Experiences with Chapel in Cosmology

Data Analysis to Simulations

Nikhil Padmanabhan [1]

[1]Dept. of Physics, Yale Univ.

2022/02/24

w/ Luna Zagorac (Yale),
Richard Easther (Auckland),
Elliot Ronaghan (HPE)

PAW-ATM, SC 2019
arXiv:2109.01920
Partially supported by NASA, DŒ

Outline

Cosmology Meets Chapel

Example I : Chapel and ULDM

- Distributed FFTs

Example II : Data Munging

Conclusions

A quick introduction to my research

- Observational/Theoretical Cosmologist
- Use large galaxy surveys to constrain underlying physics of the Universe
  - The nature of the initial conditions
  - Evolution
  - Constituents
- Get only one observation; need to use simulations to infer/constrain.

Categories of Problems

- Grids (stencils, spectral codes)
- Monte Carlo methods
  - Sampling
  - Ensembles of simulations
  - Often embarrassingly parallel!
- Miscellany
  - Numerical quadrature
  - Fitting; optimization
  - Linear algebra
  - ...

Lifecycle of a Research Problem

Process

- Conceiving of the problem
- Mathematical formulation
- Rough draft of codes required to solve
- Data munging
- Make simulated data, run tests on simulations
- Make figures, write paper, repeat as needed.

Lifecycle of a Research Problem

Process

- Conceiving of the problem
- Mathematical formulation
- Rough draft of codes required to solve
- Data munging
- Make simulated data, run tests on simulations
- Make figures, write paper, repeat as needed.

Character

- These are NOT production codes
- Lifetime usually set by the research project, rarely long-lived.
- Used by a small number of people

Productivity v. Performance

Productivity

- Time to a completed project is critical.
- Easy for students to adapt.
- Easy to develop on a variety of systems (laptops to HPC systems).
- Easy to parallelize/distribute.
- Throughput.

Productivity v. Performance

Productivity

- Time to a completed project is critical.
- Easy for students to adapt.
- Easy to develop on a variety of systems (laptops to HPC systems).
- Easy to parallelize/distribute.
- Throughput.

Performance

- Absolute performance isn't critical; fast enough is good enough.
- Codes need to scale out to characteristic sizes of problems.
- Often running on relatively small systems.

What drew me to Chapel

- Expressive parallelism
- Arrays as “first-class” objects
- No memory/performance surprises (eg. hidden copies)
- Scriptiness

Outline

Cosmology Meets Chapel

Example I : Chapel and ULDM

- Distributed FFTs

Example II : Data Munging

Conclusions

Motivating Ultralight Dark Matter

About

- In the standard cosmological model, 80% of the matter in the Universe is "dark" (i.e. non-baryonic).
- Form gravitationally bound structures : dark matter halos.
- The traditional model is a heavy particle ($\sim 100\times$ proton), with weak interactions.

Motivating Ultralight Dark Matter

About

- In the standard cosmological model, 80% of the matter in the Universe is "dark" (i.e. non-baryonic).
- Form gravitationally bound structures : dark matter halos.
- The traditional model is a heavy particle ($\sim 100\times$ proton), with weak interactions.

Successes

- Explains a large scale of observations, from the rotation of galaxies, to "Bullet" clusters, to the distribution of galaxies, to the cosmic microwave background.

Motivating Ultralight Dark Matter

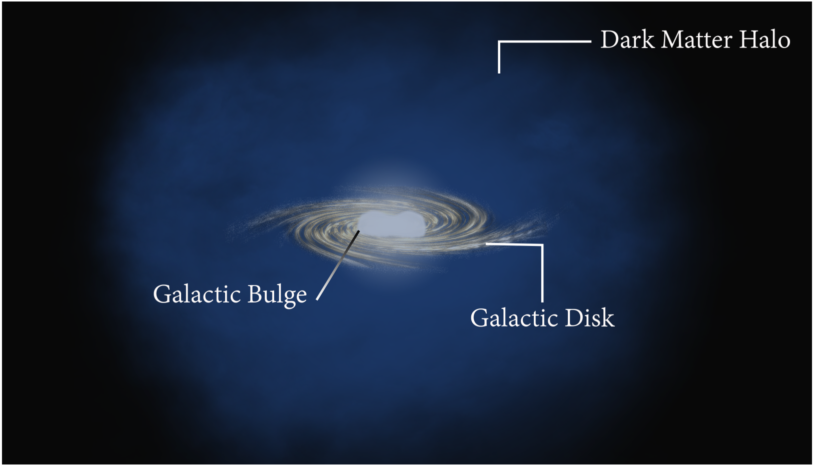
Successes

- Explains a large scale of observations, from the rotation of galaxies, to “Bullet” clusters, to the distribution of galaxies, to the cosmic microwave background.

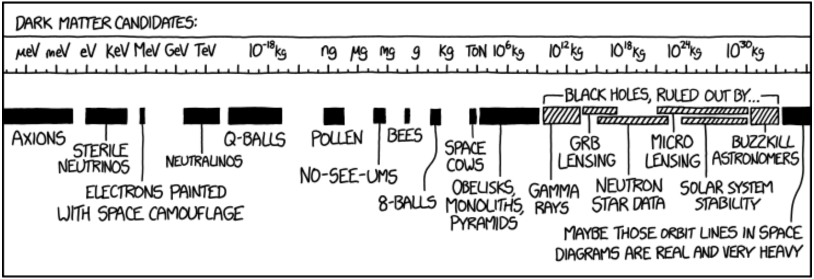
Challenges

- Possible puzzles remain on small scales from the structure of dark matter halos, to the observed abundance of dark matter halos. Note that these might well be solved by astrophysics.
- We have not detected these in the lab, or at accelerators.

Dark Matter : A Cartoon



L. Jaramillo & O. Macias/Virginia Tech.



https://xkcd.com/2035

We're waaay off to the left!

Snapshots

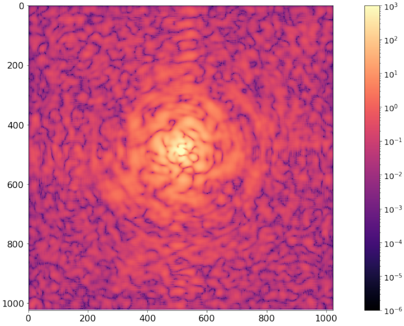


Figure: A ULDM "Halo"

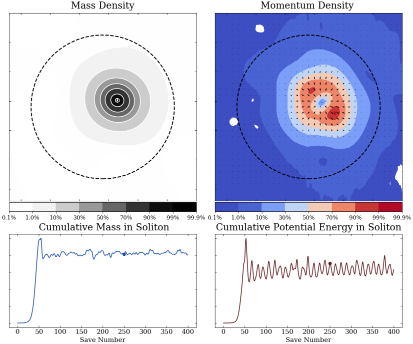


Figure: Collisions

The Schrodinger-Poisson Equations

FFTs

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi + m\Phi\psi$$

$$\nabla^2\Phi = 4\pi G m|\psi|^2$$

Isolated boundary conditions

Distributed FFTs are a key component!

Why Chapel?

- Efficiency of the Python code relied on the calling out to C for fast FFTs.
- Isolated boundary conditions required inserting steps between the various FFT stages
  - Required going back to Python
  - Looping in Python was expensive
- Memory usage
- Scaling to multiple nodes

History of Project

- `PyUltraLight`[a]: An initial code in Python, driven by Jupyter notebook
    - Easy to use and modify, allowing numerical experiments
    - Performant and multithreaded (made significant use of eg. `numexpr`, `FFTW`)
- Extending to isolated potentials hit Python bottlenecks
- Attempted a skunkworks (2019/6/22) port to Chapel for a single node. Resulting code not much longer than Python, could implement isolated potentials, better multithreaded performance.
- Distributed Code
    - Want to run larger N_{grid}, can we extend the code?
    - Isolated potential calculation led to wanting a native Chapel distributed FFT (useful for many other tasks).[b]
    - Validating the FFT led to the NAS NPB benchmark.

[a] Edwards et al, arXiv:1807.04037

[b] Note that Chapel can also interoperate with MPI.

Slab Decompositions Are Simple

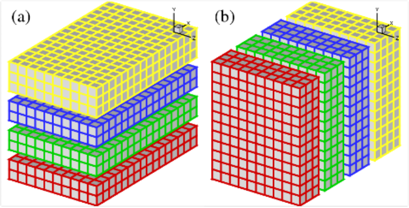


Figure: Slab Decomposition

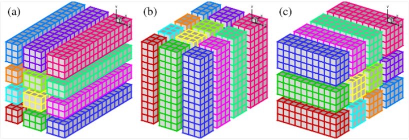


Figure: Pencil Decomposition

http://www.2decomp.org/decomp.html

- Slab decompositions are simpler (especially for the end user)
- Slab limits the amount of parallelism expressed (especially with pure MPI)
- Use 1 slab per locale/node.
- Limits $N_{\rm grid} \geq N_{\rm nodes}$, but in practice, not limiting.
- Reduce communication complexity

Chapel Code is Expressive : Pencil and Paper

The Algorithm

1. Decompose array into slabs in the x direction
2. Fourier transform in the y direction[a]
3. Fourier transform in the z direction
4. Transpose x and y (all to all)
5. Fourier transform in the x direction

[a] We use FFTW (`www.fftw.org`) for 1D serial transforms.

Chapel Code is Expressive : A Naive Implementation

```
coforall loc in Locales do on loc {
  ...
  for ix in xSrc {
    myplane = Src[{ix..ix, ySrc, zSrc}];
    // Y-transform
    forall iz in zSrc {
      yPlan.execute(myplane[0, ySrc.first, iz]);
    }
    // Z-transform
    forall iy in offset(ySrc) {
      zPlan.execute(myplane[0, iy, zSrc.first]);
      // Transpose data into Dst
      Dst[{iy..iy, ix..ix, zSrc}] = myplane[{0..0, iy..iy, zSrc}];
    }
  }
  allLocalesBarrier.barrier();
  // X-transform, similar to Y-transform
  ...
}
```

SPMD

Data parallel

FFTW 1D

PGAS Transpose

Chapel Code is Expressive : A Naive Implementation

```
coforall loc in Locales do on loc {
  ...
  for ix in xSrc {
    myplane = Src[{ix..ix, ySrc, zSrc}];
    // Y-transform
    forall iz in zSrc {
      yPlan.execute(myplane[0, ySrc.first, iz]);
    }
    // Z-transform
    forall iy in offset(ySrc) {
      zPlan.execute(myplane[0, iy, zSrc.first]);
      // Transpose data into Dst
      Dst[{iy..iy, ix..ix, zSrc}] = myplane[{0..0, iy..iy, zSrc}];
    }
  }
  allLocalesBarrier.barrier();
  // X-transform, similar to Y-transform
  ...
}
```

SPMD

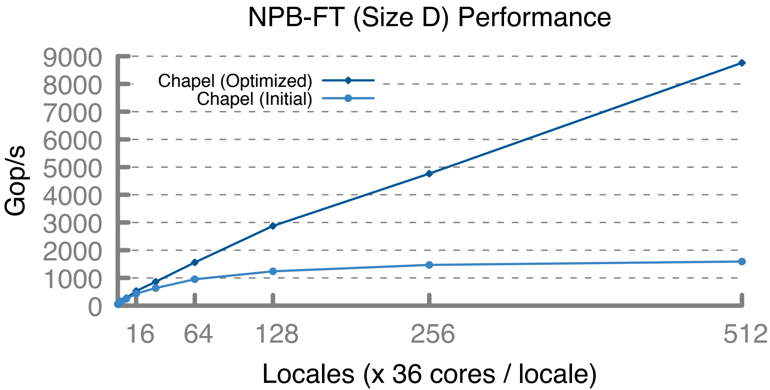
Data parallel

FFTW 1D

Reduce comm congestion!

PGAS Transpose

Chapel FFTs : Naive Performance



Chapel Code is Expressive : A Performant Implementation

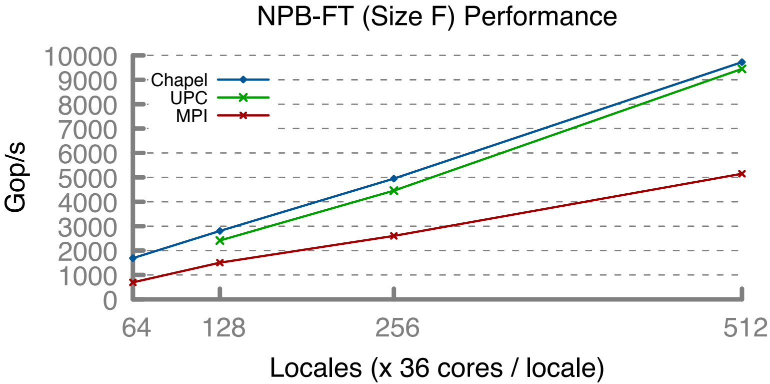
overlap computation and comm

```
...
forall iy in offset(ySrc) {
  zPlan.execute(myplane[0, iy, zSrc.first]);
  // Transpose data into Dst, and copy the next Src slice into myplane
  copy(Dst[...], myplane[...], myLineSize);
  if (ix != xSrc.last) {
    copy(myplane[...], Src[...], myLineSize);
  }
}
...
```

low-level comm

batch FFTW calls (not shown)

Chapel FFTs Scale Well Across Nodes : F = 64×D



Outline

Cosmology Meets Chapel

Example I : Chapel and ULDM

Distributed FFTs

Example II : Data Munging

Conclusions

Scriptiness

```
config const infile="/aux0/siam/catalog.fits";
config const hdu=2;
config const stringBufSize=1000;
config const iRow=10213;
```

All of these constants can be changed at runtime (no recompilation required!)

Interoperability

```
extern {
#include "fitsio.h"
}
ffopen(c_ptrTo(fptr), infile.c_str(), READONLY, c_ptrTo(status));
```

Chapel has much more robust ways to do this, but we're trying to do this quickly.

Reading : In Serial

```
// Define the array
var zcosmo : [0..#numRows] real(32);

{
  ffopen(...);

  // Get the column
  var templt = "Z_COSMO";
  ffgcnn(fptr, CASEINSEN, templt.c_str()...);
  ...
  // Read
  ffgcv(fptr, ..., colnum,
     1,..., numRows, ..., c_ptrTo(zcosmo), ...)
  ffclos(...);
}

// Complicated analysis
```

Reading : In Parallel

```
// Define the array
var zcosmo = newBlockArr(0..#numRows, real(32));

coforall loc in Locales do on loc {
  // my piece
  var myDom=zcosmo.localSubdomain();

  ffopen(...);

  // Get the column
  var templt = "Z_COSMO";
  ffgcnn(fptr, CASEINSEN, templt.c_str()...);
  ...
  // Read
  ffgcv(fptr, ..., colnum,
    (myDom.low+1),...,myDom.size,..., c_ptrTo(zcosmo[myDom.low]), ...);
  ffclos(...);
}

// Complicated analysis
```

Outline

Cosmology Meets Chapel

Example I : Chapel and ULDM

Distributed FFTs

Example II : Data Munging

Conclusions

Summarizing...

- Productive
  - Expressive when translating algorithms to code.
  - Grid-heavy codes are easy to write. Domains/arrays are first-class, and easy to “data-parallelize”.
  - Scalable across nodes/threads.
  - Scriptiness
  - Interoperability
- Expressive parallelism
- Challenges
  - Interactivity
  - Size of community/existing codes/inertia.
  - Tooling

Chapel is an expressive/productive language for research problems!