Introduction to Chapel, UPC++, and Charm++

Michelle Mills Strout

Hewlett Packard Enterprise

PAW-ATM

SC22 Dallas, TX | hpc accelerates.

Intro to Chapel, UPC++, Charm++

- Chapel programming language
  - Many styles of parallelism in a global namespace
  - Usage: `chpl myProg.chpl; ./myProg -nl 4`
- UPC++
  - PGAS provided by a C++ library
  - Usage:

```
upcxx -g myProg.cpp -o myProg
upcxx-run -n 4 ./myProg
```

- Charm++
  - Migratable objects that communicate
  - Usage:

```
charmc myProg.ci
charmc myProg.cpp -o myProg
./charmrun +p4 ./myProg
```

PAW ATM

Parallel Applications Workshop
Alternatives to MPI+X

Monday, November 14th, 2022

SC22 Dallas, TX | hpc accelerates.

CHAPEL PROGRAMMING LANGUAGE

Chapel is a general-purpose programming language that provides

ease of parallel programming,
high performance, and
portability.

And is being used in applications in various ways:

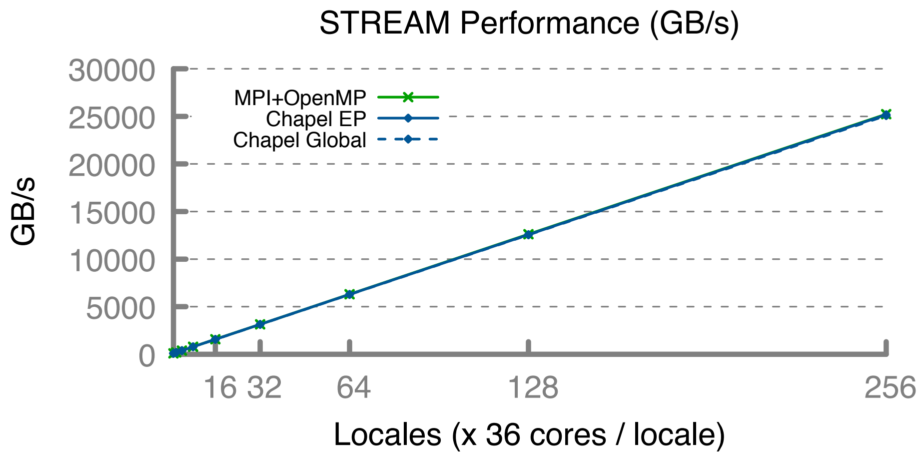
refactoring existing codes,
developing new codes,
serving high performance to Python codes **(Chapel server with Python client),** and
providing distributed and shared memory parallelism for existing codes.

EASE OF PROGRAMMING AND HIGH PERFORMANCE

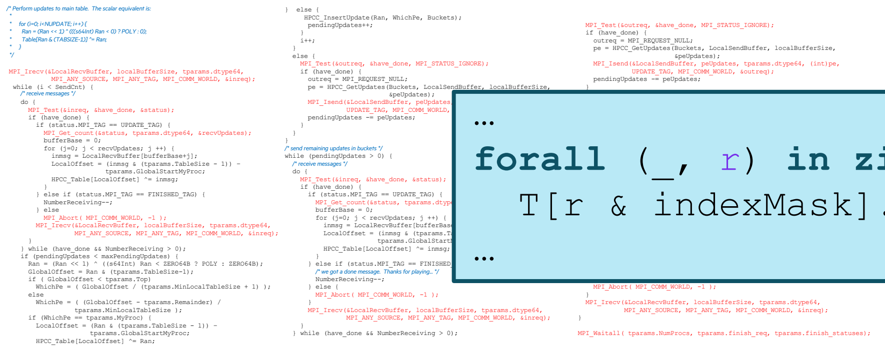
STREAM TRIAD: C + MPI + OPENMP

```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

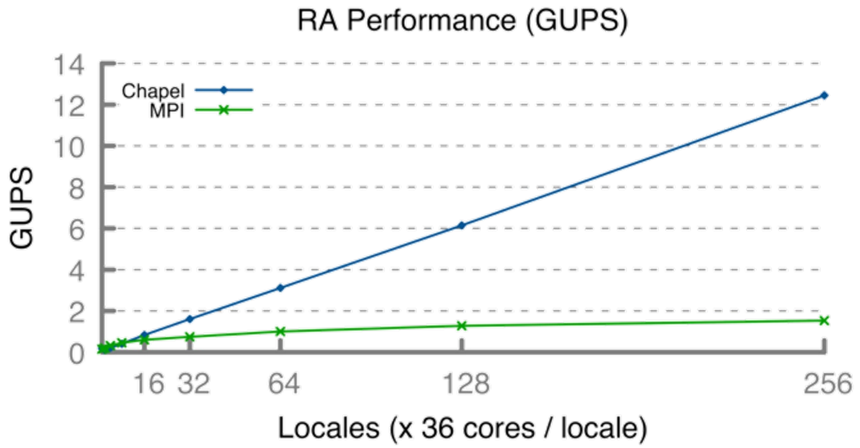
```chapel
use BlockDist;

config const m = 1000,
             alpha = 3.0;

const Dom = {1..m} dmapped …;
var A, B, C: [Dom] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```



HPCC RA: MPI KERNEL



```chapel
…
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
…
```



PORTABILITY

- **On a laptop, cluster, or supercomputer (Shared-memory parallelism)**

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 3 of 4 on n1032
Hello from task 2 of 4 on n1032
```

- **On a cluster or supercomputer (Distributed-memory parallelism)**

```
prompt> chpl helloTaskPar.chpl
prompt> ./helloTaskPar --numLocales=4
Hello from task 1 of 4 on n1032
Hello from task 4 of 4 on n1032
Hello from task 1 of 4 on n1034
Hello from task 2 of 4 on n1032
Hello from task 1 of 4 on n1033
Hello from task 3 of 4 on n1034
Hello from task 1 of 4 on n1035
…
```

EXAMPLE CODE: ANALYZING MULTIPLE FILES USING PARALLELISM

word-count.chpl

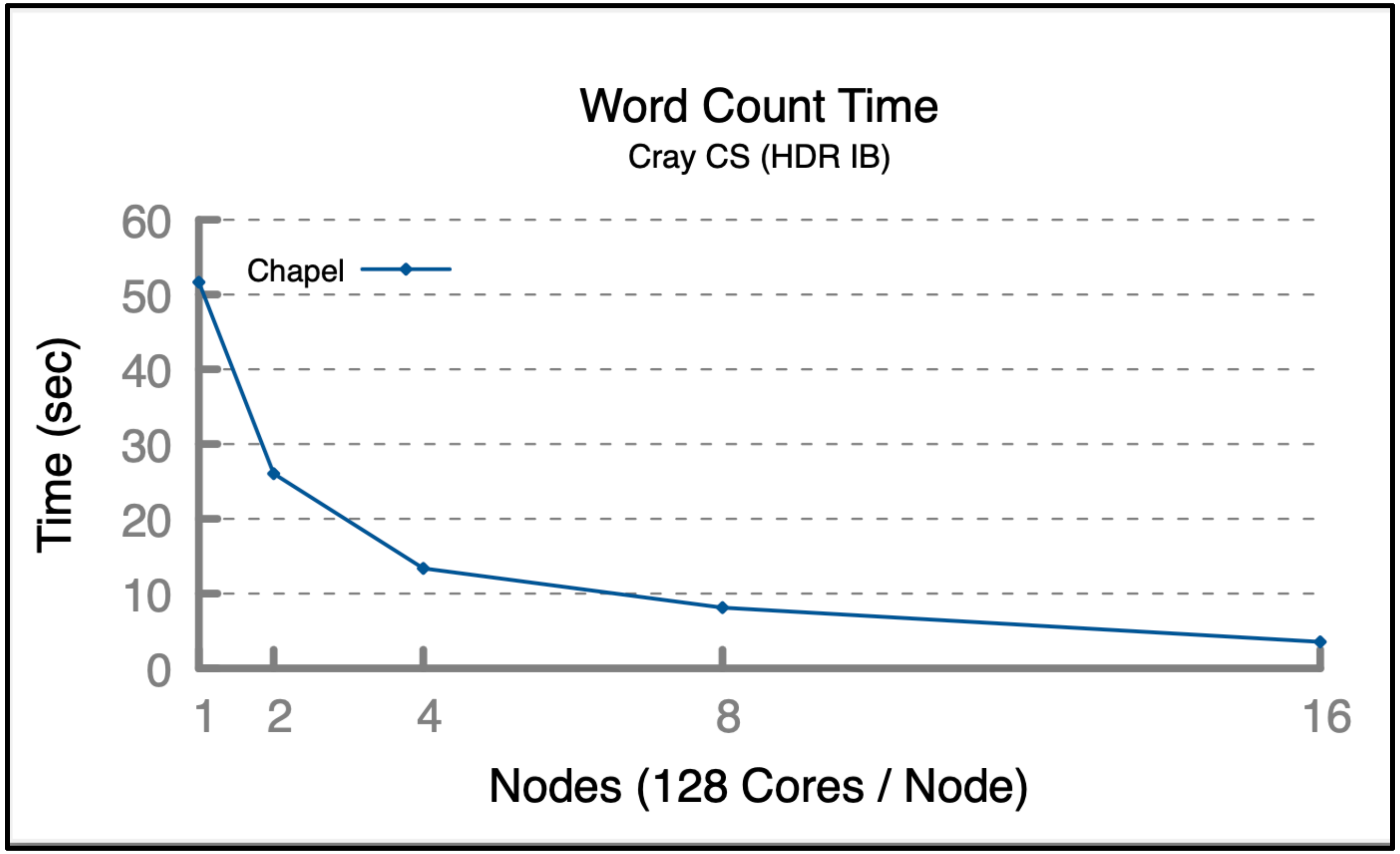
```
use FileSystem;
config const dir = "DataDir";
var fList = findfiles(dir);
var filenames
  = newBlockArr(0..<fList.size,string);
filenames = fList;

// per file word count
forall f in filenames {
  ...
  while reader.readline(line) {
    for word in line.split(" ") {
      wordCount[word] += 1;
    }
  }
 ...
}
```

```
prompt> chpl --fast word-count.chpl
prompt> ./word-count
prompt> ./word-count -nl 4
```

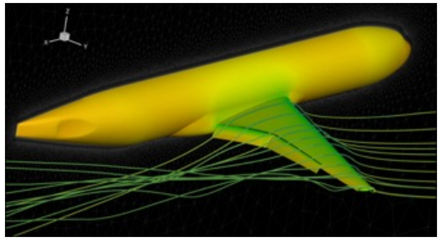
Shared and Distributed-Memory Parallelism using forall, a distributed array, and command line options to indicate number of locales

SCALING FROM LAPTOP TO SUPERCOMPUTER

- **Data Analysis Example**
  - Per file word count on all the files in a directory
  - Serial to threaded and distributed by using a forall over a parallel distributed array
  - Good scaling even for file I/O (below is for 10K files at 3MB each)



HOW APPLICATIONS ARE USING CHAPEL

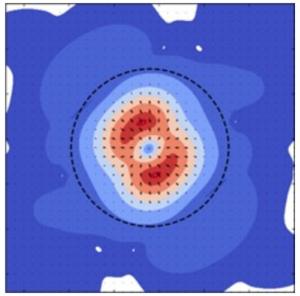


Refactoring existing codes into Chapel (~100K lines of Chapel)

CHAMPS: 3D Unstructured CFD

Éric Laurendeau, Simon Bourgault-Côté, Matthieu Parenteau, et al.

École Polytechnique Montréal



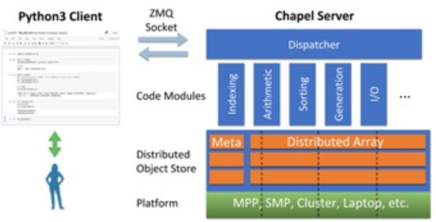
Writing code in Chapel

(~10k lines of including parallel FFT)

ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, *et al.*

Yale University / University of Auckland



Chapel server for a Python client (~25K lines of Chapel)

Arkouda: NumPy at Massive Scale

Mike Merrill, Bill Reus, et al.

US DoD

In Memoriam:

Mike Merrill passed last week, and he will be greatly missed.

CHAPEL ROADMAP

- **Generate code for GPUs**
  - Nascent support for NVIDIA
  - Exploring AMD and Intel support
- **Rearchitect the compiler**
  - Reduce compile times
    - potentially via separate compilation / incremental recompilation
  - Support interpreted / interactive Chapel programming
- **Continue to optimize performance**
- **Release Chapel 2.0**
  - guarantee backwards-compatibility for core language and library
- **Foster a growing Chapel community**

```
// Stream
// Variables stored on GPU
// vector operations executed on GPU
config var n = 1_000_000, alpha = 0.01;

coforall loc in Locales on loc {
  coforall gpu in loc.gpus do on gpu {
    var A, B, C: [1..n] real;

    A = ...;
    B = ...;
    C = ...;

    A = B + alpha * C;
  }
}
```

CHAPEL RESOURCES

Chapel homepage: https://chapel-lang.org

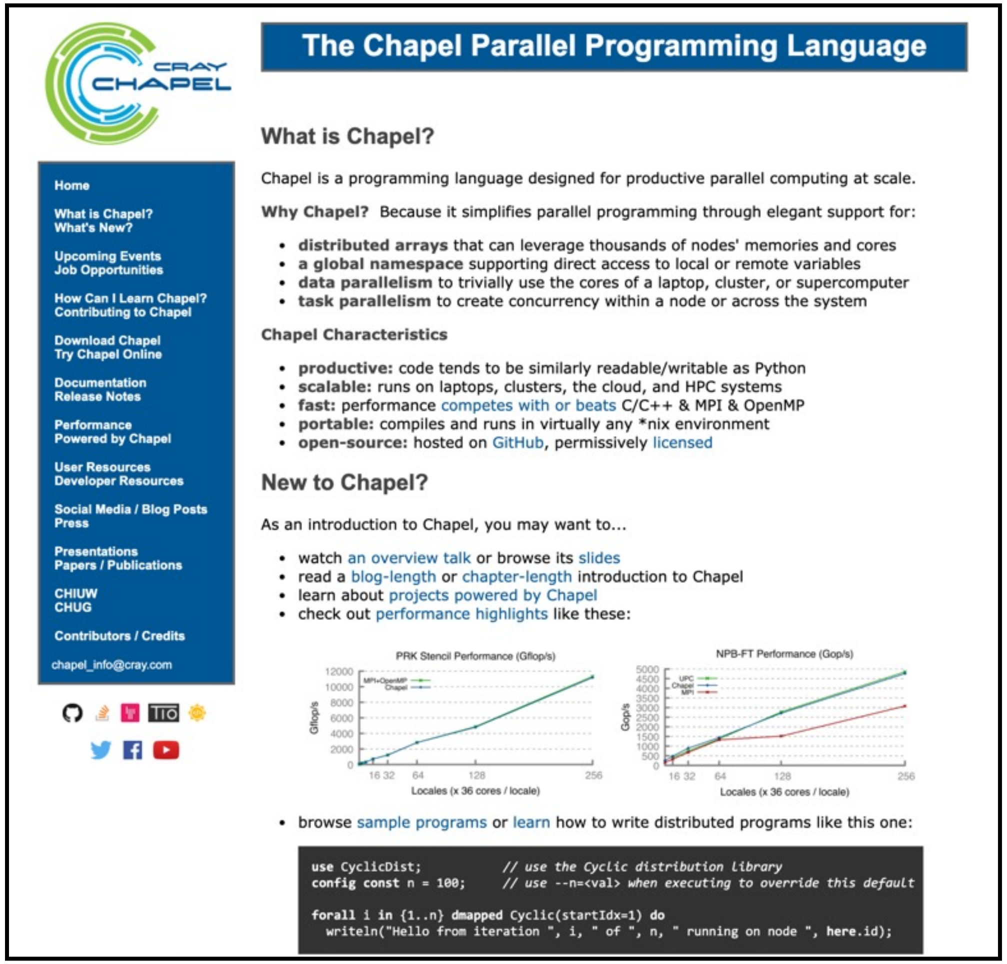
- (points to all other resources)

Social Media:

- Twitter: @ChapelLanguage
- Facebook: @ChapelLanguage
- YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

Community Discussion / Support:

- Discourse: https://chapel.discourse.group/
- Gitter: https://gitter.im/chapel-lang/chapel
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues



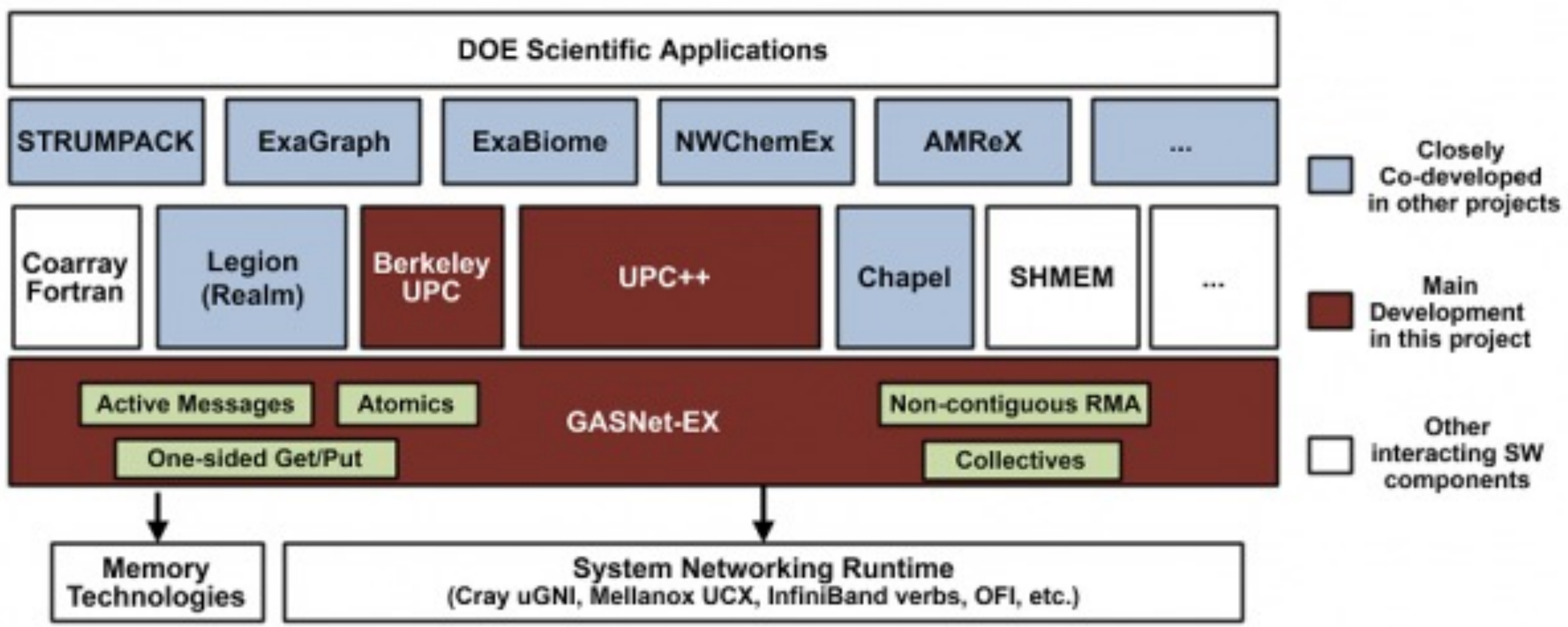
UPC++ and the Pagoda project

Slides courtesy of Dan Bonachea

https://go.lbl.gov/pagoda

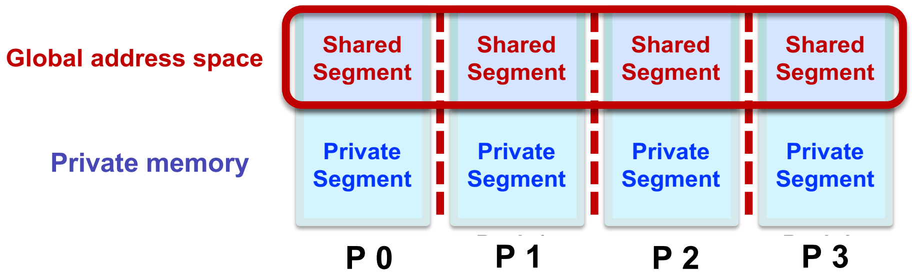
Support for lightweight communication for exascale applications, frameworks and runtimes

- **GASNet-EX** low-level layer that provides a network-independent interface suitable for Partitioned Global Address Space (PGAS) runtime developers
- **UPC++** C++ PGAS library for application, framework and library developers, a productivity layer over GASNet-EX



BERKELEY LAB

The PGAS model



Partitioned **G**lobal **A**ddress **S**pace

- Provide an abstraction of a shared memory, partitioned by locality
- One-sided RMA communication: separate synchronization from data movement
- RMA semantics leverage the network's RDMA hardware capabilities

Languages that provide PGAS:

UPC, Titanium, Chapel, X10, Fortran 2008+

Libraries that provide PGAS:

UPC++, OpenSHMEM, Co-Array C++, Global Arrays, DASH, GASPI, MPI-RMA

These slides are about UPC++

- C++ library implementation of the PGAS model
- Leverages productivity of C++
- Adds Remote Procedure Call (RPC) to complement RMA
- Extends global address space to encompass device memories (GPUs)

How does UPC++ deliver the PGAS model?

UPC++ uses a “Compiler-Free,” library approach

- UPC++ leverages C++ standards,
  needs only a standard C++ compiler

Relies on GASNet-EX for low-overhead communication

- Efficiently utilizes network hardware, including RDMA
- Provides Active Messages on which UPC++ RPCs are built
- Enables portability (laptops to supercomputers)

Designed for interoperability

- Same SPMD process model as MPI, enabling hybrid applications
- Node-level models can optionally be mixed with UPC++ as in MPI+X
  - OpenMP, C++ threads, Kokkos, CUDA, …

What does UPC++ offer?

Communication operations include:

- **Remote Memory Access (RMA)**:
  - Get/put/atomics on a remote location in another address space
  - One-sided communication leverages low-overhead, zero-copy RDMA
- **Remote Procedure Call (RPC)**:
  - Moves computation to the data

Design principles for performance and scalability

- All communication is syntactically explicit
- All communication is asynchronous: futures and promises
- Scalable data structures that avoid unnecessary replication

Easy Distributed Hash Tables using UPC++ RPC

- RPC simplifies distributed data structures
- Simple, working example:
  - Asynchronous DHT insertion
  - Leverages C++ STL hash table and provides a distributed analog

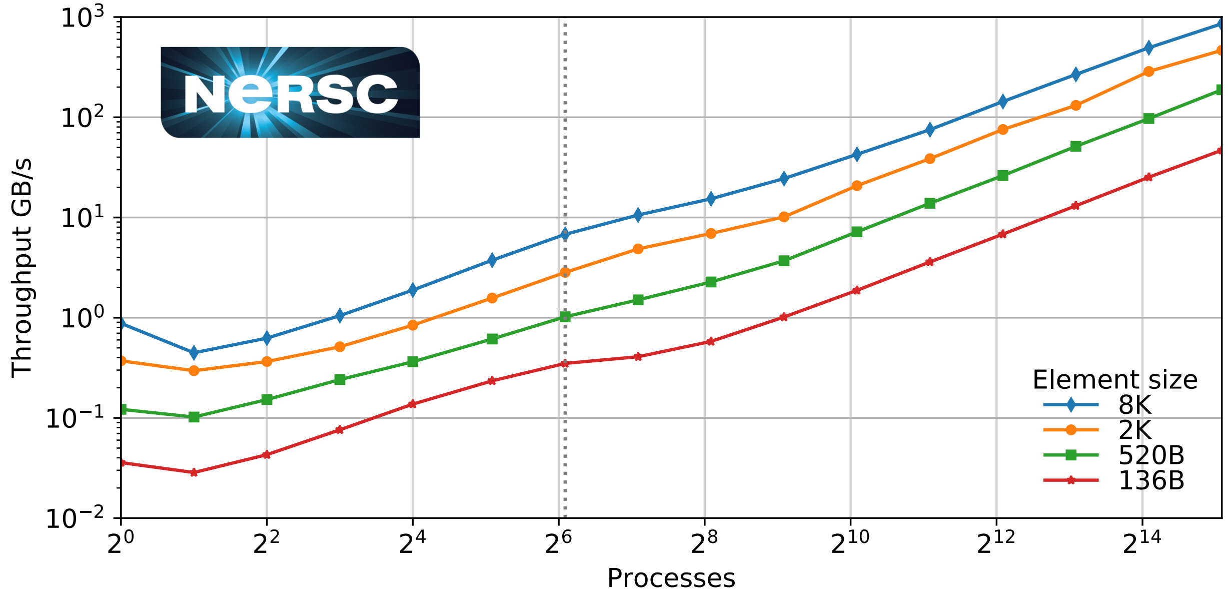
```
// C++ global variables correspond to rank-local state:
std::unordered_map<string, string> local_map;
// insert a key-value pair and return a future:
future<> dht_insert(const string& key, const string& val) {
 return rpc(hash(key) % rank_n(), // asynchronous RPC to owner
          [](const string& key, const string& val) { // lambda invoked by RPC
               local_map[key] = val;  // insert in local map
          });
}
```

High-Performance Distributed Hash Tables w/ RPC + RMA

Asynchronous DHT Insert operation shown below

- RPC inserts the key metadata at the target
- Once the RPC completes, an attached callback issues one-sided RMA Put (rput) to store the value data
- Entire operation is fully asynchronous

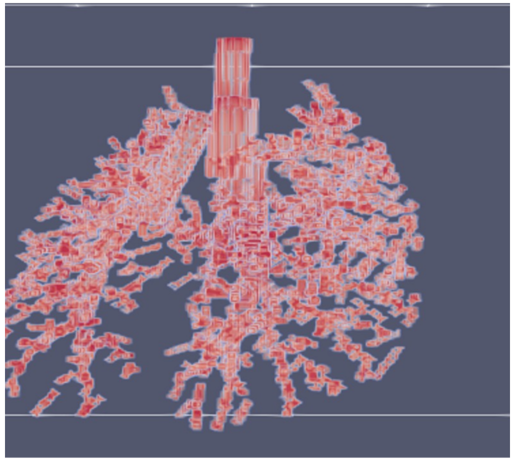
```
// C++ global variables correspond to rank-local state
std::unordered_map<uint64_t, global_ptr<char>> local_map;
// insert a key-value pair and return a future
future<> dht_insert(uint64_t key, char *val, size_t sz) {
  future<global_ptr<char>> fut =
    rpc(key % rank_n(),           // RPC obtains location for the data
            [key,sz]() -> global_ptr<char> { // lambda invoked by RPC
              global_ptr<char> gptr = new_array<char>(sz);
              local_map[key] = gptr;  // insert in local map
              return gptr;
            });
  return fut.then( // callback executes when RPC completes
       [val,sz](global_ptr<char> loc) -> future<> {
           return rput(val, loc, sz); }); // RMA Put the value payload
}
```



Weak scaling on 32k procs @ Cori KNL
see IPDPS'19 [doi:10.25344/S4V88H](https://doi.org/10.25344/S4V88H)

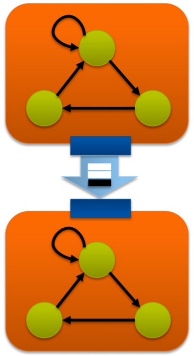
- **RPC** simplifies distributed data-structures
  - Argument passing, remote queue management and progress engine are factored out of the application code
- Asynchronous execution enables overlap

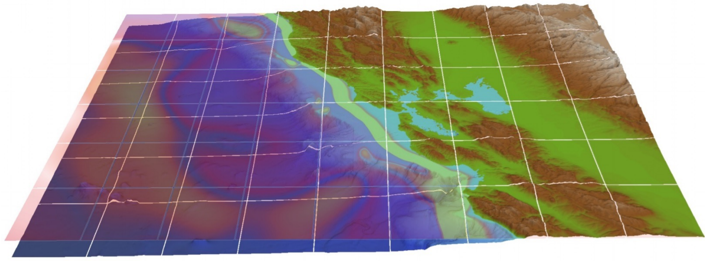
UPC++ Application Examples

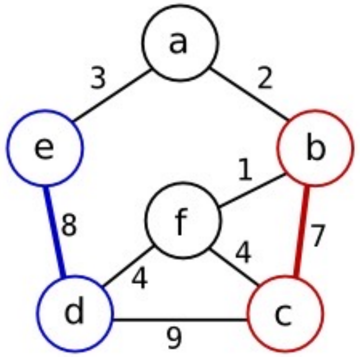


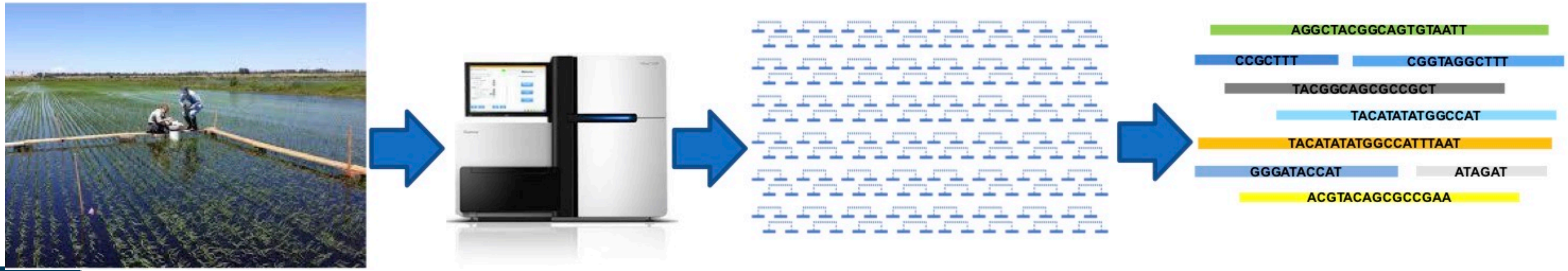
Several applications have been written using UPC++, resulting in improved programmer productivity and runtime performance. Examples include:

- MetaHipMer, a genome assembler
- symPack, a sparse symmetric matrix solver
- Pond, an actor-based tsunami simulation
- SIMCoV, agent-based simulation of lungs with COVID
- Mel-UPX, half-approximate graph matching solver
- TAMM, computational chemistry module in NWChemEX









UPC++ additional resources

Today: GASNET talk at PAW-ATM at 2:18pm

Website: upcxx.lbl.gov includes the following content:

- Open-source/free library implementation
  - Portable from laptops to supercomputers
- Tutorial resources at upcxx.lbl.gov/training
  - UPC++ Programmer's Guide
  - Videos and exercises from past tutorials
- Formal UPC++ specification
  - All the semantic details about all the features
- Links to various UPC++ publications
- Links to optional extensions and partner projects
- Contact information and support forum

"UPC++ has an excellent blend of ease-of-use combined with high performance. Features such as RPCs make it really easy to rapidly prototype applications, and still have decent performance. Other features (such as one-sided RMAs and asynchrony) enable fine-tuning to get really great performance."
-- Steven Hofmeyr, LBNL

"If your code is already written in a one-sided fashion, moving from MPI RMA or SHMEM to UPC++ RMA is quite straightforward and intuitive; it took me about 30 minutes to convert MPI RMA functions in my application to UPC++ RMA, and I am getting similar performance to MPI RMA at scale."
-- Sayan Ghosh, PNNL

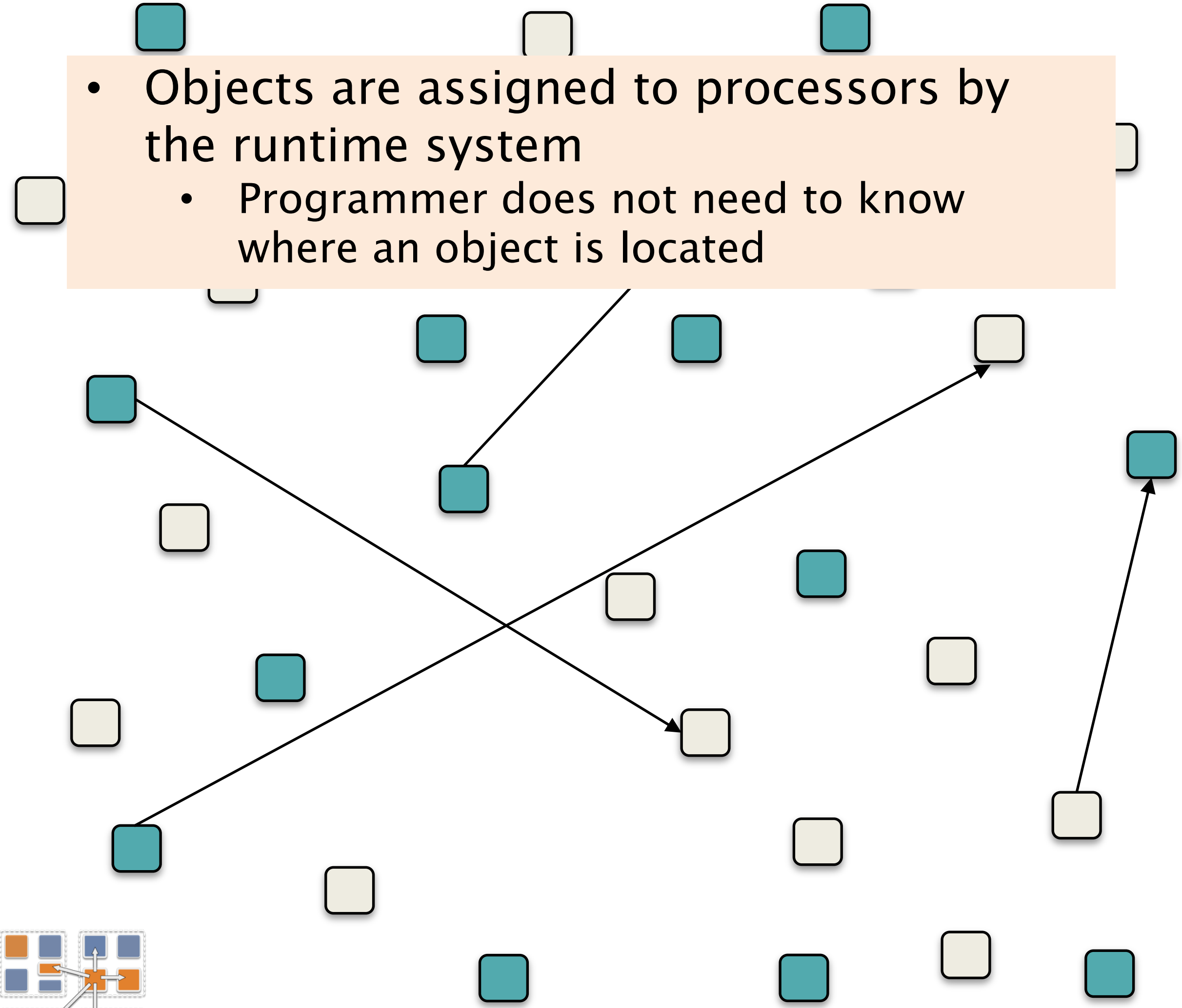
Slides courtesy of Laxmikant (Sanjay) Kale

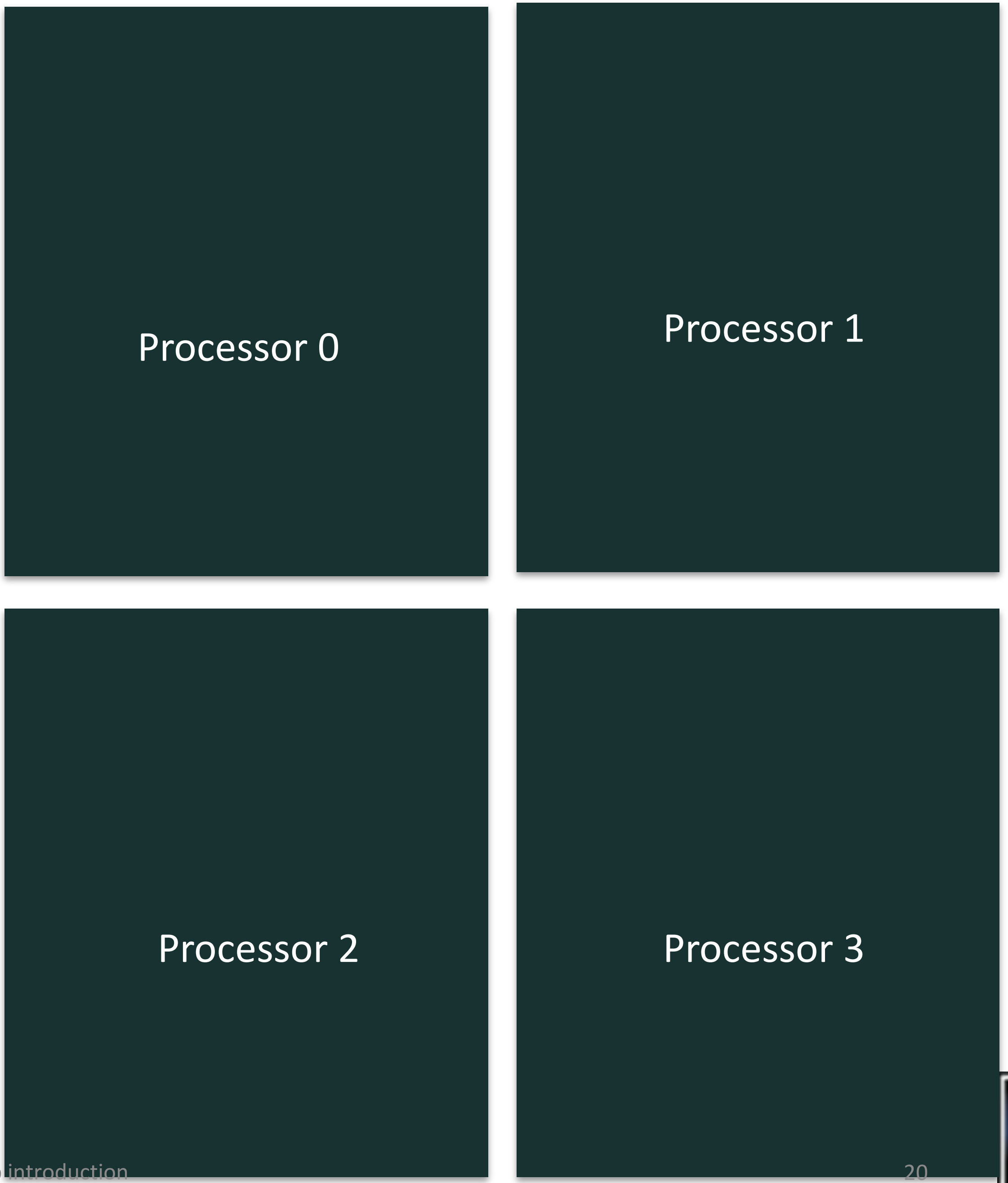
Key Ideas in Charm++

- What to automate:
  - Let the programmer do what they are good at and
  - let the system do (i.e.) automate what it can do well
- More specifically:
  - Let the programmer decide what to do in parallel
  - Express decomposition, interactions
  - Let the system decide where and when
- How: virtualize the notion of a processor
  - So as to automate resource management and associated functionalities
- The migratable objects programming model
  - Charm++ is one of the (first/foundational) programming system within this model

- A Charm++ computation consists of multiple collections of globally visible objects
- Each collection is individually indexed

- Objects are assigned to processors by the runtime system
  - Programmer does not need to know where an object is located



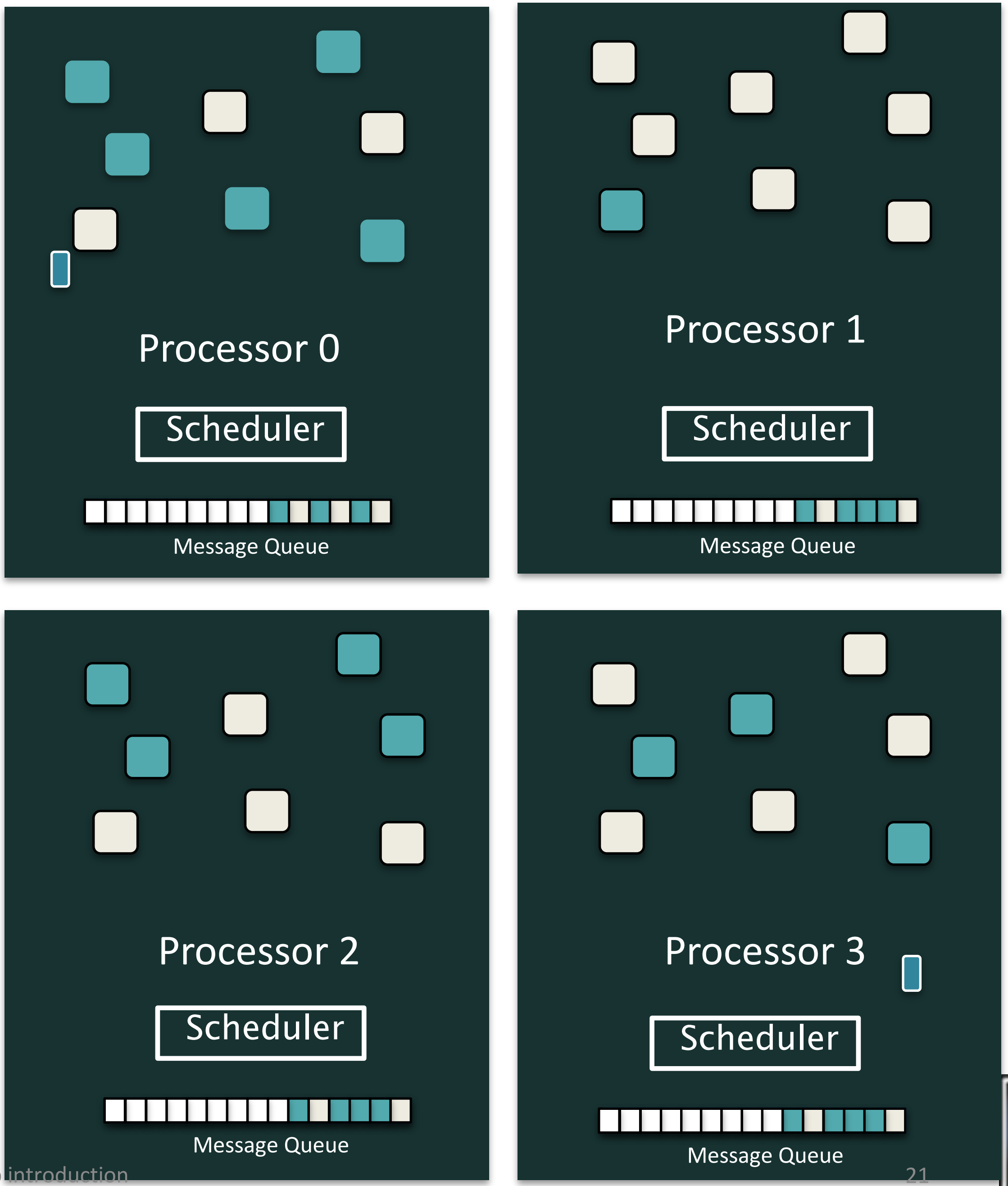


- A Charm++ computation consists of multiple collections of globally visible objects
- Each collection is individually indexed

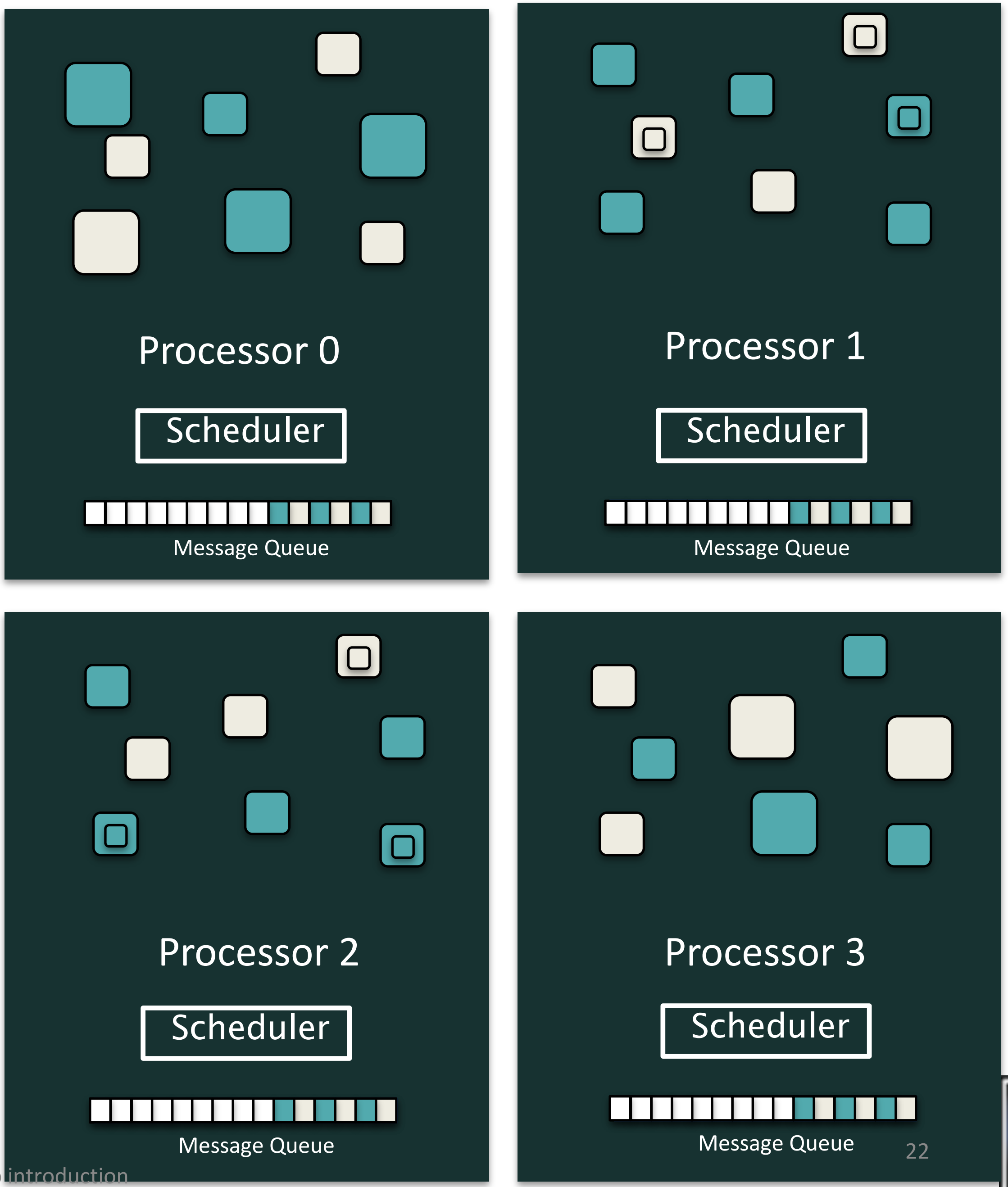
- Objects are assigned to processors by the runtime system
  - Programmer does not need to know where an object is located

- Scheduling on each processors is under the control of a user-space message-driven scheduler

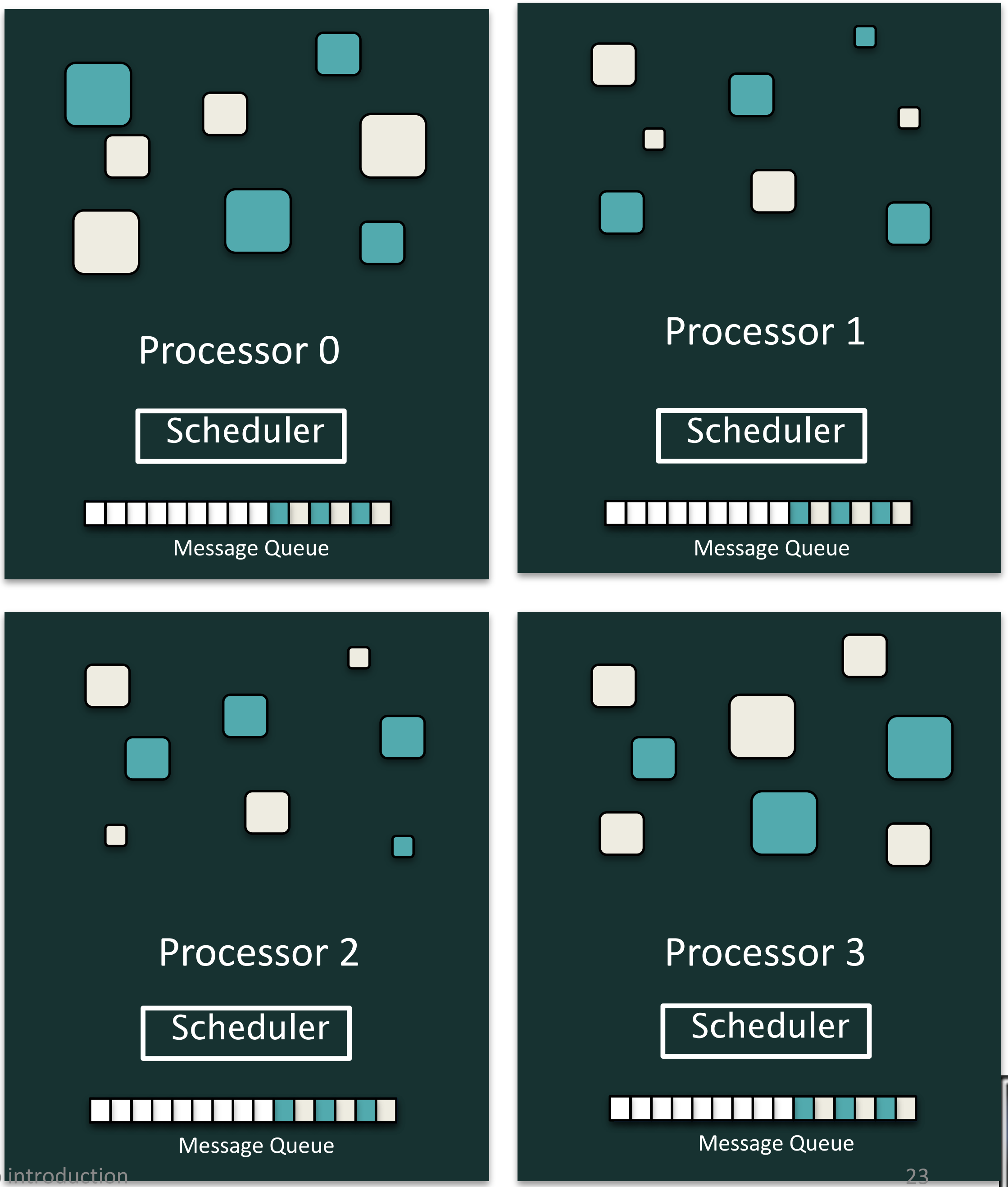
- Example: an object on 0 wants to invoke a method on object A[23]
  - The Runtime System packages the method invocation into a message
  - Locates where the target object is
  - Sends the message to the queue on destination processor
  - Scheduler invokes the method on the target object



The runtime system knows which processors are overloaded, which objects are computationally heavy, which objects talk to which



Using this information, it migrates objects to rebalance load and optimize communication



Code Example: Stencil/Jacobi Relaxation

Sequential C++ code

MPI analogue

4 send calls

4 recv calls

Asynchronous Reduction

```
entry void run() {
    while (!converged) {
        serial {
            copyToBoundaries();
            int x = thisIndex.x, y = thisIndex.y;
            int bdX = blockDimX, bdY = blockDimY;
            thisProxy(wrapX(x-1),y).updateGhosts(iter, RIGHT, bdY, rightGhost);
            thisProxy(wrapX(x+1),y).updateGhosts(iter, LEFT, bdY, leftGhost);
            thisProxy(x,wrapY(y-1)).updateGhosts(iter, TOP, bdX, topGhost);
            thisProxy(x,wrapY(y+1)).updateGhosts(iter, BOTTOM, bdX, bottomGhost);
            freeBoundaries();
        }
        for (remoteCount = 0; remoteCount < 4; remoteCount++)
            when updateGhosts[iter](int ref, int dir, int w, double buf[w])
                serial { updateBoundary(dir, w, buf);
            }
        serial { double error = computeKernel();
            int conv = error < DELTA;
            if (iter % 5 == 2)
                contribute(sizeof(int), &conv, CkReduction::logical_and,
                           CkCallback(CkReductionTarget(Jacobi, checkConverged), thisProxy));
        }
        if (++iter % 5 == 0)
            when checkConverged(bool result) if (result) serial { mainProxy.done(iter); converged = true; }
        if (iter % 20 == 0) { serial { AtSync(); }  when resumeFromSync() {} }
    }
};
```

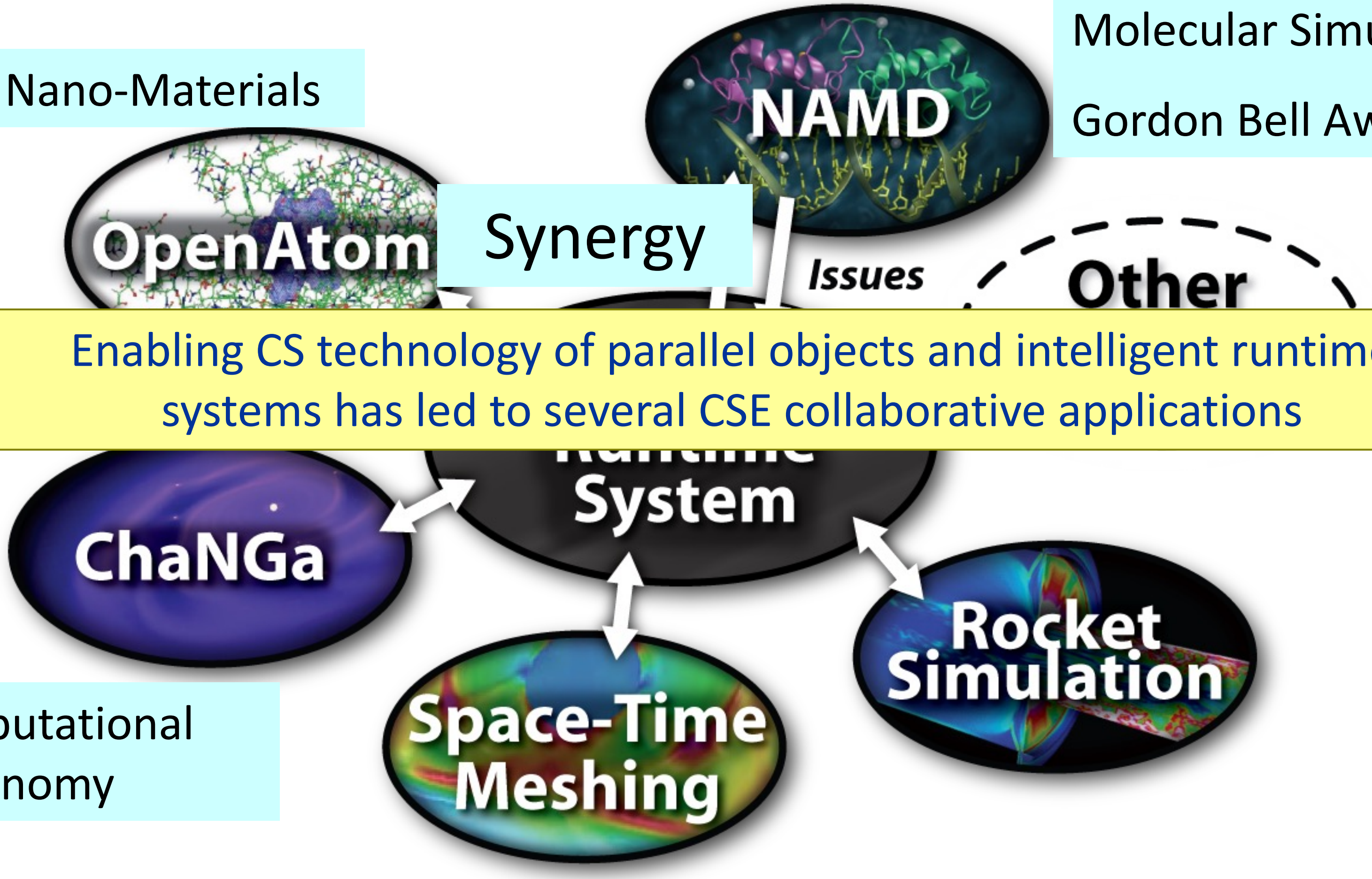
Charm++ and CSE Applications

Well-known Biophysics Molecular Simulation App

Gordon Bell Award, 2002

Nano-Materials

Synergy



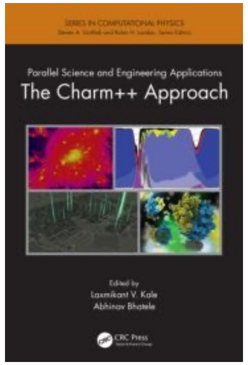
Enabling CS technology of parallel objects and intelligent runtime systems has led to several CSE collaborative applications

Computational Astronomy

Charm++ Additional Resources

AMPI talk at ExaMPI at SC’22 (Sunday)

- Papers and research projects: https://charm.cs.Illinois.edu
- Recent workshop: https://charmworkshop22.github.io/
  - Includes talk videos and slides for the last 20 years’ workshops
- More learning material : https://charmplusplus.org
- Commercial support: https://hpccharm.com
- A nice demo of load balancing and fault tolerance:
  - https://www.hpccharm.com/demo (on a raspberry pi cluster)
- Tutorial book: ask me for a draft
- Book with languages and applications:



Higher level abstractions:

- AMPI (Adaptive MPI), Charm4Py
- Multiphase shared arrays, Charisma
- ParaTreeT
- CharmTyles (new)

- Applications:
  - NAMD (BioPhysics)
  - CHaNGa (Astro)
  - SPeCTRE (Astro)
  - Enzo-P (Astro)
  - OpenAtom (elec structure)

- 4 out of 21 applications project selected by TACC for leadership apps are Charm++ based

Summary for Chapel, UPC++, and Charm++

- Major applications *(see previous slides for more)*
  - **Chapel**: CHAMPS (aerodynamics code), Arkouda server (data analytics)
  - **UPC++**: MetaHipMer (genome assembler), SIMCoV (biology)
  - **Charm++**: NAMD (molecular dynamics), ChaNGa (astronomy)
- All work from laptop to supercomputers
- All provide interoperability with MPI
- Key differentiator is the overarching programming model
  - **Chapel**, one thread and then indicates which locales tasks should compute on
  - **UPC++,** SPMD with each process executing the same program
  - **Charm++,** main object that starts other objects and seeds communication