Personal Information
Name: Debjoy Saha
Major: Electronics and Electrical Communication Engineering
University: Indian Institute of Technology, Kharagpur
Github: @Debjoy10
Email: sahadebjoy10@gmail.com
Phone: (+91)8448508987
Address: Flat No. 3/A, Sakuntala Apartment, 224-A Anandamath, Ichapur, North 24 PGNS, West Bengal, India, PIN - 743144
Timezone: Indian Standard Time (UTC+05:30)

Project proposal for Google Summer of Code 2020

Optimisation of TARDIS-SN codebase using Numba

OVERVIEW

TARDIS-(Temperature And Radiative Diffusion In Supernovae) is an open-source Monte Carlo radiative-transfer spectral synthesis software for running 1D models of supernova ejecta. TARDIS software needs to process loads of data ranging from explosion time and model information to detailed composition of its constituent elements. With such a large number of parameters, the bottlenecks of the simulation are iterations and matrix operations, both of which can be optimised using suitable libraries and optimal algorithms. Also, due to multiple operations taking place simultaneously, there is a huge scope of parallelization.

This project aims at optimising TARDIS software in these areas using Numba.

Numba: Numba is a just-in-time(JIT) compiler for python that optimises an already existing codebase written using numpy with minimal changes. Numpy arrays, functions and operations(broadcasting, assignment, loops etc) are compiled by Numba to machine code that can run at native machine code speed.

Numba has many decorators for specific purposes, the principal ones being-

1. `@jit` - Basic numba decorator. Very versatile.
2. `@njit` - Alias for `@jit` with nopython mode, that allows it to run without the involvement of a python interpreter. It does not support some functions.

3. `@vectorize` - Converts functions taking scalar inputs into NumPy ufuncs.
4. `@guvectorize` - Allows users to write ufuncs working on an arbitrary number of elements of input arrays.
5. `@jitclass` - A `@jit`-aware class. All methods of a jitclass are compiled into nopython functions.

Using Numba with Dask: Dask is another parallel computing library in python which has built-in support for some numba decorators. Using Numba and Dask decorators simultaneously may/may not be the favourable step for every function. There is a limit to the optimisation that one can achieve for a specific function. An additional decorator to an already optimised function will only include the overhead computation of compiling that extra decorator, as described in detail in this notebook.

1. **Dask** is better when you have a complicated task-graph where there is scope for parallelization(Sample PR - #1108).
2. **Numba** is more versatile and usable under all scenarios. It optimises everything, from multiple sequential array operations to nested for loops(Sample PR - #1111).

For most functions, using only one of them is usually more effective. A basic assumption on which tool to be used can be made based on the task-graph and computations involved.

I will be focussing on Numba since its integration with code-base is comparatively easier and it is way more versatile. The principal task is to properly identify the type of decorator(among `@jit`, `@vectorize`, `@guvectorize` etc), and modify the function definition for maximum improvement. Moreover, Numba provides automatic parallelization (`parallel`) and even an option of writing ufuncs in python. Speed up varies depending on the application but can be one to two orders of magnitude.

DETAILED PROJECT DESCRIPTION

The following improvement **milestones** are proposed.

Optimisation of Monte-Carlo scripts

The Monte Carlo module contains all programs related to probabilistic techniques for modelling the radiative transfer problems in supernovae. Current monte-carlo scripts use

Cython and C for fast computation at present, which contains a lot of extra boilerplate code that makes the code harder to follow and more difficult to maintain. Integration with numba can provide better performance and at the same time, is simple and easier to maintain. The framework for integration with Numba had been developed previously in this PR, where I made some further improvements in this PR.

Changes introduced in said PR -

- Replaced lazily compiled `@jit` with eagerly-compiled `@vectorize`, used like- `@vectorize([float64(float64, float64, float64, int64)]).` It improved performance by 0.2 seconds for the compilation, at just the expense of additionally specifying the argument data types and return type of the function(it's signature). Converting into an eagerly-compiled function reduces compilation time since, for lazy compilation, the kernel is dynamically compiled at runtime, which needs extra processing power.
- For functions that have non-numeric(bool/object) inputs cannot be directly improved by using `@vectorize`. They were broken down into two separate functions, a jitted wrapper function which extracted numeric data from all variables and called the other vectorized function to perform all computations.

```
@vectorize(signature)
def calculate(numerical_parameters):
     return_val = ..... # All Computations
     return return_val

@njit
def wrapper(parameters):
    numerical_parameters = unwrap(parameters)
     return calculate(numerical_parameters)
```

- Vectorized functions cannot have multiple return values. In case of multiple return values, the wrapper function calls the vectorized function with different values of a temporary variable and depending upon that value, the vectorized function returns one of each return variable at a time. The wrapper function, therefore, calls the actual function once for each return value.

  Even though this method involved some extra calculations, time of compilation decreased by 0.2 seconds.

- Parallelized all for loops using numba's automatic parallelization using keyword `parallel=True` and `prange`.

Key deliverables would be the complete integration of monte-carlo scripts with Numba, including the above changes as well as others.

Optimisation of Plasma scripts

The role of the plasma module is to determine the ionisation and excitation states of the elements of the supernova ejecta, given the basic structure, including the elemental abundances, densities and radiation temperature. Similar optimisation problems are present as in the monte-carlo module. Key deliverables for this module-

- Complete integration of classes with `@jitclass`.
- Optimise looping with `@njit` and `prange`.
- Try to maximise numpy array broadcasting to allow usage of eagerly compiled decorators `@vectorize` and `@guvectorize`.
- Convert all cython code to python and numba code. Like in this PR, where I converted plasma utility function macro_atom.py, which was initially written in Cython.

Optimisation of classes in other scripts with @jitclass

Model, gui, io directories mostly contain classes related to different phenomena and their helper functions. Integration with `@jitclass` would be the major objective. Further scope of improvement, if observed, will be included. Also, scripts, stats, util directories contain many small utility functions that can be improved slightly by integrating with Numba `@jit` and `@vectorize`.

Integration and Testing

Extensive testing will be performed after the integration of each feature and overall compatibility and semantic correctness will be thoroughly tested once a week. This testing will entail-

1. **Semantic and syntactic correctness -** Whether code compiles for all possible input values and produces the same output as the previous version. Also, properly test all scripts that import the changed function. Common errors -
    a. Mismatch in the eager compilation signature and the datatypes of the arguments they are called with.
    b. Calling any function incompatible with nopython mode while in nopython mode.
2. **Improvement testing -** Note improvements due to changes in the script. Also, document the change in performance due to other decorators and why currently used one is the best.
3. **Profiling -** Profile the code in between changes to identify the key bottlenecks of the code.
4. **Overall integration -** Edit the overall testing script, to include changes(if any), The documentation should be updated accordingly.

Quickstart for Numba

An interactive playground notebook should be included for new users and developers unfamiliar with Numba to get insights on how and why numba is used in the TARDIS codebase. Also, if possible, it should be integrated into **Quickstart.**

Additional Objectives

- **Quickstart for developers**: The current Quickstart in TARDIS-SN website is meant just for users. There is no documentation regarding the scripts and the explanation of the physical quantities that each one of them computes. I propose to write a developer quickstart similar to this example, possibly in collaboration with other contributors, that calls each important function and elaborates on the output and explains their overall integration.
- Contribute to **Integration with Dask**.

DETAILED PROJECT TIMELINE

Phase 0 [Pre-GSOC Period] (April 1 - May 4)

> **Key Milestone:** My major objective will be solving issues and properly understanding the code flow of the entire TARDIS-SN codebase.

Phase 1 [Community Bonding Period] (May 4 - June 1)

> **Key Milestone:** During this period, my major objective will be to discuss the project with my mentors and formulate a roadmap for this project. Slight changes to this timeline can be made during this period. Also, my time will be utilised in properly understanding all Numba functions and how they can be best used. I will also be exploring the other libraries meant for code optimisation in Python.

Phase 2 [Coding Period 1] (June 1 - June 29)

> **Key Milestone:** Principal objective during this period will be finishing all major changes to the monte-carlo codebase and finish documentation. As soon as this goal is achieved, optimisation of plasma scripts will be started.

Phase 3 [GSoC Phase 1 Evaluations] (June 29 - July 3)

> **Key Milestone:** All codes written in coding period-1 will be integrated and documented. The focus will be on submission of a detailed work report, complete with proper tests, improvements and docs for my evaluation.

Phase 4 [Coding Period 2] (July 3 - July 27)

> **Key Milestone:** Optimisation of plasma scripts will be completed. Integration of trivial changes to functions and class improvements will also be finished in this period. All proposed Numba optimisation targets will be completed.

Phase 5 [GSoC Phase 2 Evaluations] (July 27 - July 31)

> **Key Milestone:** Similar to phase-1 evaluations, All changes to the codebase will be properly documented, with proper testing and improvements in the final versions and reports submitted for evaluations.

Phase 6 [Coding Period 3] (July 31 - August 24)

> **Key Milestone:** Focus will be on further testing and improvement of higher-level functions (those that users will usually use) and proper documentation for their usage. Quickstart for developers will also be started during this period.

Phase 7 [Final Evaluation] (August 31 - September 7)

> **Key Milestone:** Documentation and Testing will be finished completely. Quickstart will be completed. The report will be finalised and submitted for final evaluations.

TIME AVAILABILITY

I will be working on the code 7-8 hours a day on weekdays and 3-4 hours on weekends. My working hours will be between 10 am morning (04:30 UTC) to 1 am night (19:30 UTC) (flexible), and am available to work anytime in between. Absences(if any) will be informed well in advance(some are already listed under **OTHER COMMITMENTS** below).

OTHER COMMITMENTS

1. **Aerial Robotics Lab, IIT Kharagpur (Duration: 1 month):** I will be devoting 2-3 hours per day, out of my non-working hours, for mentoring first-year students at my university to prepare for robotics competitions. This will, in **no** way, hamper the quality or quantity of work at TARDIS. Any emergencies, as stated, will be informed in advance.
2. **University Classes (Duration: 1 month or less)*:** If live classes are announced after the lockdown period, their hours can vary from 4 to 8 hours per day and are always within IST 8 AM(02:30 UTC)- 6 PM(12:30 UTC), Monday to Friday. For recorded lectures, I can adjust my time accordingly and the duration will be shorter than the actual classes. My working hours will reduce to 5-6 hours per day during this period. I will make sure to make up for any lost time on weekends.

3. **Semester Exams at university (Period: TBD, Duration: 17-20 days)***
   (Post virus-outbreak schedule will be communicated as soon as possible)

    a. **Preparatory phase (Duration: First 7-10 days):** During this period, my working hours will be reduced to 2 hours a day, unless urgent. I would try to remain updated on Gitter and look out for any updates on my submissions or issues.
    b. **Examination phase (Rest of the duration):** I will be completely unavailable during this interval. I will, however, try to respond to all mentions on Gitter and attend to any immediately solvable issues.

*Tentatively in Community Bonding period and Coding Phase 1.

WHY TARDIS-SN?

Some things that I have been interested in during my brief stint with open source development have always been solving those little optimisation problems that improve the quality and performance of the code-base and produce optimal end programs. Also my language of expertise is mostly Python and C++ and I have little experience with other programming languages.

I found my ideal project in the Numba optimisation project that only demands expertise in python functions and libraries. Selecting this project ensured that I can make myself productive from day 1 and waste less time getting familiar with the programming requirements, keeping in mind the time-bound. That said, I am always open to learning any new skills that might come up during the project.

TARDIS has made immense contributions in helping the world understand the inner workings of the universe. I am very excited to work with the brilliant multi-disciplinary team here at TARDIS. It will surely provide me with good exposure at working with open-source projects and as an added advantage, I might also learn some things about supernovae.

SOMETHING ABOUT ME

I am a second-year BTech-Mtech Dual degree student at IIT Kharagpur, India, majoring in Electronics and Electrical Communication Engineering and a student member of undergraduate research group Aerial Robotics Lab. Involvement with multiple robotics software development and perception projects over the last year has helped me gain experience in Python, C++ and all related libraries, ROS, OpenCV and Tensorflow.

I am quite interested in software development and open-source. I believe in maintaining clear documentation of my work for showcasing my project and aiding those who develop it further in the future(Check out some of my ongoing personal projects on my github page). I am regular and quite dedicated. I properly respond to all mentions and can perform under pressure.

When I am not working, I will probably be playing the keyboard, playing table-tennis or listening to songs.

I am applying for GSOC for the first time, and TARDIS is the only organisation for which I am submitting a proposal. I am excited and very hopefully anticipating a fruitful next couple of months.

ALL PRs made till now (Also included in relevant contexts above)

Optimisation of Montecarlo scripts using numba

Convert cython code in macro_atom to numpy with numba

[First Objective] Optimised function intensity_black_body using Numba.

[First Objective] Running multiple TARDIS instances in parallel using DASK