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

Damir Pulatov, Bo Zhang, Supreeth Suresh, Cena Miller

National Center for Atmospheric Research

Motivation

- As computation is optimized, I/O and post processing becomes the next major bottleneck in MURaM workflow.
- After I/O challenges are addressed, the post processing analysis scripts written in IDL need to be ported and optimized to take advantage of GPUs.
- IDL is proprietary and has a small community of programmers (mostly astrophysics researchers).
- Python is an alternative worth exploring for analysis: open source, large community, can be optimized for different hardware.

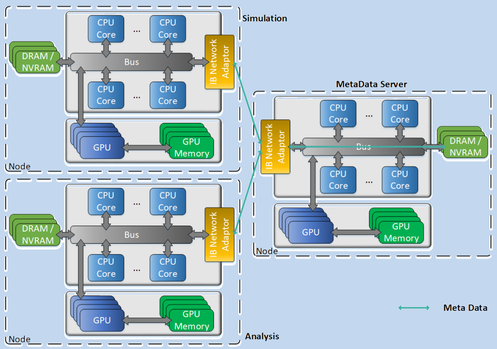

Fig. 1: Proposed workflow to address the I/O issues by GPU-to-GPU communication with the help of DataSpaces framework. The analysis is shown in the lower square.

Goals

- Port analysis IDL programs into Python.
- Optimize Python code.
- Explore parallelization of Python program for both CPUs and GPUs.

Porting

- Porting done with Numpy, Scipy and Xarray.
- Automatically extracted and converted data from MURaM into Zarr.
- Zarr is format for storing compressed, chunked N-dimensional arrays.

Plots

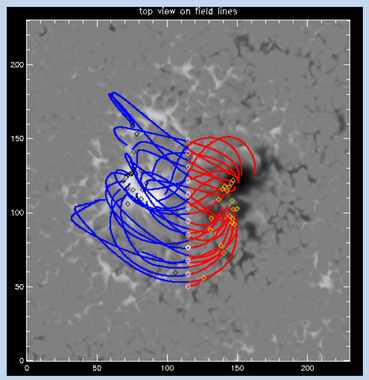

Fig. 5: Top view plot of results produced by IDL

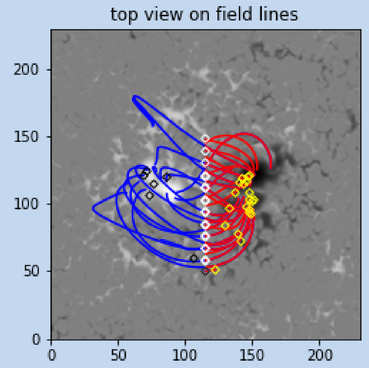

Fig. 6: Top view plot of results produced by Python

Benchmarks

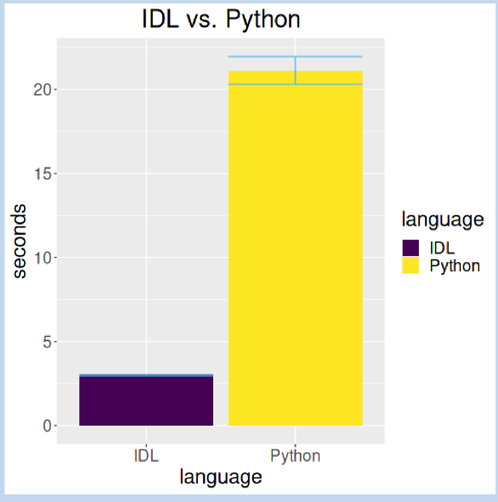

Fig. 7: Benchmark of two implementations of the same algorithm

Figure 7 shows the benchmarking information extracted for both IDL and Python programs. The data represents ten runs on Casper for each IDL and Python implementations. The runs are aggregated with mean and error bars. Lower and upper whiskers show standard deviation of runs for each implementation. IDL is almost 10 times faster than Python because IDL's interpolation is parallel by default.

Parallelism

There are two potential routines that could be parallelized: tracing and interpolation. These were were both explored through the use of several libraries.

Focus	Library	Explanation
Tracing	Dask[1]	Algorithm too complex for Dask to parallelize
	Cupy[2]	Limited support for scipy functions used in our implementation
Interpolation	Numba[3]	No parallelism due to mixing of different data types
	Cython[4]	Issues with GIL (Global Interpreter Lock) in CPython

Fig. 8: Results of exploring parallelism with various libraries

Future Work

- Investigate the use of alternative Python implementations (Jython, IronPython, etc.) due to CPython's limitations.
- Implement interpolation in C++.
- Investigating the use of transcompilers.
- Extend idlwrap Python library which provides IDL-like interface for Python

Acknowledgements

Administrative: Jerry Cyconne, Max Cordex Galbraith, Virginia Do, AJ Lauer
Technical: Anderson Banihirwe, Sheri Mickelson, Jian Sun, Brian Dobbins, John Dennis, Richard Loft

References

[1] Dask Development Team. Dask: Library for dynamic task scheduling. 2016. url: https://dask.org.

[2] Ryosuke Okuta et al. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS). 2017. url: http://learningsys.org/nips17/assets/papers/paper_16.pdf.

[3] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-Based Python JIT Compiler". In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. LLVM '15. Austin, Texas: Association for Computing Machinery, 2015. isbn: 9781450340052. doi: 10.1145/2833157.2833162. url: https://doi.org/10.1145/2833157.2833162.

[4] S. Behnel et al. "Cython: The Best of Both Worlds". In: Computing in Science Engineering 13.2 (2011), pp. 31–39. issn: 1521-9615. doi: 10.1109/MCSE.2010.118.