques (mechanism not policy)
  - Each CD provides own recovery mechanism
  - Analytical model: 90%+ efficiency at 2 EF
    vs. 0% for conventional checkpointing
- **Berkeley Lab Checkpoint Restart**
  - BLCR is a system-level Checkpoint/Restart
    - Job state written to filesystem or memory; works on most HPC apps
  - Checkpoint/Restart can be used for roll-back recovery
    - a course-grained approach to resilience
    - BLCR also enables use for job migration among compute nodes
  - Requires support from the MPI implementation
  - Impact: part of standard Linux release



- ***Preserve*** data on domain start
- ***Compute*** *(domain body)*
- ***Detect*** faults before commit
- ***Recover*** from detected errors

DEGAS Software Stack