Parallelized Physics Simulations in Julia

Liam Doherty

Outline

The Julia Ecosystem

- DifferentialEquations.jl, MLJ.jl, Flux.jl, Optim.jl, SciML
- Centered around pure Julia
- Focused on C-like performance with best syntax fused from Python, MATLAB, R

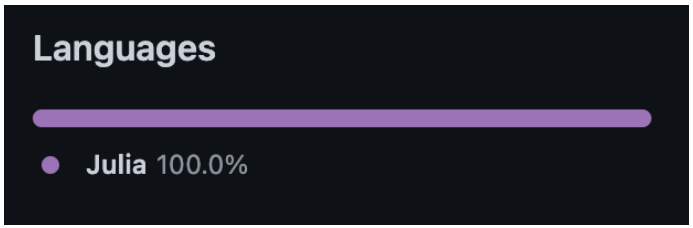

Figure: Flux.jl Languages

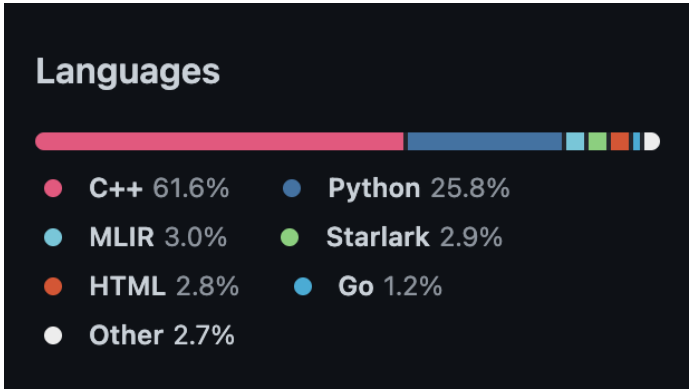

Figure: Tensorflow Languages

The Julia Ecosystem

- Fantastic support for differential equation solving (ODEs, PDEs, SDEs, etc.)
- Extremely flexible options for parallelism/performance engineering including multithreading, distributed computing, GPU acceleration via CUDA, and more!
- Also domain-specific packages are plentiful: BioJulia, JuliaImages, JuliaDynamics, and QuantEcon just to name a few.

The Problem

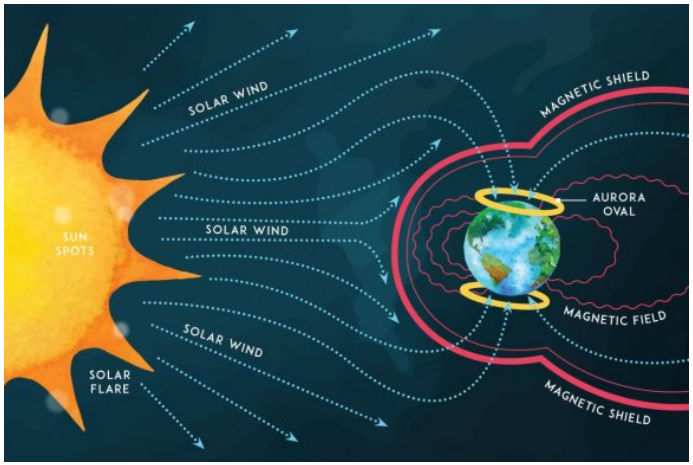

Figure: Magnetic Field Lines

Figure: Aurora Produced

The Problem

System of ODEs (after normalization and dropping small terms):

$$\dot{p} = -\frac{\mu}{\gamma}\frac{\partial b}{\partial \lambda}\frac{\partial \lambda}{\partial z} - kU(\lambda,\mu)\cos(\varphi)$$
$$\dot{\mu} = -U(\lambda,\mu)\cos(\varphi)$$
$$\dot{\lambda} = \frac{\partial \lambda}{\partial z}\frac{p}{\gamma}$$
$$\dot{\varphi} = \frac{\chi b(\lambda)}{\gamma} + \frac{\partial U}{\partial \mu}\sin(\varphi) + \chi(k\frac{p}{\gamma} - \omega_m)$$

Here, λ, p are the coordinate and momentum, and μ, φ are the angular momentum and gyrophase of a charged particle (i.e., an electron) traveling in Earth's magnetic field.

The Problem

- Want to integrate 20,000 trajectories (16 initial angles; 5 initial energy levels each affecting p, μ, λ) and 250 (random) initial phases φ for each angle-energy pair.
- These trajectories are independent, so they can be computed in parallel.
- We look to integrate from $t = 0$ to $t = 3000$, stopping the trajectory if at any point $\mu < \mu_{min} = 0.001$.
- Finally, we want statistics (how many/when did trajectories terminate early, compute their ending energy).

Solving and Performance Analysis

There are a few things that need to be done for implementation:

- Code the system in a flexible way (module, similar to a class in OOP)
- Code a solver
- Stay careful to make serial code as efficient as possible
- Parallelize solving across large sets of initial conditions
- Test the results!

Solving and Performance Analysis

- A lot of the heavy lifting is done for us (at least in terms of the solver) through DifferentialEquations.jl. We use a basic 4th order Runge-Kutta method for our system.
- Keeping type stability (e.g., making sure variables don't change from integers to floats) makes serial code much faster.
- Declaring physical constants in a non-mutable structure with predefined types helps with this!

But how do we parallelize? Again, DifferentialEquations.jl comes to the rescue!

- Ensemble problems allow us to "remake" our ODE system each evaluation of the system, so we can set our initial conditions to the next set in our list for each trajectory.
- This framework has options for serial, multithreaded, distributed, and GPU-accelerated evaluation.
- For our comparison, we use serial as a baseline, and then distributed for parallelism. For larger simulations, we can use GPU-based acceleration.

Solving and Performance Analysis

Now, it is time to see how the distributed code does against the serial code...

Solving and Performance Analysis

Now, it is time to see how the distributed code does against the serial code...

```
Beginning Parallel...
262.239733 seconds (14.21 M allocations: 821.738 MiB, 0.07% gc time, 0.87% compilation time)
Beginning Serial...
1690.302925 seconds (1.58 M allocations: 100.209 MiB, 0.00% gc time, 0.12% compilation time)
julia>
```

Figure: 6.45 times speed-up!

Here we ran 100 identical trajectories, with the distributed version actively making use of 10 Julia processes.

Solving and Performance Analysis

- At this scale, it doesn't quite make sense to use GPUs yet.
- We will be considering how modulation to the input of the system affects the output statistics; stay tuned for results there!
- Going forward, we will be using Drexel's Picotte cluster for our computations.

Future Directions for Research

- Data-Driven Physics
- Physics-Informed Neural Networks (PINNs)
- Large-scale multiphysics modeling
- Parallel computing methods for all of the above

Thank you!