Cosmological Particle Mesh Simulations in Chapel

Nikhil Padmanabhan & Ben Albrecht
Yale University
Cray Inc
PAW 2017

My perspective

Astrophysicist who occassionally writes code.

Need to be able to easily prototype algorithms.

Limited by time to think

- Days
- Final runs on large number of data sets for final results.

Want to be able scale out relatively easily.

Yes, I like free lunches!

Goals of this Work

Scientific

- Exploring Chapel in a research environment
- Common motifs in a PM code and other data analysis codes

Usability of Chapel

- Ease of parallel programming
- Interoperability with MPI
- Interoperability with existing codes

Performance of Chapel

- Balance "performance" with "productivity"

Where I'd like to end up

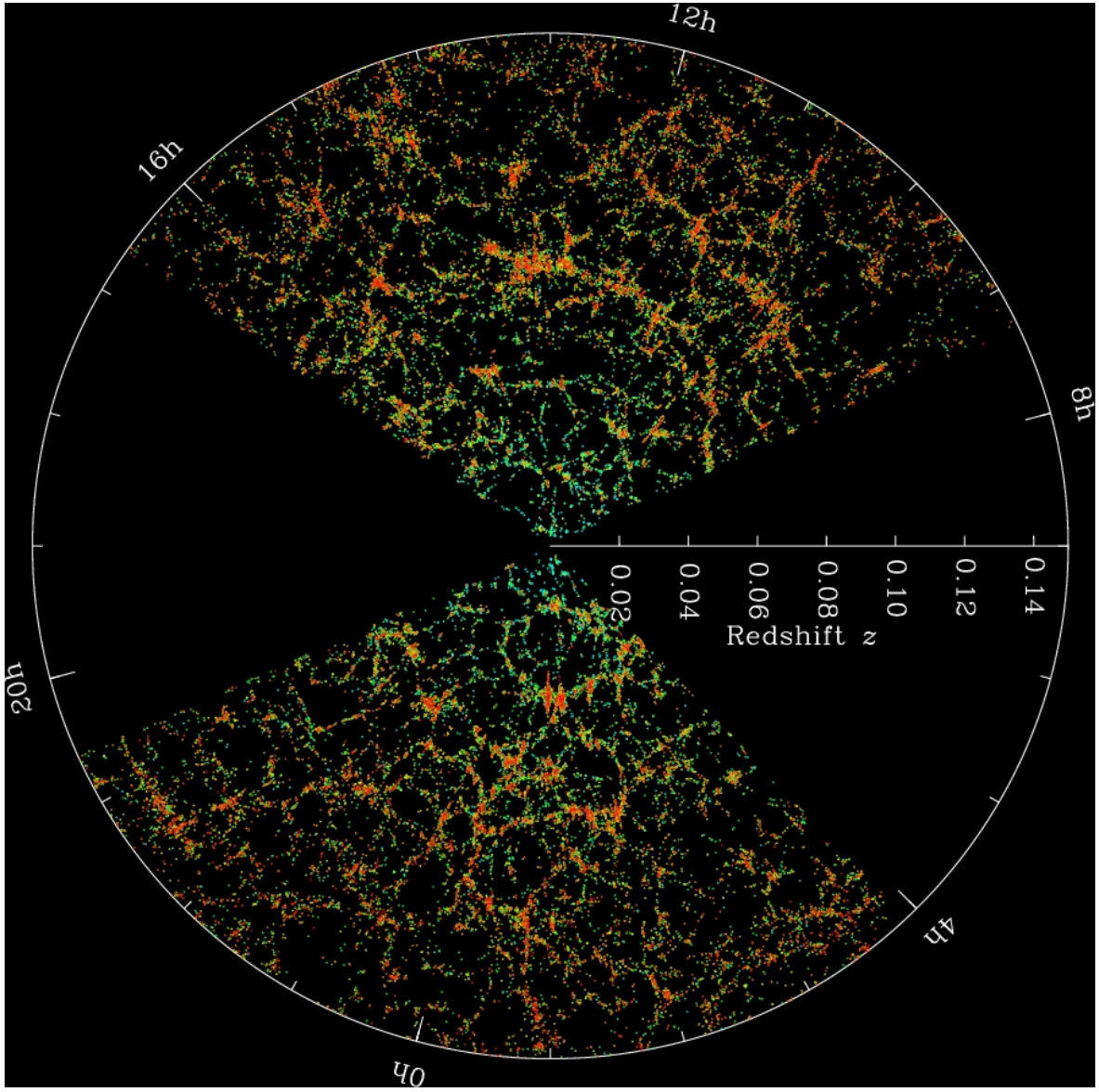
Chapel is a **usable, productive** language today.

- Ease of hybrid parallel/distributed programming
  - Assuming key abstractions are in place.
  - Not hard to write : the PM code uses FFTW-compatible grids and skyline arrays.
- Performance within 2x of C+MPI for this application.
- Strong interop story with C
- Functional interop story with MPI

Challenges

- Tooling remains a challenge

The Role of Cosmological Simulations in Cosmology



SDSS Collaboration

The Role of Cosmological Simulations in Cosmology

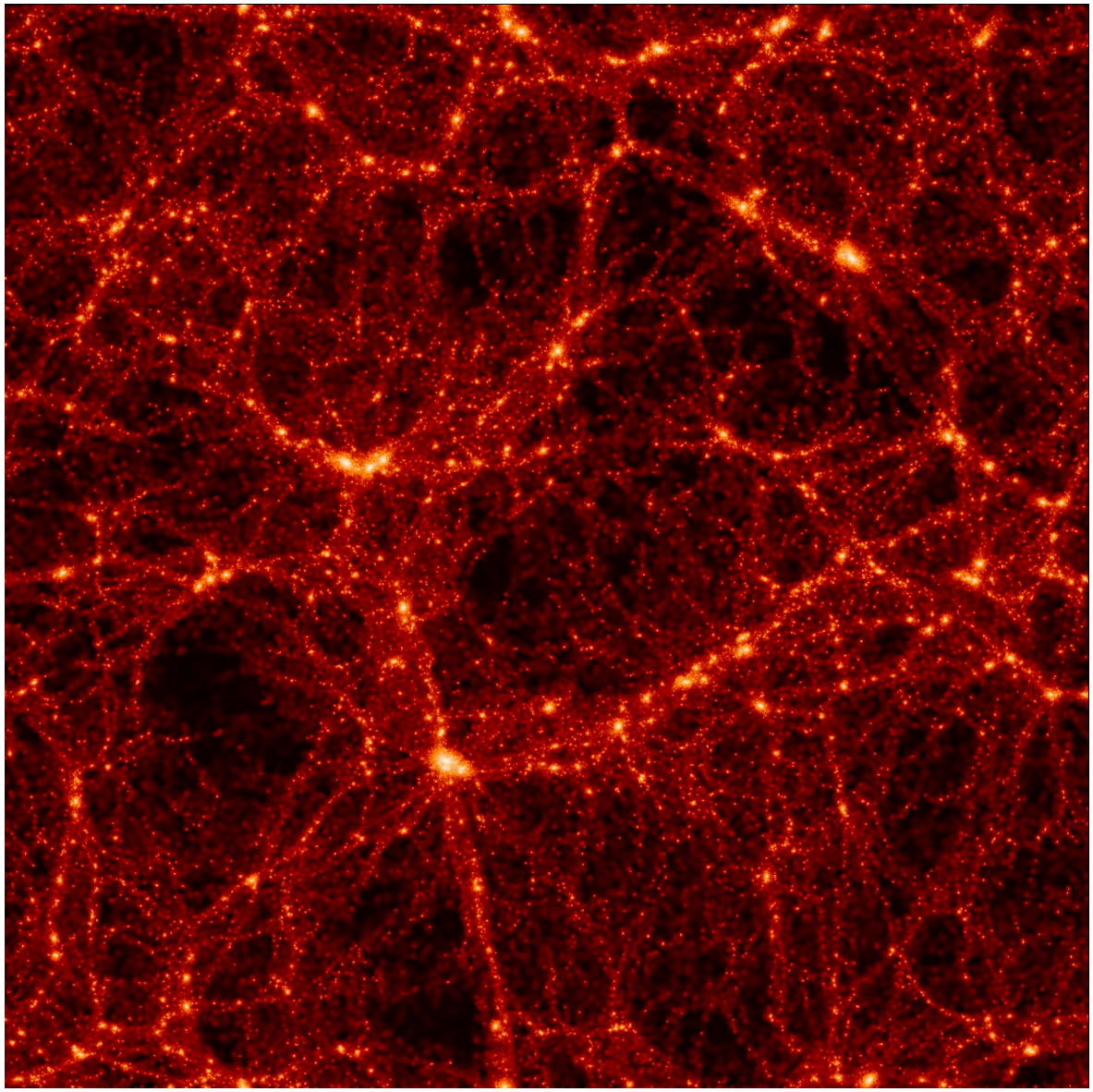


Figure courtesy MPA

An Overview of Chapel

A next-generation, high-productivity PGAS language.

- Developed as an open-source project at Cray Inc.
- Runs on laptops to Crays

Key features

- Native data and task parallelism
- Multiresolution design
  - High-level abstractions
  - Low-level communication/computation when necessary
- Data abstractions are in Chapel
  - Allow users to extend these
- Interoperability with C and MPI

Status

- Chapel is under active development
- The current implementation is advanced enough to be usable.

Evolving Gravity in an Expanding Universe

Update particle positions and velocities

$$\frac{d\mathbf{p}}{da} = -\frac{\nabla\phi}{\dot{a}}$$

$$\frac{d\mathbf{x}}{da} = \frac{\mathbf{p}}{\dot{a}a^2}$$

Compute the gravitational potential

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

Solve this with Fourier transforms

The Anatomy of a Particle-Mesh Simulation

Gravitation PM codes have two principal phases.

Forces

- Forces are computed on a grid
- Deposit paricles onto grid to define density field.
- Use FFTs to solve the Poisson equation for the gravitational potential
- Finite difference to compute forces (gradients of potential)

Update particle positions and velocities

- Simple S(tream)-K(ick)-S(tream) integrator

The Code

An abbreviated top-level view of the code.

Module imports

- Include MPI and FFTW support

```
/* This is the main driver program. */

use MPI;
use FFTWDist;
```

Global view of storage

- Provide a global view of the grid and particles
- Chapel introduces the concept of a domain on an array
  - Abstract data distribution/data parallel operations.
  - Chapel domains are written in Chapel.
  - We extend the `Block` domain to be compatible with FFTW.

```
var AA,BB: [FFTW_Domain_Ghosted]real;
var PP = initializeParticlesOnGrid(Nc);
```

A Deep Dive into Domains

- Chapel's domain arithmetic allows for specifying the problem space in an intuitive way.
- `D` is automatically distributed across all nodes, as expected by FFTW, and accounting for ghost cells.

```
const DSpace = {0.. #Ng, 0.. #Ng, 0.. #(Ng+2)};
const D : domain(3) dmapped FFTW3D(Ng, nghosts=nghosts) = DSpace;
```

- Use domain slicing to access subdomains
  - Access the real and imaginary parts of the FFT.

```
const Dreal = D[..,..,0.. #Ng]; // In real space
const Dre = D[..,..,0.. #(Ng+2) by 2 align 0]; // Real part
const Dim = D[..,..,0.. #(Ng+2) by 2 align 1]; // Imaginary part
const Dk = D[..,..,0.. #(Ng/2+1)]; // Frequency
```

The Main Loop

```
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);
}
```

Kicking/Streaming Particles

- Particle data `PP` stored in SOA manner
- Coordinates, momenta etc are stored in skyline arrays
  - Implemented in user code in Chapel
  - Transparent
- Reshuffling particles benefits from PGAS

```
proc streamParticles(logai: real, logaf: real) {
  const tfac = (symx(logaf) - symx(logai)):real(32);
  for param idim in 1..Ndim {
    [(x1, p1) in zip(PP.r(idim), PP.p(idim))] x1 = periodic(x1+p1*tfac);
  }
}
```

- Param loops are unrolled by the compiler
- Chapel automatically parallelizes across cores/nodes
- Code is the same as a serial code

The Potential Calculation

```
// Put the particles onto the grid
CIC(PP, AA);

// Work out the potential
AA.fftForward(transposeOpt=true);
forall (ik, ire, iim) in zip(Dk, Dre, Dim){
  local { // For safety
    const ikt = (ik(2), ik(1), ik(3)); // Transpose
    const kk = kFreq(ikt, Ng);
    var k2 = 0.0;
    for param idim in 1..Ndim do k2 += kk[idim]**2;
    k2 *= twopi2;
    const fac = -(1.0/k2)*1.5*om*scale;
    BB.localAccess[ire] = fac*AA.localAccess[ire];
    BB.localAccess[iim] = fac*AA.localAccess[iim];
  } // End of local
}
BB.fftReverse(transposeOpt=true);
BB.updateGhosts();
```

- Notice the global view on to the grid.
- `localAccess` is an current required manual optimization.
- Moving to a more data-centric implementation.

The Force Calculation

```
// Now finite difference in each direction
for idim in 1..Ndim {
  var tmp = (0,0,0);
  tmp(idim) = 1;
  const dir = tmp;
  forall idx in Dreal {
    local {
      // 4-pt difference scheme, do this by hand
      var plus1 = idx + dir;
      var plus2 = plus1 + dir;
      var minus1 = idx - dir;
      var minus2 = minus1 - dir;


      var p1 = BB.localAccess[plus1],
        p2 = BB.localAccess[plus2],
        m1 = BB.localAccess[minus1],
        m2 = BB.localAccess[minus2];

      AA.localAccess[idx] = (-2.0/3.0)*(p1-m1)+(1.0/12.0)*(p2-m2);
      AA.localAccess[idx] *= Ng;
    }
  }
  AA.updateGhosts();
```

- Note the automatic parallelization.
- The iteration over `Dreal` is a 3D iteration.

Interoperability

Declaring external C functions (including MPI ones)

```
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

```
Barrier(CHPL_COMM_WORLD);
fwd = fftw_mpi_plan_dft_r2c_3d(Ng, Ng, Ng,
         myElems[idx], myElems[idx], CHPL_COMM_WORLD, fftwPlanner);
```

MPI Subtleties

- Impedance mismatches between the Chapel tasking layer and MPI
- Prefer non-blocking calls or protect with barriers.
- The Chapel MPI module provides non-blocking alternatives.
- The Chapel MPI module wraps MPI 1.1.

Specifications

Hardware

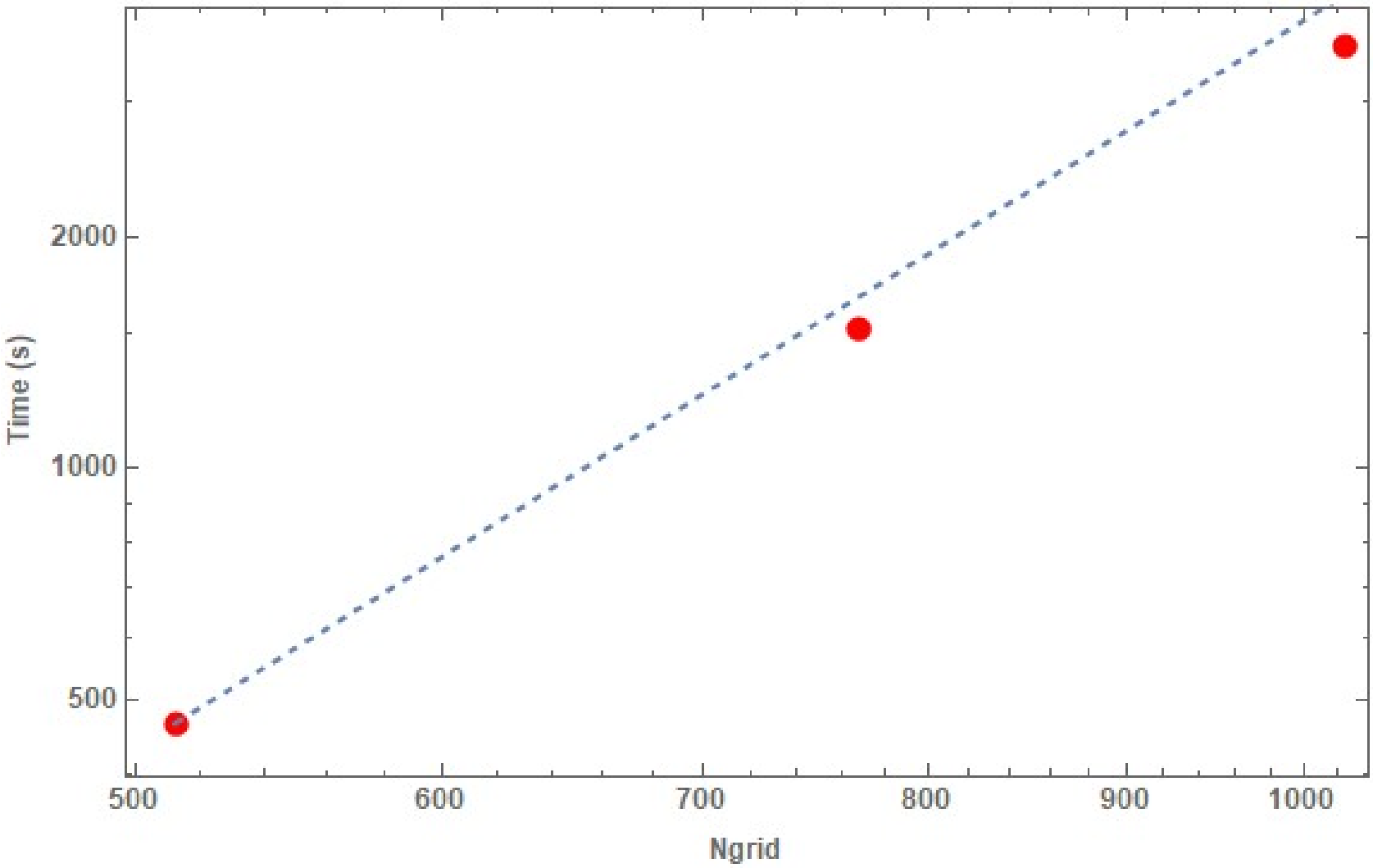
- Broadwell Intel Xeon 2.2 GHz
- 44 cores (dual socket) per node
- 128 GB memory
- Aries network

Software

- Chapel 1.16 pre-release (3274f16b43)
- compiled with intel-17.0.4
- ugni communication layer
- Qthreads tasking layer
- Compiled with: --fast

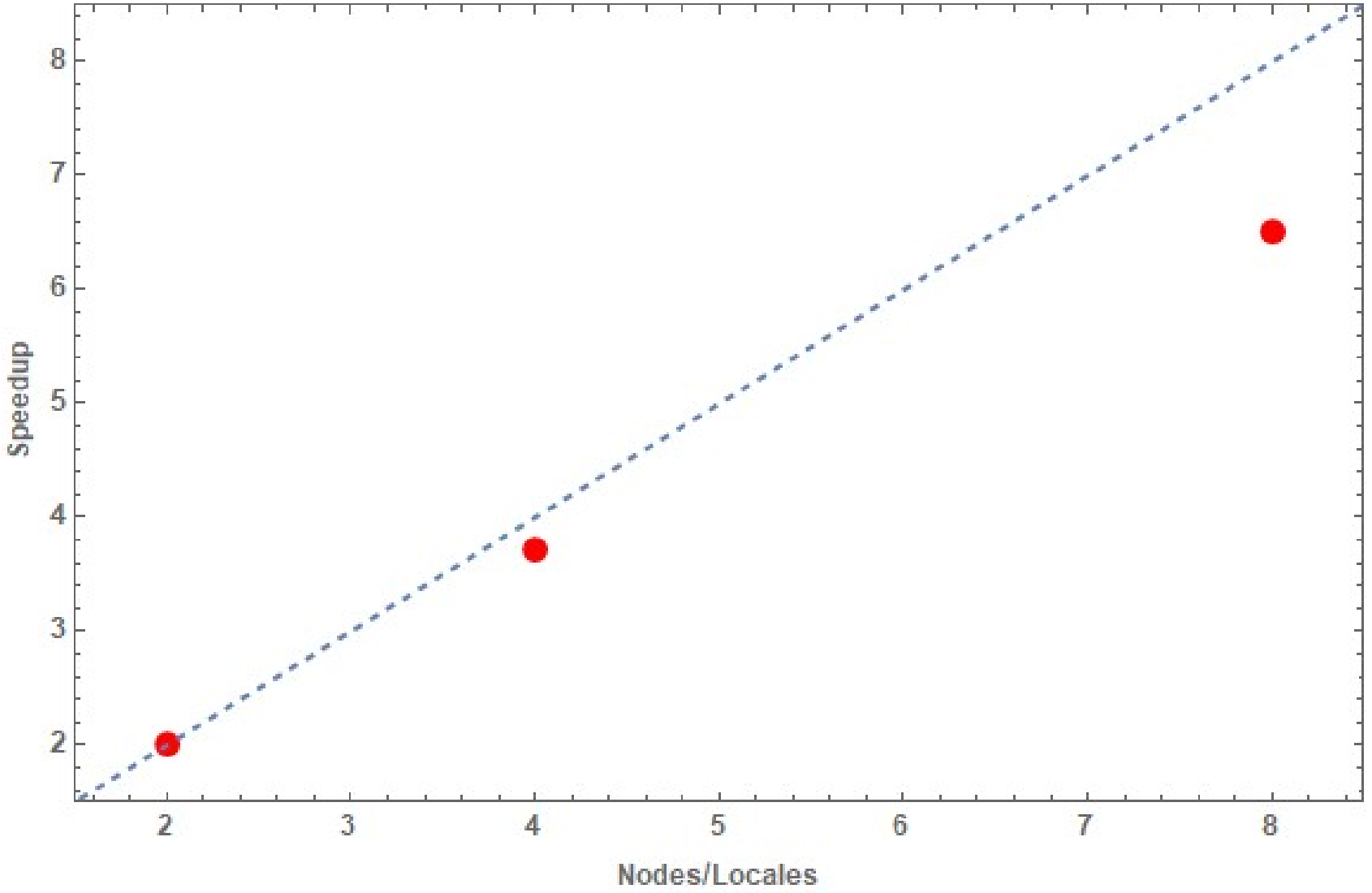
Scaling with Problem Size

- Time dominated by FFT
- Expected scaling $O[N_g^3 \log(N_g^3)]$



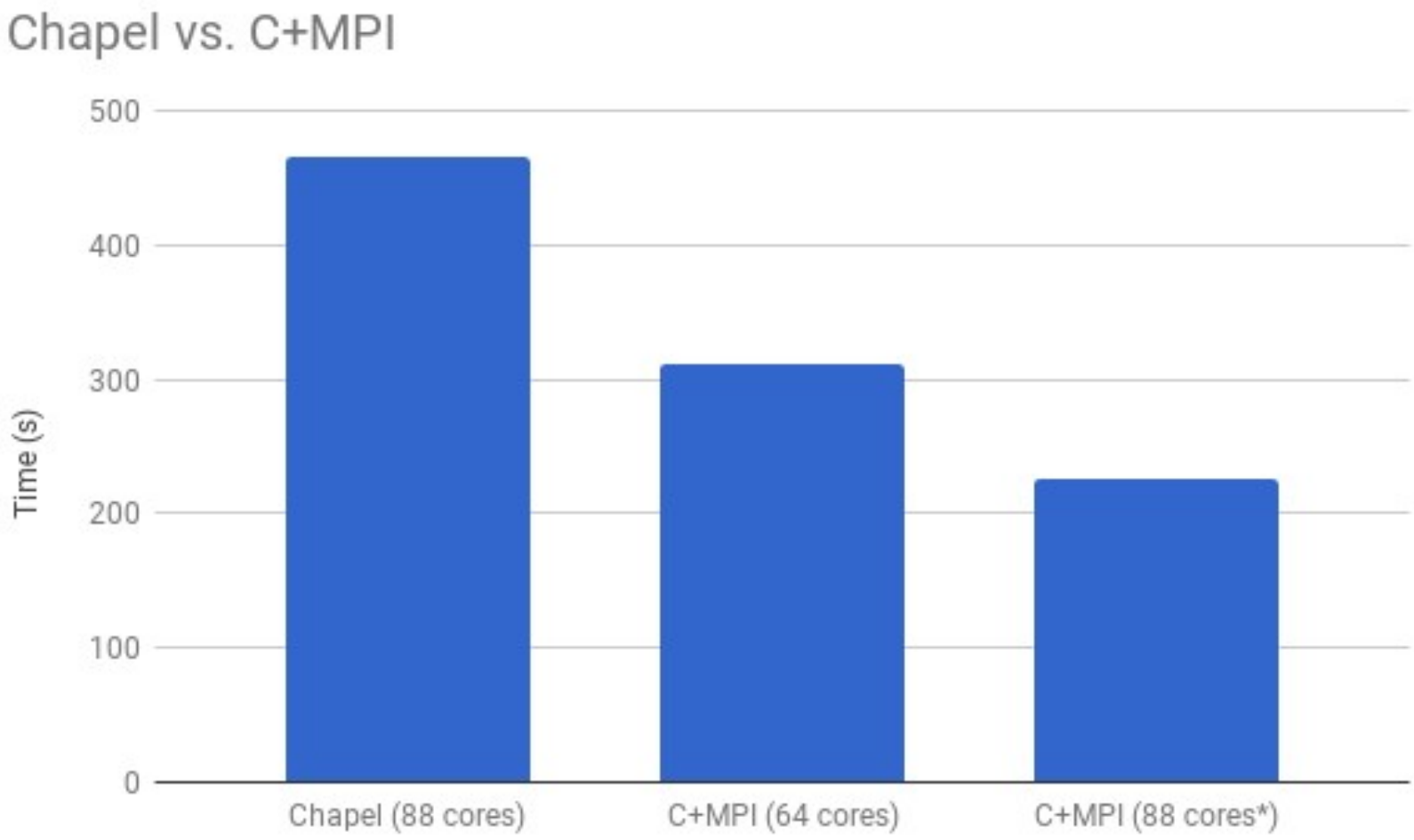
Strong Scaling

- Grid size=1024^3, # particles=1024^3



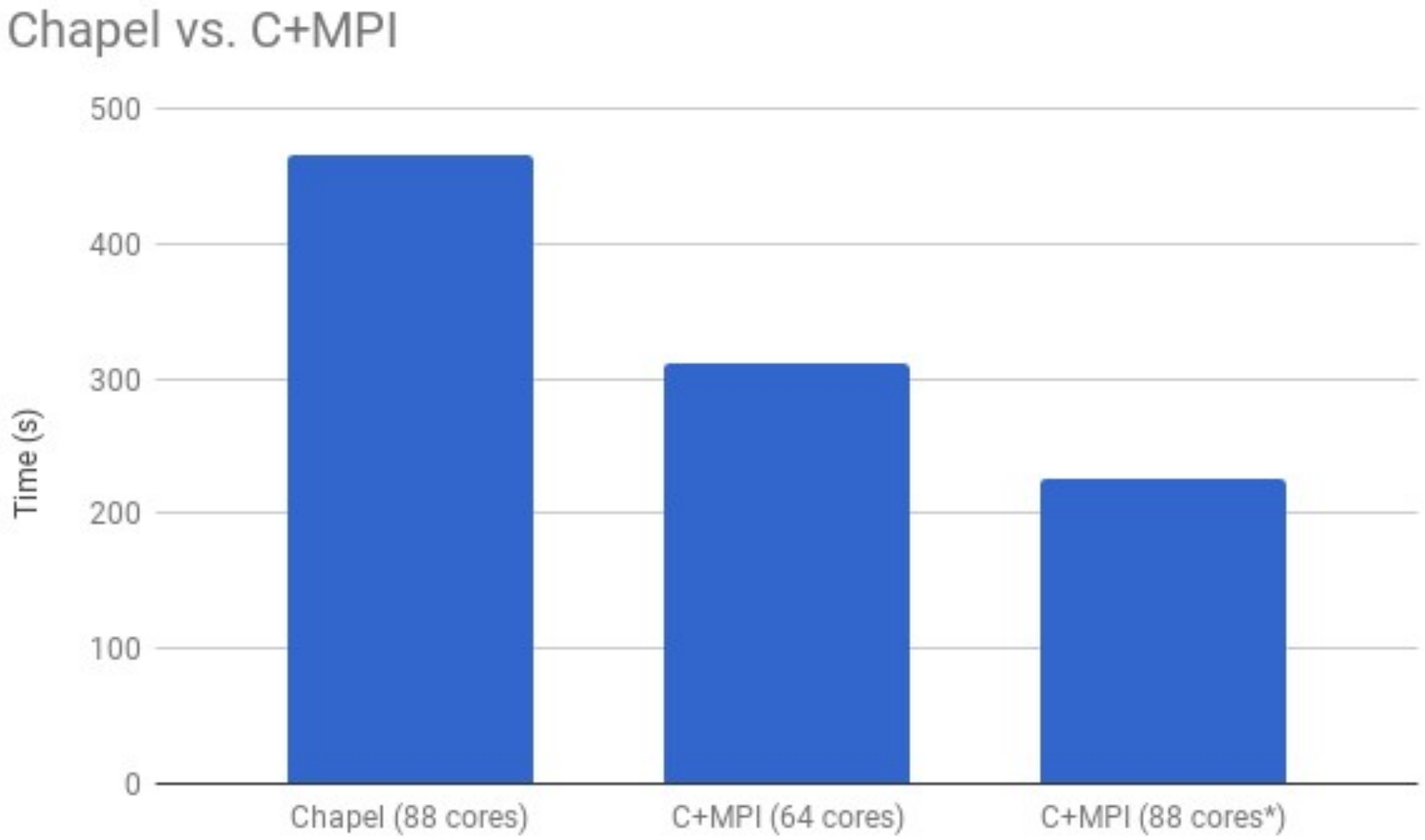
Chapel vs MPI

- C code requires number of ranks to divide grid size.
  - 88 core version assumes perfect scaling from 64 cores.



Performance Caveats

- C code is pure MPI
  - Contention when accumulating onto grid in Chapel
  - Chapel code uses atomics for convenience
- FFTW transposes are single-threaded



Conclusions

Chapel is a **usable, productive** language today.

- Ease of hybrid parallel/distributed programming
  - Assuming key abstractions are in place.
  - Not hard to write : the PM code uses FFTW-compatible grids and skyline arrays.
- Performance within 2x of C+MPI for this application.
- Strong interop story with C
- Functional interop story with MPI

Challenges

- Tooling remains a challenge

Programming in Chapel is fun!