12th ANNUAL WORKSHOP 2016

UPC++

Yili Zheng

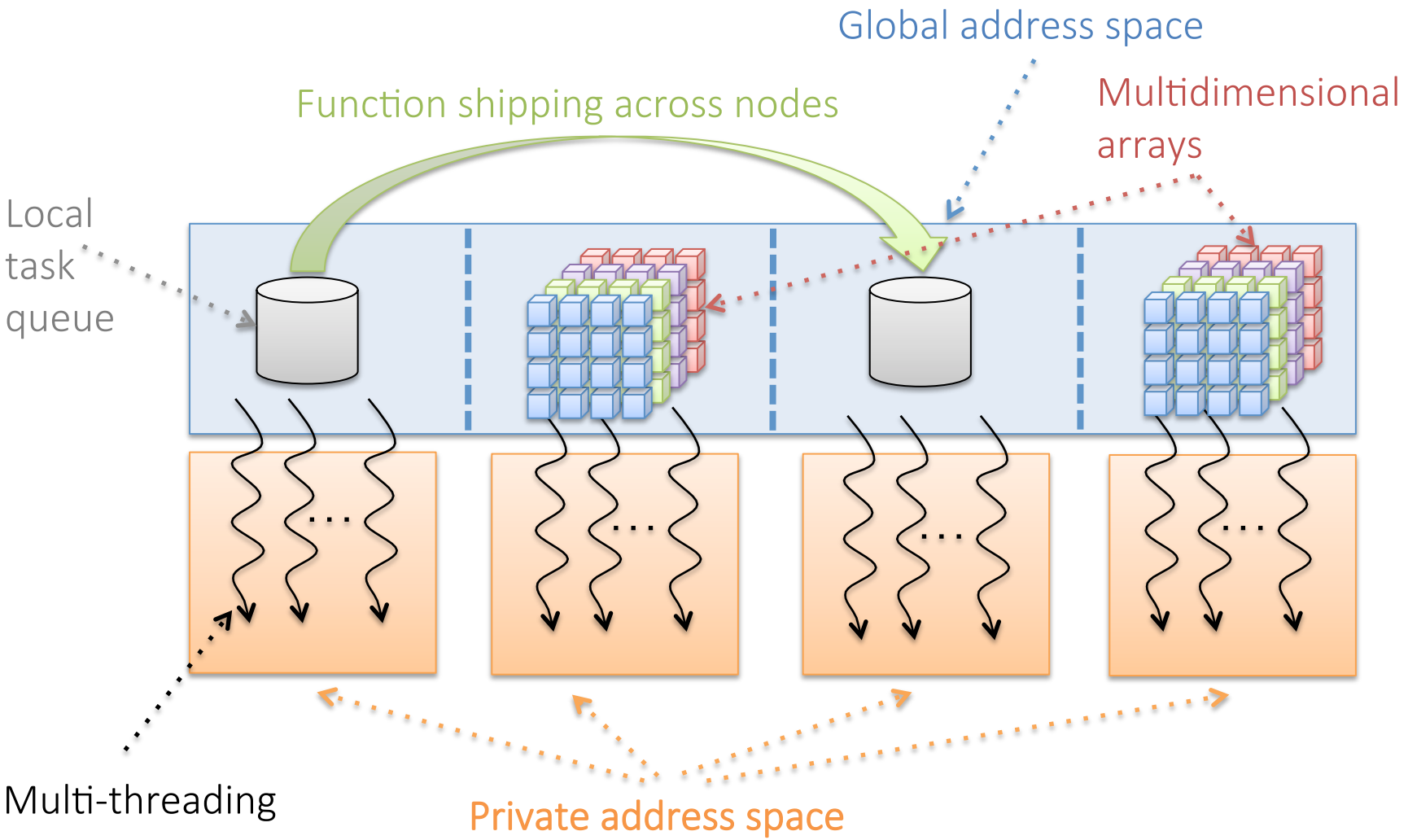
Lawrence Berkeley National Laboratory

[April 8th, 2016]

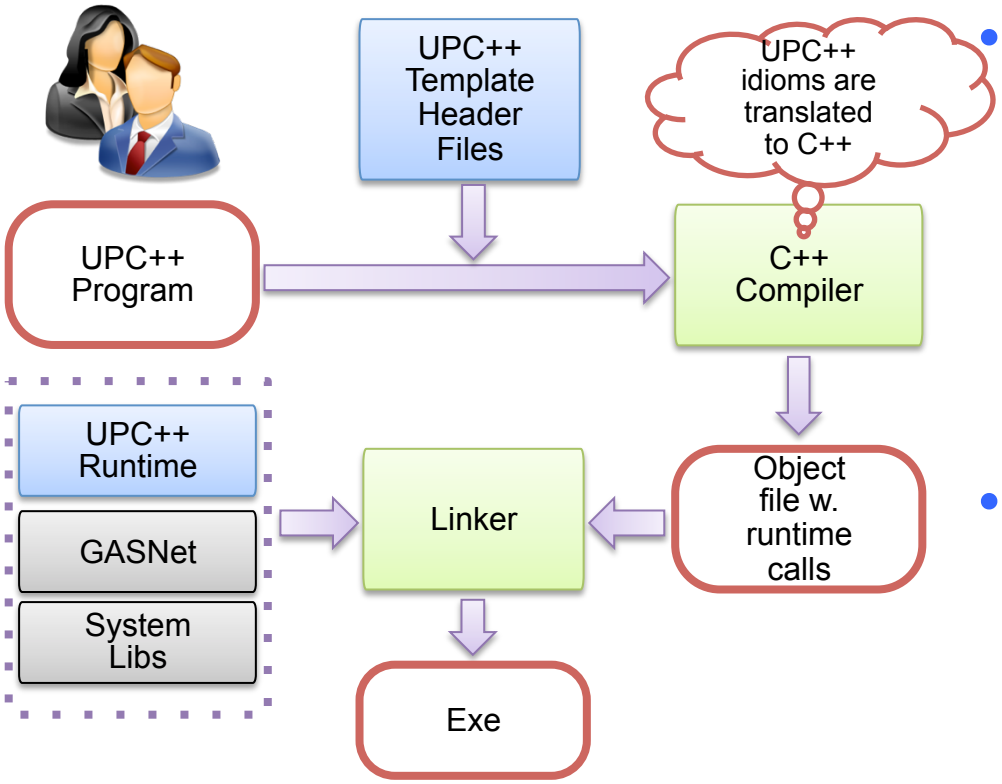
UPC++ OBJECTIVES

- **PGAS and beyond**
  - UPC-style PGAS
  - Asynchronous remote function invocation and dynamic event-driven task execution
  - Titanium-style multidimensional arrays
- **Easy on-ramp for existing applications**
  - Based-on C++ (and C++11 if available)
  - Compiler-free (but the user doesn't need to know how we implement it)
  - Interoperate with existing codes, e.g., MPI/ OpenMP/CUDA/…

PGAS WITH COMPOSABLE EXTENSIONS



A "COMPILER-FREE" APPROACH FOR PGAS



- Leverage C++ standards and compilers
  - Implement UPC++ as a C++ template library
  - C++ templates can be used as a mini-language to extend C++ syntax
- Many new features in C++11
  - E.g., type inference, variadic templates, lambda functions, r-value references
  - C++ 11 is well-supported by major compilers

RANDOM ACCESS BENCHMARK

```
// shared uint64_t Table[TableSize]; in UPC
shared_array<uint64_t> Table(TableSize);

void RandomAccessUpdate()
{
  uint64_t ran, i;
  ran = starts(NUPDATE / ranks() * myrank());
  for(i = myrank(); i < NUPDATE; i += ranks()) {
    ran = (ran << 1) ^ ((int64_t)ran < 0 ? POLY : 0);
    Table[ran & (TableSize-1)] ^= ran;
  }
}
```

Main update loop

Global data layout

local data layout

DYNAMIC GLOBAL MEMORY

- **Global address space pointers**

```
global_ptr<Type> ptr;
```

- **Dynamic global memory allocation**

```
global_ptr<T> allocate<T>(uint32_t where,
                          size_t count);
void deallocate(global_ptr<T> ptr);
```

Example: allocate space for 512 integers on rank 2

```
global_ptr<int> p = allocate<int>(2, 512);
```

ONE-SIDED DATA TRANSFER

```
// Copy count elements of T from src to dst
copy<T>(global_ptr<T> src, global_ptr<T> dst,
        size_t count);

// Implicit non-blocking copy
async_copy<T>(global_ptr<T> src, global_ptr<T> dst,
              size_t count);

// Synchronize all previous asyncs
void async_wait();

// Return 1 if previous asyncs are done; 0 otherwise
int async_try();
```

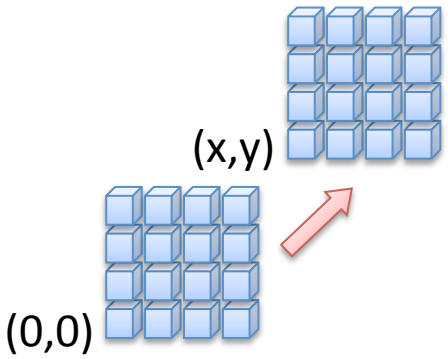
ASYNC TASK EXAMPLE

```
#include <upcxx.h>

void print_num(int num) {
  printf("myrank %u, arg: %d\n", myrank(), num);
}

int main(int argc, char **argv) {
  ...
  for (int i = 0; i < upcxx::ranks(); i++) {
    upcxx::async(i)(print_num, 123);
  }
  upcxx::async_wait();
  ...
}
```

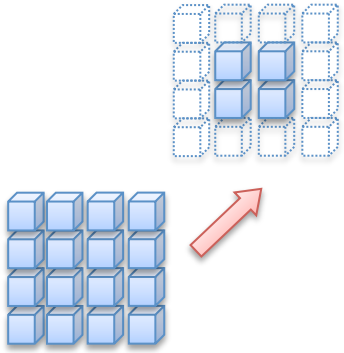
FINISH-ASYNC PROGRAMMING IDIOM

```
using namespace upcxx;

// Thread 0 spawns async tasks
finish {
  for (int i = 0; i < ranks(); i++) {
    async(i)([] (int num)
             { printf("num: %d\n”, num); },
             1000+i);
  }
} // All async tasks are completed
```

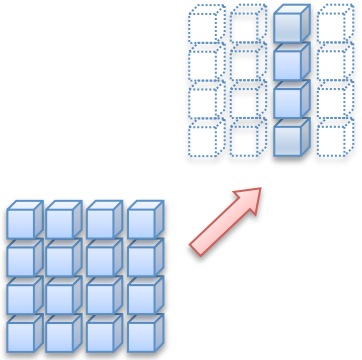
UPC++ MD ARRAY LIBRARY



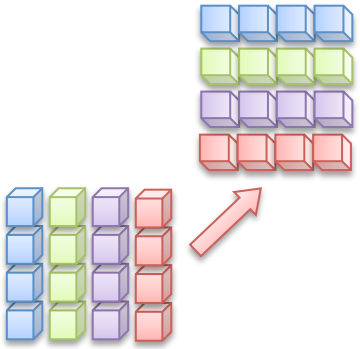
Translate



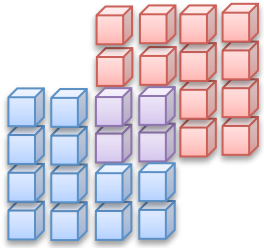
Restrict



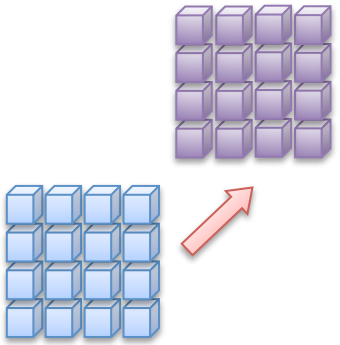
Slice (n dim to n-1)



Transpose

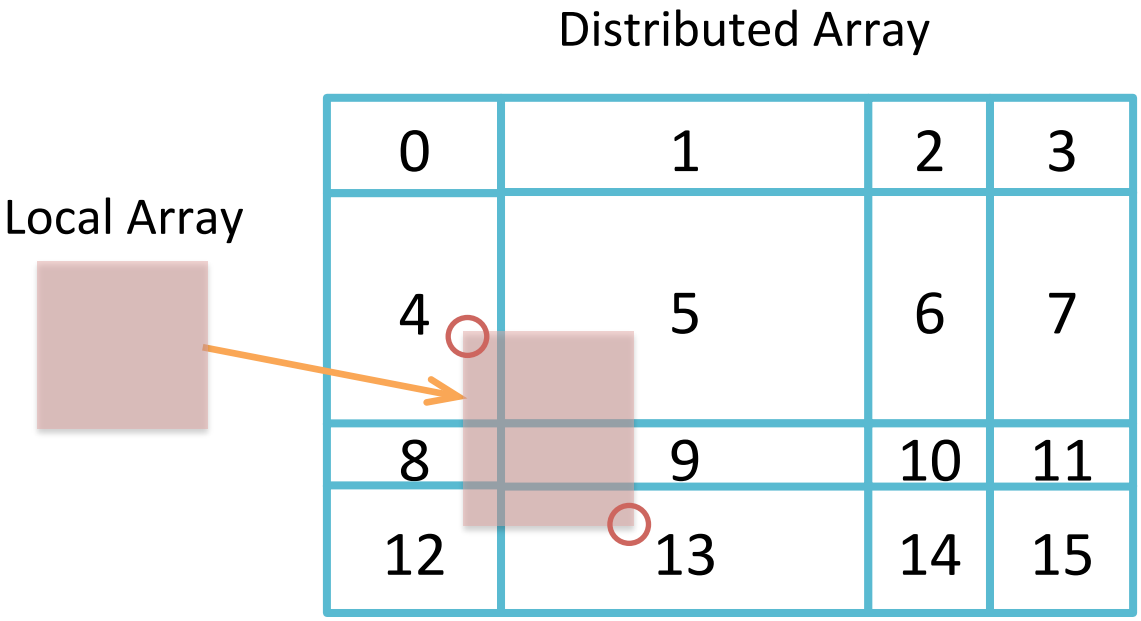


Intersect



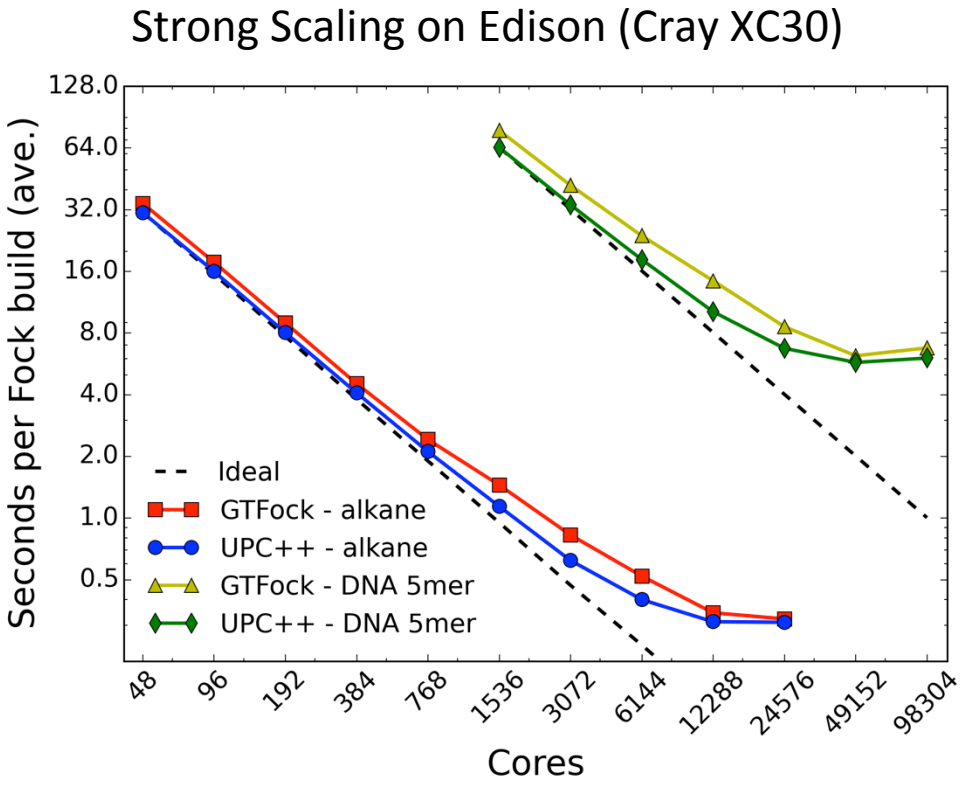
Copy

IRREGULAR SUBMATRIX UPDATE

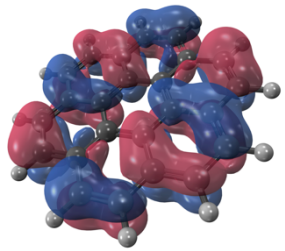


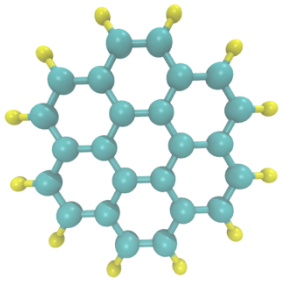
- Dynamic work stealing and fast atomic operations enhance load balance
- New distributed array abstraction delivers productivity and performance

UPC++ FOCK SCALES TO 96K CORES



Lower is better

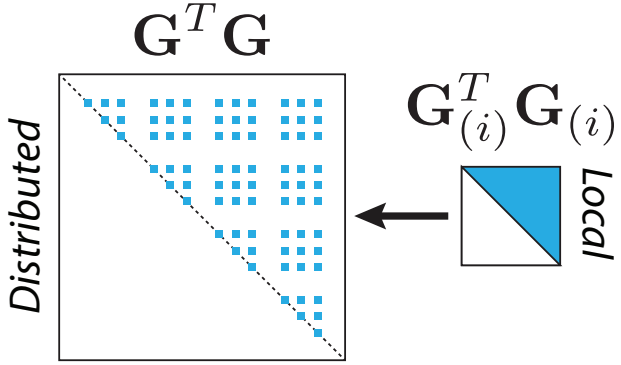


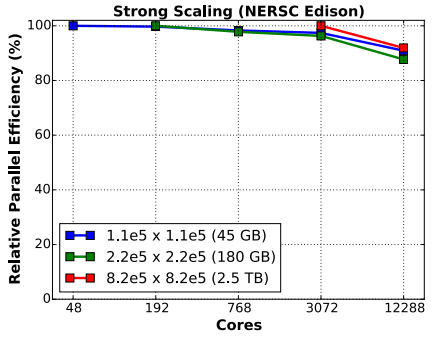


20% faster than the best existing solution

D. Ozog, A. Kamil, Y. Zheng, P. Hargrove, J. R. Hammond, A. Malony, W. de Jong, K. Yelick; *"A Hartree-Fock Application using UPC++ and the New DArray Library",* IPDPS 2016

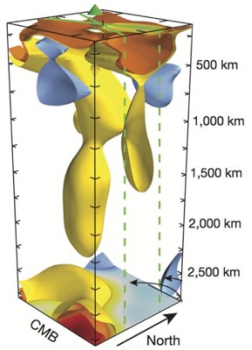
ENABLED NEW SEISMIC DISCOVERY





- Problem `GtG[ix,ix] += GtG_i[:,:]`
  - Massive data don't fit in single memory
  - Dynamic and irregular update patterns
- Solution
  - PGAS + Asynchronous Remote Task Execution using Customized App Logics
- Impact
  - *First-ever whole-mantle seismic model from numerical waveform tomography*
  - *Reveals new details of deep structure not seen before*

Excellent parallel efficiency for strong scaling due to near complete overlap of computation and communication (IPDPS'15)



3D rendering of low-velocity structure beneath the Hawaii hotspot

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

NERSC 2016 Achievement Award for Innovative Use of HPC

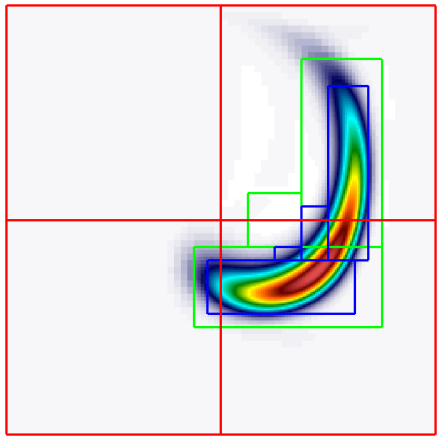
ALTERNATIVE IMPLEMENTATION: MPI-3 RMA

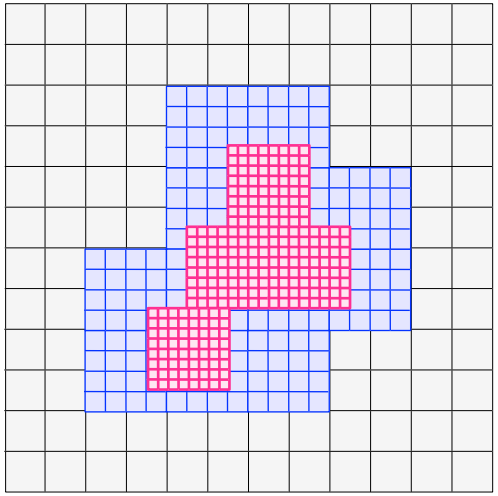
`upcxx::async` (general *functions*)	vs.	**`MPI_Accumulate`** (general *data types*)
• Explicit buffer management • Customized update function with domain knowledge • Progress at both source and target is controllable • One way bulk data movement can be guaranteed		• Opaque internal MPI buffers • Generalized MPI data types + pre-defined merge ops • Progress is impl.-specific and not controllable at target • Data may take more than one trip to ensure passive target (ex: bulk accumulate in foMPI)

More opportunities to exploit problem / domain specific knowledge

ADAPTIVE MESH REFINEMENT

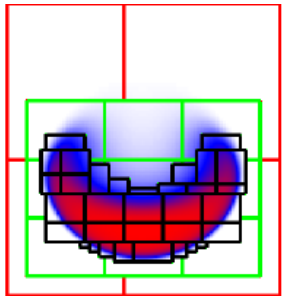
- **AMR allows the method to dynamically adapt the multilevel grid hierarchy on which the equations are solved.**
- **Finer level composed of union of regular subgrids but the union may be irregular**
- **Intensive and dynamic data exchange communication required**
  - Between levels
  - Neighbors within the same level



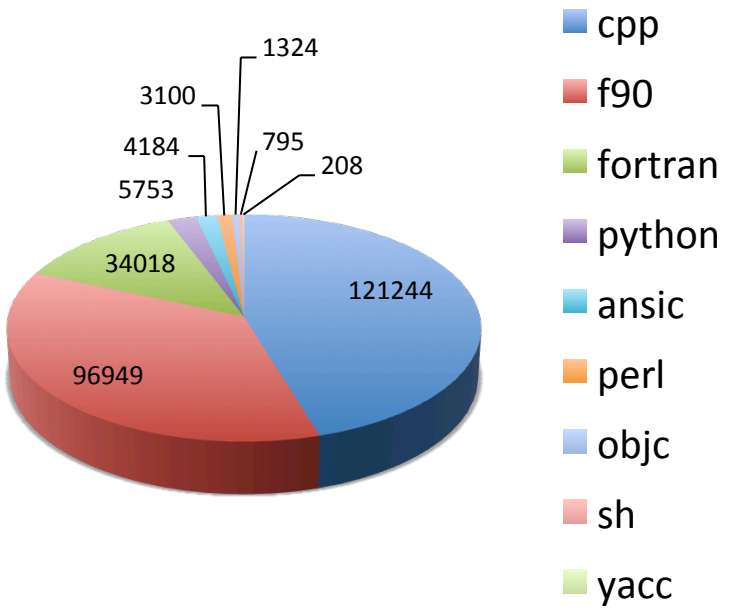


BOXLIB AMR FRAMEWORK

- BoxLib mostly written in C ++ and Fortran 90 with MPI+OpenMP
  - BoxLib development effort estimated by SLOCCount: 70.77 Person-Years ($24.77M)
- Need an incremental adoption strategy with maximum code reuse
- Collaboration and integration are key!



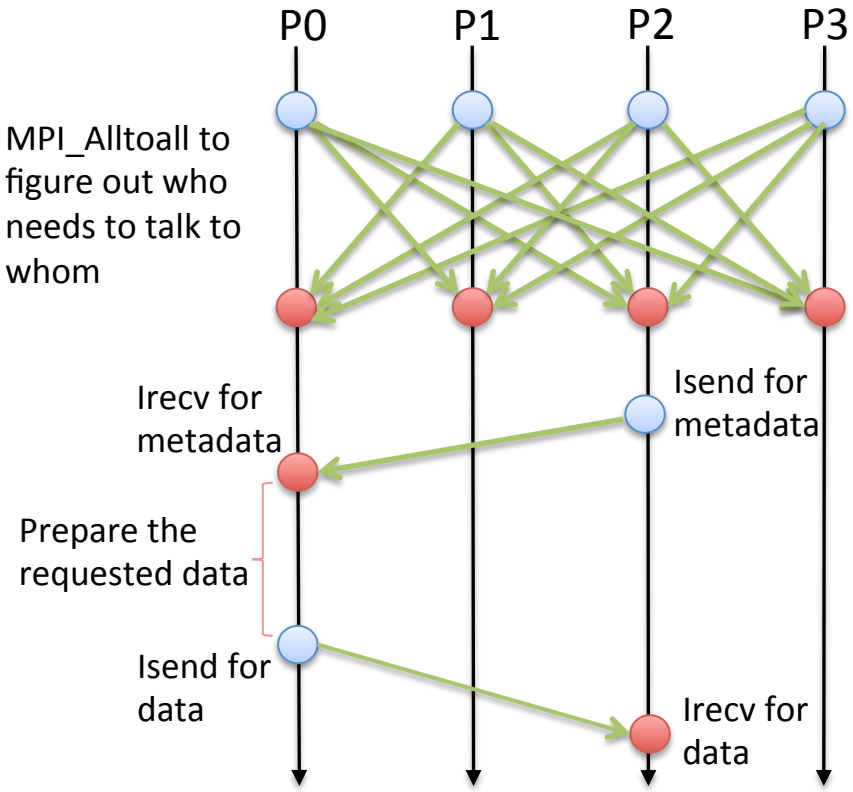
Source Lines of Code



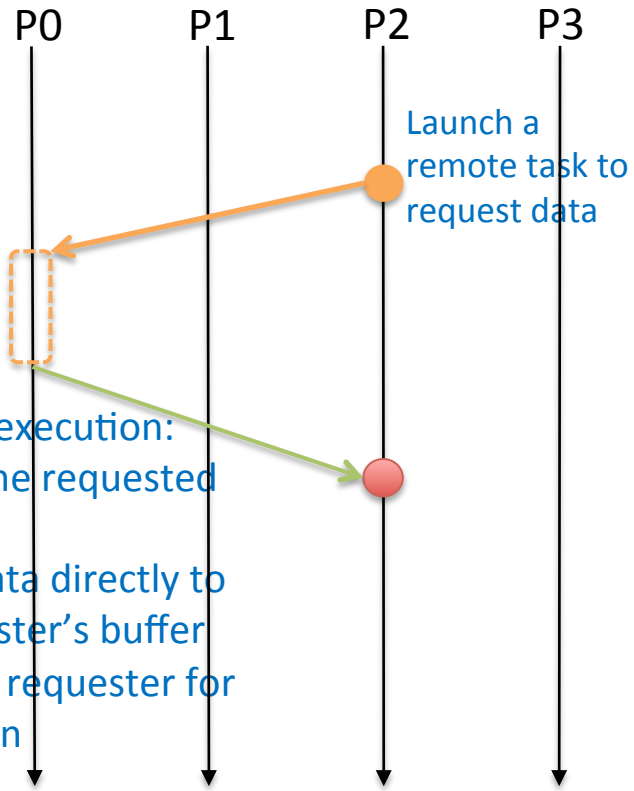
ACTIVE MESSAGES SIMPLIFY COMMUNICATION WORKFLOW

In the following example, P2 needs data from P0.

Original Communication Pattern in BoxLib



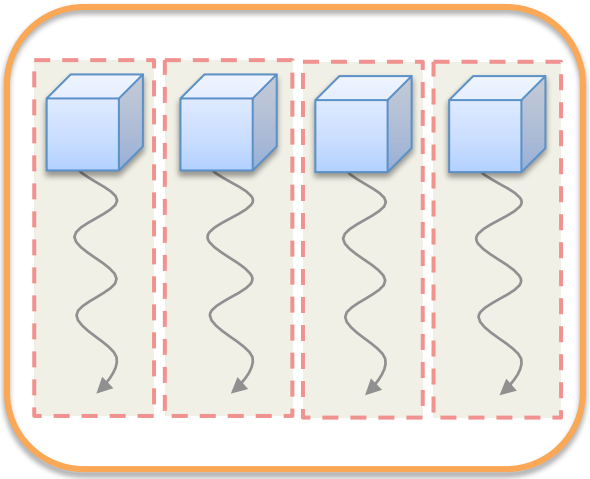
Active Message Model in BoxLib



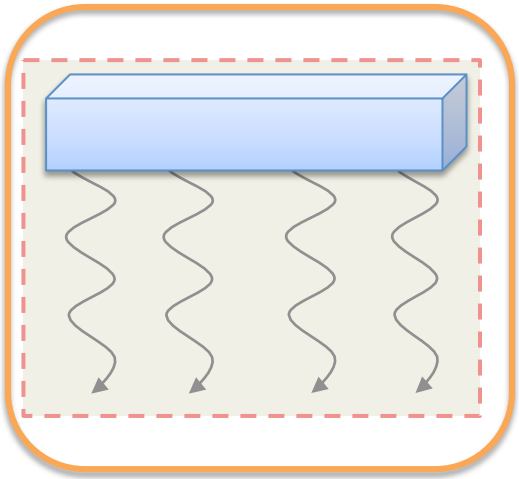
Remote task execution:

- Prepare the requested data
- Put the data directly to the requester's buffer
- Signal the requester for completion

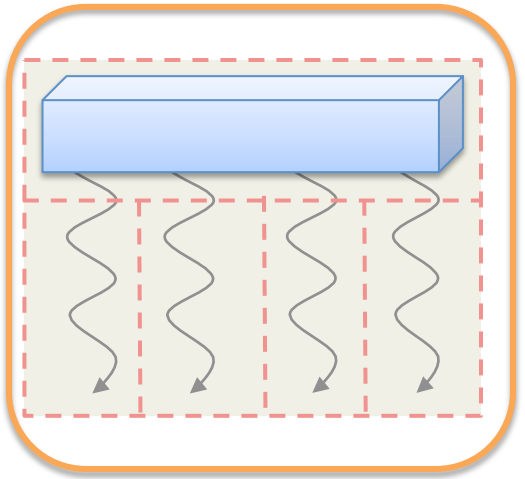
PGAS FOR EFFICIENT COMMUNICATION AND DATA SHARING WITHIN A NODE



Pure Message Passing
Model: no data sharing



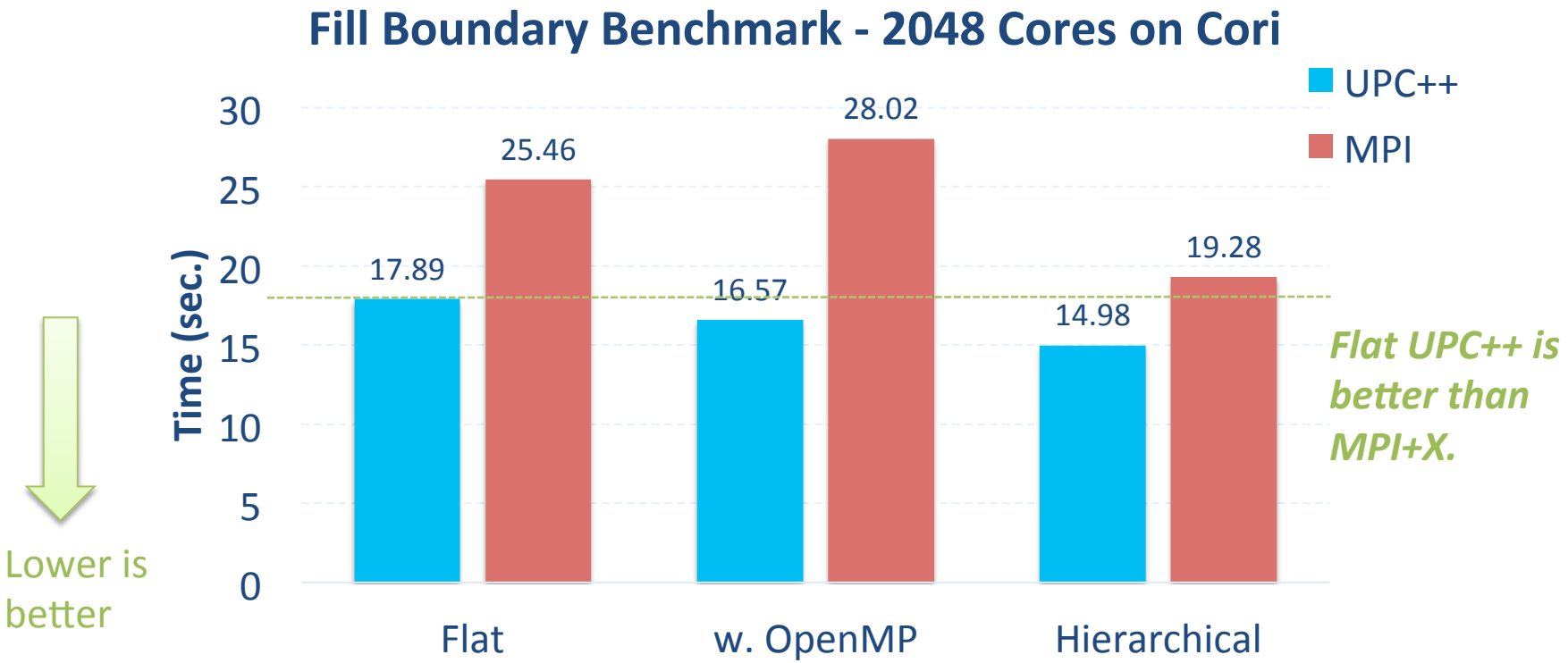
Pure Multi-threading
Model: share all data



PGAS Model: selectively share data

- Pure message passing model is good for data protection and parallel network injection.
- Pure multi-threading model is good for sharing data and intra-node communication.
- PGAS (process-shared-memory) provides both advantages.

COMMUNICATION PERFORMANCE IMPROVEMENT IN BOXLIB



Flat: use only one programming model.
Hierarchical: use one programing model but handle on-node communication through shared-memory (e.g., MPI+MPI)

FULL APPLICATION PERFORMANCE: COMPRESSIBLE ASTROPHYSICS (CASTRO)

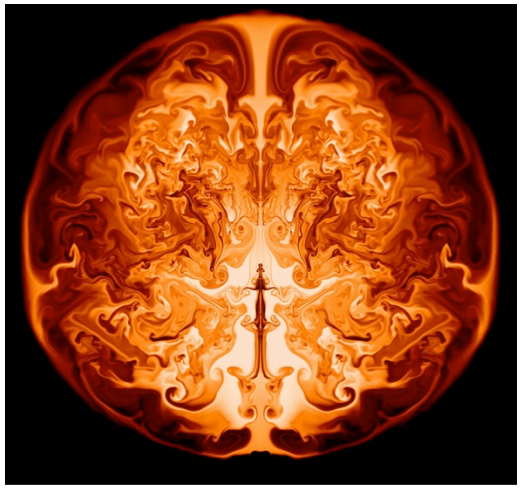
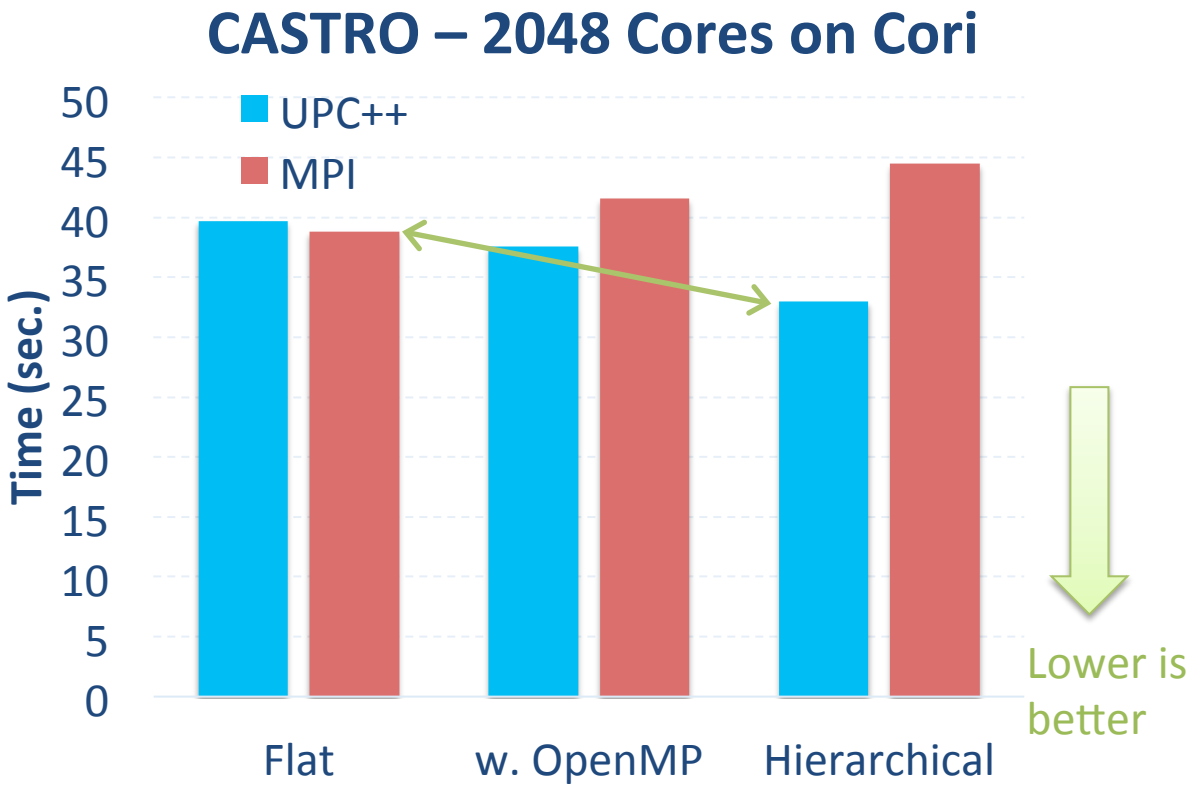


Image Credit: Ken Chen, University of California, Santa Cruz

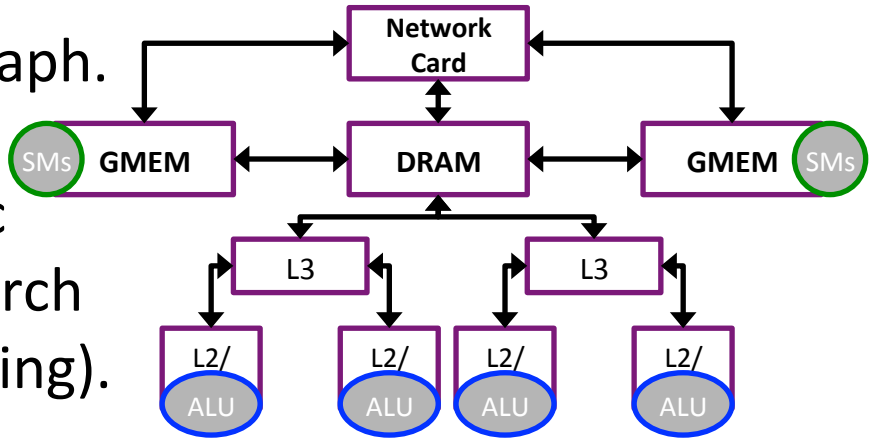
The best UPC++ version (Hierarchical) is 18% faster than the best MPI version (flat).

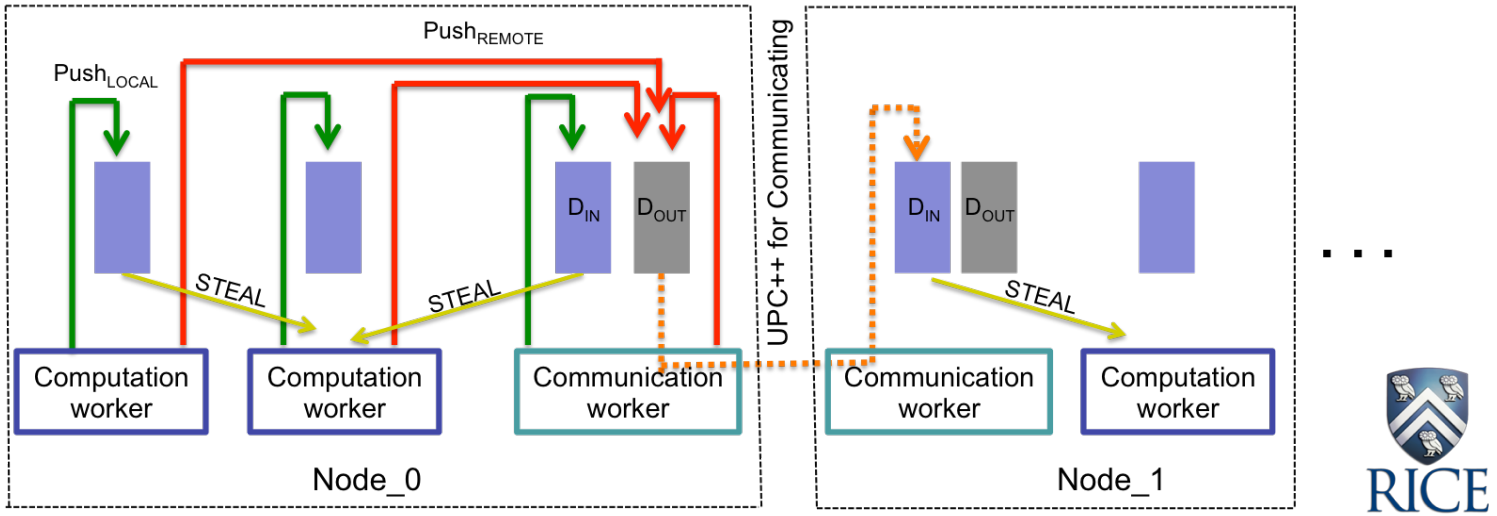


Flat: use only one programming model.
Hierarchical: use one programing model but handle on-node communication through shared-memory.

HABANERO-UPC++

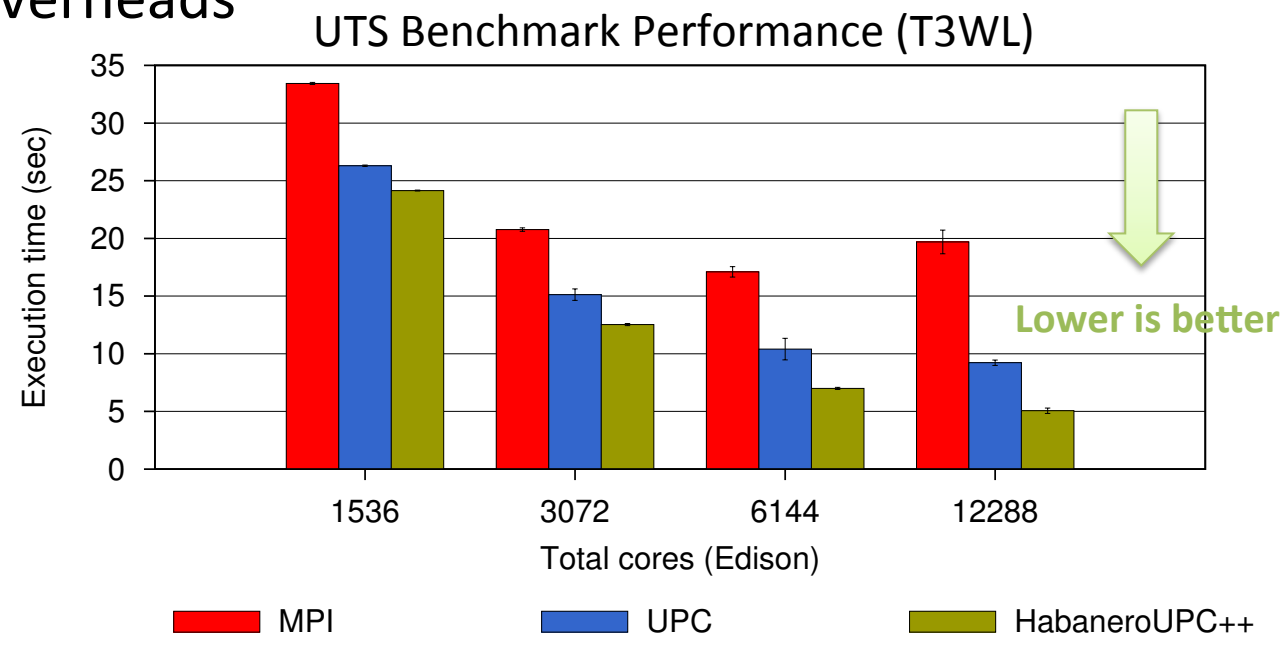
- Represent machine layout as a graph.
- Tasks are each associated with a locale; worker threads have static locale paths along which they search for tasks (generalized load balancing).





DISTRIBUTED LOAD BALANCING

- A simple API to declare a **"locality-free"** task, which can participate in distributed load-balancing
- Habanero-UPC++ runtime uses a novel distributed work-stealing strategy that maximizes balance and minimizes overheads



SUMMARY

- **Communicate more often**
  - use non-blocking one-sided operations
- **Move computation instead of data**
  - use async and event-driven execution
- **Express algorithms with high-level data structures**
  - E.g., use Titanium-style multidimensional arrays
- **Easy on-ramp**
  - interoperate w. existing MPI+OpenMP codes

UPC++: https://bitbucket.org/upcxx/upcxx

OPENFABRICS
ALLIANCE

12th ANNUAL WORKSHOP 2016

THANK YOU

upC++