DEGAS Program System via C++ Library Extension

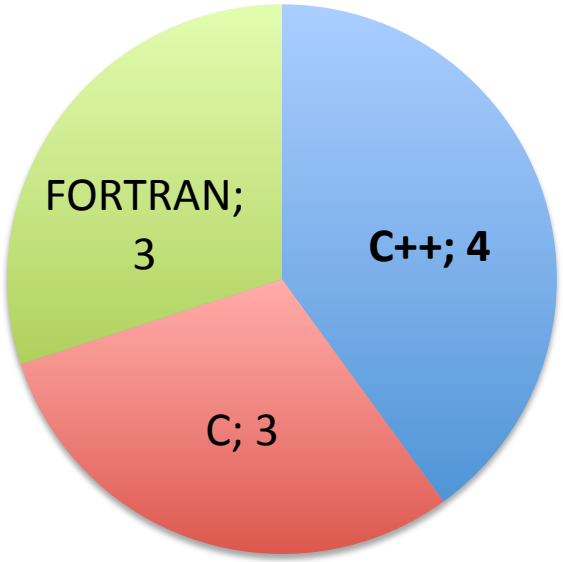
Yili Zheng

Computational Research Division

Lawrence Berkeley National Laboratory

C++ is Increasingly Important

Co-Design Center Proxy Apps



10 Apps: 4 in C++, 3 in C, 3 in FORTRAN (source: Steve Hofmyer)
Other C++ examples: BoxLib, Combinatorial BLAS, …

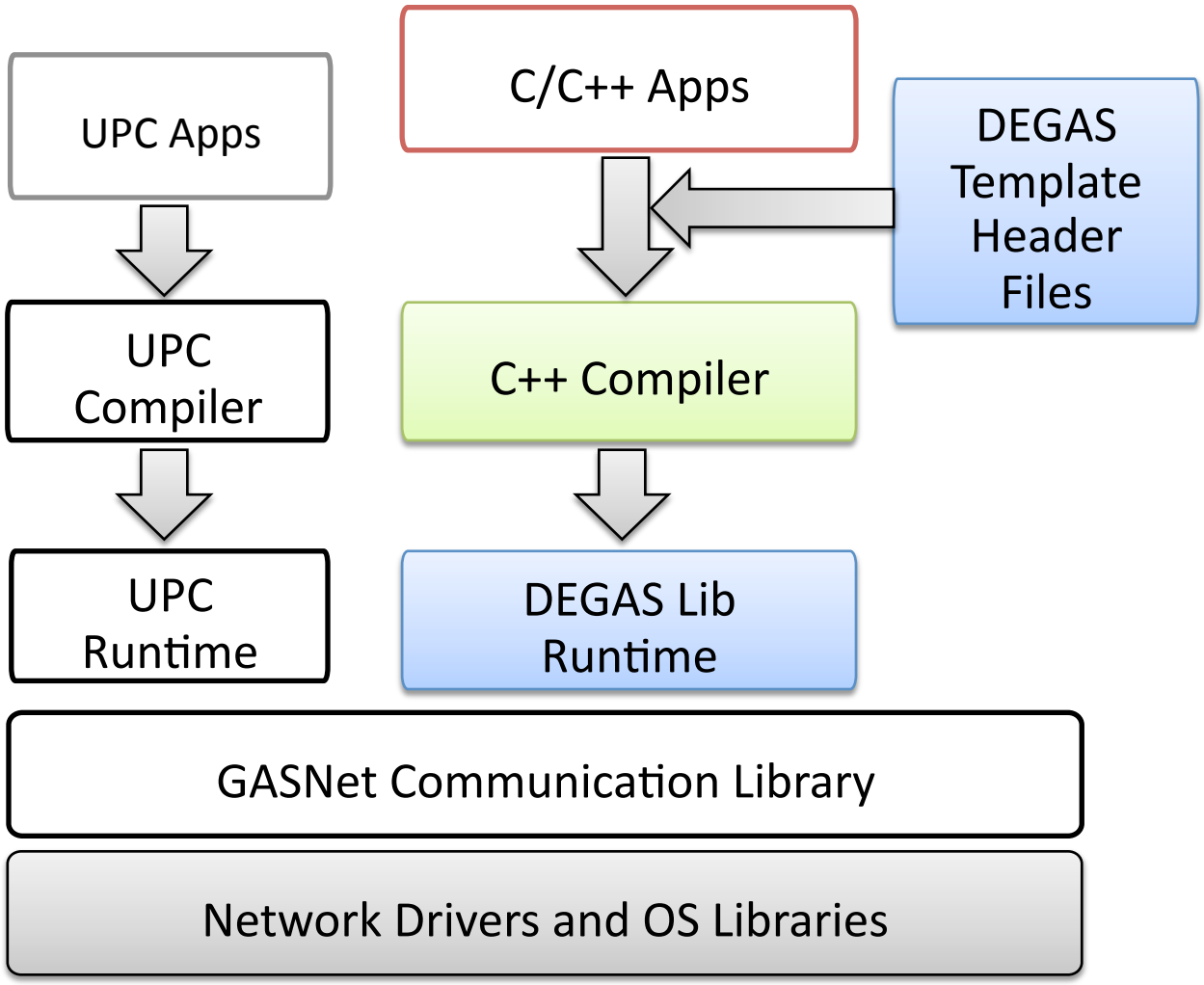
DEGAS Lib supports C/C++ apps,
and can be used by FORTRAN.

A "Compiler-Free" Approach

- Leverage the standardized C++ specifications and their compiler implementations
  - New features in C++ 11 for DEGAS lib

    async, future, lambda functions
- C++ 11 is well supported

DEGAS Lib Software Stack



Design Goals

- PGAS
  - global address space memory management
  - one-sided communication
  - statically shared variables and arrays
  - synchronization primitives
- Dynamic
  - asynchronous task execution
- Hierarchical PGAS

C++ Templates for PGAS Shared Data

- C++ templates enable generic programming
  - Declaration

```
template<class T>
 struct matrix { T *elements; };
```

  - Instantiation

```
matrix<double> A;  matrix<complex> B;
```

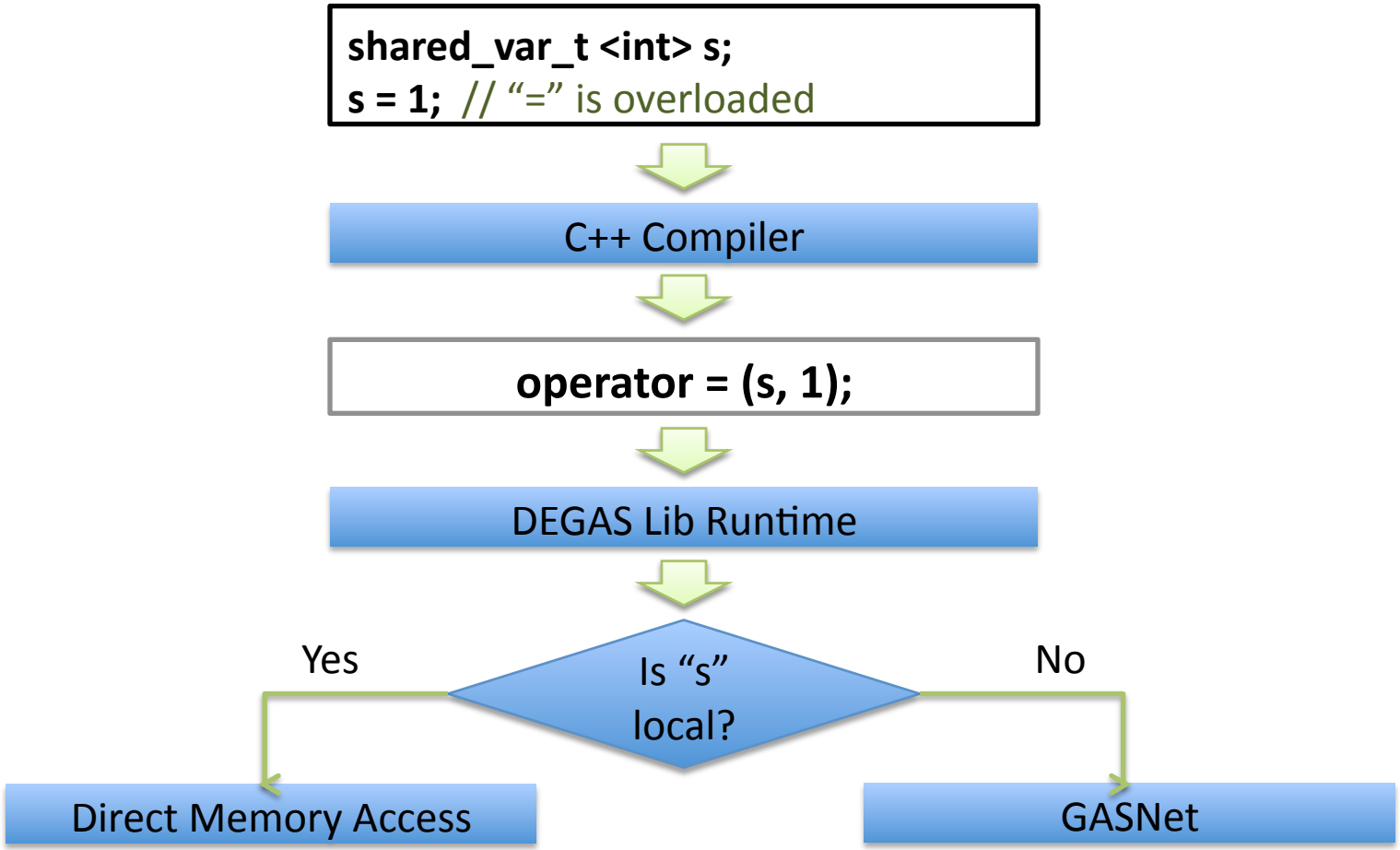
- In DEGAS we use templates to describe shared data

```
shared_var_t<int> s;  // = shared int s; in UPC
shared_array_t<int> sa(8); // = shared int sa[8]; in UPC
```

Shared Variable Example

```
#include <degas.h>
using namespace degas;

shared_var_t<int> s = 0; // shared by all processes
global_lock_t l;
// SPMD
void update() {
  for (int i=0; i<100; i++)
    s = s + 1; // a race condition
  barrier();
  // shared data accesses with lock protection
  for (int i=0; i<100; i++)
    { l.lock(); s = s + 1; l.unlock(); }
}
```

Translation Flow



Dynamic Shared Data

- Global address space pointers

  global_ptr_t<data_type, place_type> p;

- Dynamic global memory allocation

  global_ptr_t **allocate**(place_type place, size_t count);

- Data transfer functions (memory copy)

  copy(global_ptr_t src, global_ptr_t dst, size_t count);

Allocate and *copy* are function templates. C++ compliers can instantiate the right version of *allocate* and *copy* based on the types of the pointers (meta-programming).

Advanced Communication Features

- Non-blocking one-sided communication
  - *async_copy()*
  - *async_copy_fence()*
- MPI-style collective operations built directly on GASNet
  - *broadcast, gather, scatter, reduce, all-to-all*, …
- Can *put* to and *get* from any memory locations
  - Leverage GASNet's dynamic memory registration feature

Asynchronous Task Execution

- C++ 11 async library

```
future = std::async(Function &&f, Args&&… args);
future.wait();
rv = future.get();
```

- DEGAS async library

```
future = degas::async(Function &&f, Args&&… args)
degas::async(place)(Function &&f, Args&&… args);
degas::async_after(place, future)(Function &&f,
                                  Args&&… args);
```

Async Task Example

```
#include <degas.h>
// A regular function can be called by remote processes
void print_num(int num)
{
  printf("arg: %d\n", num);
}
// Only the master process executes the main function
int main(int argc, char **argv)
{
  node_range_t group(1, node_count, 2); // odd num. proc.
  async(group)(print_num, 123); // call a function remotely
  degas::wait(); // wait for the remote tasks to complete
  return 0;
}
```

Async with Lambda Function

```
// Process 0 spawns async tasks
for (int i = 0; i < node_count(); i++) {
  // spawn a task at place “i”
  // the task is expressed by a lambda function
  async(i)([] (int num) { printf("num: %d\n”, num); },
           1000 + i); // argument to the lambda function
  degas::wait(); // wait for all tasks to finish
}
```

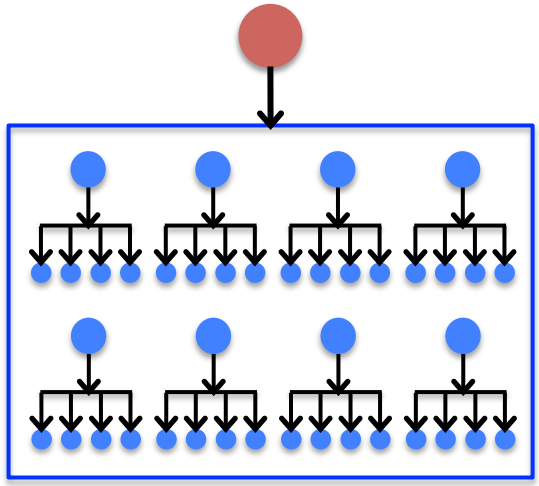
mpirun –n 4 ./test_async

Output:

```
num:  1000
num:  1001
num:  1002
num:  1003
```

DEGAS Async is a Building Block

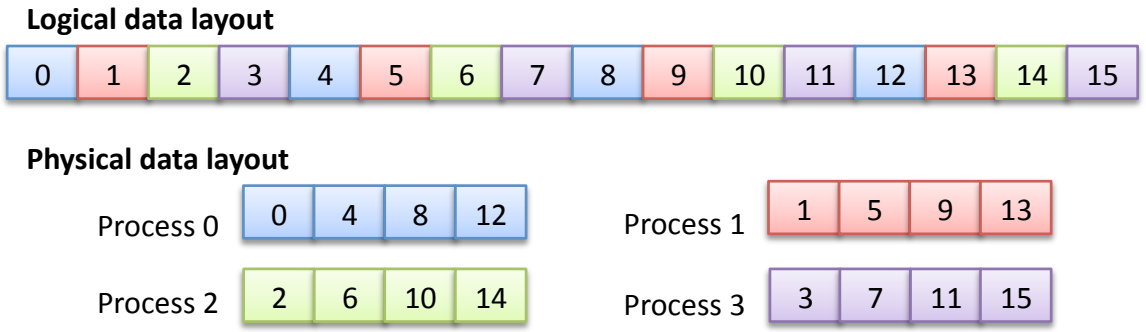
- One-sided collective communication
- Remote atomic operations
- Active Message Broadcast



Random Access Benchmark (GUPS)

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

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

Main update loop



Experimental System: MIC

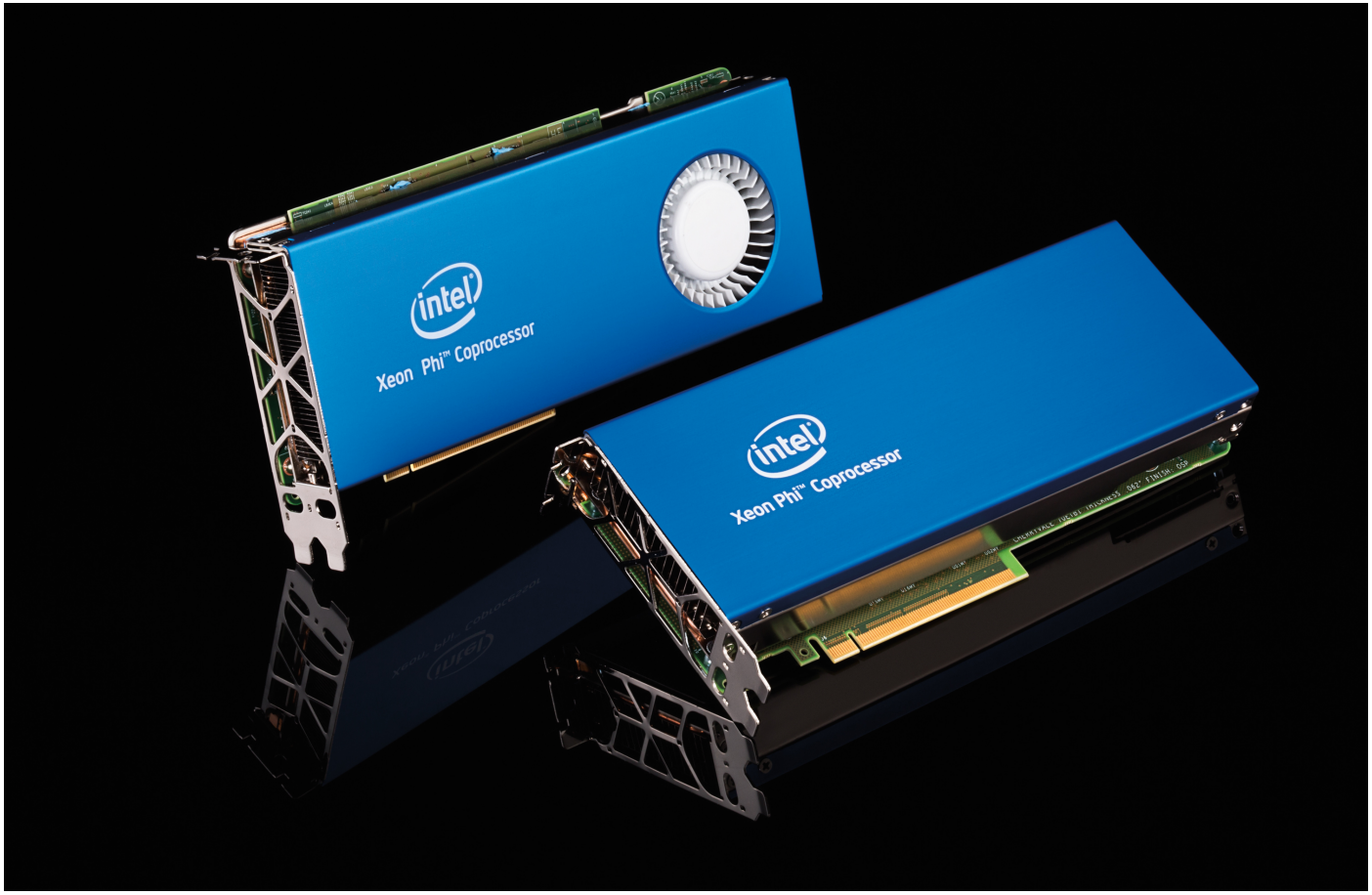
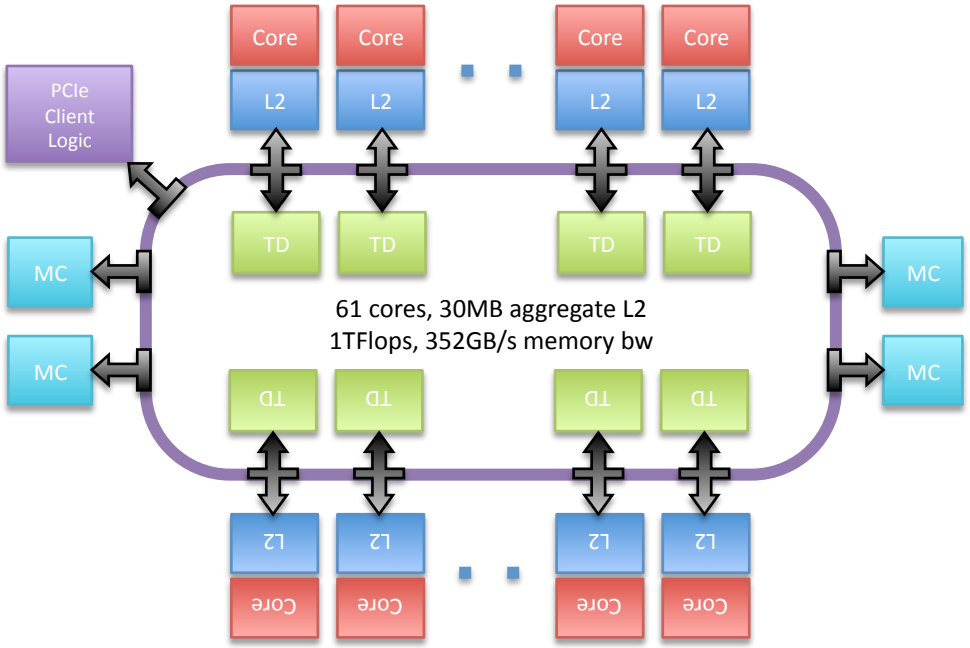
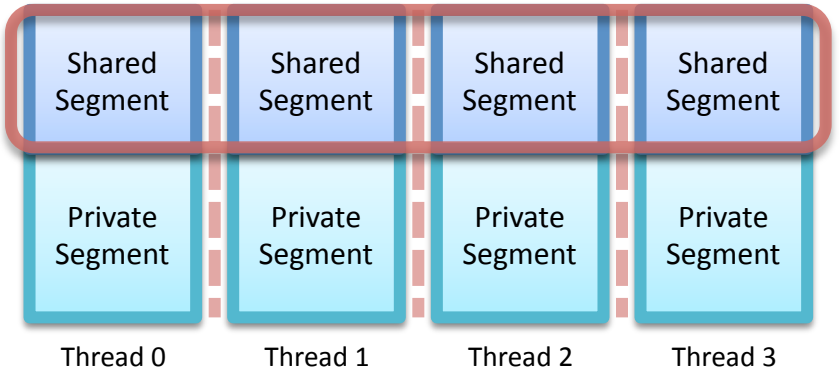


Photo Source: http://newsroom.intel.com/community/intel_newsroom/blog/2012/11/12/

Manycore - A Good Fit for PGAS

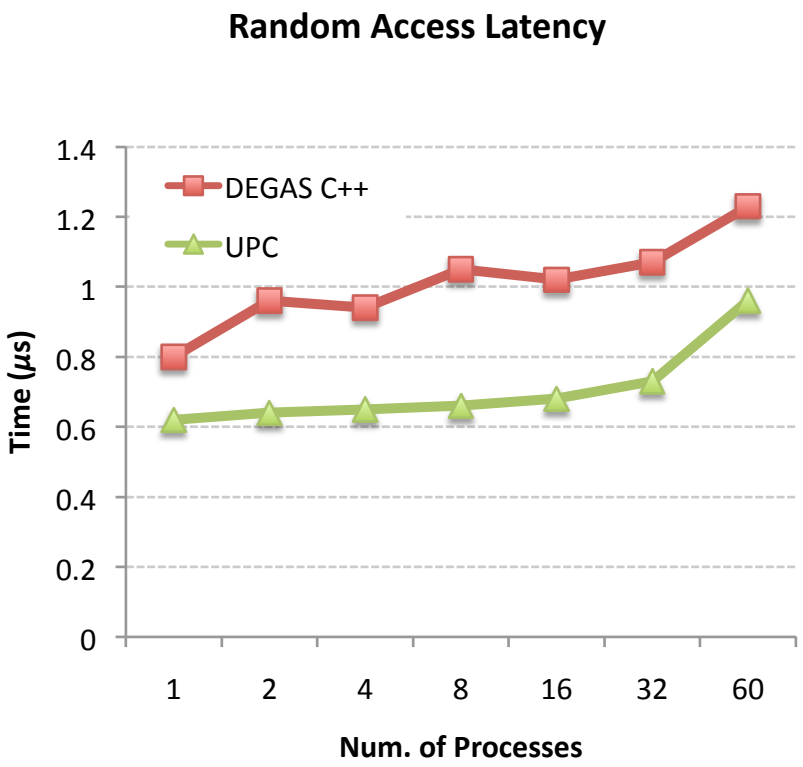


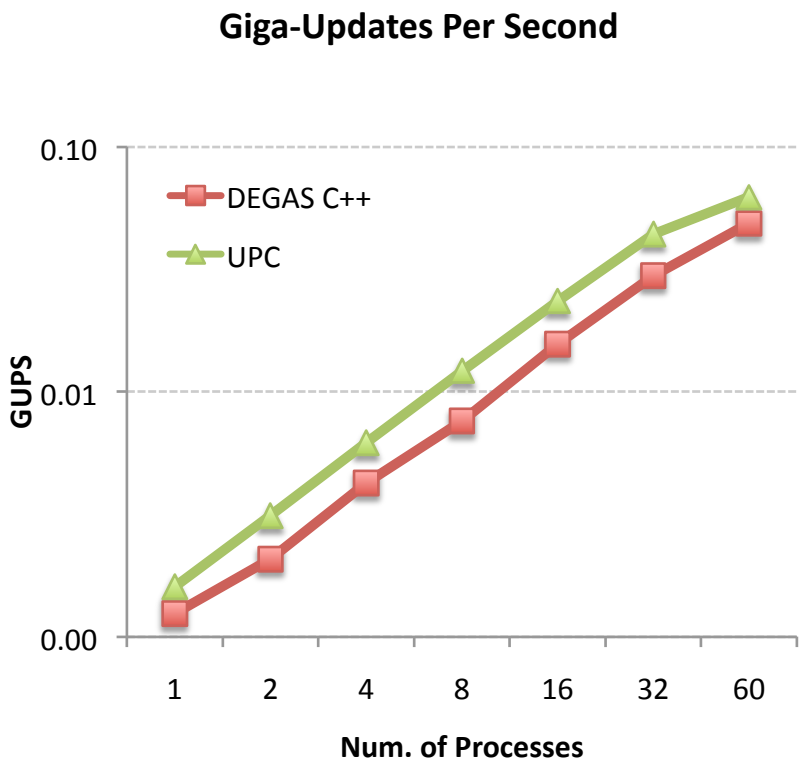
Block Diagram of the Knights Corner (MIC) micro-architecture



PGAS Abstraction

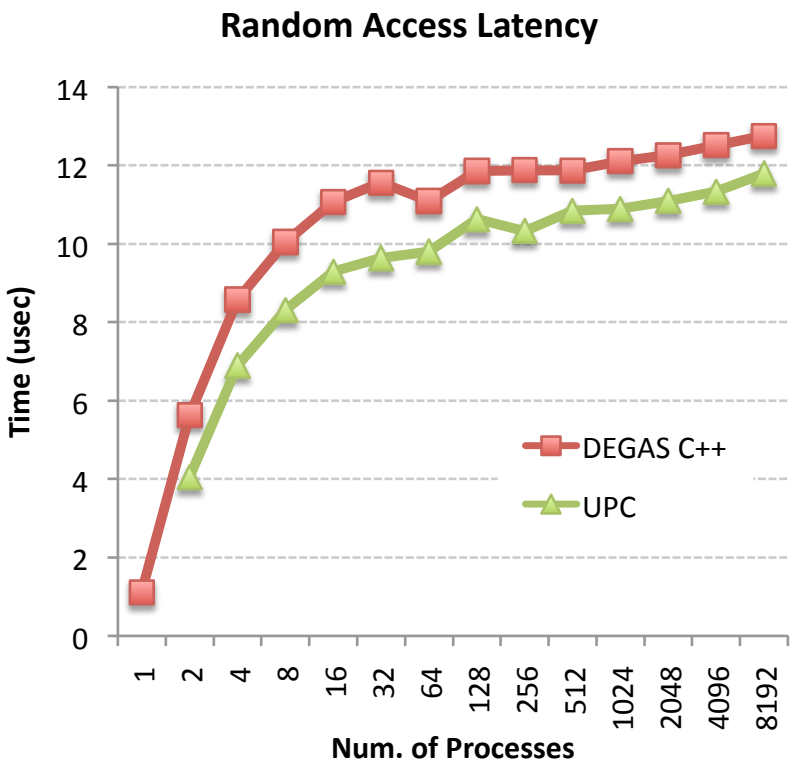
GUPS Performance on MIC

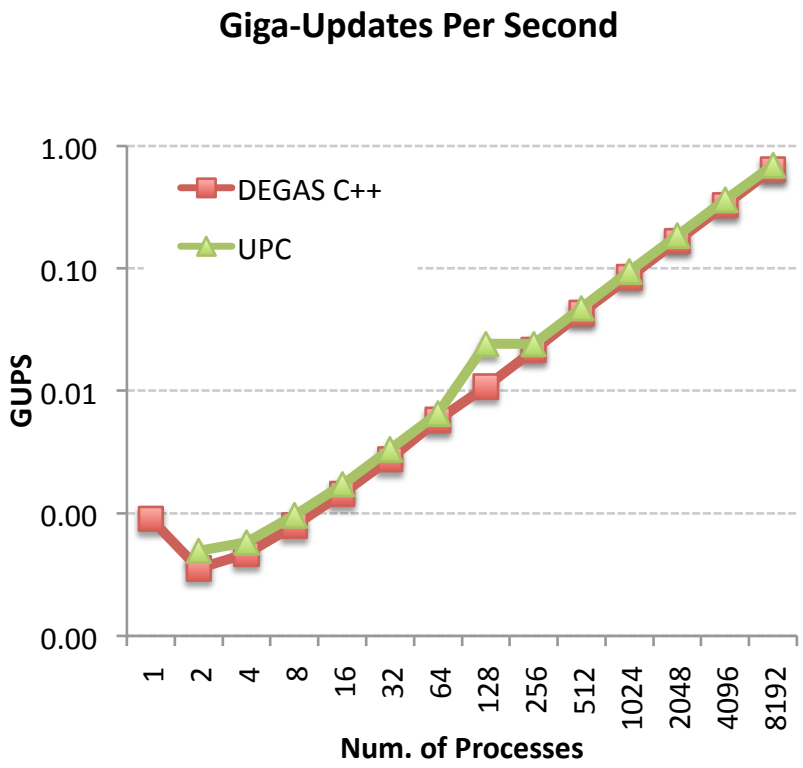




Overhead is only about 0.2 μs (~220 cycles).

GUPS Performance on BlueGene/Q

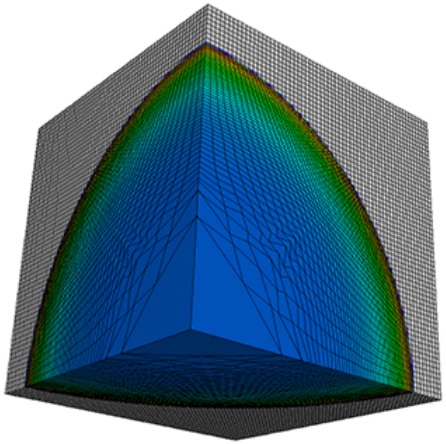




Overhead is negligible at large scale.

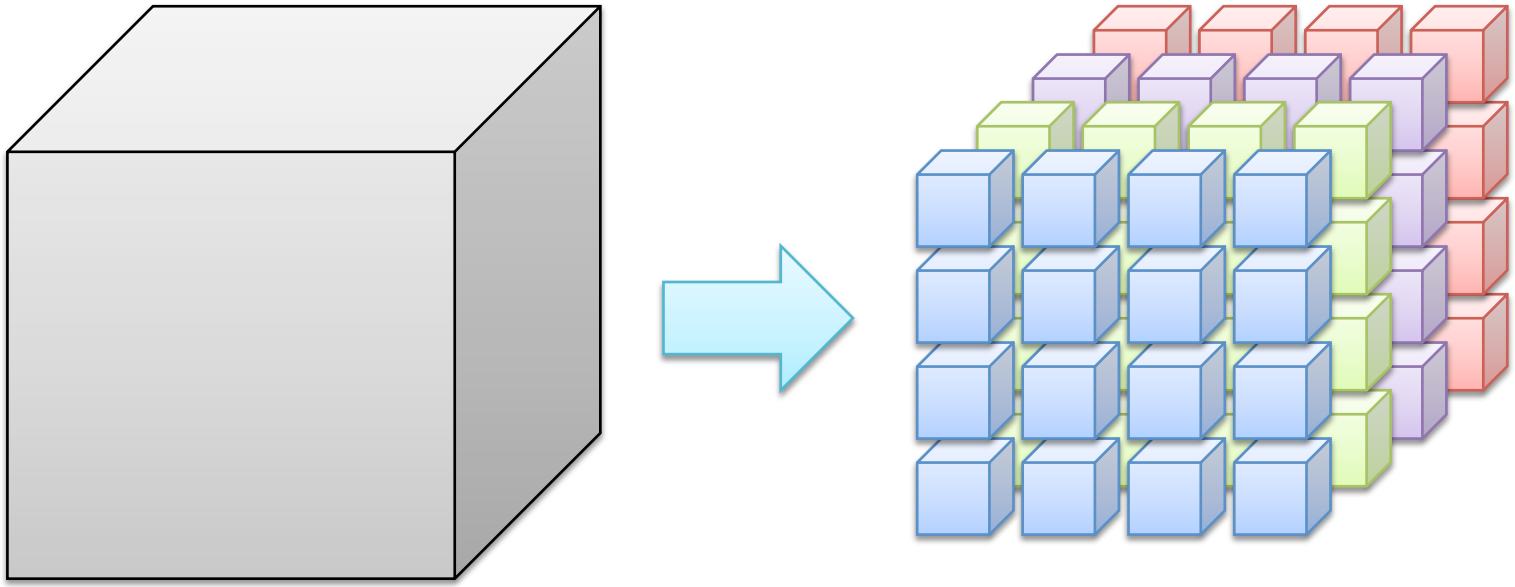
LULESH Proxy App

- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics
- Proxy App for UHPC, ExMatEx, and LLNL ASC
- Written in C++ with MPI, OpenMP, and CUDA versions

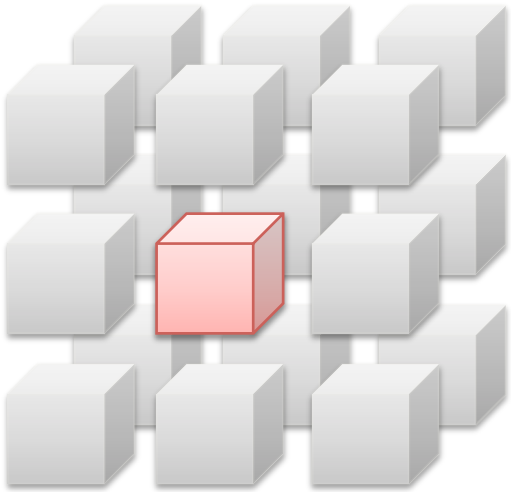


https://codesign.llnl.gov/lulesh.php

LULESH 3-D Data Partitioning



LULESH Communication Pattern

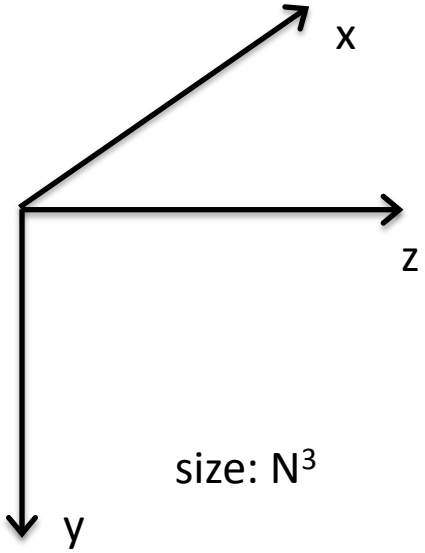


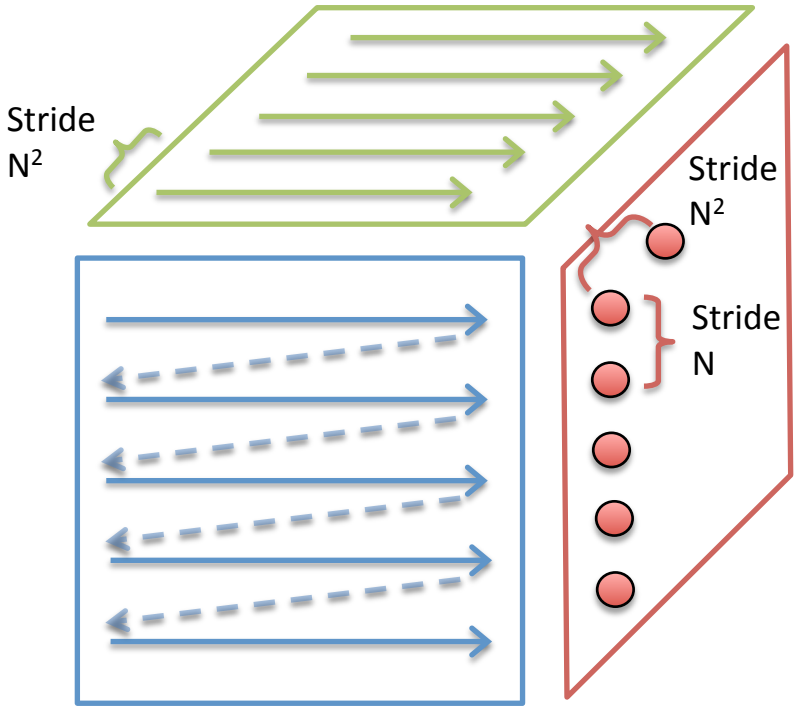
Cross-section view of
the 3-D processor grid

26 neighbors

- 6 faces
- 12 edges
- 8 corners

Data Layout of Each Partition





- 3-D array A[x][y][z]
- row-major storage
- z index goes the fastest

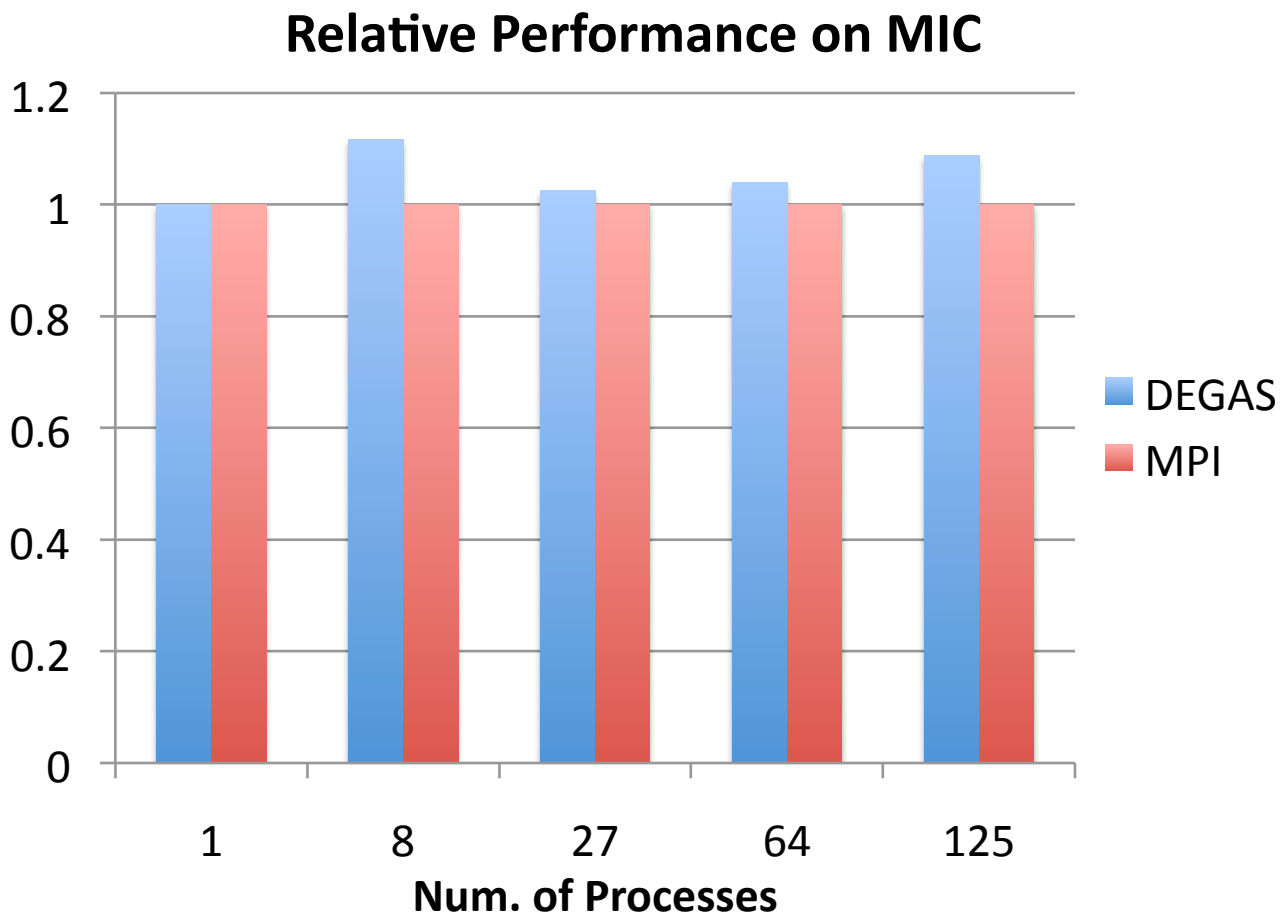
- Blue planes are contiguous
- Green planes are stride-N^2 chunks
- Red planes are stride-N elements

Convert MPI to DEGAS Lib

```
// Post Non-blocking Recv
MPI_Irecv(RecvBuf_1);
…
MPI_Irecv(RecvBuf_N);

// Post Non-bocking Send
Pack_Data_to_Buf();
MPI_Isend(SendBuf_1);
…
MPI_Isend(SendBuf_N);

MPI_Wait(…);
…
Unpack_Data(…);
```

```
Pack_Data_to_Buf();

// Post Non-blocking Copy
async_copy(SendBuf_1, RecvBuf_1);
…
async_copy(SendBuf_N, RecvBuf_N);

async_copy_fence(…);
…
Unpack_Data(…);
```

LULESH Performance on MIC



Porting LULESH to DEGAS is "Easy"

- DEGAS lib and LULESH speak the same language – C++
- LULESH is well modularized – easy to find out the communication components
- Non-blocking one-sided data copy functions are handy and efficient – leverage hardware RDMA and shared-memory for PGAS
- MPI-style collectives are *necessary*

Concluding Remarks

- DEGAS Lib: a Minimally Invasive Technique for bringing DEGAS functionality to existing apps
- Many-core architectures with hardware shared-memory (no cache coherence is OK) will be ideal for PGAS, HPGAS, DEGAS
- High-level abstractions in the base language are crucial to productivity


See You Tonight for PyGAS

Thank You!