Simulating Ultralight Dark Matter with Chapel

Nikhil Padmanabhan \(^1\)  Elliot Ronaghan \(^2\)
J. Luna Zagorac \(^1\)  Richard Easther \(^3\)

\(^1\)Yale Univ.
\(^2\)Cray/HPE
\(^3\)Univ. of Auckland

2020/05/22
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.
We’re waaay off to the left!
Motivating Ultralight Dark Matter

- A different paradigm is a very light particle \((10^{-31} \times \text{proton})\).
- Many names: fuzzy dark matter, Bose-Einstein dark matter, ...
- Small mass means that quantum-mechanics can smear it out over astrophysically interesting scales.
- High enough density that it forms a Bose-Einstein condensate.
- Different phenomenology: eg. interference patterns.
- Anything by the most idealized situations requires simulations.
Our Motivation

- Want a code to do numerical experiments with.
- Need scalability
  - Must resolve soliton cores: large dynamic range
  - Simulation time scales as $N^5$; need to scale to large numbers of nodes.
- Initial problem: revisit aspects of the formation of ultra-light dark matter halos from collisions of soliton cores.
- This is an area of very active research (we are newcomers).
- Several codes exist - including adaptive codes, codes built on existing large astrophysical simulations. Challenges to large boxes still exist.
History of Project

- **PyUltraLight**: 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 $N_{\text{grid}}$, can we extend the code?
  - Isolated potential calculation led to wanting a native Chapel distributed FFT (useful for many other tasks).
  - Validating the FFT led to the NAS NPB benchmark.

---

*Note that Chapel can also interoperate with MPI.*

---

*a* Edwards et al, arXiv:1807.04037

*b* Note that Chapel can also interoperate with MPI.
The Schrodinger-Poisson Equations

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

\[ \nabla^2 \Phi = 4\pi Gm |\psi|^2 \]

Isolated boundary conditions

Distributed FFTs are a key component!

Padmanabhan et al

ULDM in Chapel
Slab Decompositions Are Simple

- 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 $N_{\text{grid}} \geq N_{\text{nodes}}$, but in practice, not limiting.
- Reduce communication complexity

Figure: Slab Decomposition

Figure: Pencil Decomposition

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

}
Chapel FFTs: Naive Performance

NPB-FT (Size D) Performance

Locales (x 36 cores / locale)

Chapel (Optimized)
Chapel (Initial)
overlap computation and comm

```chapel
class myplane:
    static var myplane: float;
    myplane[...];

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)
## Machine/Compiler Specifications

- **Scalability Hardware (Cray-XC):**
  - 36-core (72 HT), 128 GB RAM
  - dual 18-core (36 HT) “Broadwell” 2.1 GHz processors

- **Software**
  - CLE 7.0.UP01
  - Intel Compilers 19.0.5.281
  - FFTW 3.3.8.4
  - Chapel 1.20.0
  - Cray 9.0.2 (classic)
Benchmarks

- Use the NAS NPB-FT benchmark
  - NPB v3.4
  - Class D (2048 $\times$ 1024 $\times$ 1024), E (8$\times$), F(8$\times$)
- Compare Chapel, MPI reference and UPC (with non-blocking overlapped comm)
- MPI and UPC use pencil decompositions for large problems/node counts.
- MPI and UPC use 32 cores/node (require a power of 2)
  - Restricting Chapel to 32 cores does not significantly change timings, indicating memory/communication bound.
Chapel FFTs Scale Well Across Nodes: Class D

NPB-FT (Size D) Performance

Locales (x 36 cores / locale)

Gop/s

Chapel
UPC
MPI
Chapel FFTs Scale Well Across Nodes: $E = 8 \times D$

NPB-FT (Size E) Performance

- Chapel
- UPC
- MPI

Locales (x 36 cores / locale)

Gop/s
Chapel FFTs Scale Well Across Nodes: $F = 8 \times E$

NPB-FT (Size F) Performance

- Chapel
- UPC
- MPI

Locales (x 36 cores / locale)

Gop/s
Simulations Scale Well

ULDM Simulation (100 Steps)

Locales (x 36 cores / locale)

Time (s)

Ng=512
Ng=1024
Ng=2048
Ng=4096

Padmanabhan et al

ULDM in Chapel
Writing a Research Code

**Plumbing**

- Parallel HDF5 for saving full simulations (interop with MPI)
- HDF5 Tables for serializing arrays of records
- (C)TOML for input parameters

*All of the above are good examples of C interop.*

**Analysis**

- Summary statistics run in-line; need to be fast.
- eg. Density/Energy profiles/histograms

*These get modified often; do not want to introduce a bottleneck.*
Computing Density Profiles/Histograms : Elegant

```chapel
var counts : [ProfDom] real;
forall (i,j,k) in Dom with (+ reduce counts) {
    const rr = sqrt(i*i + j*j + k*k):int;
    counts[rr] += 1.0;
}
```

‘Dom’ is a distributed domain

That’s elegant!
Timings : Elegant

Degraded perf more severe at higher locale count

![Graph showing time against number of locales]

Time(s)

NumLocales (x44 cores)

Elegant
Computing Density Profiles/Histograms : Atomic

```chapel
var priv_counts : [PrivateSpace][ProfDom] chpl_processorsAtomicType(real);
var counts : [ProfDom] real;
forall (i,j,k) in Dom {
    const rr = sqrt(i*i + j*j + k*k):int;
    priv_counts[here.id][rr].add(1.0);
}
for iloc in 0.. #numLocales {
    var loc_counts = priv_counts[iloc];
    forall ii in ProfDom do counts[ii] += loc_counts[ii].read();
}
```

Local counts
Avoid network atomics

Manual reduction
Timings: Atomic

Atomic contention?

Time(s)

NumLocales (x44 cores)

Elegant

Atomic
Computing Density Profiles/Histograms: SPMD

```chapel
var priv_counts : [PrivateSpace][ProfDom] real;
var counts : [ProfDom] real;
coforall loc in Locales do on loc {
    const myDom = Dom.localSubdomain();
    ref mycounts = priv_counts[here.id];
    forall (i,j,k) in myDom with (+ reduce mycounts) {
        const rr = sqrt(i*i + j*j + k*k):int;
        mycounts[rr] += 1.0;
    }
}
for iloc in 0.. #numLocales {
    var loc_counts = priv_counts[iloc];
    forall ii in ProfDom do counts[ii] += loc_counts[ii];
}
```

SPMD, local reduction
Timings : SPMD

comm congestion?

![Graph showing timings for SPMD and Elegant. The y-axis represents Time in seconds, and the x-axis represents NumLocales (x44 cores). The graph shows two lines: one for Elegant and one for SPMD. The line for SPMD starts at a lower point and then increases significantly, indicating potential communication congestion.]
var counts : [ProfDom] real;
coforall loc in Locales do on loc {
    const myDom = Dom.localSubdomain();
    var mycounts, recvbuf : [ProfDom] real;
    forall (i,j,k) in myDom with (+ reduce mycounts) {
        const rr = sqrt(i*i + j*j + k*k):int;
        mycounts[rr] += 1.0;
    }
    MPI.Barrier(CHPL_COMM_WORLD);
    if (here.id==0) {
        MPI_Reduce(mycounts[0], recvbuf[0], (2*Ng):c_int,
                    MPI_DOUBLE, MPI_SUM, 0, CHPL_COMM_WORLD);
        counts = recvbuf;
    } else {
        MPI_Reduce(mycounts[0], recvbuf[0], (2*Ng):c_int,
                    MPI_DOUBLE, MPI_SUM, 0, CHPL_COMM_WORLD);
    }
}

Use MPI if needed
Timings: MPI

Could likely match MPI with a parallel reduction

![Graph showing comparison between Elegant and MPI in terms of time (s) vs number of locales (x44 cores).]
Science, Powered by Chapel

In Active Use

- Actively being used by Luna Zagorac for her thesis.
- All plots/movies are courtesy Luna, simulations are Chapel powered!
- Code is being actively developed, with new science modules being added in!

Binary Collisions

- Cosmological structures form hierarchically.
- Run simulations colliding pairs of solitons, exploring different initial conditions.
- Final state of system?
- Time scales involved?
Mass Density

Momentum Density

Cumulative Mass in Soliton

Cumulative Potential Energy in Soliton
Where Chapel could do better

1. Tooling
   - Identifying communication - how much and from where? How to recognize a sub-optimal pattern.
   - Easier profiling
   - Compiler improvements, including speed.

2. Easier to express low-level communication/locality
   - Low level communication primitives are not exposed to user (useful when the user can reason better about the communication patterns).
   - Verbose to express locality of computation and have the compiler optimize appropriately.

3. Fewer hidden performance traps
   - Unexpected communication
   - Promotion of operations over N-d arrays can be slow.

None of these are new issues to the Chapel team (and many have Github issues).
My Thoughts

• HPC:
  ○ Productivity: Chapel design has scientific codes in mind.
    • Domains/Arrays
    • Expressive Parallelism - where/when you need it.
    • Interoperability - C (and now Python!)
  ○ Performance: Chapel code can perform(scale very well without heroic efforts.

• It’s a fun language to write. Easy to throw together prototype code in. And it largely does the right thing!

• I’m getting to the point where I’m just working in Chapel.