



# DEGAS

## *Dynamic Exascale Global Address Space*

*Katherine Yelick, LBNL PI*

*Vivek Sarkar & John Mellor-Crummey, Rice*

*James Demmel, Krste Asanović UC Berkeley*

*Mattan Erez, UT Austin*

*Dan Quinlan, LLNL*

*Paul Hargrove, Steven Hofmeyr, Costin Iancu, Khaled*

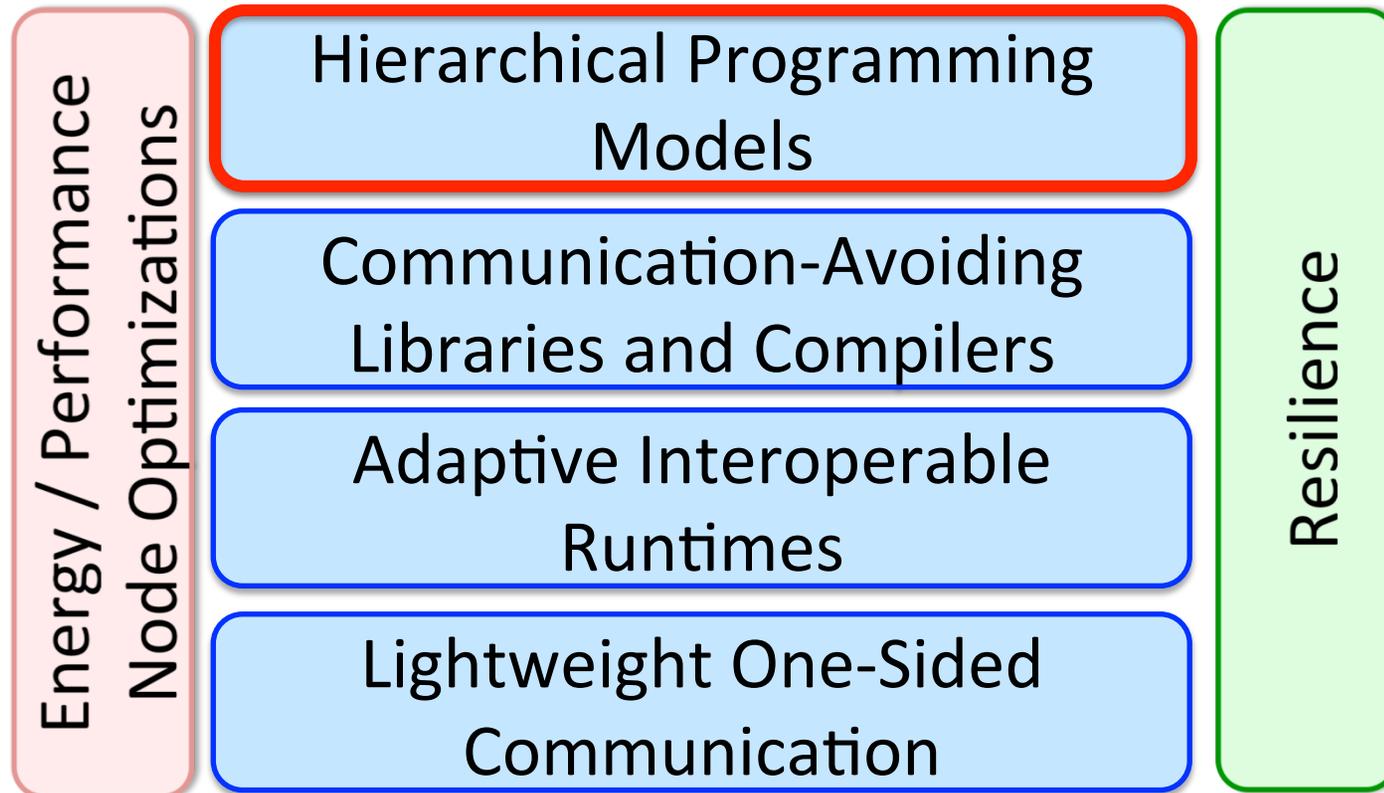
*Ibrahim, Leonid Oliker, Eric Roman, John Shalf, Erich*

*Strohmaier, Samuel Williams, Yili Zheng, LBNL*

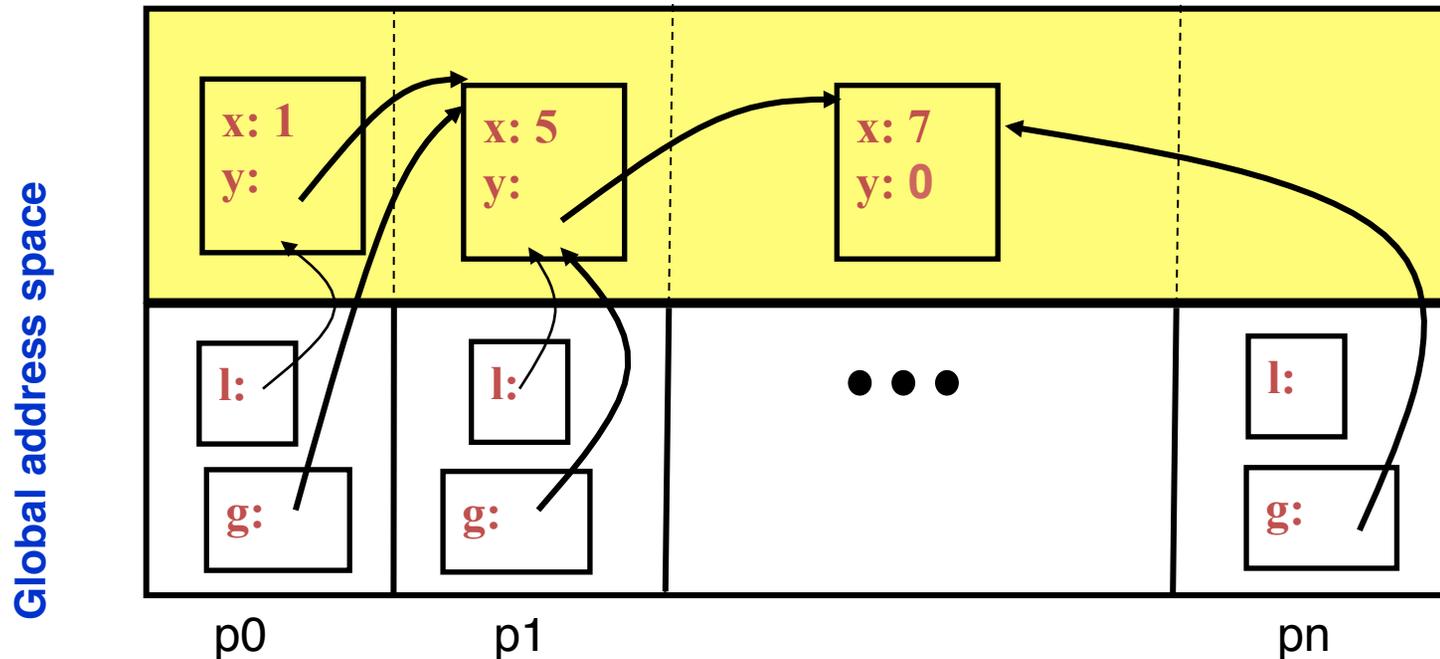
*+ Several postdocs and students!*

# DEGAS: Dynamic Exascale Global Address Space

---

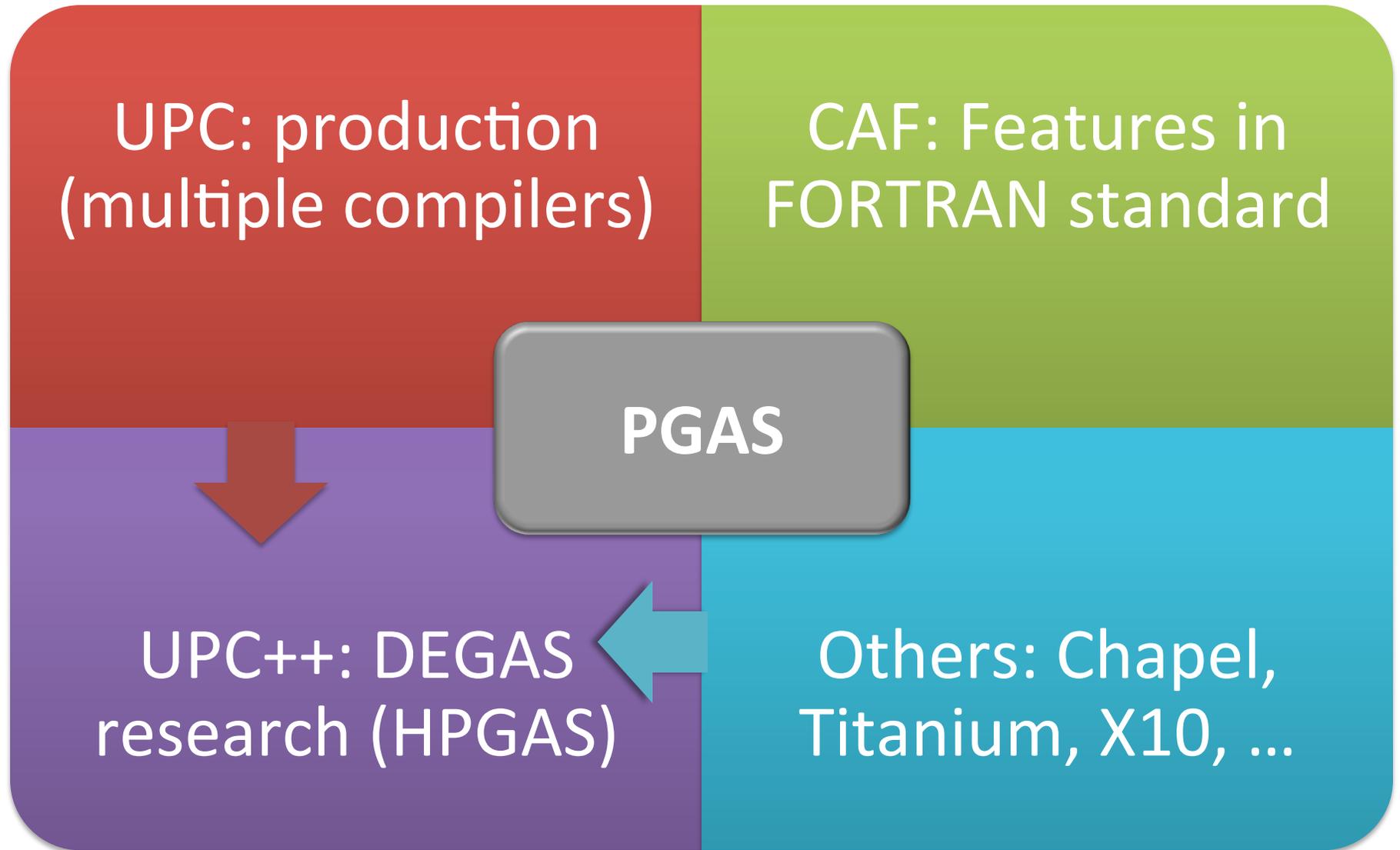


# PGAS: Partitioned Global Address Space



- **Global Address Space: Directly read/write report memory**
- **Partitioned: data layout and control layout through partitioning**
- **Runs on shared memory and scales on distributed memory**

# PGAS Languages



# UPC++ DEGAS: PGAS with “Mixins”

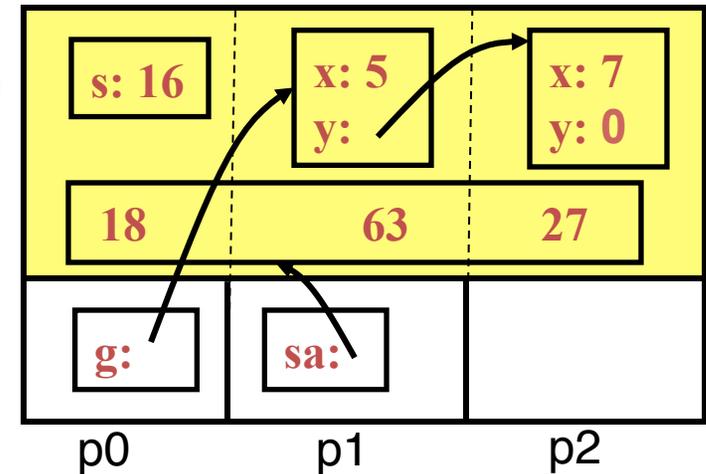
- UPC++ uses templates (no compiler needed)

```
shared_var<int> s;  
global_ptr<LLNode> g;  
shared_array<int> sa(8);
```

- Default execution model is SPMD, but

- Teams for hierarchical algorithms and machines

```
teamsplit (team) { ... }
```



- Remote methods, async

```
async(place) (Function f, T1 arg1,...);  
wait();      // other side does poll();
```

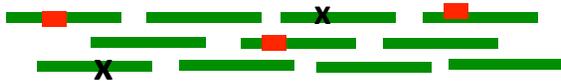
- Interoperability is key; UPC++ can be use with OpenMP or MPI

---

# How to Use DEGAS for Application and Machine Challenges at Exascale

# App Challenge #1: Random Access to Large Memory

## Meraculous Assembly Pipeline reads



## k-mers



## contigs



**Human:** 44 hours to 20 secs  
**Wheat:** “doesn’t run” to 32 secs

## Scaffolds using Scalable Alignment



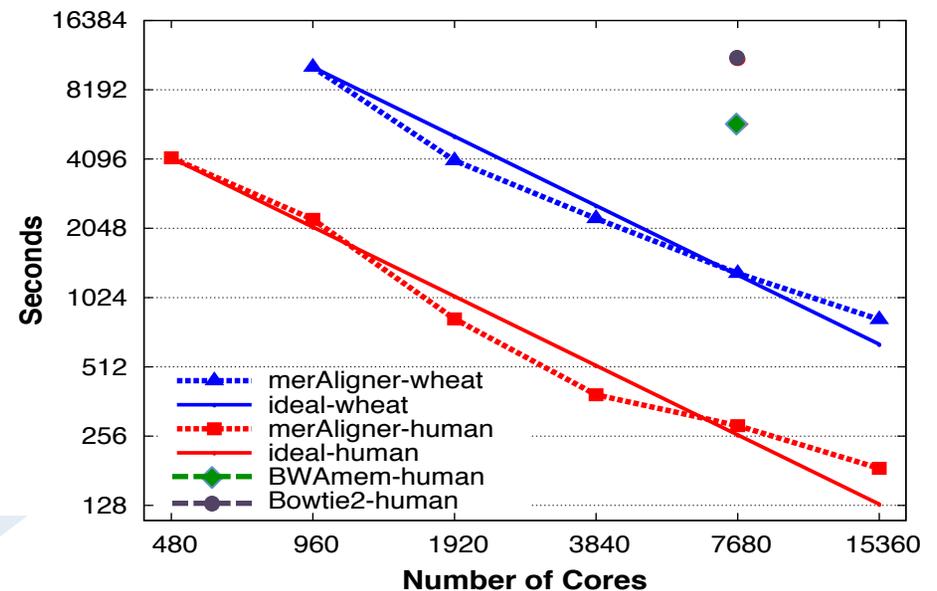
Combines with new algorithm to anchor 92% of wheat chromosome

Transforms process of discovery for de novo assembly

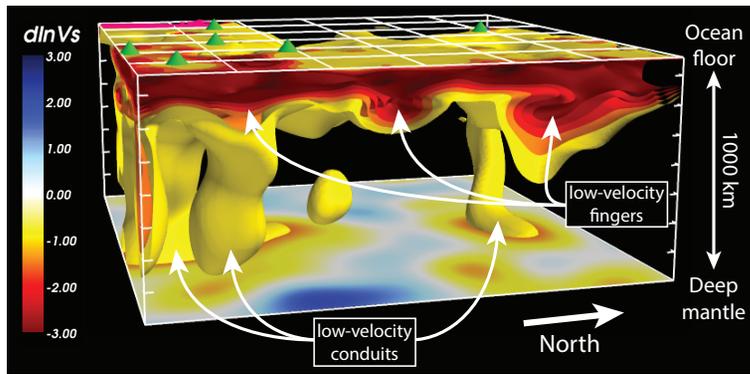
## Perl to PGAS: Distributed Hash Tables

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

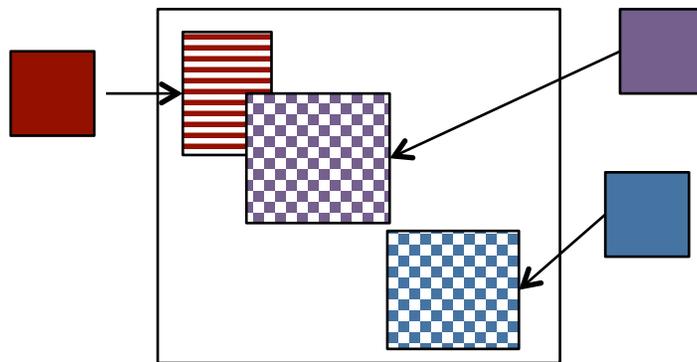
→ UPC++ Hash Table with “tunable” runtime optimizations



# App Challenge #2: Data Fusion in UPC++

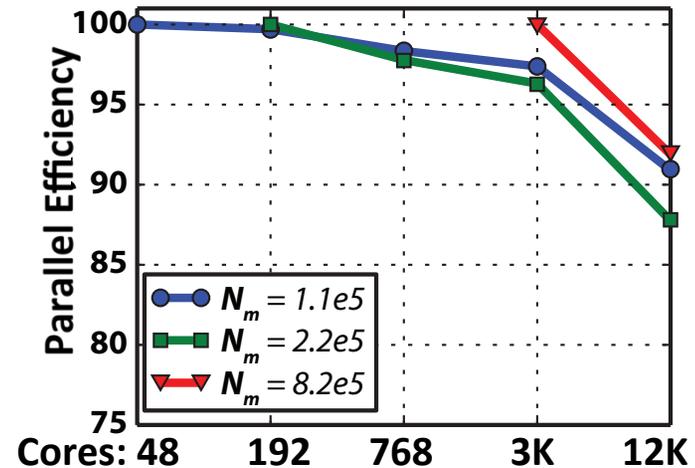


- Seismic modeling for energy applications “fuses” observational data into simulation
- With UPC++, can solve larger problems



## Distributed Matrix Assembly

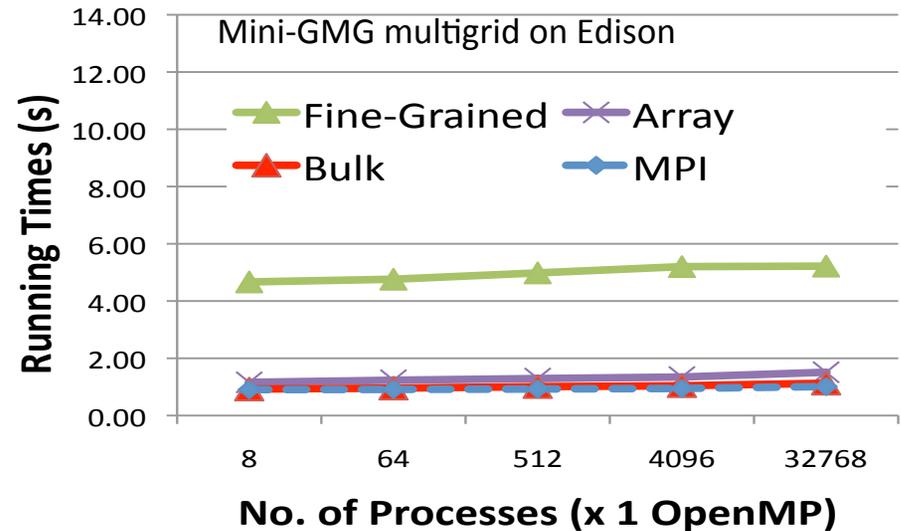
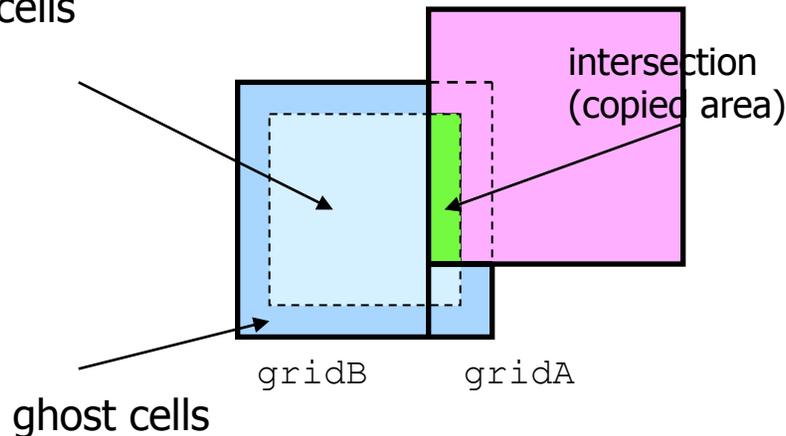
- Remote asyncs with user-controlled resource management
  - Team idea to divide threads into injectors / updaters
  - 6x faster than MPI 3.0 on 1K nodes
- Improving UPC++ team support



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). See F et al, IPDPS 2015 for parallelization overview.

# App Challenge #3: Toward Adaptive Mesh Refinement

“restricted” (non-ghost) cells Useful in grid computations including AMR

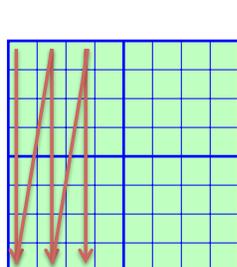


```
ndarray<double, 3, global> gridB = bArrays[i, j, k];  
...  
gridA.async_copy(gridB.shrink(1));
```

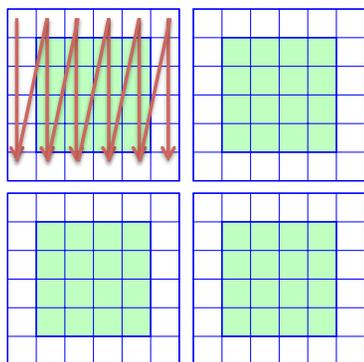
- The UPC++ arrays from Titanium; “views” for slicing, shrinking, and arbitrary index (not 0 or 1-based)
- The performance is close to that of a version that explicitly packs/unpacks (bulk) and to MPI
- The flat (no OpenMP) version is faster than hybrid

# TiDA: Tiling as a Durable Abstraction

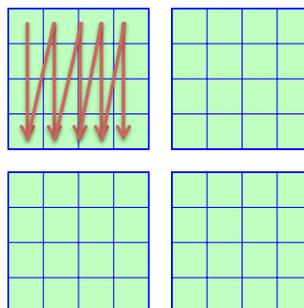
Didem Unat, Cy Chan, Weiqun Zhang, John Bell, John Shalf



Logical tiles



Isolated tiles



Contiguous tiles

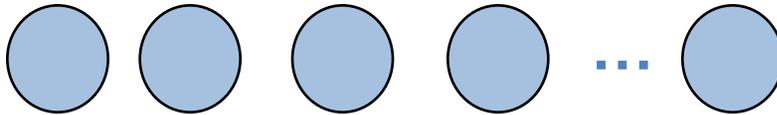
```
1 type(tile) :: tl
2 integer    :: tileno, tlo(2), thi(2), i, j
3 double precision, pointer :: ptrA(:, :)
4
5 do tileno=1, ntiles(tilearr)
6
7     ptrA => dataptr(A, tileno)
8     tl = get_tile(tilearr, tileno)
9     tlo = get_lwb(tl)
10    thi = get_upb(tl)
11
12    do j=tlo(2), thi(2) !element loop 1
13        do i=tlo(1), thi(1) !element loop 2
14            !loop body
15            ptrA(i,j) = do_something(i,j)
16        end do
17    end do
18
19 end do !end of tile loop
```

- **Plans to include TiDA-style optimizations into UPC++ arrays**
  - Add loop nests so inner ones fit in cache, e.g., 3-loop matmul → 6-loop
  - TiDA: Add tile shape/size information to each array
  - Optionally change the data layout to match
  - Can also add ghost regions as needed

# App Challenge #4: Dynamic Load Balancing

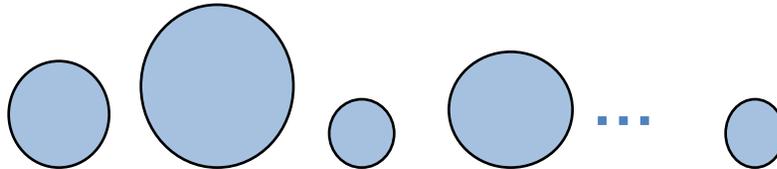
---

- **Static:** Equal size tasks



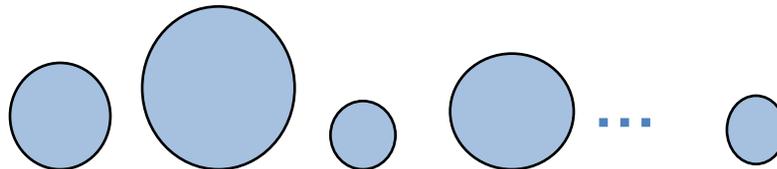
*Regular meshes, dense matrices, direct n-body*

- **Semi-Static:** Tasks have different but estimable times



*Adaptive and unstructured meshes, sparse matrices, tree-based n-body, particle-mesh methods*

- **Dynamic:** Times are not known until mid-execution

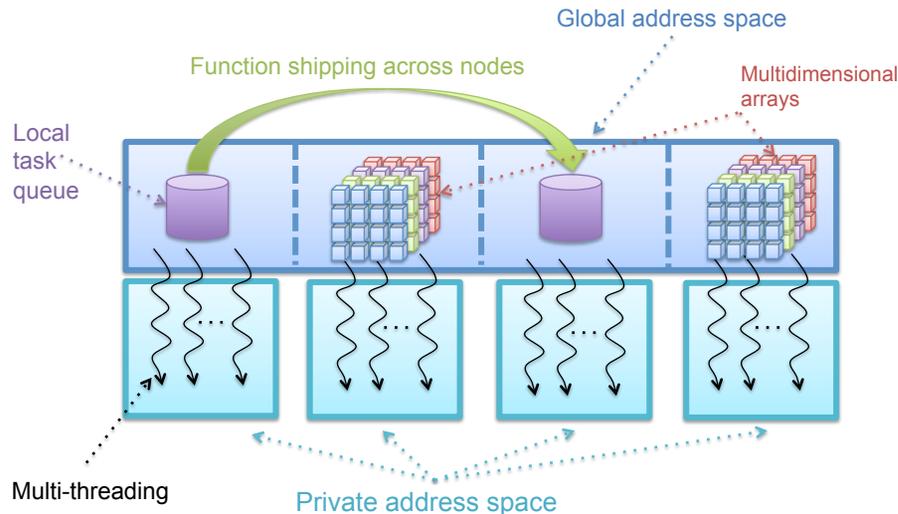


*Search (UTS), irregular boundaries, subgrid physics, unpredictable machines*

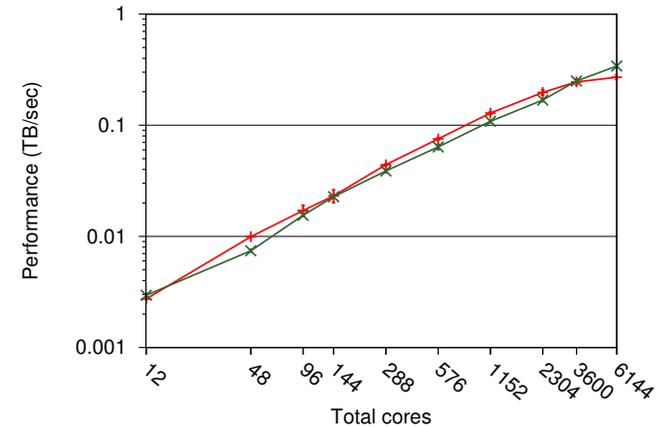
**Dynamic (on-the-fly) useful when:**

**Load imbalance penalty > communication to balance**  
**Load balancing can't solve lack of parallelism**

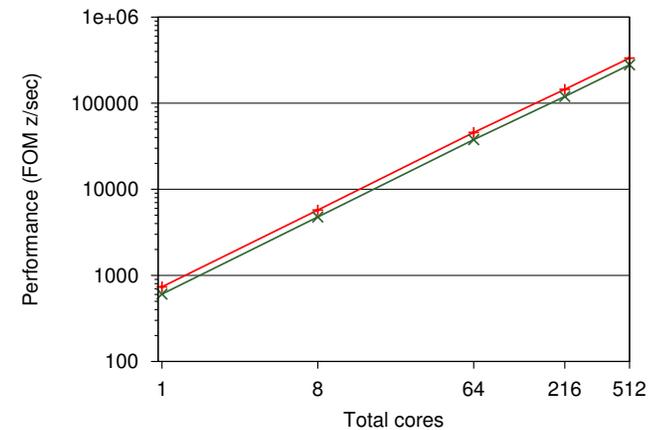
# Dynamic Load Balancing in UPC++ with Habanero and OCR



- **Dynamic tasking option in UPC++**
    - Demonstrated with library version of Habanero, runs on OCR on node
    - Combines with remote async
- **Dynamic load balancing library for domain-specific runtime in UPC++**

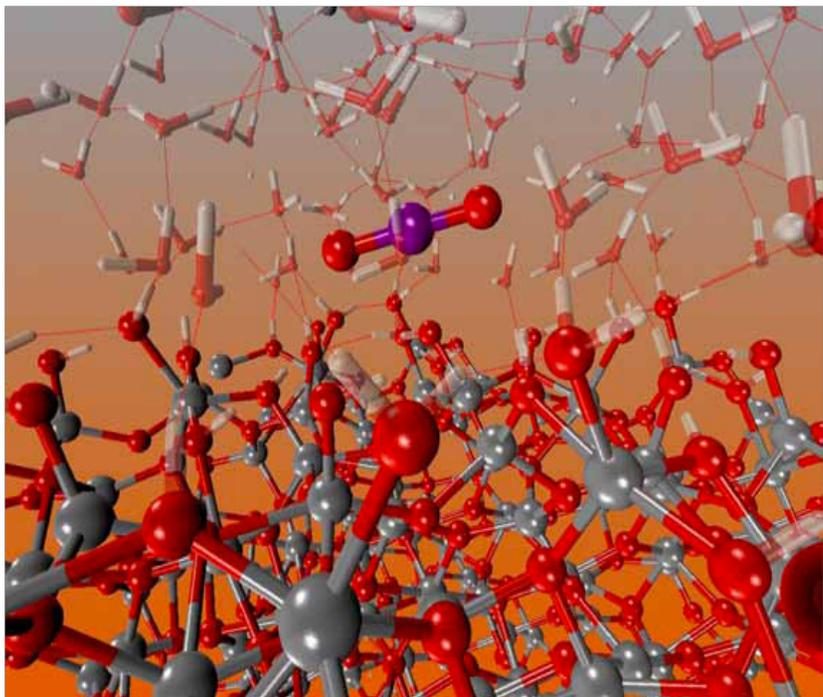


(a) SampleSort



(b) LULESH

# Towards NWChem in UPC++



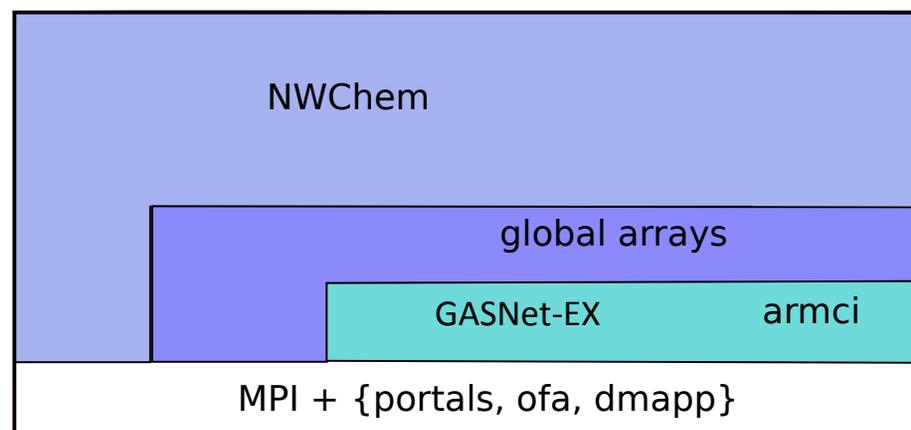
credit:nwchem-sw.org

## Internal tasking, memory management, and application checkpoint/restart

- New implementation on GASNet replacing ARMCI
- Plans for new Hartree Fock algorithm using global work stealing in UPC++

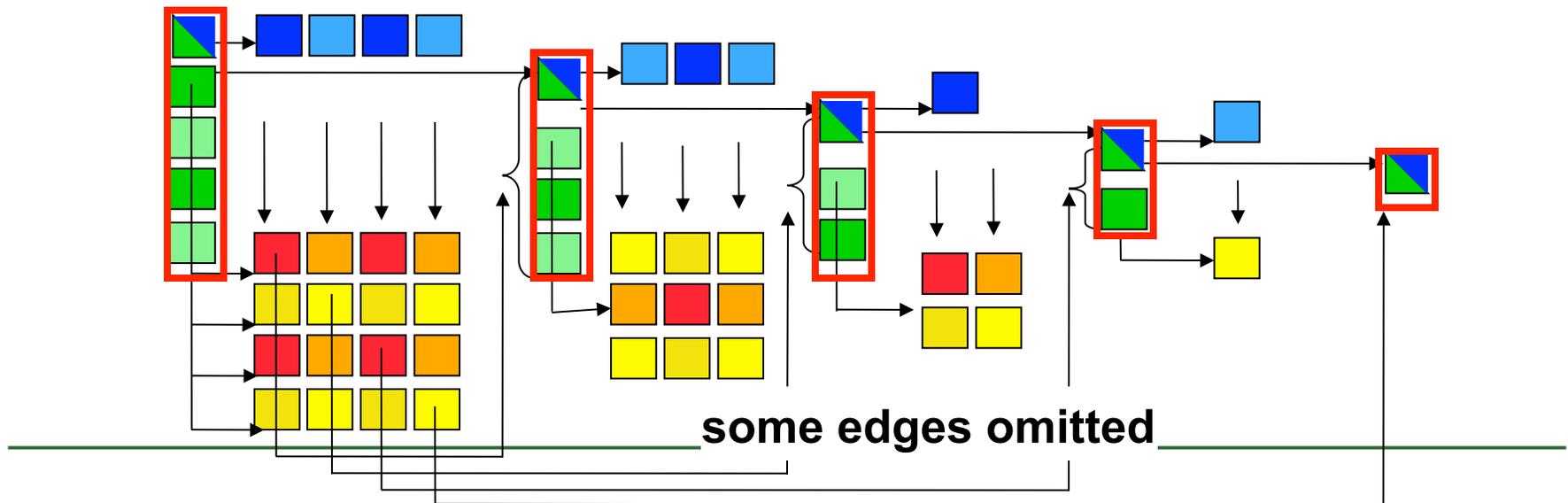
- **High-performance computational chemistry code**

- 60K downloads world wide
- 200-250 scientific application publications per year
- Over 6M LoC, 25K files
- Scales to 100K+ processors



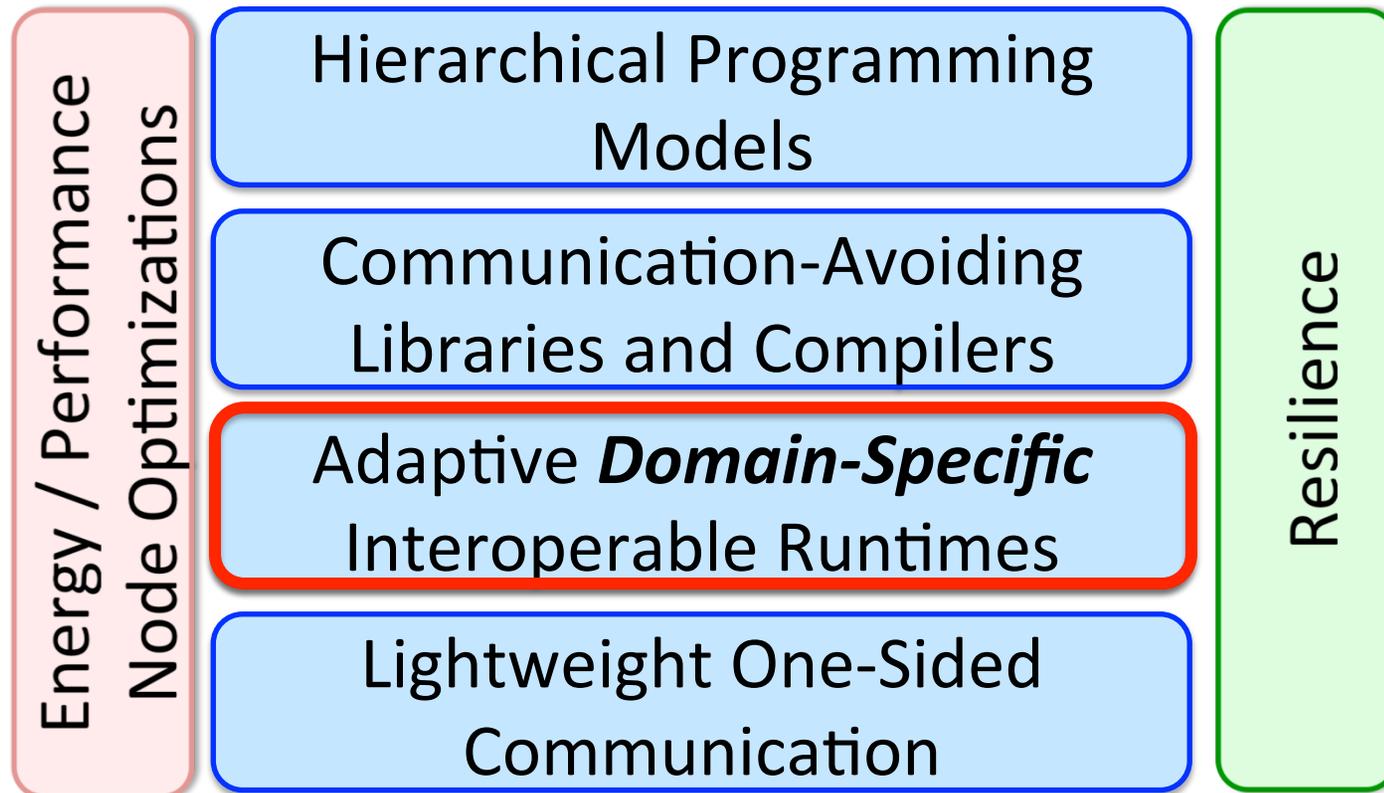
# App Challenge #5: DAG Scheduling

- **UPC++ async's can specify an implicit DAG**
- **Experience with UPC DAG Scheduling before it's time**
  - Assignment of work is static; schedule is dynamic
- **Currently developing Sparse Cholesky in UPC++**
  - Joint with FastMath (M. Jacquelin, E. Ng) and Y. Zheng
  - Minimum degree ordering
- **Two issues: dynamic scheduling in partitioned memory**
  - Can deadlock in memory allocation from overlapping communication
  - Solution: “memory constrained” lookahead



# DEGAS: Dynamic Exascale Global Address Space

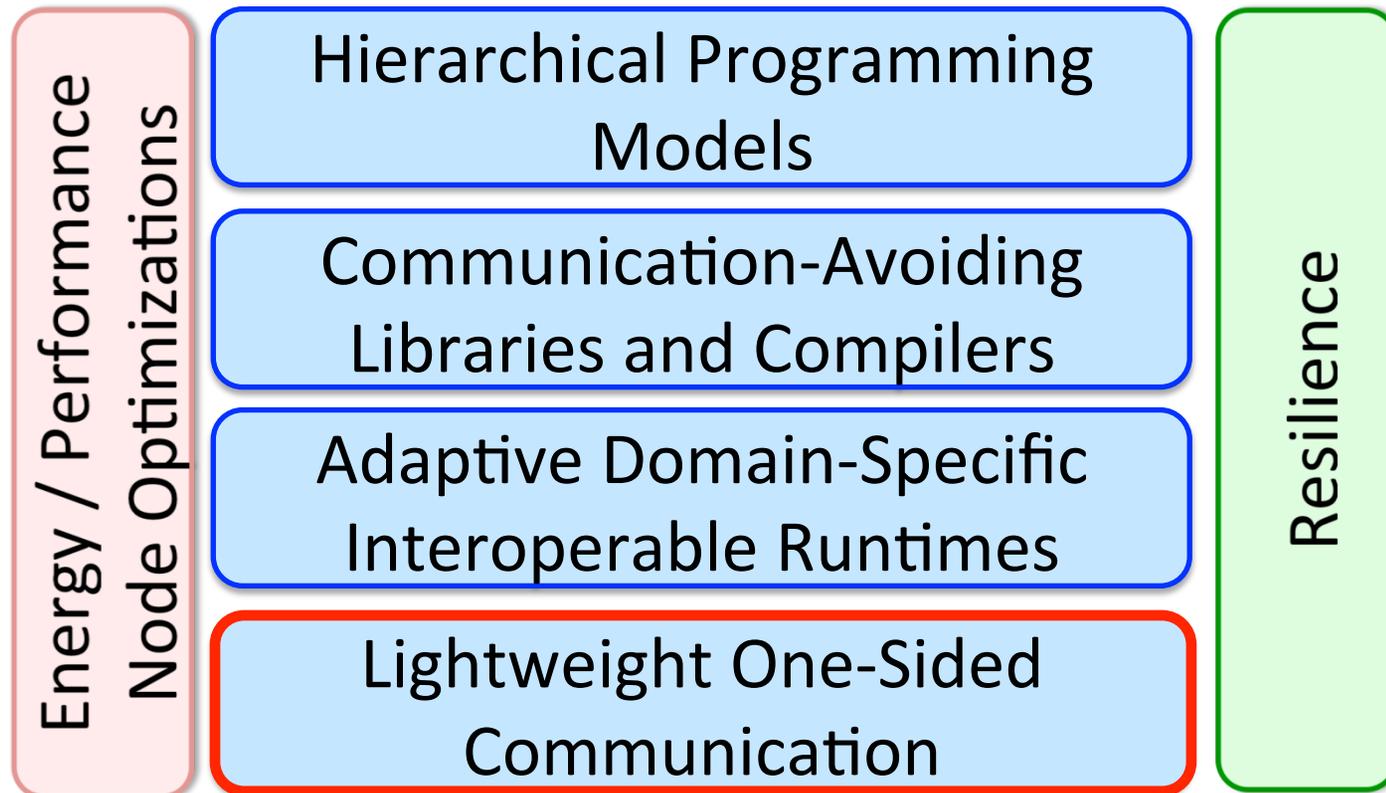
---



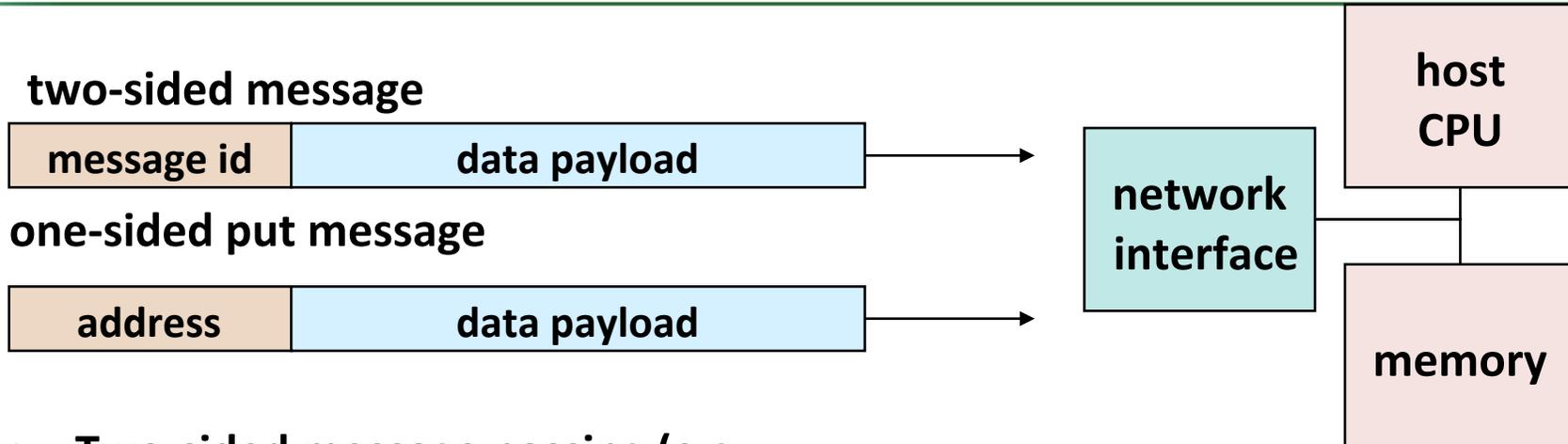
**Distinction in runtimes: who owns “main()”?**

# DEGAS: Dynamic Exascale Global Address Space

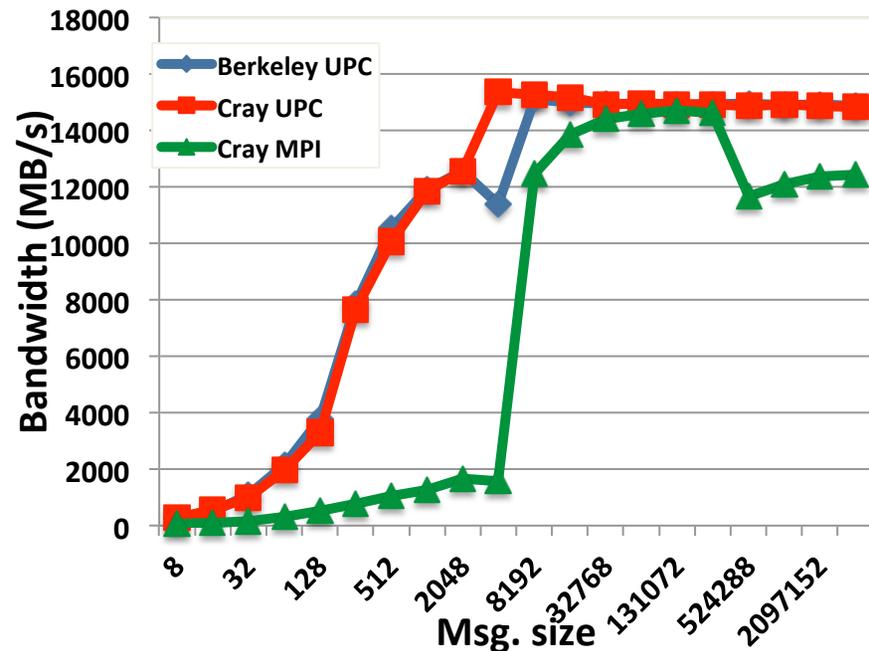
---



# Machine Challenge #1: Manycore Nodes



- **Two-sided message passing (e.g., send/receive in MPI)**
  - Couples data transfer/synchronization; sometimes what you want
- **Global address space decouples synchronization**
  - Very lightweight (often used under 2-sided)
  - Pay for what you need!
- **With many cores, want to avoid bottleneck from:**
  - Large software stack on 1 (LW) core
  - Cross-model cost from hybrid (MPI+X)



## Machine Challenge #2: PGAS on a Chip?

---

- Exascale nodes may not be cache coherent: domains of coherence
- PGAS lesson: don't cache remote values (trivially coherent)
  - Remote accesses do have to “see” cache; but not all cores
- MPI 3.0: TBD inter-node; less likely the right model within nodes

```
*p = ...;  ... = a[i];
```

Translates to:

```
void upc_memput(shared void *dst, const void *src, size_t n);
```

VS

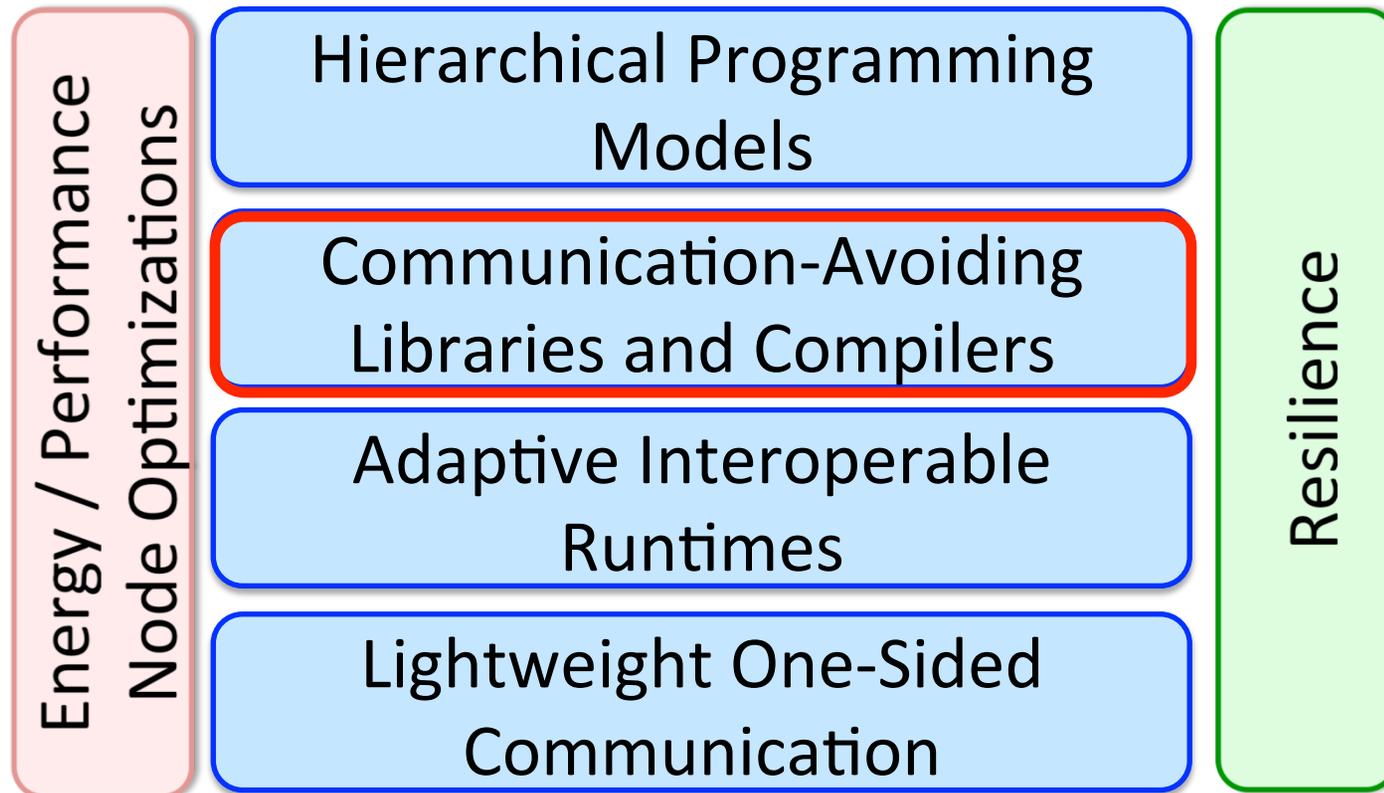
```
MPI_Put(const void *origin_addr, int origin_count,  
        MPI_Datatype origin_datatype, int target_rank,  
        MPI_Aint target_disp, int target_count,  
        MPI_Datatype target_datatype, MPI_Win win)
```

Explicit memory levels (local store and NVRAM) well suited to “vertical” PGAS

---

# DEGAS: Dynamic Exascale Global Address Space

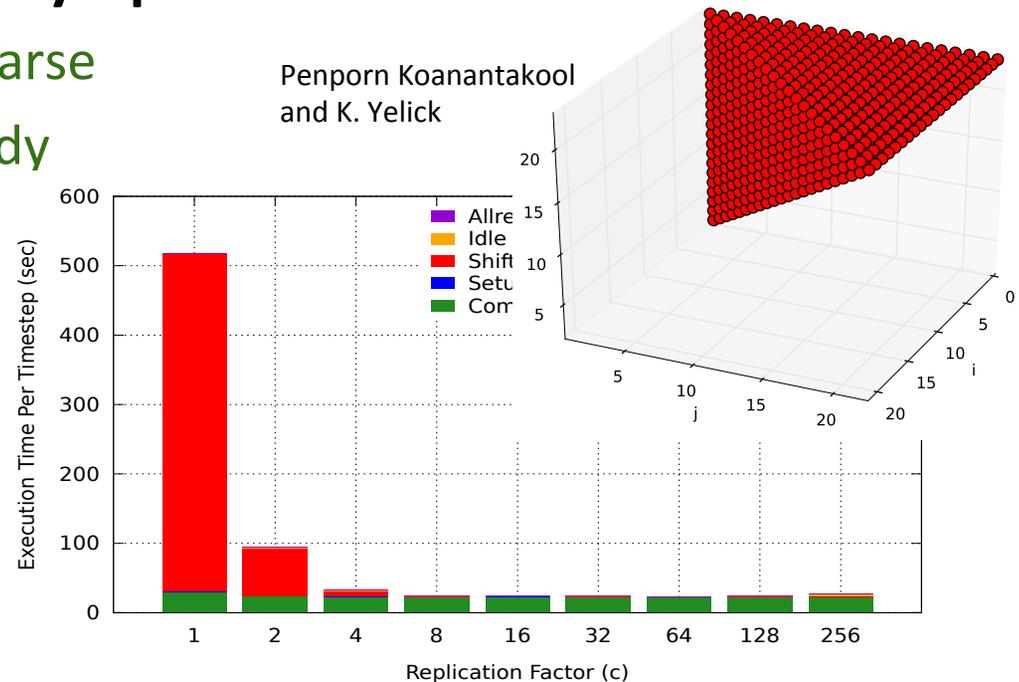
---



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

# Machine Challenge #3: Communication is Expensive

- Many algorithms have provably-optimal variants
  - Linear algebra, dense and sparse
  - Direct N-body and now K-body
- Generalize to compilers
  - Mellor-Crummey demo'd in compiler for MatMul
  - Theory being generalized



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

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

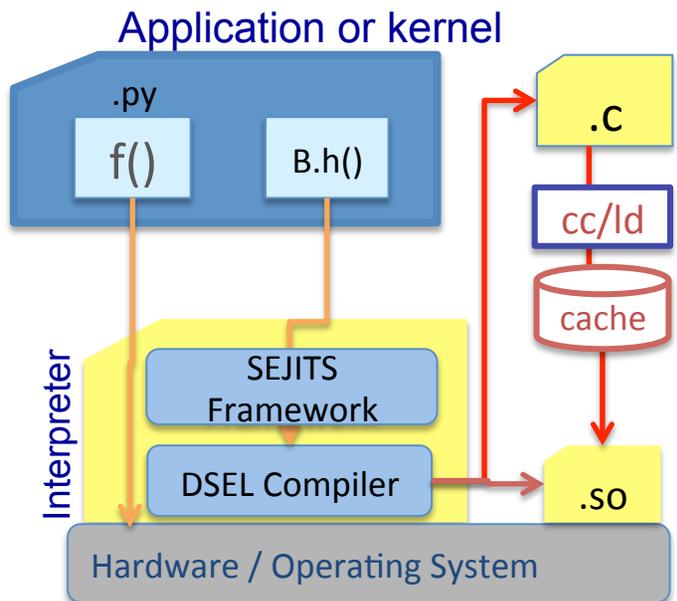
for some  $e$  we can determine

Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes up to constant factors

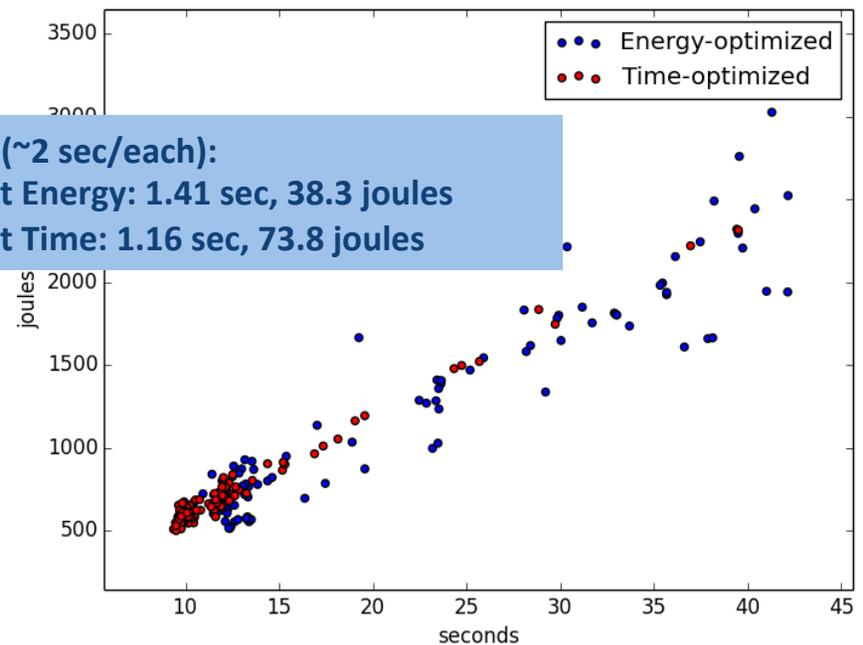


# Machine Challenge #4: Accelerators

- I was hoping these would go away, and they still might, but meanwhile...
- **Code generation options: compiler, DSL, annotations,...**
  - DEGAS CTree uses Python introspection on ASTs (joint with ASPIRE)
  - Domain-Specific Compiler uses LLVM for code generation
- **Automatic performance tuning to reach limits (uses OpenTuner)**



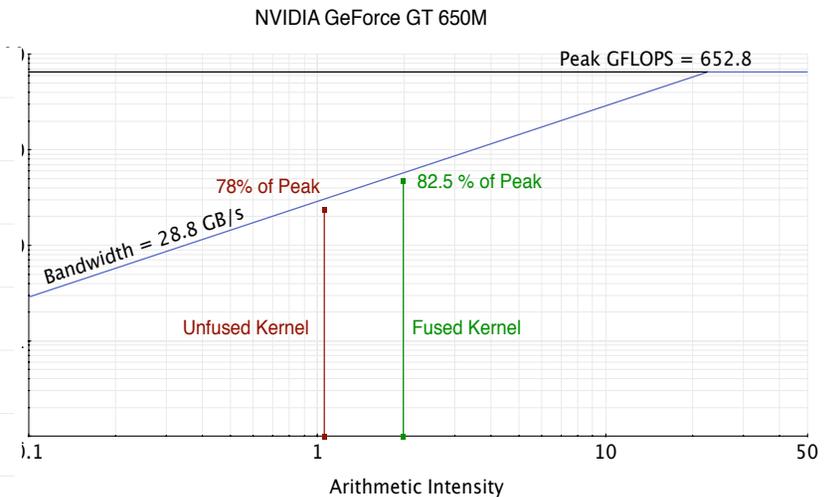
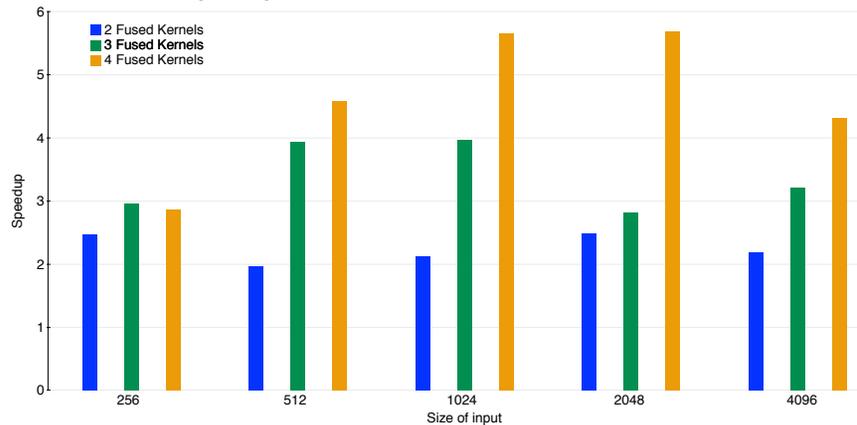
100 trials (~2 sec/each):  
Best Energy: 1.41 sec, 38.3 joules  
Best Time: 1.16 sec, 73.8 joules



# Towards Shift Calculus in SEJITS

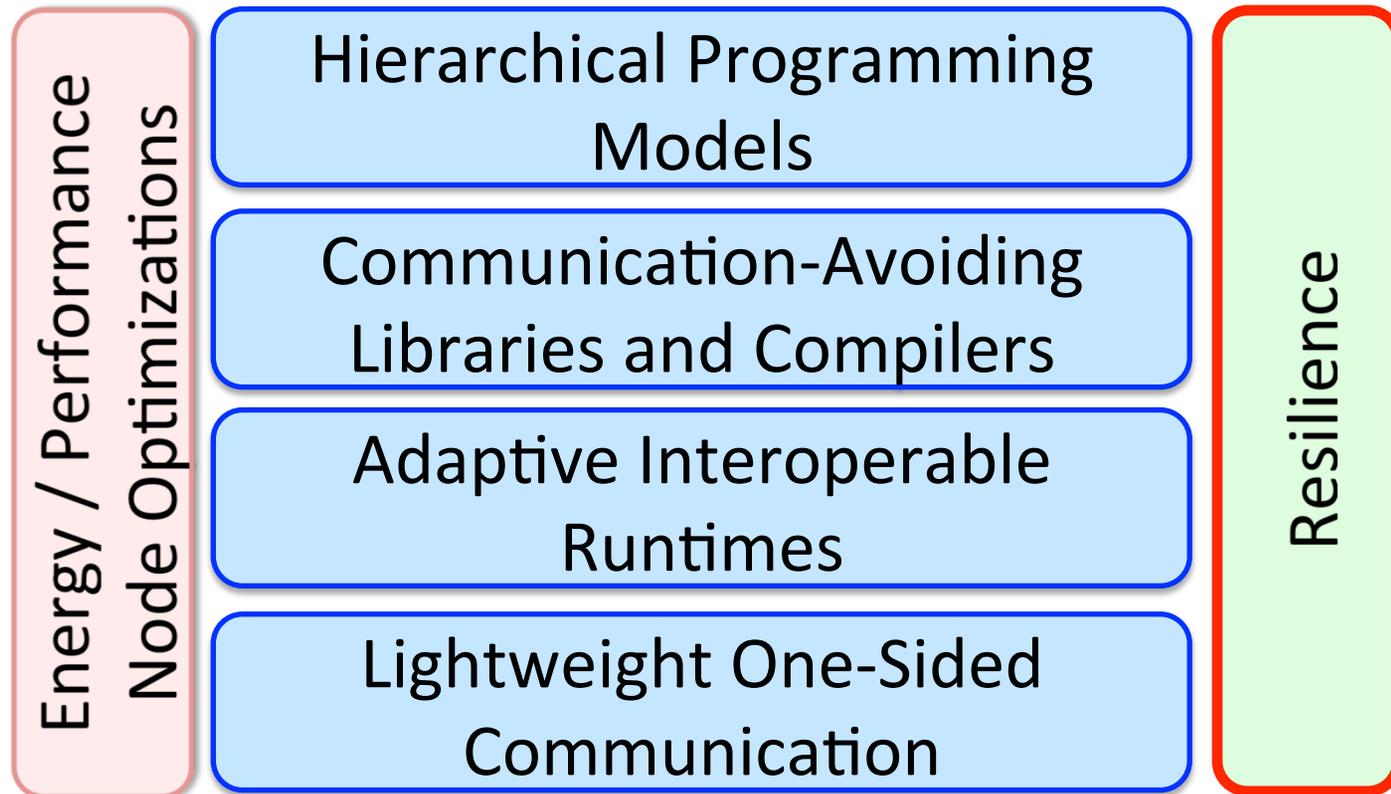
- Working toward 2<sup>nd</sup> (beside D-TEC) Shift Calculus DSL
- Pieces so far:
  - Arrays in UPC++
  - New optimization framework (Ctree) by M. Driscoll
  - Currently implemented part of HPGMG benchmark in stencil DSL
  - Other results on simpler stencils from ASPIRE collaborators

Speedup of Kernel Fusion For Stencils



# DEGAS: Dynamic Exascale Global Address Space

---



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



# DEGAS Programming Components

Hierarchical PGAS with Interoperability and Dynamic Scheduling as Needed

UPC++

UPC++  
+ DAGs

UPC++  
+ TaskQ

MPI +  
UPC++

UPC++  
+ OpenMP

UPC++  
+ OpenACC

PGAS Data Structures

Distributed Data Structures  
(Arrays, Hash Tables, Graphs, etc.)

Adaptive Scheduling Structures

Domain-Specific Runtime Structures  
(DAGs, TaskQs, etc.)

Domain-Specific  
Communication-Avoiding  
Code Generators

SEJITS

Domain-Specific  
Resilience

PGAS Contain-  
ment Domains

PGAS  
BLCR

Node  
Runtimes

PGAS-on-  
a-Chip

Haba-  
nero

Ram-  
butan

OCR

XPI

Communication

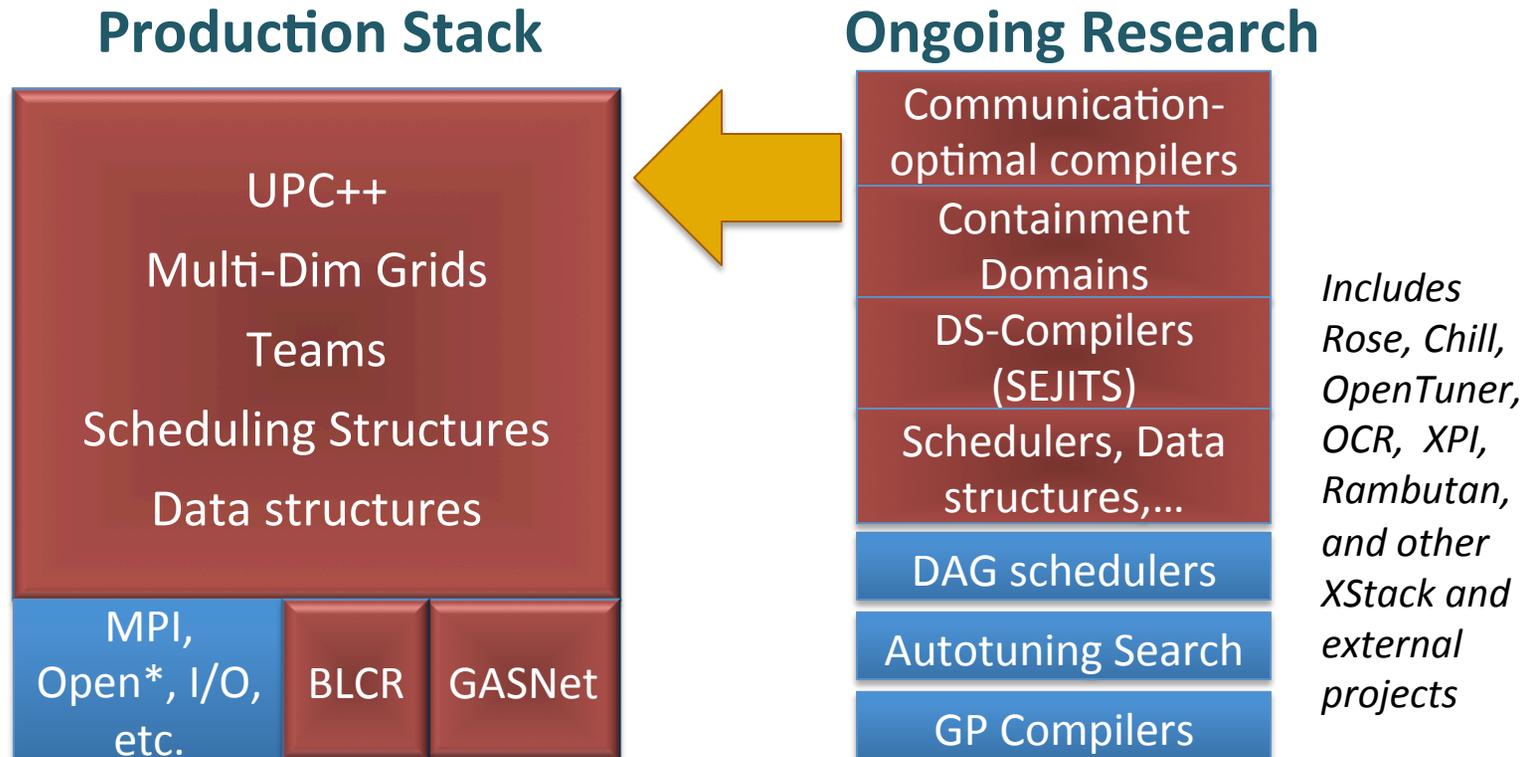
GASNet-  
EX

MPI

Manycore and Multicore nodes with scratchpad  
memory and limited cache-coherence

Interconnect

# Managed Discovery in DEGAS



- **New programming constructs and implementation techniques transition from research to production based on:**
  - Demonstrated application need and feasibility (performance,...)
- **Research provides risk tolerance and upside potential**

# Comments

---

- **A “bad” machine can turn easy problems to a hard ones**
  - Must be enough to overwhelm locality advantages of a (semi)static
  - Bad things happen between nodes; easier to fix (load balance) within

## **Two reasonable approaches to irregularity in hardware / applications:**

- **Expose all parallelism in a DAG (with all synchronization and communication on edges); write “mapper” of some kind**
  - Make dynamic execution the default and infer locality structure
- **Provide “abstract” hierarchical model of machines and let programmers write to them**
  - Offer dynamic scheduling / load balancing as options

## **Two reasonable approaches to architectural node diversity:**

- **Full fledged compiler**
- **Small compiler (maybe)**

**A good graduate student can make any programming model look good**

---

# Q&A

---

- **How are programming models differentiated from programming environments and what roles to they serve that are distinct but mutually supporting?**
  - Programming environment generally includes tools; this is not a well-accepted distinction so not important to focus on
  - Clearly tools are important
- **What are the key new abstractions for parallelism that the community must adopt to succeed at exascale? How should parallelism be identified and concurrency managed in these models?**
  - Abstraction of future systems
- **Are there breakthroughs in programming models and environments that we should explore, in addition to continued incremental improvements to existing ones?**
  - Compilers: High risk, high reward for productivity and performance
  - DSLs: many companies are supporting their own languages (and compilers), including DSLs
  - Full task-based models
- **What are the most promising ideas for programming abstractions to represent data and its distribution across the lateral and hierarchical memory structures?**
  - HPGAS

- 
- **How should PM/E represent persistent objects and the storage system to programmers?**
    - Containment domains: hierarchical model that can be tailored to application needs
  - **Are there innovative ideas for integrating resilience and debugging into the programming model?**
    - See Corvette project on reproducible computations
    - Tools to detect races, and
  - **Many application teams are beginning to explore task-based and data-driven programming models. Are there common abstractions and key features? How do they differ?**
    - Yes: Explicit DAGs and implicit DAGs
    - Tasks that run-to-completion without synchronization or communication vs more general tasks (with various types of comm/synch allowed)
  - **Are there lessons to be learned from other communities that we can apply?**
    - Many companies are building their own DSLs or even own general purpose languages; DOE as a hold has the ability to do that with sufficient commitment
-