Porting IDL programs into Python for GPU-Accelerated In-situ Analysis

Damir Pulatov
NCAR & University of Wyoming
Partner: Bo Zhang
Mentors: Supreeth Suresh, Cena Miller

July, 2021
MURaM is the primary solar model used for simulations of the upper convection zone, photosphere and corona.

100x acceleration is needed to keep up the simulation with the real time data from telescope.

MURaM have been ported to use scalable GPUs to achieve this!

As computation is optimized, I/O and post processing becomes the next major bottleneck.

Thus, both converting this workflow to an in-situ approach and a staging-based IO subsystem for this in-situ workflow are critical problems need to be addressed.
• One bottleneck is post processing analysis

• A way to reduce the bottleneck is to parallelize data analysis

• Current analysis programs are in IDL

• IDL is proprietary has a small community (astrophysics researchers)

• Python is a better choice for analysis: open source, large library selection, can be optimized for different hardware
Goals

- Port analysis IDL programs into Python
- Optimize Python code (better data structures, efficient libraries, etc.)
- Parallelize Python code for both CPUs and GPUs
- Integrate Python analysis scripts with the larger workflow
- If time permits, look into automating IDL to Python conversion
Algorithm

Main

read file

all points done?

trace points

no

yes

all points done?

trace points

no

trace points

tracing in a positive direction

yes

tracing in a negative direction

combine results
Top View

Python

top view on field lines

IDL

top view on field lines
More about my experience + comparison of the two languages: https://wiki.ucar.edu/display/~dpulatov/Comparison+of+IDL+and+Python
Benchmarking

IDL vs. Python

- IDL: 0 seconds
- Python: 20 seconds

Language: IDL, Python
Code Profiling

Python:

#1: 6.189  _evaluate_linear  scipy/interpolate/interpolate.py:2534  call tree depth: 4
#2: 5.111  [self]  call tree depth: 5
#3: 4.326  _find_indices  scipy/interpolate/interpolate.py:2554  call tree depth: 4

IDL:

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Count</th>
<th>Only(s)</th>
<th>Avg.(s)</th>
<th>Time(s)</th>
<th>Avg.(s)</th>
<th>LinesRun</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLARR</td>
<td>(S)</td>
<td>3</td>
<td>1.190213</td>
<td>0.396738</td>
<td>1.190213</td>
<td>0.396738</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FINDGEN</td>
<td>(S)</td>
<td>2</td>
<td>0.000021</td>
<td>0.000010</td>
<td>0.000021</td>
<td>0.000010</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FLTARR</td>
<td>(S)</td>
<td>1</td>
<td>0.000071</td>
<td>0.000071</td>
<td>0.000071</td>
<td>0.000071</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HELP</td>
<td>(S)</td>
<td>1</td>
<td>0.000046</td>
<td>0.000046</td>
<td>0.000046</td>
<td>0.000046</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INTERPOLATE</td>
<td>(S)</td>
<td>4488</td>
<td>1.003250</td>
<td>0.000224</td>
<td>1.003250</td>
<td>0.000224</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Numpy
- Numerical computation library for Python
- Fast array operations written in C

Xarray
- Extends Numpy with labels
- Intuitive data access thanks to metadata
- Tailored to work with NetCDF format
Xarray allows easy read/write with Zarr/NetCDF formats
Implemented a variable reader for MURaM that saves data into Zarr
Zarr is format for storing compressed, chunked arrays

```python
xarray.Dataset

- Dimensions: (x: 286, y: 144, z: 576)
- Coordinates: (0)
- Data variables:
  - vx (z, x, y) float32 3.898e+03 519.4 ... -1.061e+06
  - shape: (576, 288, 144)
  - by (z, x, y) float32 -412.6 -320.7 ... -0.6506 -0.4422
  - bx (z, x, y) float32 -352.5 -126.0 214.5 ... 13.29 13.28
  - bz (z, x, y) float32 623.5 -419.3 ... 2.681 2.735
  - rho (z, x, y) float32 0.0004166 0.0004166 ... 1.654e-16
  - vy (z, x, y) float32 537.7 307.7 ... -4.539e+05
- Attributes:
  - description: MURaM files converted into zarr format
```
Parallelism in Python

Dask
- Library for parallel computing
- Integrates well with Numpy and Xarray

Cupy
- Array library for GPU computing
- Almost drop-in replacement for Numpy

Numba
- Just-in-time compiler for Python
- Translates Python to machine code

Cython
- Static compiler
- Makes writing C extensions easy
Exploring Parallelism

There are two potential routines to parallelize: tracing and interpolation. Both were explored during this stage.

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dask</td>
<td>Algorithm too complex for Dask to parallelize</td>
</tr>
<tr>
<td>Cupy</td>
<td>Limited support for Scipy functions in our</td>
</tr>
<tr>
<td></td>
<td>implementation</td>
</tr>
<tr>
<td>Numba</td>
<td>No parallelization due to mixing of data types</td>
</tr>
<tr>
<td>Cython</td>
<td>No parallelization due to GIL in CPython</td>
</tr>
</tbody>
</table>
Future Work

- Reimplement interpolation in C++ with native support for parallelism instead of Python

- idlwrap library provides IDL-like interface for Python
  Not complete, possible avenues for improvement

- Using/extending IDL to Python translators
  pyIDL, Pike, i2py
  None are complete, all projects are abandoned
Acknowledgement

- Administrative: Jerry Cycone, Max Cordes Galbraith, Virginia Do, AJ Lauer

- Technical & Feedback: Anderson Banihirwe, Sheri Mickelson, Jian Sun, Brian Dobbins, John Dennis, Richard Loft
Questions?