Cosmological Particle-Mesh Simulation in Chapel

Ben Albrecht, Nikhil Padmanabhan

Chapel

Chapel is a modern parallel programming language designed for productivity at scale.

Chapel supports:

General parallelism
A single syntax to describe any flavor of parallelism
Separation of Parallelism and Locality
Better suited for emerging heterogeneous architectures
Multiresolution Design
Parallelism and locality interface spans multiple layers of abstraction

Chapel is:

Open source
Licensed under Apache v2, hosted on GitHub
Portable
Runs on laptops, desktops, to Cray supercomputers
Productive
Includes a wide range of modern language features
Performant
Shared-memory: typically competitive with C+OpenMP
Distributed-memory: varies, but closing in on C+MPI
A PGAS language
Not restricted to static SPMD parallelism

Multiresolution Design

Domain Maps
Data Parallelism
Task Parallelism
Base Language
Locality Control
Target Machine

```
// Task parallelism in Chapel
// This will create 'numTasks' tasks to compute each iteration
coforall taskId in 0..#numTasks do
  writeln("Hello, world! from task " + taskId);

// Data parallelism in Chapel
// This will create a number of tasks proportional to the available hardware
//   and the size of the loop
forall msg in 0..#numMessages do
  writeln("Hello, world! from message " + msg);
```

Simulating the Universe

Motivation:

The goal of cosmology is to understand the origin, evolution, and fate of the Universe. Cosmology relies on modeling observations in their entirety in order to make inferences about the underlying physics.

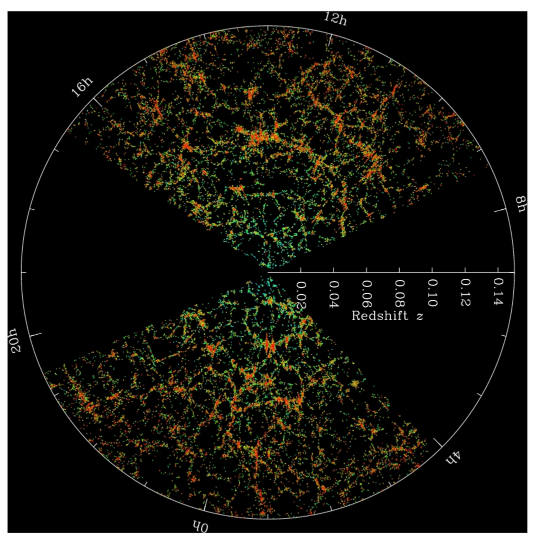
Physics:

1. **Update particle positions and velocities:**

$$\frac{d\mathbf{x}}{da} = \frac{\mathbf{p}}{\dot{a}a^2} \qquad \frac{d\mathbf{p}}{da} = -\frac{\nabla\phi}{\dot{a}},$$

2. **Update particle positions and velocities:**

$$\nabla^2\phi = 4\pi G\rho_0\frac{\delta}{a}$$



Map of Universe by Sloan Digital Sky Survey

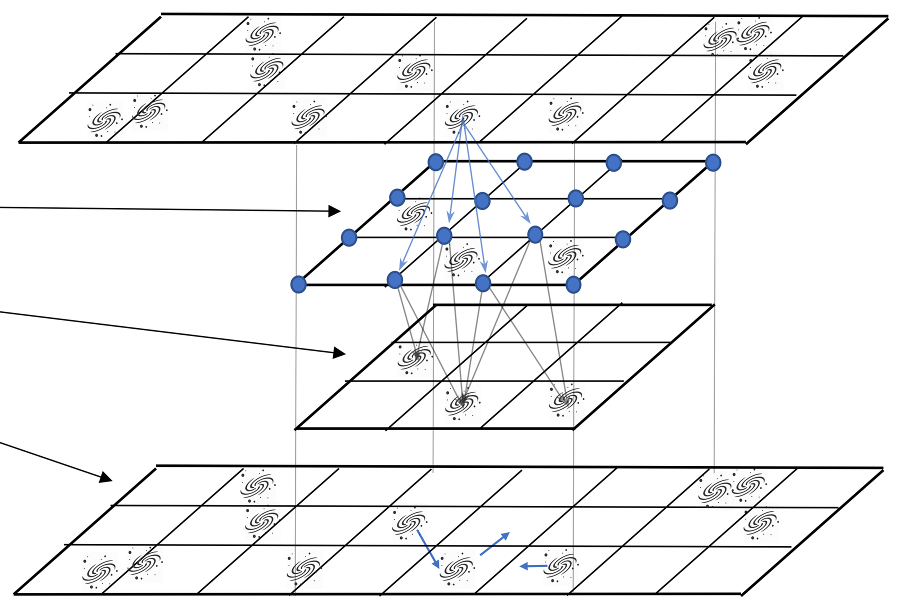
Computation:

N-body simulation is performed using the *particle-mesh (PM)* algorithm.

A single time step in the N-body simulation:

Particle-mesh algorithm visualized in 2D

1. Deposit particles onto grid to define grid density
2. Use FFTs to solve the Poisson equation for the gravitational potential
3. Finite difference to compute forces (gradients of potential)
4. Galaxy positions and velocities are updated



Productivity

```
// The main loop of particle-mesh

proc main() {
  // Initial conditions
  makeZeldovichInitialConditions();
  slabDecompose(PP);
  ...

  do {
    // Stream particles
    streamParticles(log(aa), log(ahalf));


    // Reshuffle particles across domains
    slabDecompose(PP);


    // Compute forces
    pmforce(aa);


    // Kick particles
    kickParticles(log(aa),log(aa)+dloga);


    // Stream particles
    streamParticles(log(ahalf),
                    log(aa)+dloga);
  } while (aa <= afinal);
}
```

```
// Domains & FFTW Distribution

/*
  Global view of storage abstracts data
  distribution and parallel operations

  Block distribution was extended to
  support FFTW
 */
var AA,BB: [FFTW_Domain_Ghosted] real;
var PP = initializeParticlesOnGrid(Nc);
```

```
// Domain slicing

// FFTW-distributed domain
const D : domain(3) dmapped
  FFTW3D(Ng, nghosts=nghosts) = DSpace;

/* Domain slicing allows accessing
   subdomains, e.g. real and imaginary
   parts of FFT split up
 */

// Real space
const Dreal = D[..,..,0..#Ng];

// Real part
const Dre = D[..,..,
              0..#(Ng+2) by 2 align 0];

// Imaginary part
const Dim = D[..,..,
              0..#(Ng+2) by 2 align 1];

// Frequency
const Dk = D[..,..,0..#(Ng/2+1)];
```

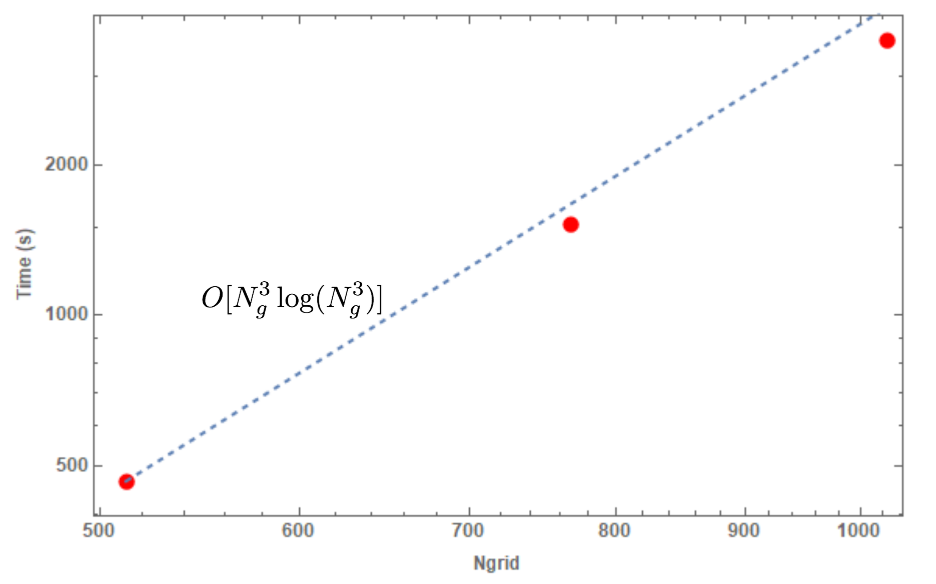
```
// C/MPI Interoperability

// Extern declaration exposes FFTW
extern proc fftw_mpi_plan_dft_r2c_3d(
  n0 : c_ptrdiff, n1 : c_ptrdiff,
  n2 : c_ptrdiff, ref inarr ,
  ref outarr, comm : MPI_Comm,
  flags : c_uint) : fftw_plan;

// Calling MPI and FFTW from Chapel
Barrier(CHPL_COMM_WORLD);
fwd = fftw_mpi_plan_dft_r2c_3d(
  Ng, Ng,Ng, myElems[idx],
  myElems[idx], CHPL_COMM_WORLD,
  fftwPlanner);
```

Performance

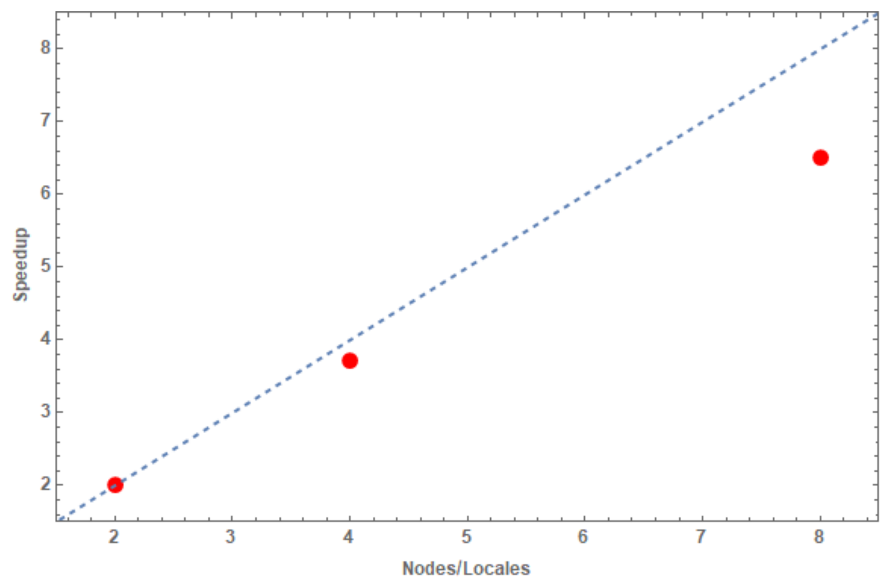
Hardware
Broadwell Intel Xeon 2.2 GHz
44 cores (dual socket) per node
128 GB memory
Aries network

Software
Chapel 1.16.0.1
compiled with intel-17.0.4
ugni communication layer
Qthreads tasking layer
Compiled with: `--fast`

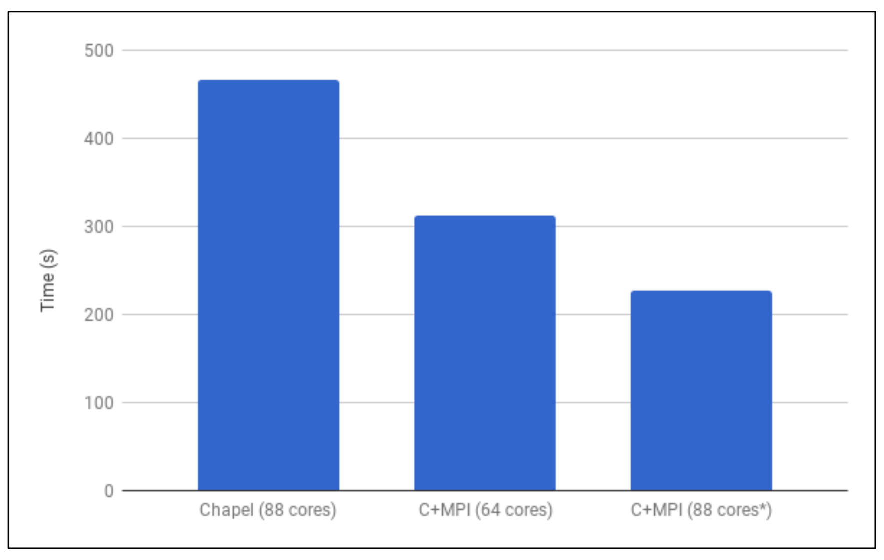
Scaling with Problem Size



Strong Scaling



Chapel vs. C+MPI



** C+MPI implementation requires number of ranks to be divisible by N_{grid}, therefore C+MPI 88 core timing assumes perfect scaling*

Conclusions and Next Steps

Productivity

Chapel is usable as a productive language today.

- Hybrid parallel/distributed programming is made easy in Chapel, assuming key abstractions are in place.
- It was not difficult to implement the PM code, the FFTW-compatible distribution, and skyline arrays.
- Chapel's first class C interoperability feature made interfacing with legacy code simple.
- Interoperability with MPI is functional, though performance is not ideal.
- Tooling was the biggest weakness felt during this work, including:
  - Debugging
  - Profiling
  - Compilation times

Performance

- Chapel performance is within factor of 2 of C+MPI performance
- C code is pure MPI vs. Chapel being hybrid Qthreads+MPI
  - This caused contention when accumulating onto grid in Chapel
- FFTW transposes are single-threaded, which penalizes Chapel's performance
- Problem size scaling is largely dominated by FFT
- Chapel code uses atomics for assigning particles to grid points, trading performance for simplicity

References:

1. N. Padmanabhan, B. Albrecht, "Cosmological Particle-Mesh Simulations in Chapel", *Proceedings of the PGAS Application Workshop (PAW)*, November 2017
2. PM GitLab: https://gitlab.com/bja/pm-paw2017
3. Chapel GitHub: https://github.com/chapel-lang/chapel