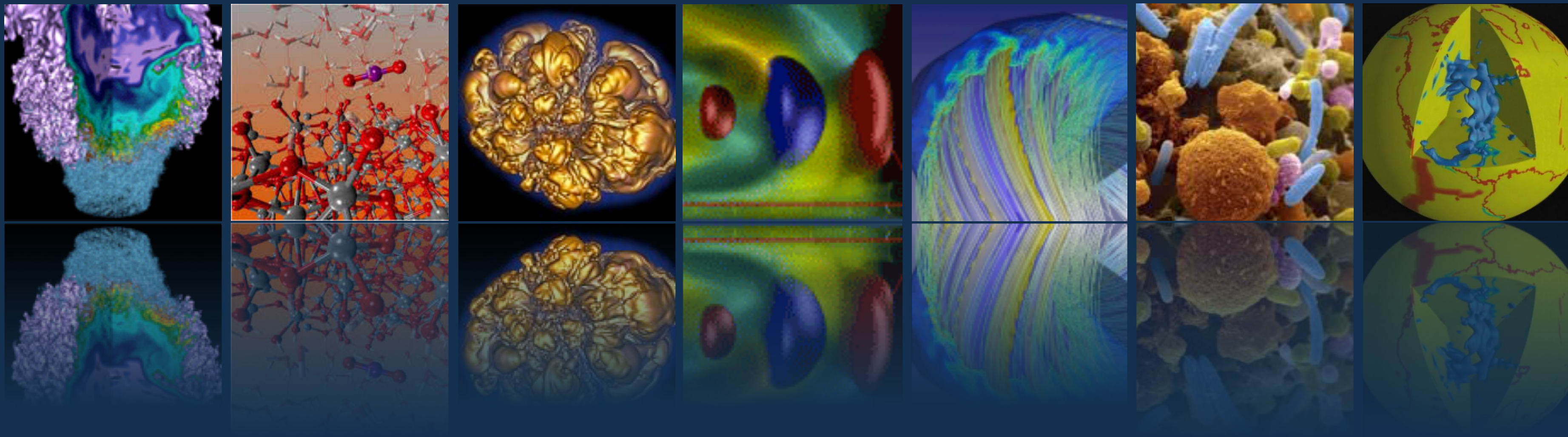
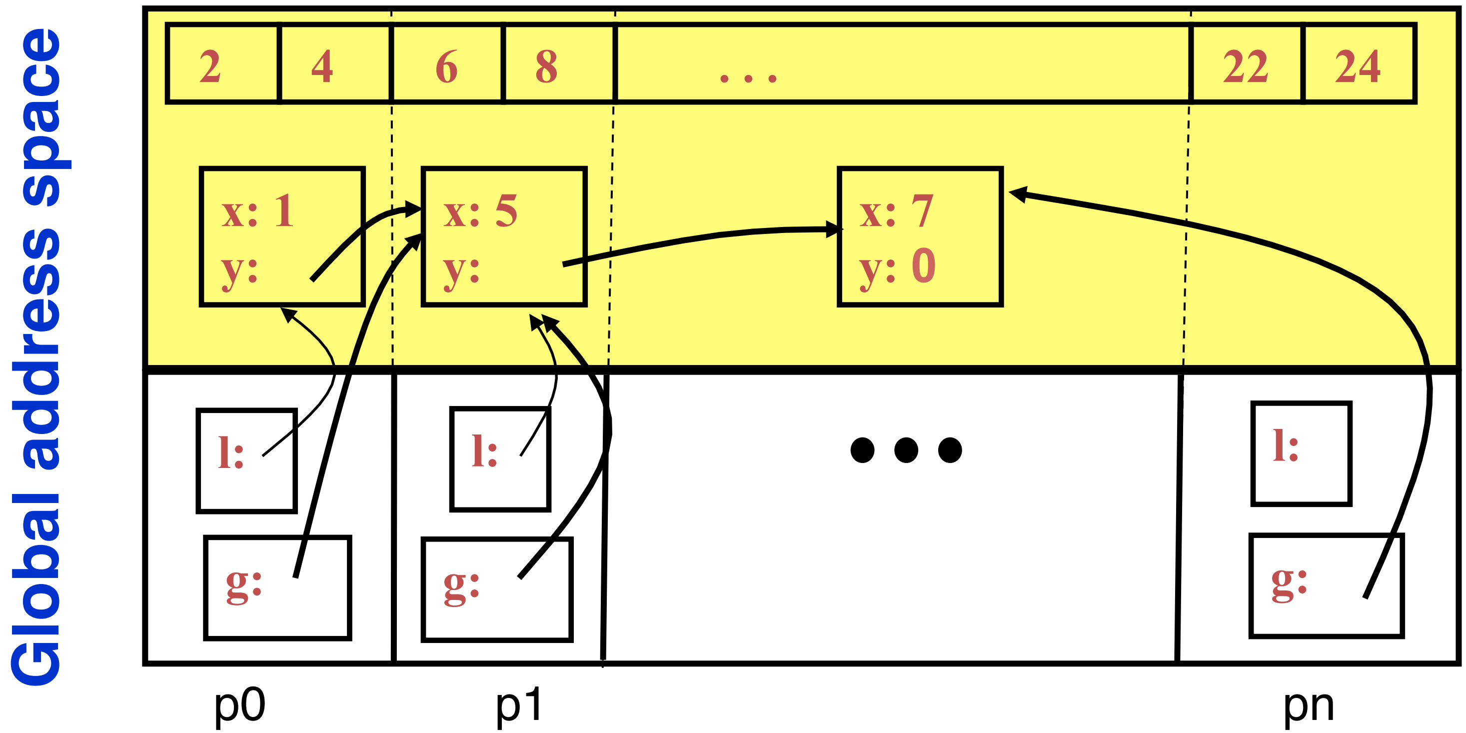


PGAS Applications
What, Where and Why?

Kathy Yelick

Professor of Electrical Engineering and Computer Sciences
University of California at Berkeley
Associate Laboratory Director for Computing Sciences
Lawrence Berkeley National Laboratory

What is PGAS? Partitioned Global Address Space



- **Global Address Space: Directly access remote memory**
- **Partitioned: Programmer controls data layout for scalability**

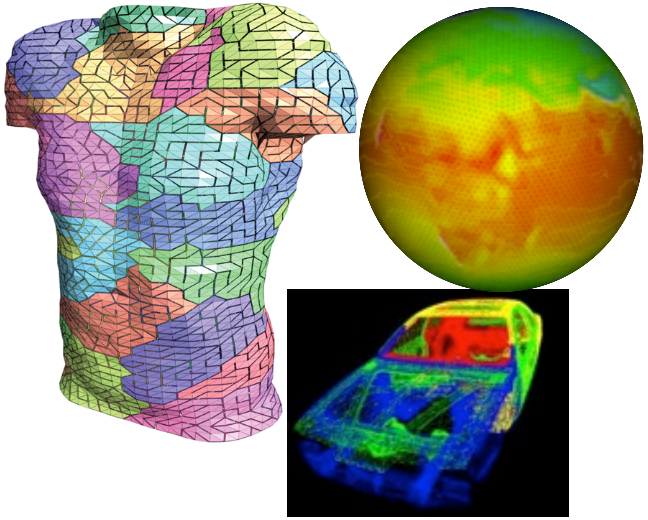
PGAS Languages and Libraries

- **Language mechanisms for distributed arrays**
  - (CoArray) Fortran REAL:: X(2,3)[*]
    X(1,2) …. X(1,3)[5]
  - UPC shared double X [100]; or double X[THREADS*6];
    X[] …. X[MYTHREAD]
  - Chapel const ProblemSpace= {1..m}
    dmapped Block(boundingBox={1..m});
    var X: [ProblemSpace] real;
  - UPC++ upcxx::shared_array<Type> X;
    X.init(128); or X.init(128,4)
    X [upcxx:ranks()] ... X[6]

Many others...

Programming Data Analytics vs Simulation

Simulation: More Regular



Message Passing Programming

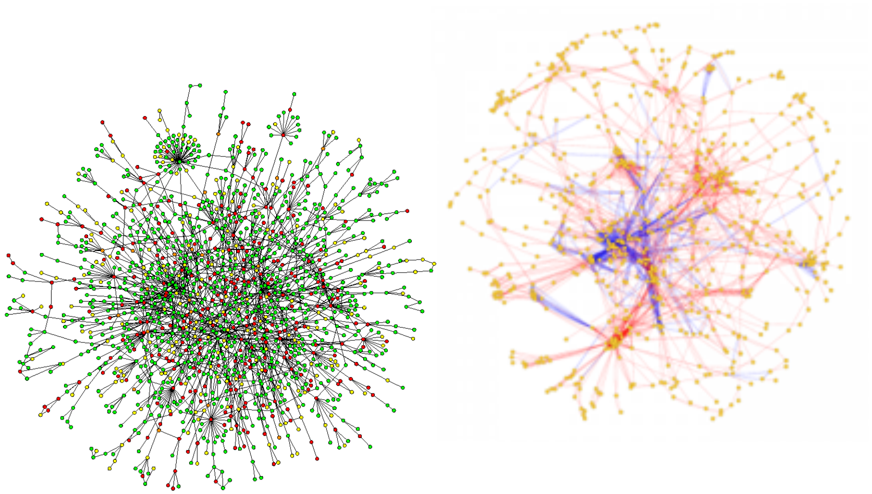
Divide up domain in pieces

Compute one piece

Send/Receive data from others

MPI, and many libraries

Analytics: More Irregular



Global Address Space Programming

Each start computing

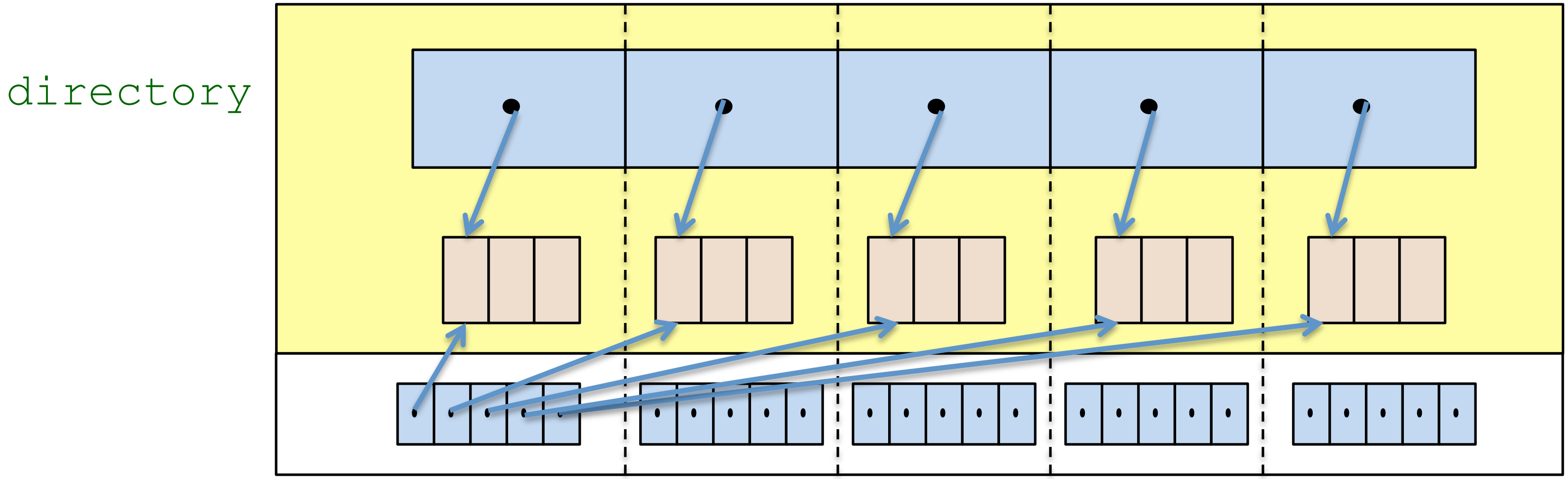
Grab whatever / whenever

UPC,UPC++, CAF, X10, Chapel, Shmem, GA

Distributed Arrays Directory Style

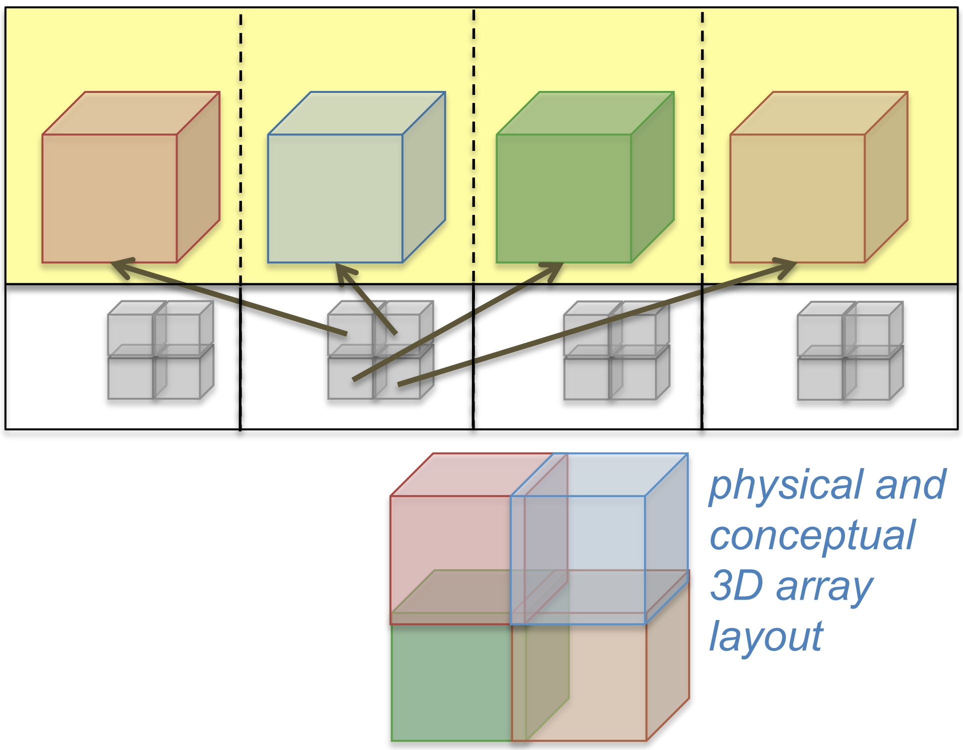
Many UPC programs avoid the UPC style arrays in factor of directories of objects

```
typedef shared [] double *sdblptr;
shared sdblptr directory[THREADS];
directory[i]=upc_alloc(local_size*sizeof(double));
```

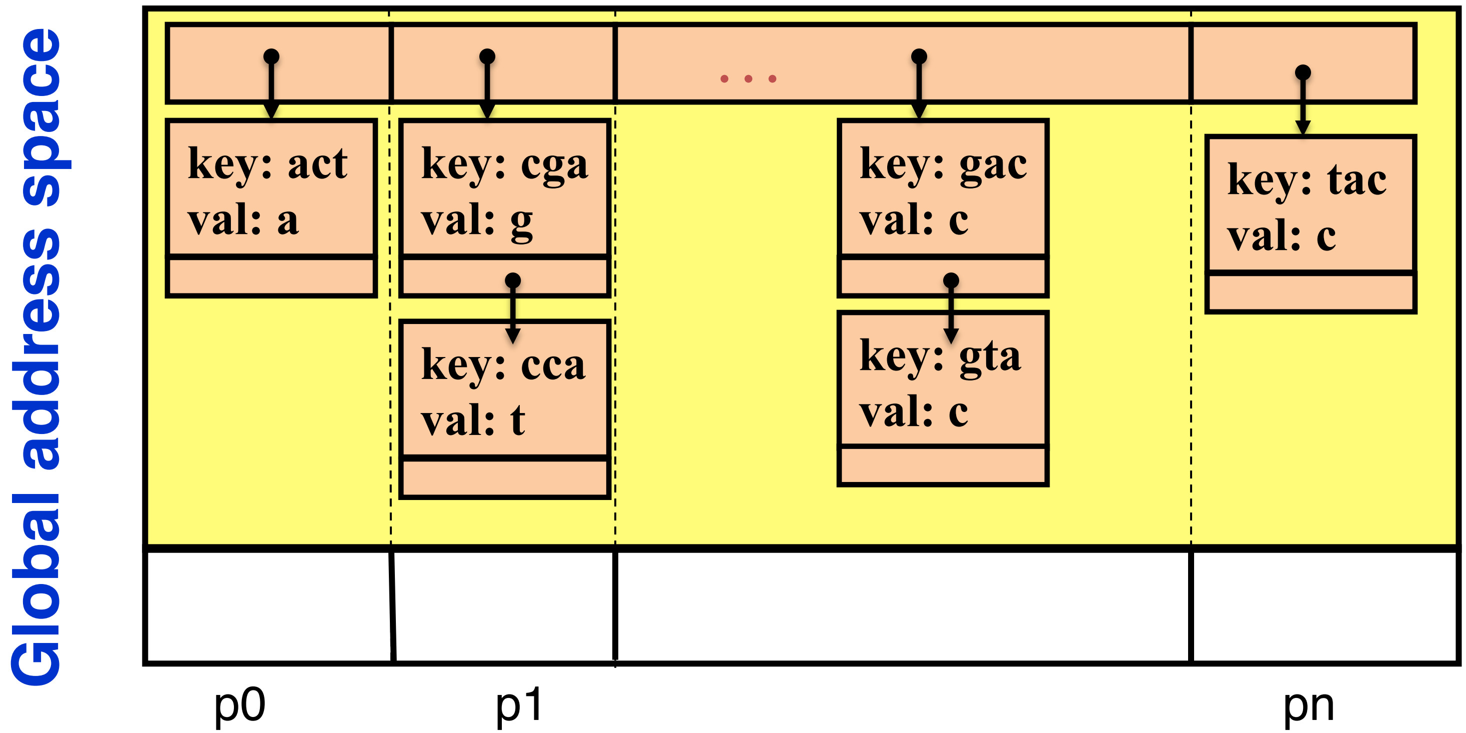


Distributed Arrays Directory Style

- **These are also more general:**
  - Multidimensional, unevenly distributed
  - Ghost regions around blocks



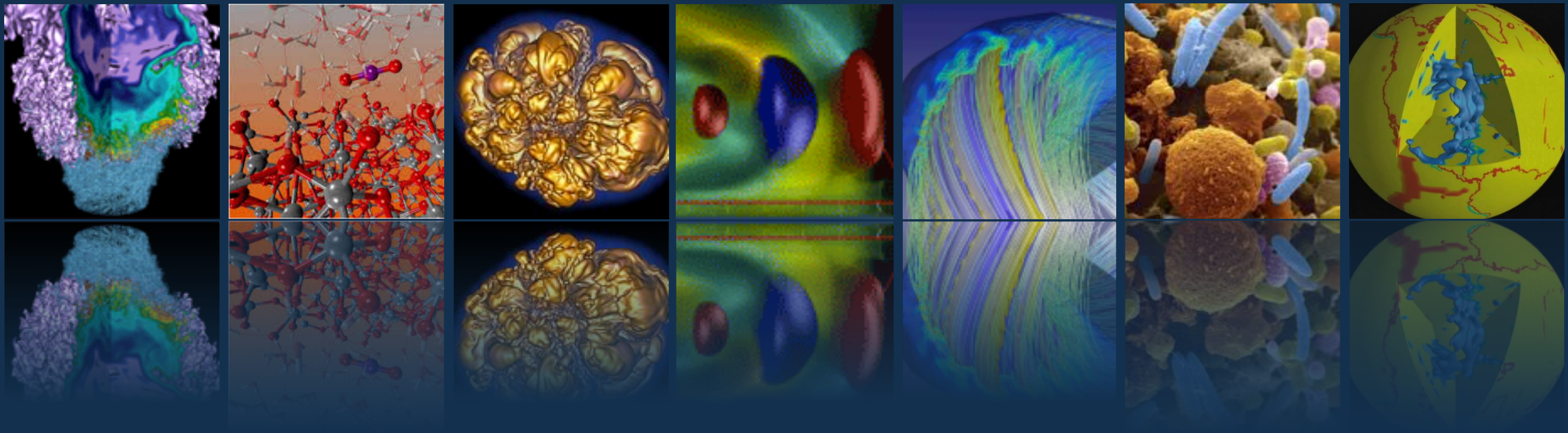
Example: Hash Table in PGAS



- <key, value> pairs, stored in some bucket based on hash(key)
- One-sided communication; never having to say "receive"
- Allows for Terabyte-Petabyte size data sets vs ~1 TB in shared memory

Other Programming Model Variations

- **What can you do remotely?**
  - Read, Write, Lock
  - Atomics (compare-and-swap, fetch-and-add)
  - Invoke functions
  - Signal processes to wake up (task graphs)
- **What type of parallelism is there**
  - Data parallel (single threaded semantics, e.g., A = B + C)
    - Collective communication
  - Single Program Multiple Data (SPDM): if (MYTHREAD == 0)….
  - Hierarchical SPMD (teams): if (MYTEAM….)...
  - Fork-join: fork / async
  - Task graph (events)



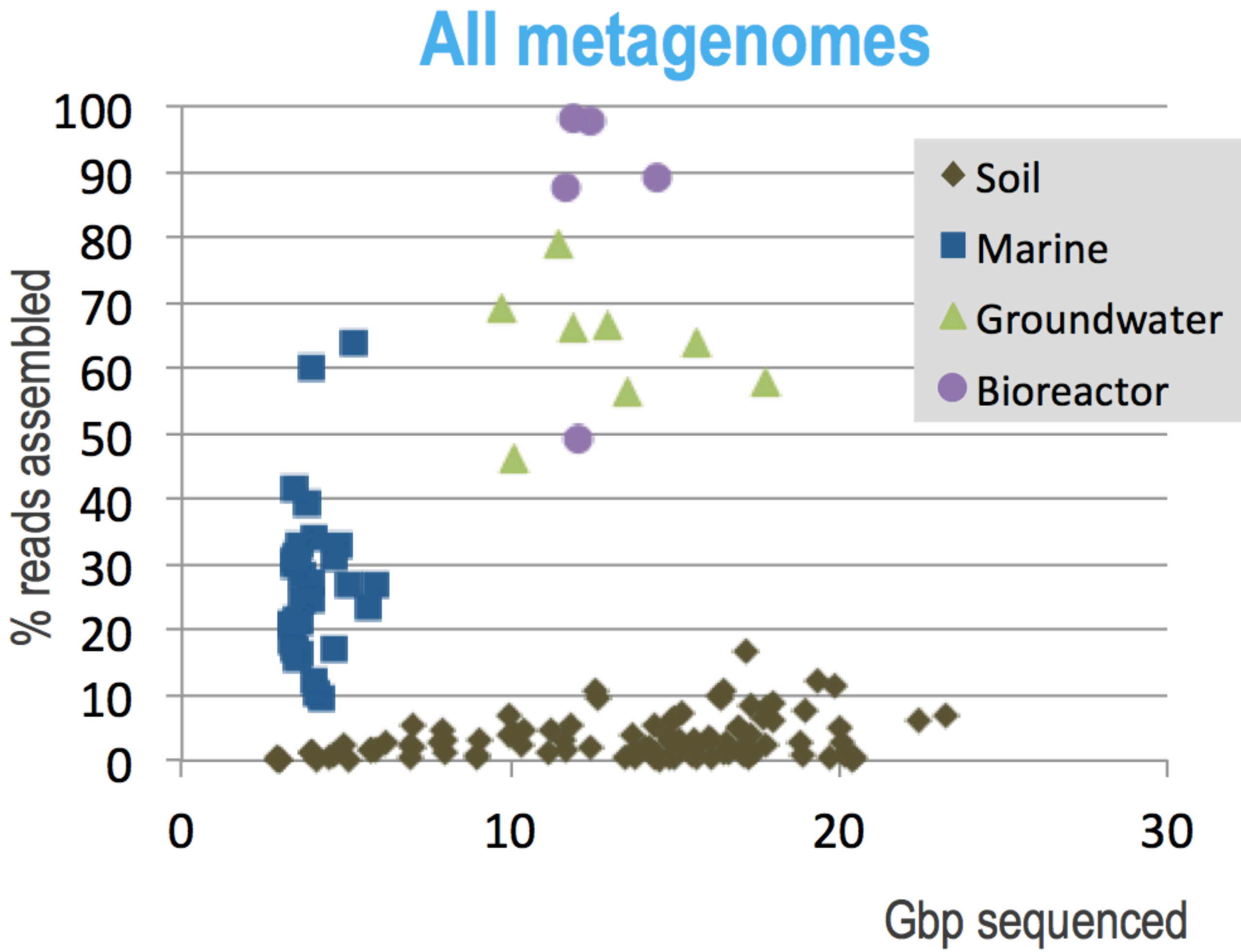
Where is PGAS programming used?

1. Asynchronous fine-grained reads/write/atomics

De novo Genome Assembly

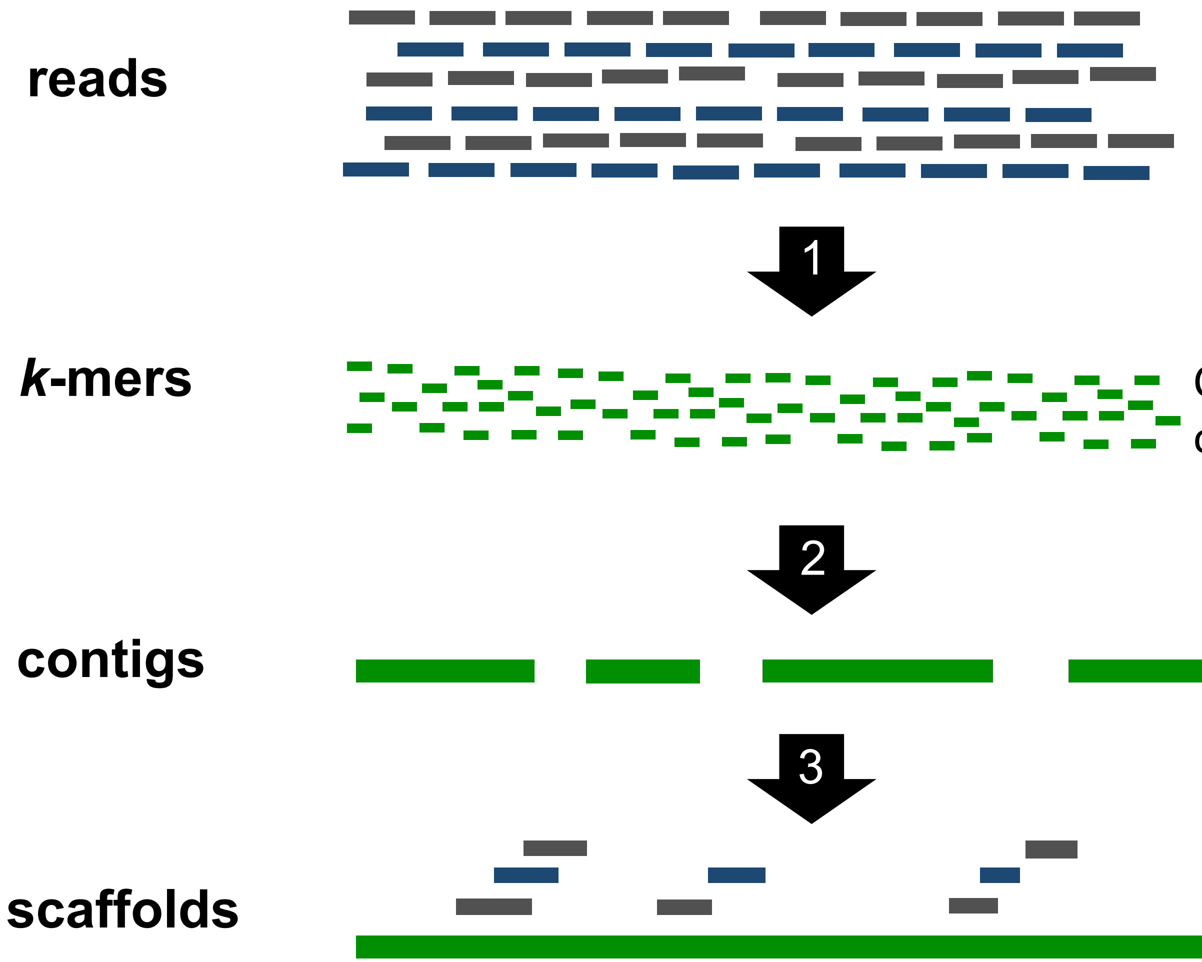
- DNA sequence consists of 4 bases: A/C/G/T
- Read: short fragment of DNA
- De novo assembly: Construct a genome (chromosomes) from a collection of reads

Metagenome Assembly: Grand Challenge



For complex metagenomes (soil) most of the reads cannot be assembled

De novo Genome Assembly *a la* Meraculous



Input: Reads that may contain errors

Chop reads into k-mers, process k-mers to **exclude errors**

- **Bloom filter (statistical)**
- **Intensive I/O**
- **High memory footprint**

Construct & traverse de Bruijn graph of k-mers, generate contigs

- **Huge graph as a hash table**
- **Irregular accesses**
- **Injection limited**

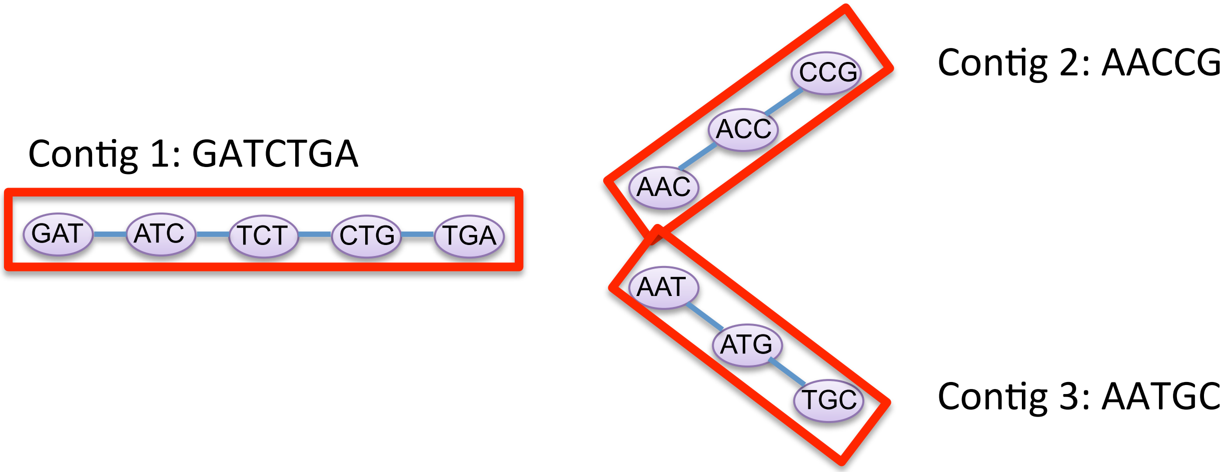
Leverage read information to link contigs and generate scaffolds.

- **Multiple hash tables**
- **High memory requirements**
- **Intensive computation**
- **Intensive I/O**

Georganas, Buluc, Chapman, Oliker, Rokhsar, Yelick, [Aluru,Egan,Hofmeyr] in SC14, IPDPS15, SC15

Application Challenge: Random Access to Large Data

- **Parallel DFS (from randomly selected K-mers) to compute contigs**
- **Some tricky synchronization to deal with conflicts**

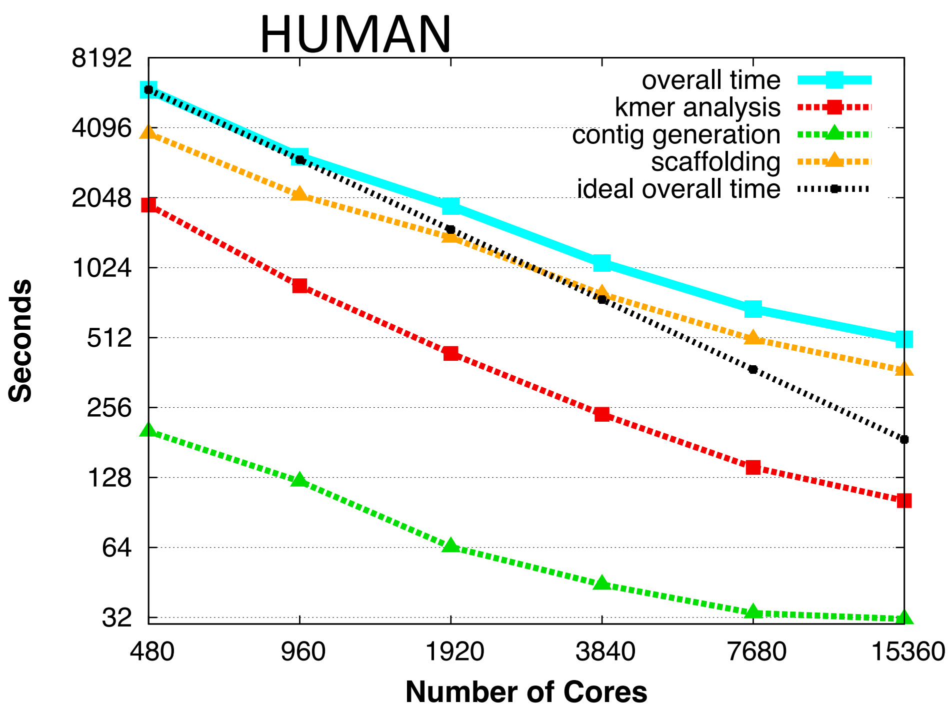


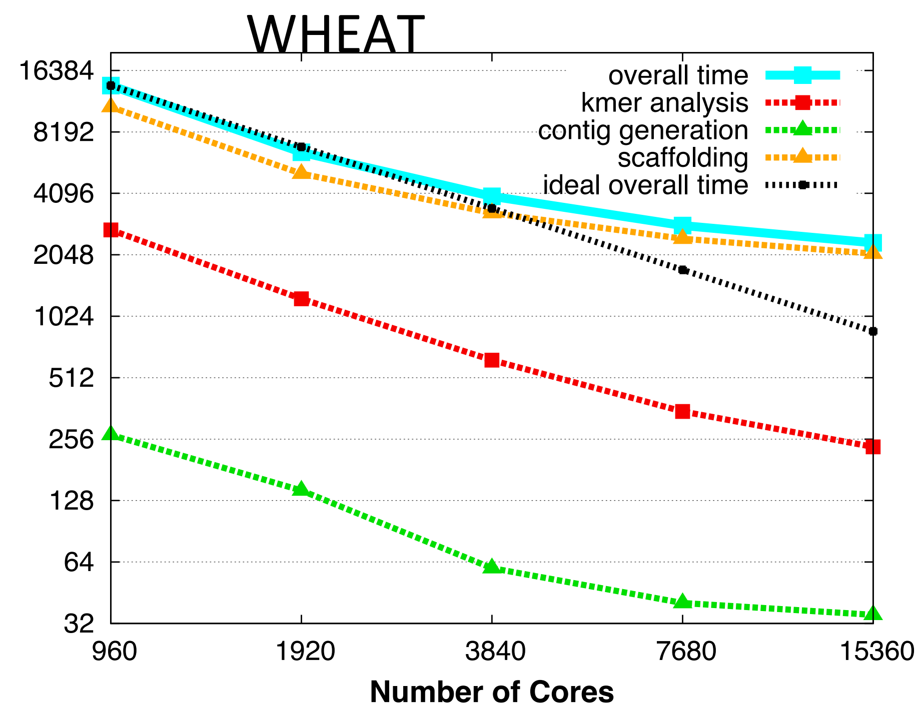
- **Hash tables used in all phases**
  - Different use cases, different implementations
- **No a priori locality: that is the problem you're trying to solve**

HipMer (High Performance Meraculous) Assembly Pipeline

Distributed Hash Tables in PGAS

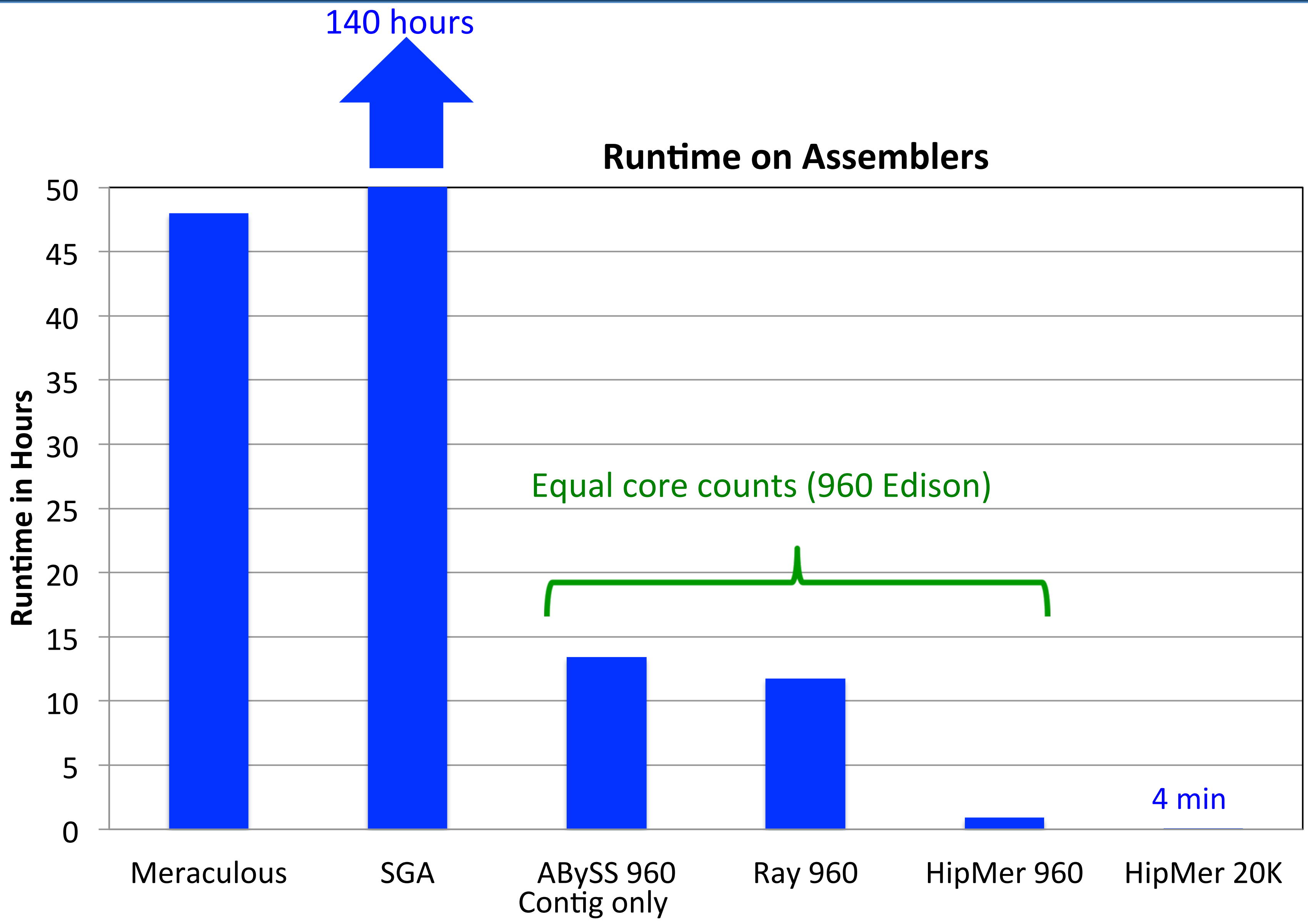
- Remote Atomics, Dynamic Aggregation, Software Caching
- 13x Faster than another HPC/MPI code (Ray) on 960 cores





Evangelos Georganas, Aydın Buluç, Jarrod Chapman, Steven Hofmeyr, Chaitanya Aluru, Rob Egan, Lenny Oliker, Dan Rokhsar, and Kathy Yelick. **HipMer: An Extreme-Scale De Novo Genome Assembler**, SC'15

Comparison to other Assemblers



Science Impact: HipMer is transformative

- **Human genome (3Gbp) “de novo” assembled :**
  - Meraculous: 48 hours
  - HipMer: 4 minutes (720x speedup Meraculous)

Makes unsolvable problems solvable!

- **Wheat genome (17 Gbp) “de novo” assembled (2014):**
  - Meraculous (did not run):
  - HipMer: 39 minutes; 15K cores (first all-in-one assembly)

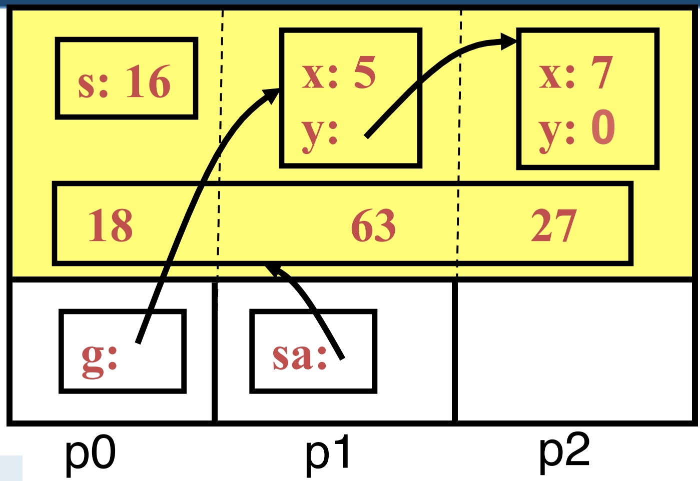
- **Pine genome (20 Gbp) “de novo” assembled (2014) :**
  - Masurca : 3 months; 1 TB RAM

- **Wetland metagenome (1.25 Tbp) analysis (2015):**
  - Meraculous (projected): 15 TB of memory
  - HipMER: Strong scaling to over 100K cores (contig gen only)

DEGAS

Georganas, Buluc, Chapman, Oliker, Rokhsar, Yelick, [Aluru,Egan,Hofmeyr] in SC14, IPDPS15, SC15

UPC++: PGAS with “Mixins” (Teams and Asyncs)



- **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**

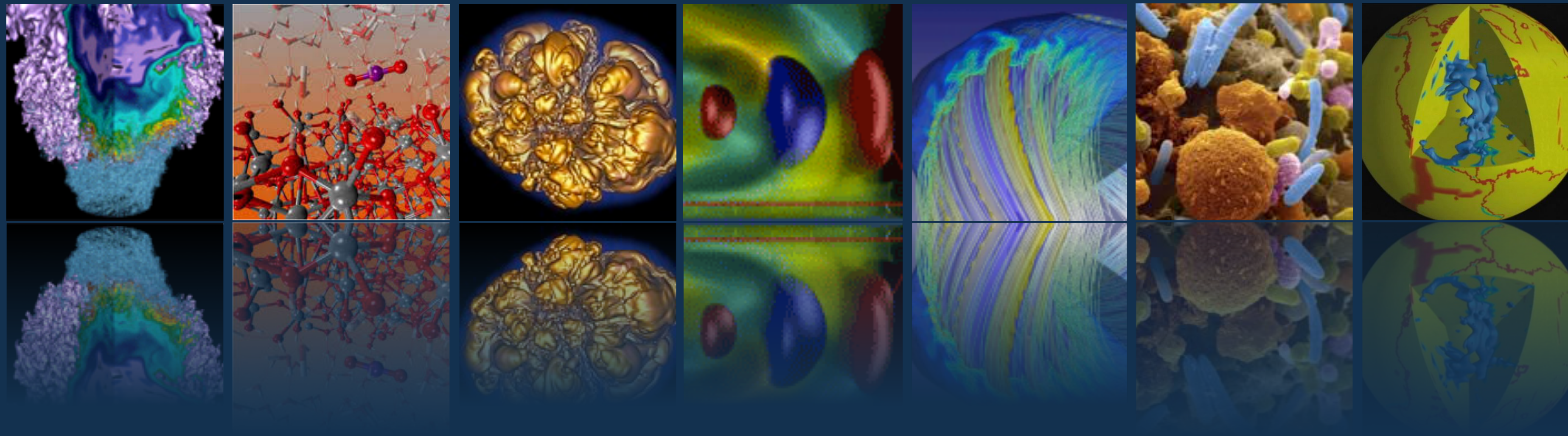
- **Remote methods, async**

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

- **Research in teams for hierarchical algorithms and machines**

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

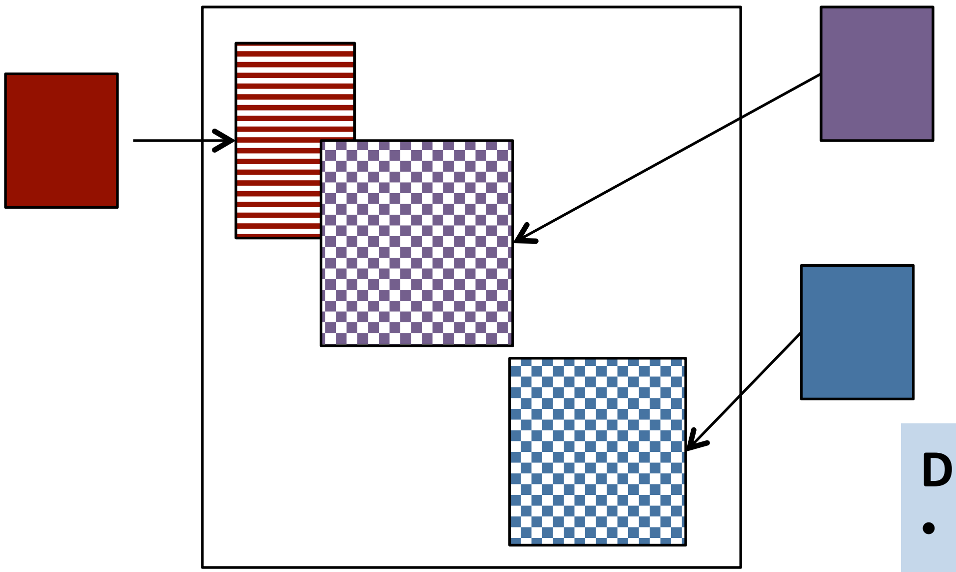
- **Use these for “domain-specific” runtime systems**



Where is PGAS programming used?

1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Strided irregular updates (adds) to distributed matrix

Application Challenge: Data Fusion



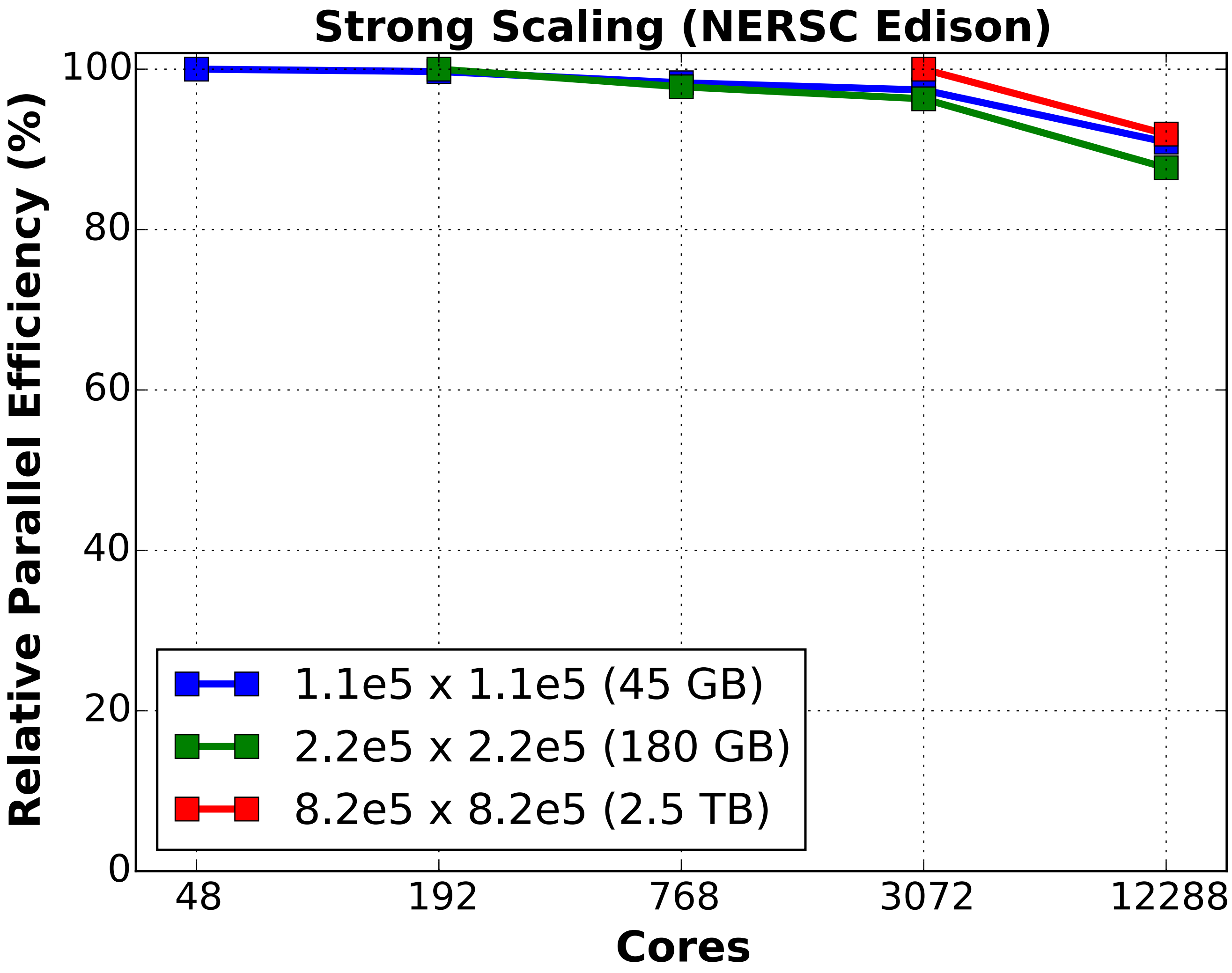
Distributed Matrix Construction

- Remote asyncs with user-controlled resource management
- Divide threads into injectors / updaters
- 6x faster than MPI 3.0 on 1K XE6 nodes

- "Fusing" observational data into simulation
- Interoperates with MPI/Fortran/ScaLAPACK

Scott French, Yili Zheng, Barbara Romanowicz, Katherine Yelick; *"Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates"*, IPDPS 2015

Application Challenge: Data Fusion

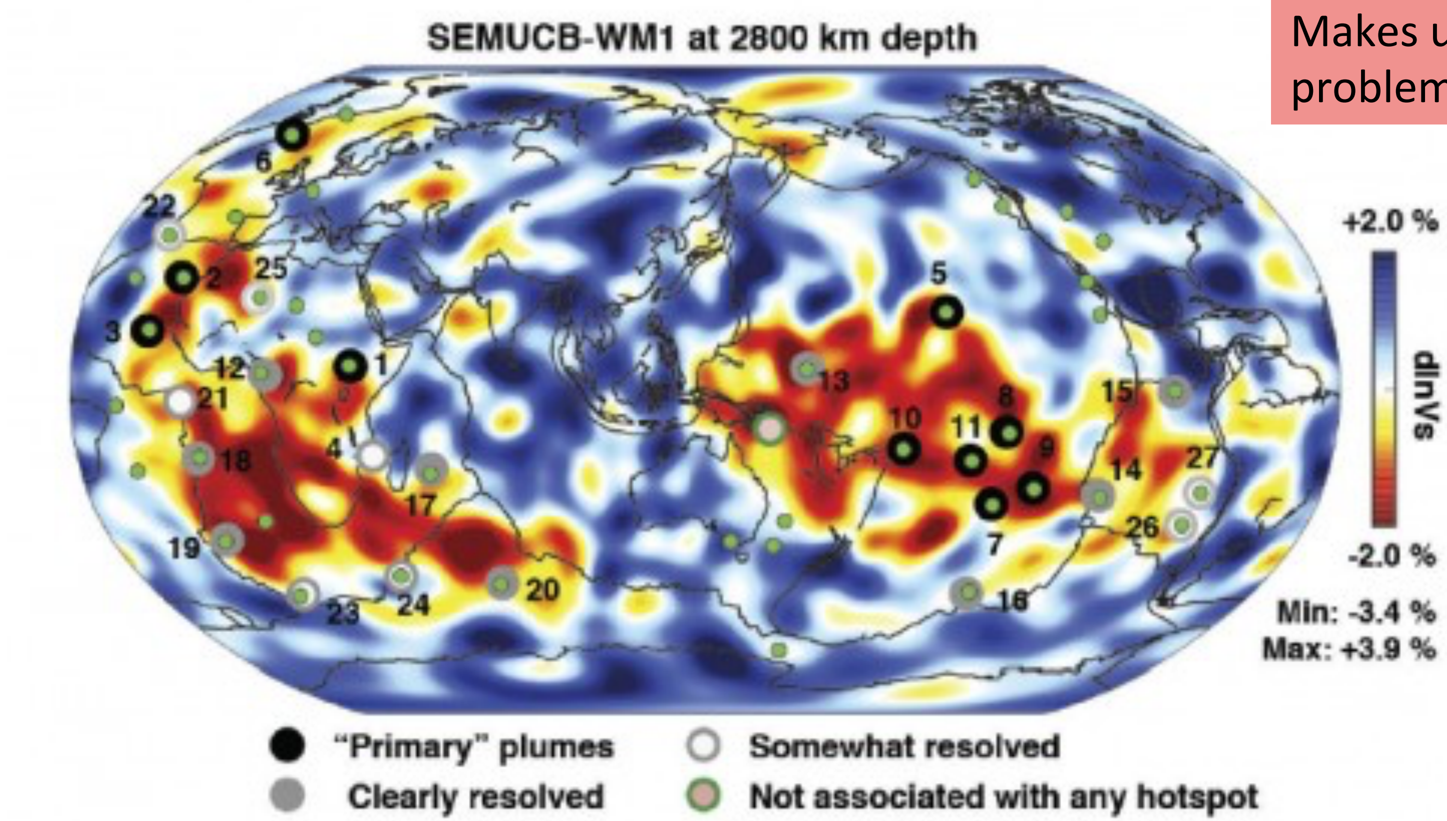


Scott French, Yili Zheng, Barbara Romanowicz, Katherine Yelick; *"Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates"*, IPDPS 2015

Science Impact: Whole-Mantle Seismic Model

- *First-ever whole-mantle seismic model from numerical waveform tomography*
- *Finding: Most* volcanic hotspots are linked to two spots on the boundary between the metal core and rocky mantle 1,800 miles below Earth's surface.

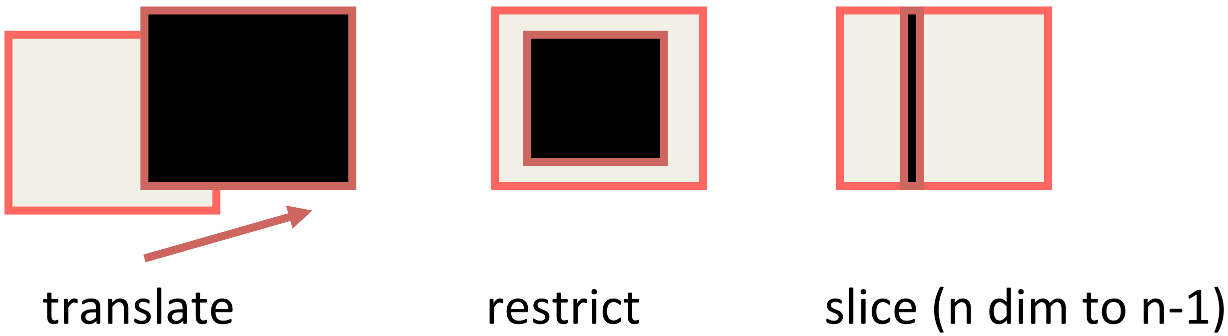
Makes unsolvable problems solvable!

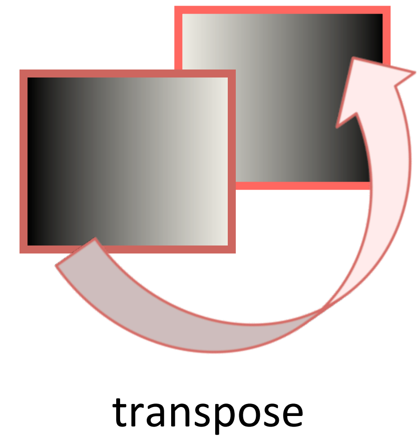


Scott French, Barbara Romanowicz, *"Broad plumes rooted at the base of the Earth's mantle beneath major hotspots"*, **Nature, 2015**

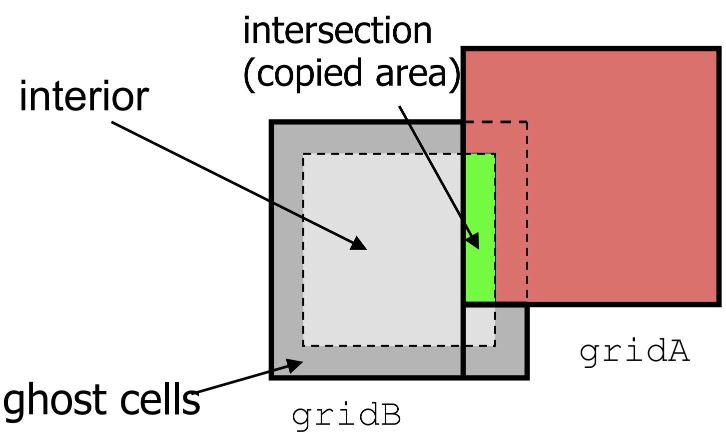
Multidimensional Arrays in UPC++

- **UPC++ arrays have a rich set of operations**

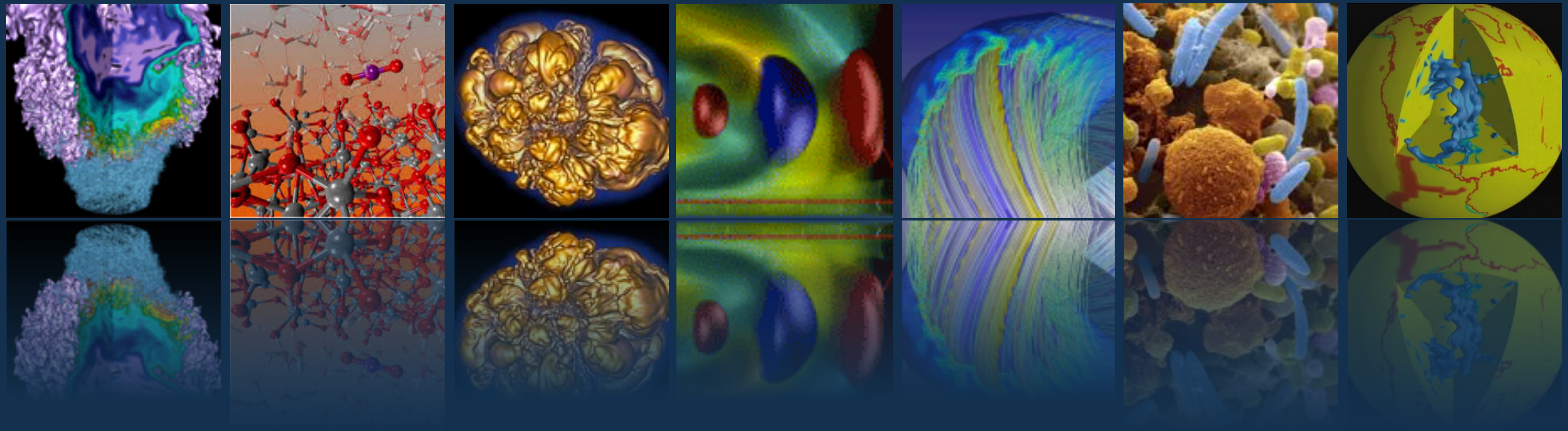




- **Create new views of the data in original array**
- **Example: ghost cell exchange in AMR**




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

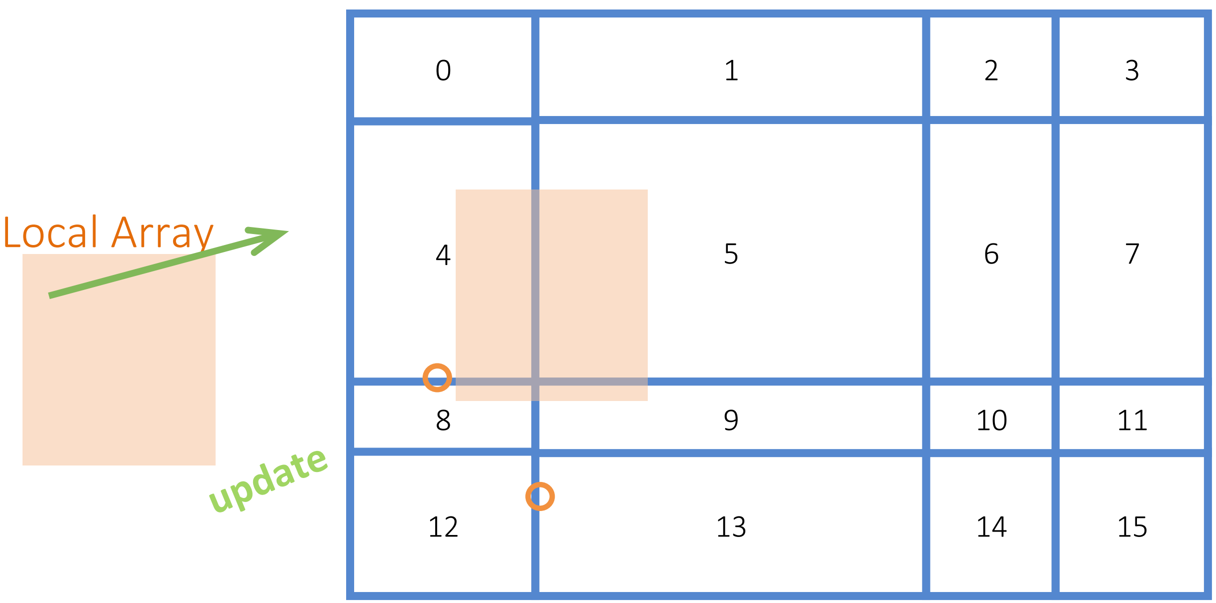


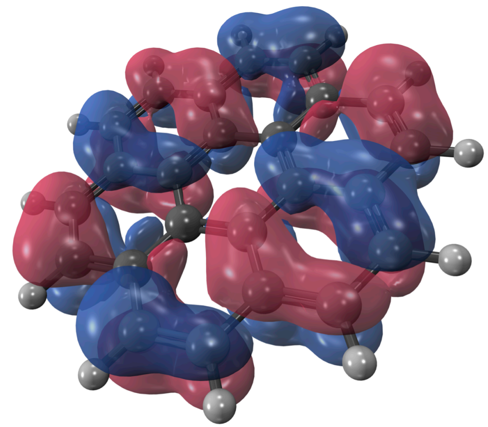
Where is PGAS programming used?

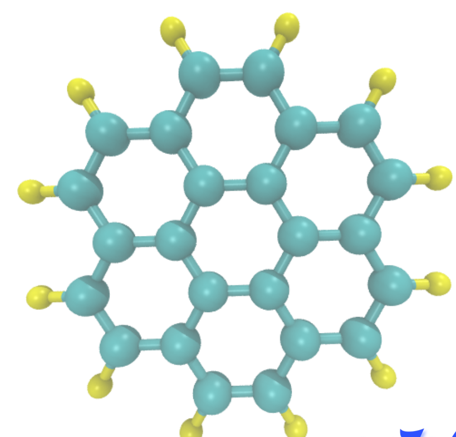
1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Strided irregular updates (adds) to distributed matrix
3. Dynamic work stealing

Application Challenge: Dynamic Load Balancing

- **Hartree Fock example (e.g., in NWChem which is already PGAS)**
  - Inherent load imbalance
- **UPC++ version**
  - Dynamic work stealing and fast atomic operations enhanced load balance
  - Transpose an irregularly blocked matrix

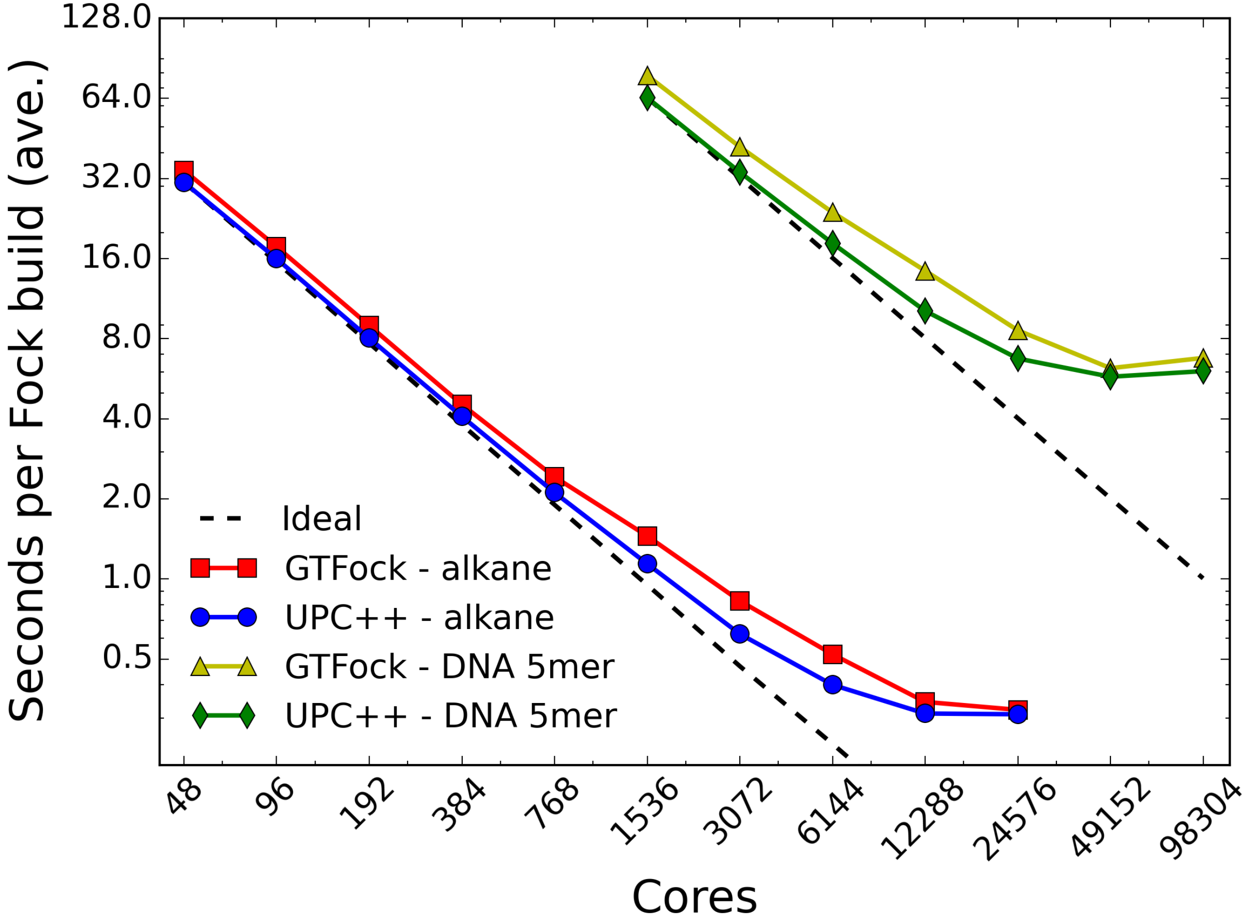






David Ozog (CSGF Fellow), A. Kamil, Y. Zheng, P. Hargrove, J. Hammond, A. Malony, W. de Jong, K. Yelick

Hartree Fock Code



Improved Scalability

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

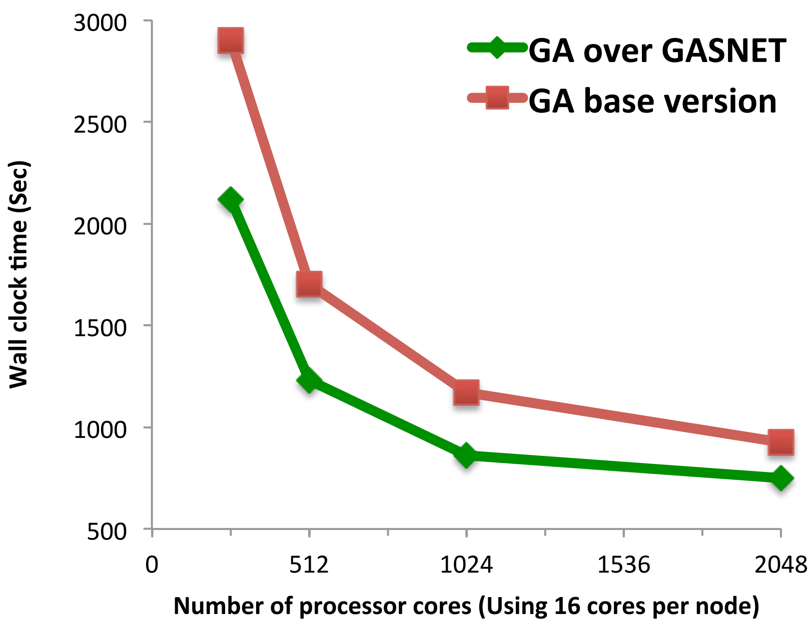
David Ozog (CSGF Fellow), A. Kamil, Y. Zheng, P. Hargrove, J. Hammond, A. Malony, W. de Jong, K. Yelick

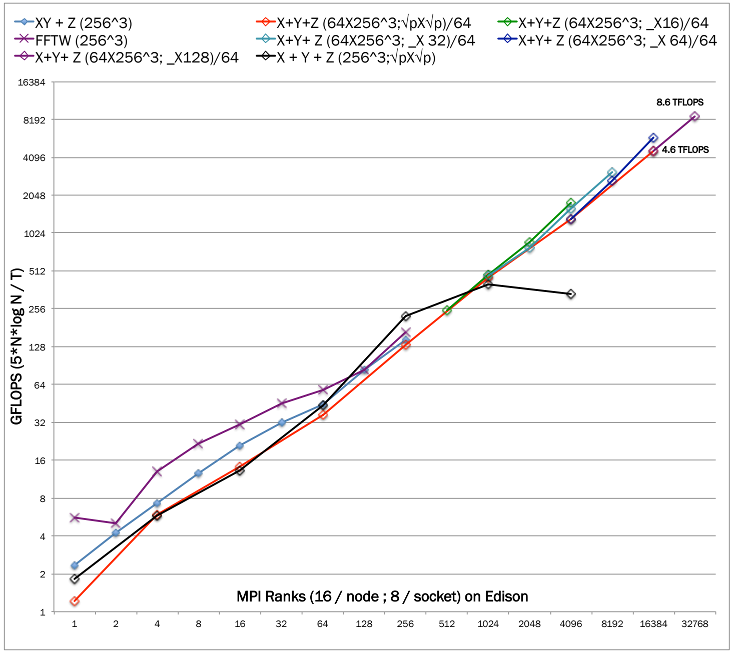
Towards NWChem in UPC++

- **New Global Arrays Toolkit over GASNet**
  - *Over 20% faster on Infiniband*

Increase scalability!

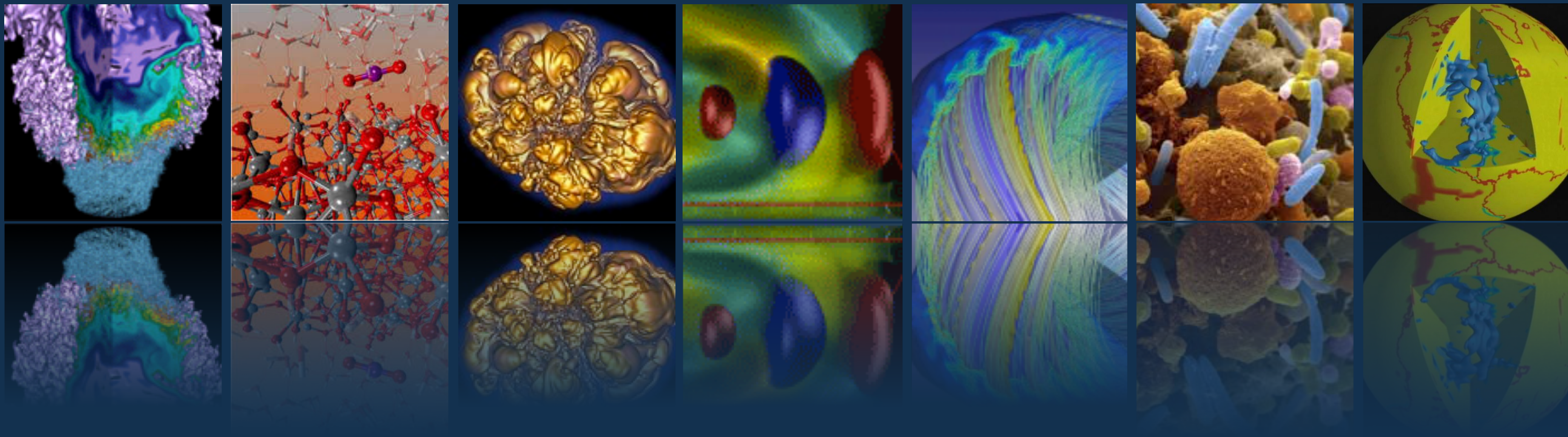
- **More scalable aggregate FFTs than FFTW**





- **Goal of making this ready for production use (Bert de Jong)**

E. Hoffman (GAGA), H. Simhadri (FFTs)

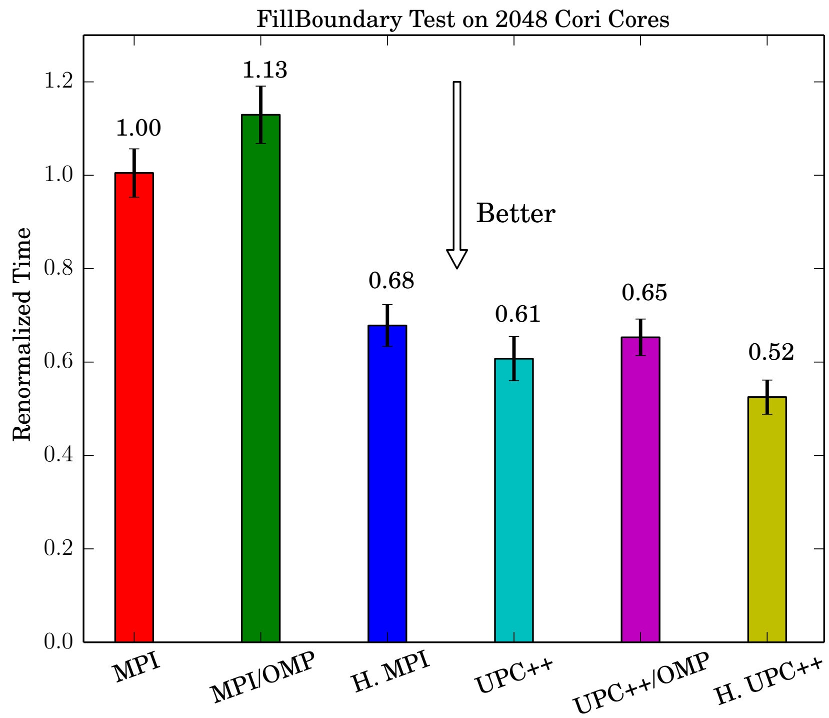


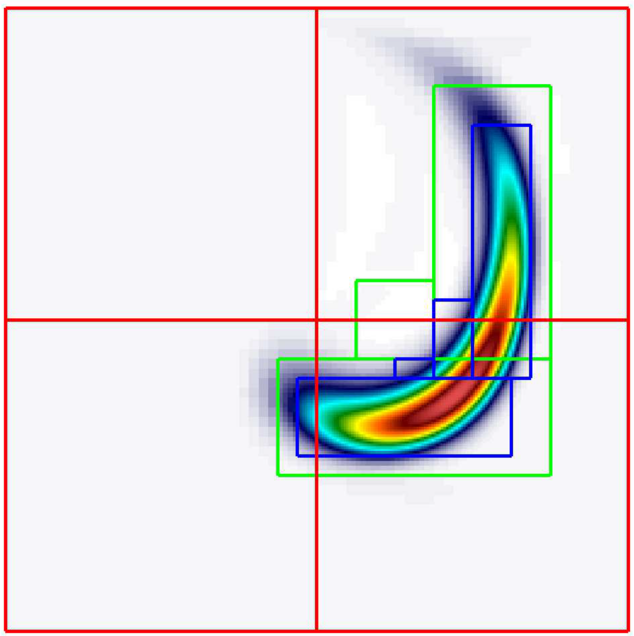
Where is PGAS programming used?

1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Strided irregular updates (adds) to distributed matrix
3. Dynamic work stealing
4. Hierarchical algorithms / one programming model

UPC++ Communication Speeds up AMR

- **Adaptive Mesh Refinement on Block-Structured Meshes**
  - Used in ice sheet modeling, climate, subsurface (fracking), astrophysics, accelerator modeling and many





Hierarchical UPC++ (distributed / shared style)

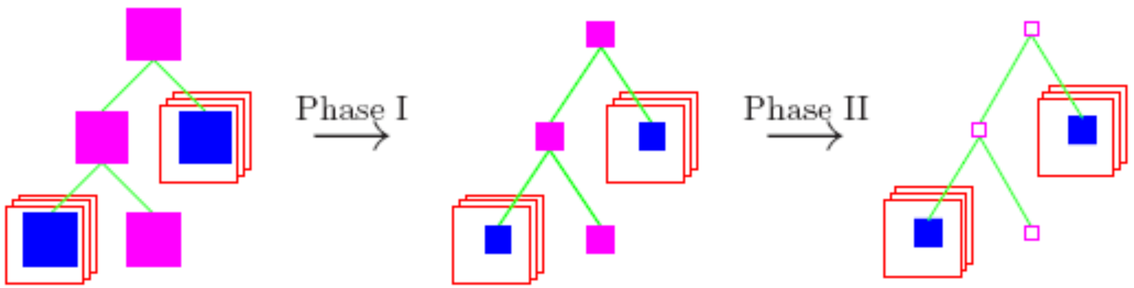
- UPC++ plus UPC++ is 2x faster than MPI plus OpenMP
- MPI + MPI also does well

Reducing Metadata Overhead in AMR

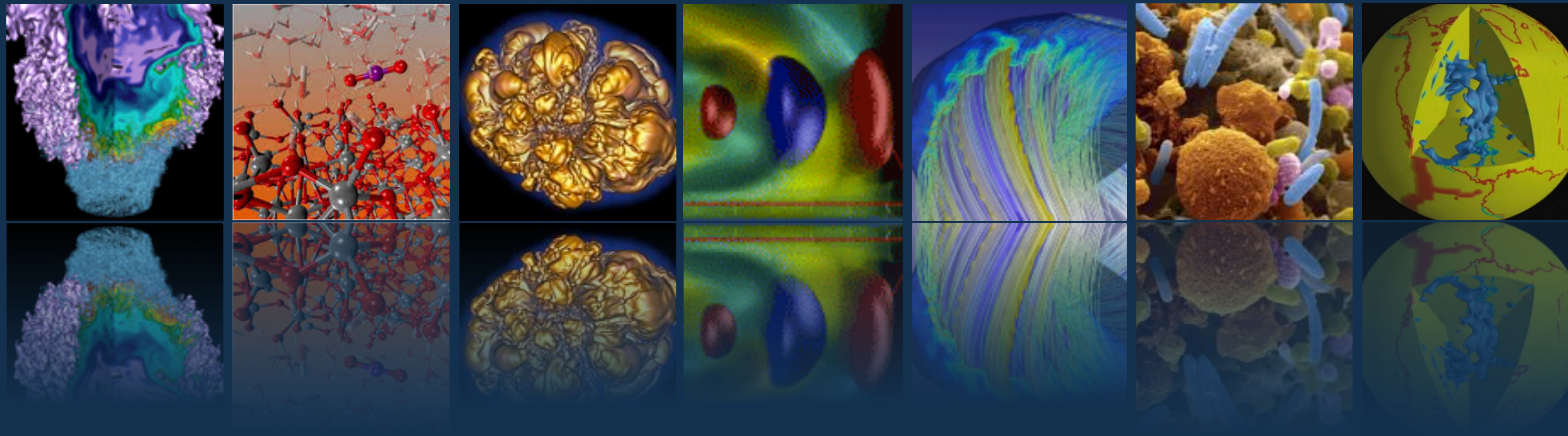
- Reducing the metadata size

phase I: Reduce the size of the grid class

phase II: Split the grid class into grid_local and grid_remote



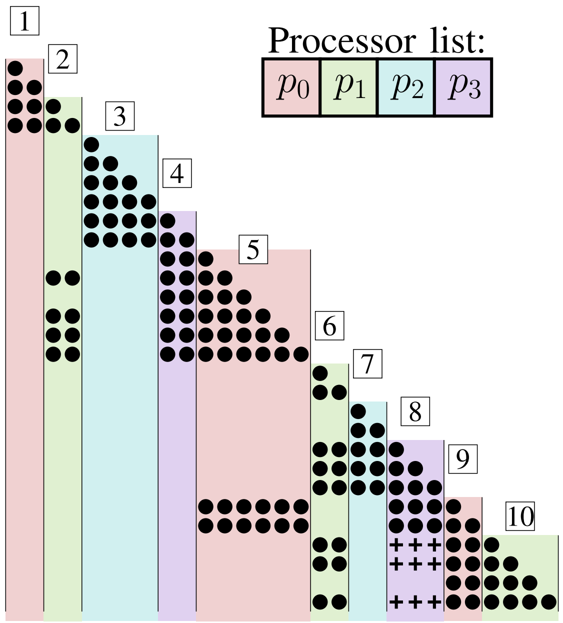
- Distribute the grid hierarchy data structure using UPC

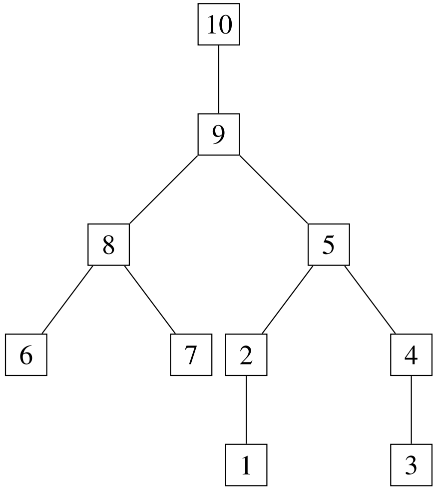


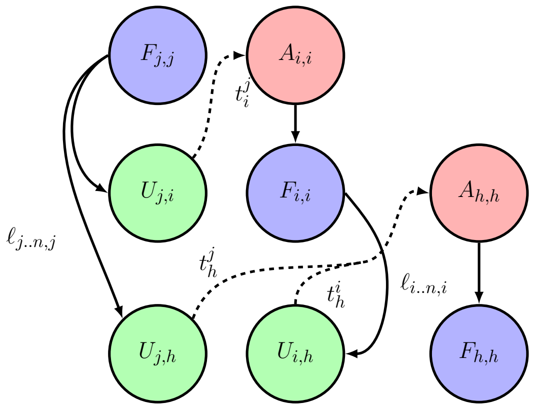
Where is PGAS programming used?

1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Strided irregular updates (adds) to distributed matrix
3. Dynamic work stealing
4. Hierarchical algorithms / one programming model
5. Task Graph Scheduling (UPC++)

Sparse Cholesky as a Parallel Task Graph



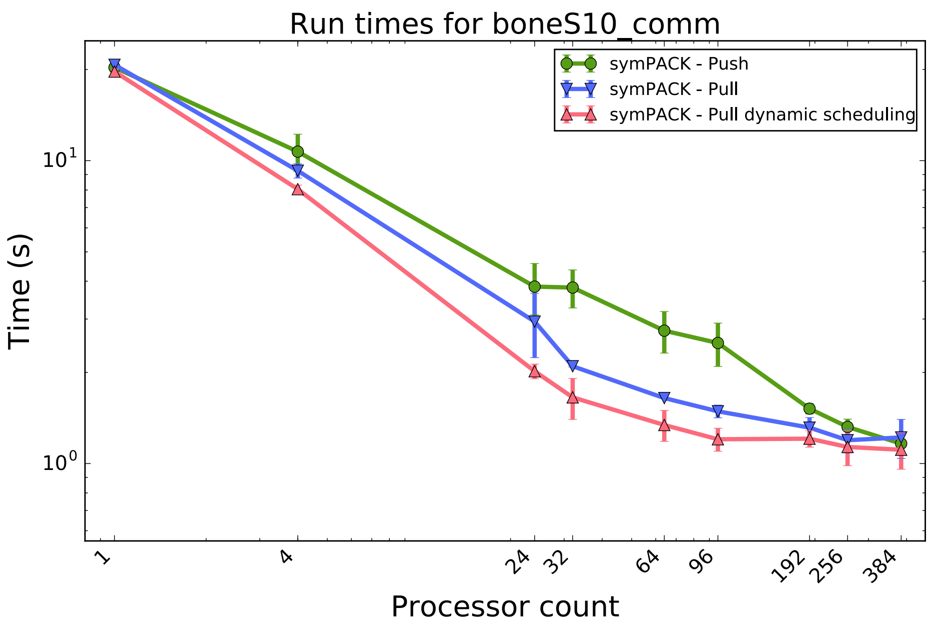


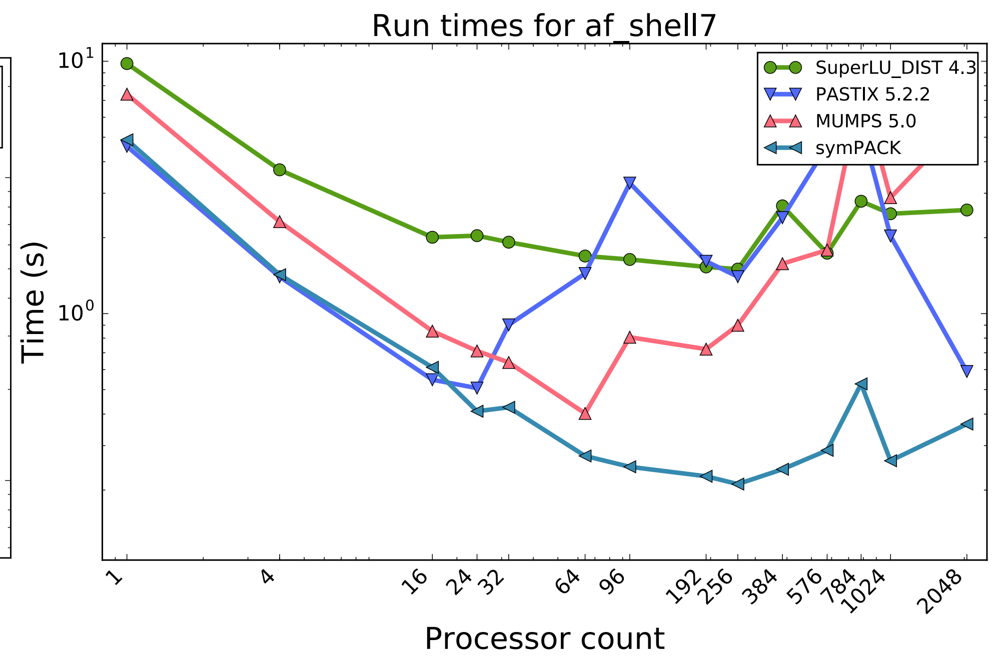


- **Sparse matrix factorization (Cholesky)**
- **Novel fan in/out algorithm programmed in UPC++**

Mathias Jacqueline, Esmond Ng, Yili Zheng, Kathy Yelick

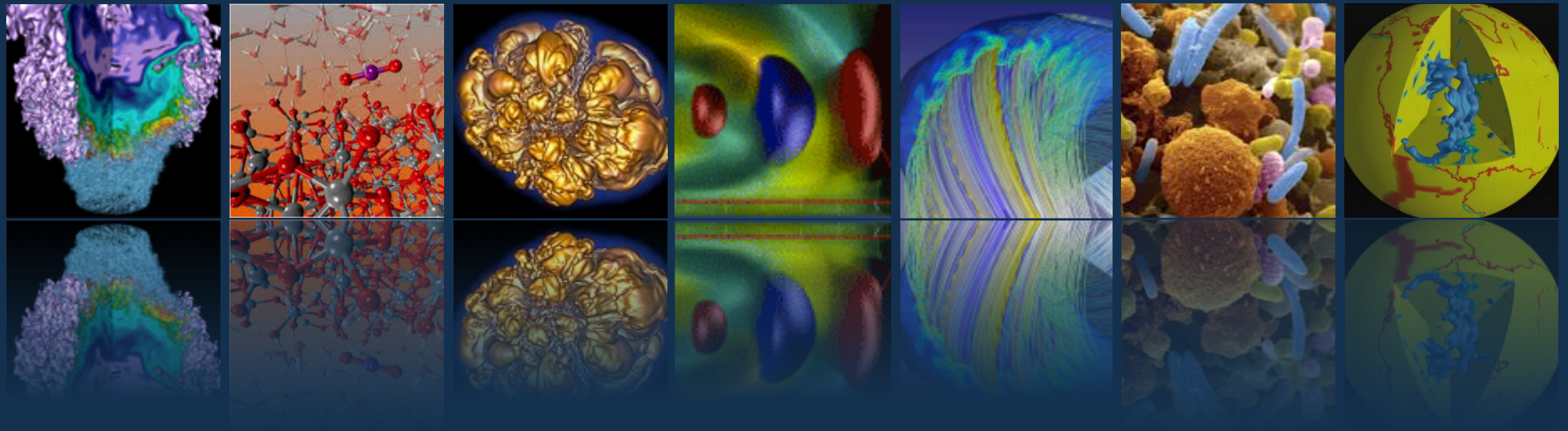
Sparse Cholesky Comparisons





- **Dynamic scheduling outperforms other**
- **The combination of algorithm and implementation (in UPC++) outperforms the competition**

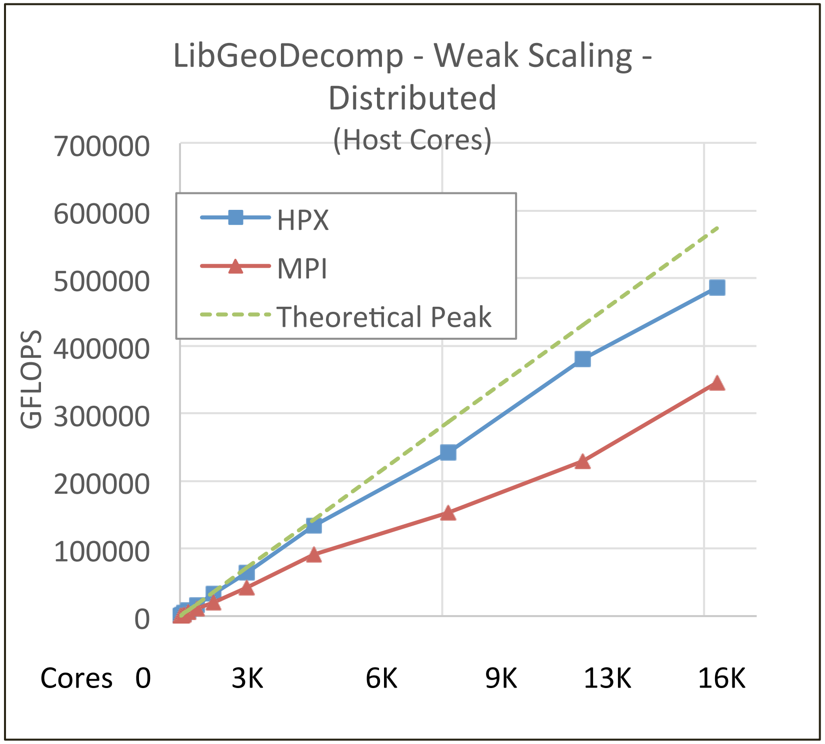
Mathias Jacqueline, Esmond Ng, Yili Zheng, Kathy Yelick

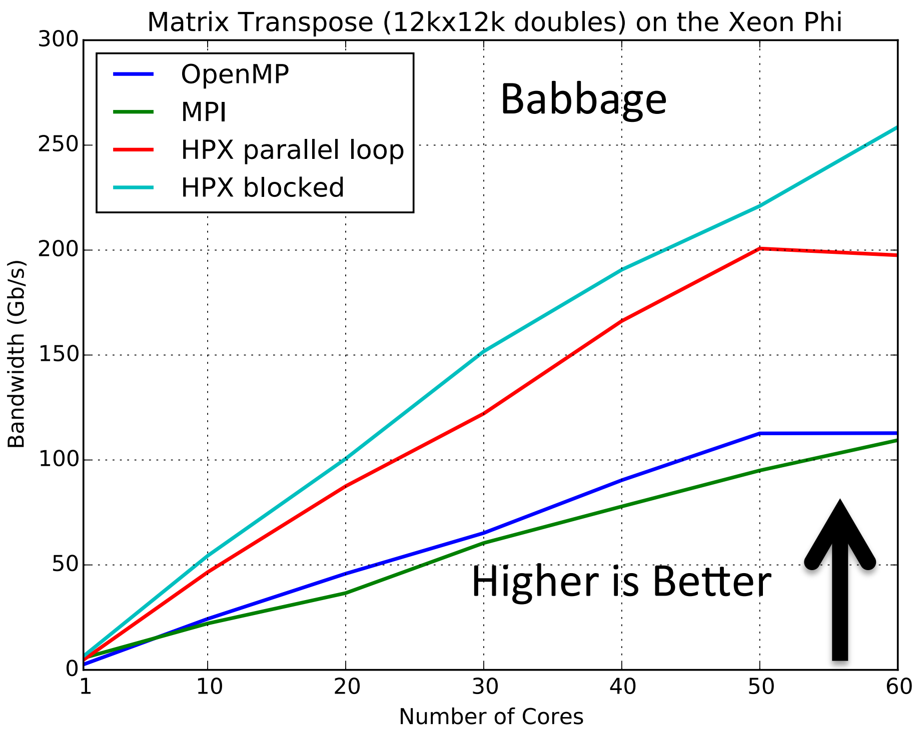


Where is PGAS programming used?

1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Dynamic work stealing
3. Strided irregular updates (adds) to distributed matrix
4. Hierarchical algorithms / one programming model
5. Task Graph Scheduling (UPC++)
6. Dynamic runtimes (CHARM++, Legion, HPX)

HPX Asynchronous Runtime Performs on Manycore

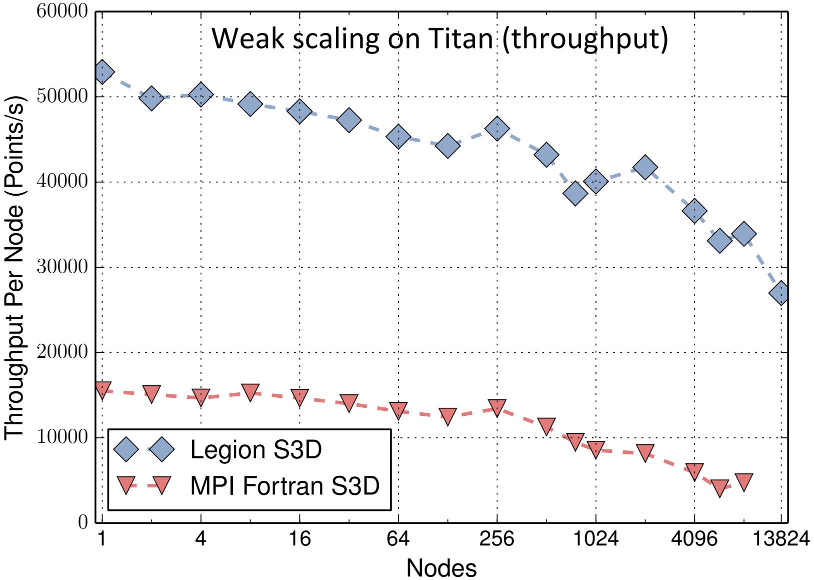


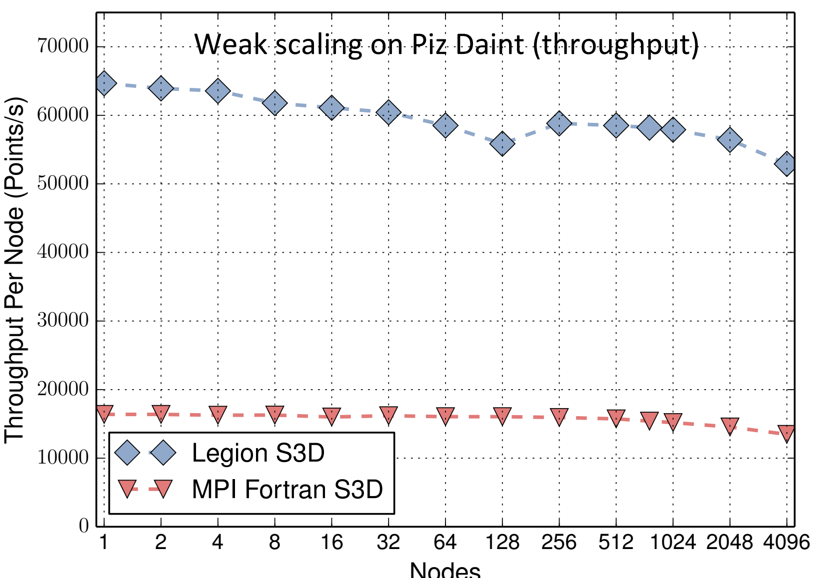


Credit: Harmut Kaiser, LSU and HPX team

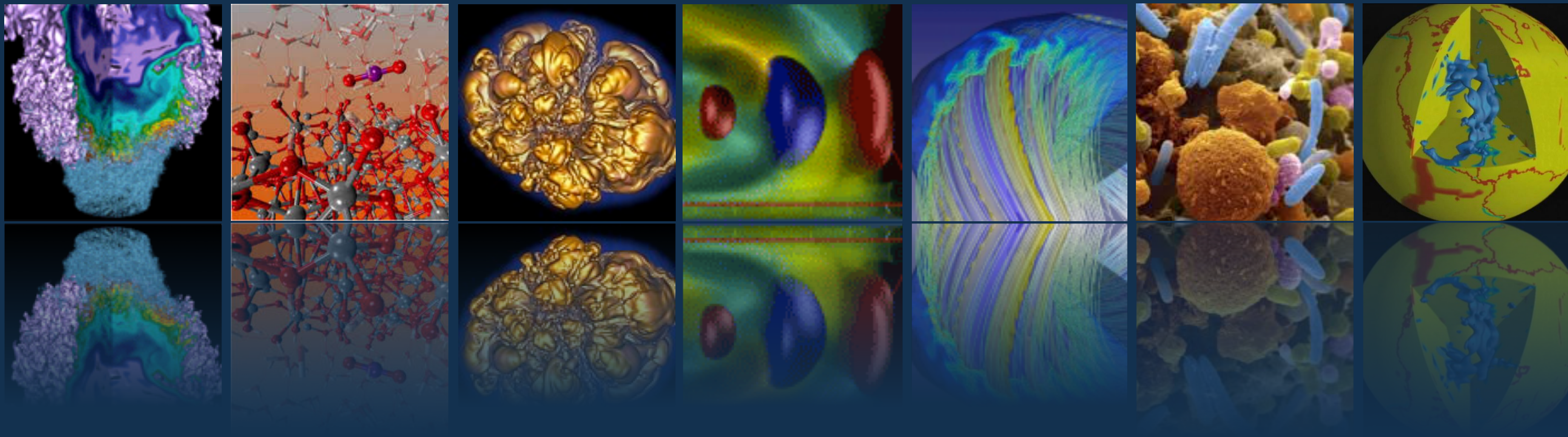
Legion Programming Model & Runtime

- **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
- **Declarative specification of task graph in Legion**
  - Serial program
  - Read/Write effects on regions of data structures
  - Determine maximum parallelism
- **Port of S3D complete**



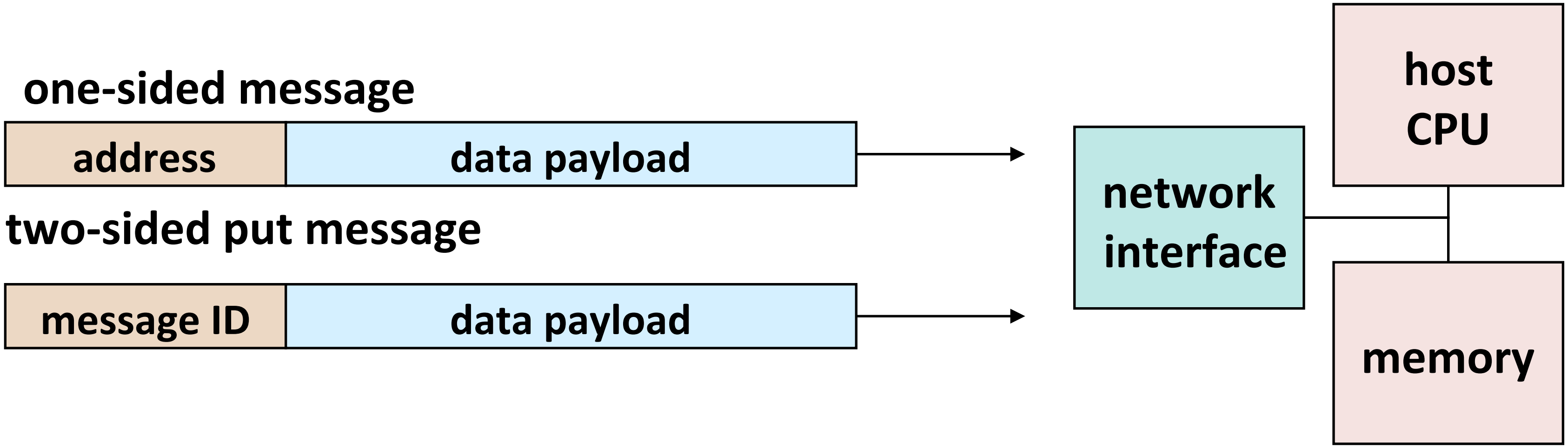


Legion team from Stanford, ExaCT Co-Design Center



Why is PGAS used? (Besides Application Characteristics)

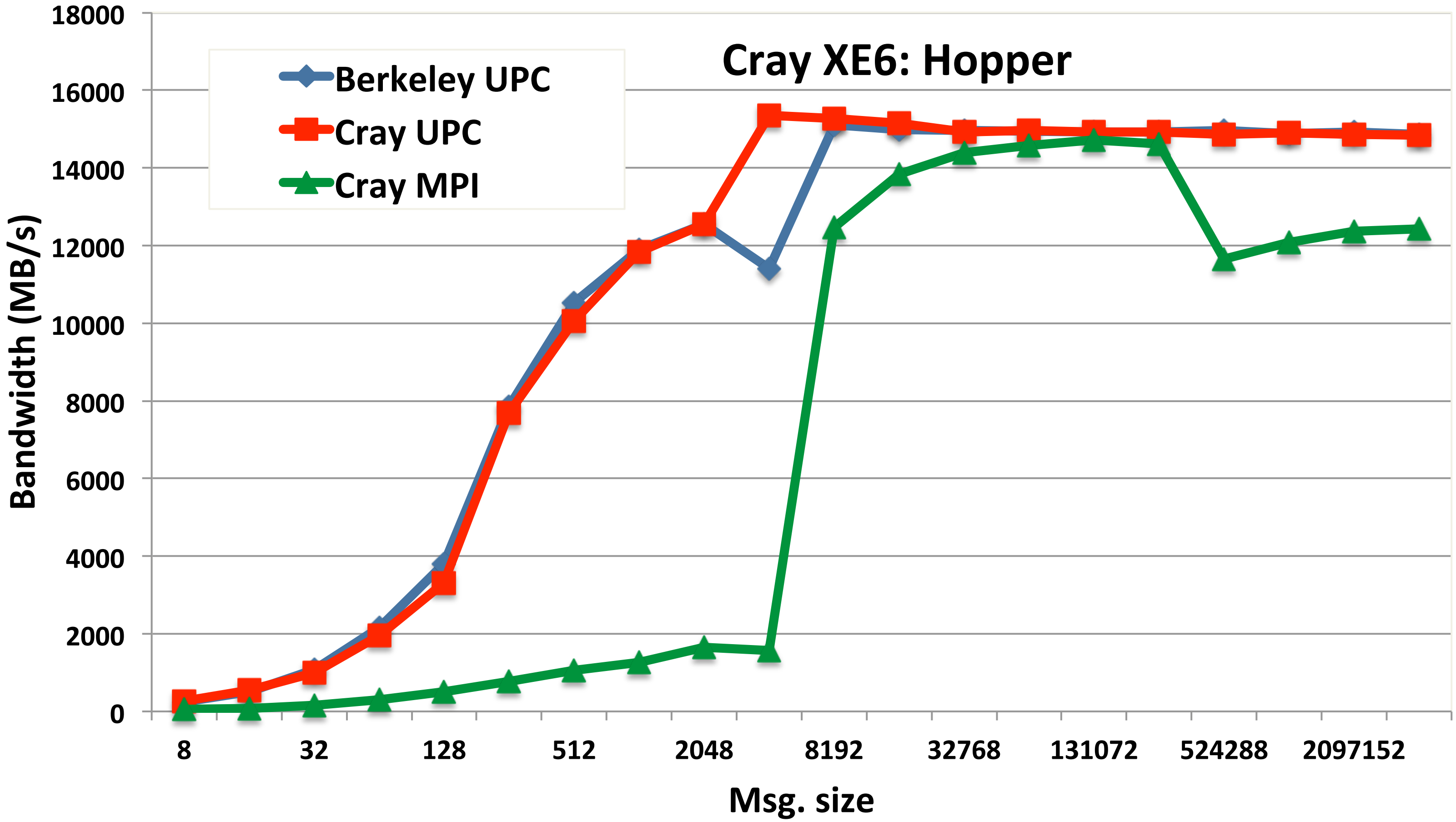
One-Sided Communication is Closer to Hardware



- **One-sided communication (put/get) is what hardware does**
  - Even underneath send/receive
  - Information on where to put the date is in the message
  - Decouples synchronization from transfer
- **Two-sided message passing (e.g., send/receive in MPI)**
  - Requires matching with remote side to "find" the address to write data
  - Couples data transfer with synchronization (often, but not always what you want)

Exascale should offer programmers / vendors a lightweight option

One-Sided Communication Between Nodes is Faster



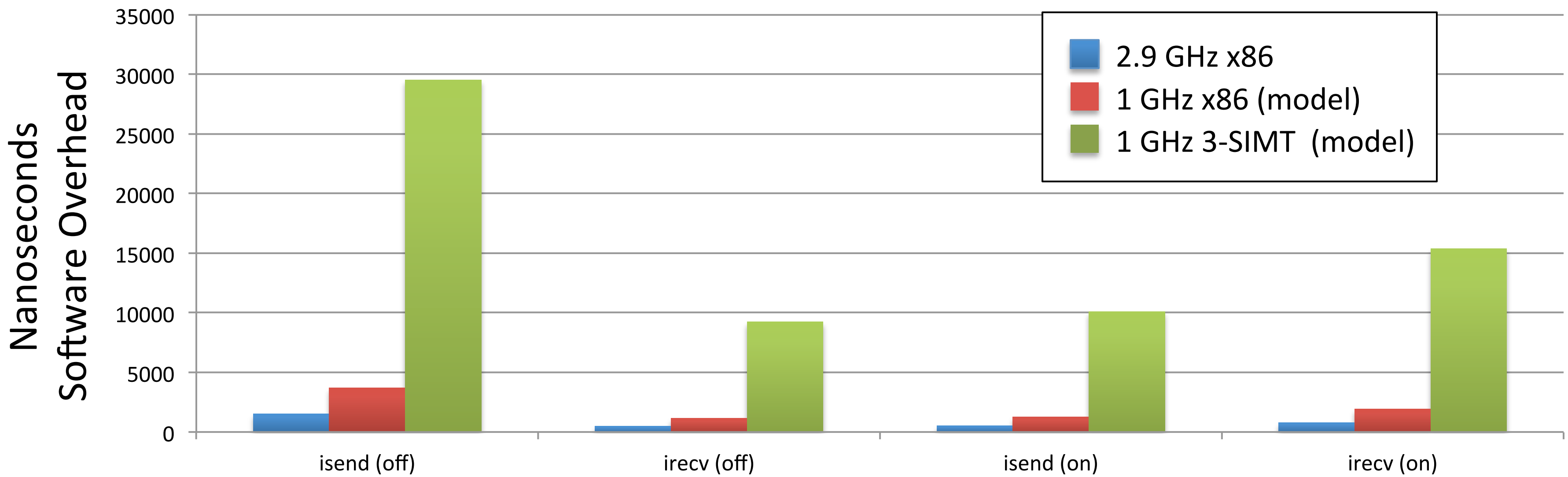
- **For communication-intensive problems, the gap is substantial**
  - Problems with small messages
  - Bisection bandwidth problems (global FFTs)

Hargrove, Ibrahim

Overhead for Messaging

- **Overhead (processor busy time) gets worse on "exascale" cores**
- **Having a low overhead option is increasingly important**

Avg cycles per call *(to do nothing)* On Intel Ivybridge	**Off Node**	**On-Node**
iSend()	3,692 cycles	1,262 cycles
iRecv()	1,154 cycles	1,924 cycles



Summary

- Successful PGAS applications are mostly asynchronous

1. Asynchronous fine-grained reads/write/atomics (aggregation and software caching when possible)
2. Dynamic work stealing
3. Strided irregular updates (adds) to distributed matrix
4. Hierarchical algorithms / one programming model
5. Task Graph Scheduling (UPC++)
6. Dynamic runtimes (CHARM++, Legion, HPX)

- Exascale architecture trends