ADonea_2024b Report

This Merit Allocation Project was supported as a code audit, to help better understand the nature of the work required to improve performance.

Project Summary

Enhancing Solar Farside Magnetogram Generation through Deep Learning Analysis of Satellite Data

Our primary objective is to optimize an existing deep learning algorithm cGANs for supercomputer platforms, utilizing the latest Python environments, TensorFlow packages, and other dependencies. This will enable us to perform detailed performance testing on solar data pairs. The algorithm's performance is hampered by its slow execution speed and inefficient configuration on GPU/CPU setups. The intrinsic competition between the generator and discriminator within the algorithm aids in generating realistic data. However, a significant challenge lies in assessing the quality of the generated data due to limitations in our testing methods. Our goal is to enhance the computational efficiency of our model, aiming to reduce the processing time from 12 hours to 3 hours for even a larger input of datasets. Additionally, optimizing how the code processes and reads data pairs is essential. This fourfold increase in speed is crucial for deeper insights into the algorithm's functioning.

Key findings

Code design

- The script was initially developed with TensorFlow v2.3.0. However, due to the rapid evolution of TensorFlow and numerous breaking changes since then, the script cannot run "as-is" with the latest TF version (2.17).
- In TF version 2.11, the base class for "optimizers" was changed, however the legacy versions can still be imported from the "legacy" module. i.e.

```
-from tensorflow.keras.optimizers import Adam
+from tensorflow.keras.optimizers.legacy import Adam
```

- This is required in order to make use of the `get_updates()` method.
- Starting from version 2.16, TensorFlow includes Keras version 3 instead of version 2, which significantly alters the API and programming paradigm. Refactoring the code is necessary to use TF version 2.16 or later and Keras 3. Note: it is possible to use Keras 2 with TF >= 2.16 using the `tf_keras` package. See https://keras.io/getting_started/#tensorflow--keras-2-backwards-compatibility
- Additionally, the code is written in TF v1 compatibility mode, i.e.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

- The main consequence of this is that the code executes in graph mode, instead of using eager execution. This is not inherently a bad thing, but using TF v2 may allow the use of newer features to improve performance. Again, it may involve some effort to refactor the code.
- The code uses a custom training loop. "A custom training loop might come in handy when you have complicated models with non-trivial loss/gradients calculation."
- This means we can't use TF `model.fit()`, and therefore can't easily use TensorBoard callbacks for profiling.

Running on OzSTAR

- TensorFlow is a large package with many files, which can cause performance issues on the Lustre filesystem. This makes it challenging to install and work with in a conda or Python virtual environment.
- It is recommended to use the provided TensorFlow module or to containerize the environment with Apptainer.

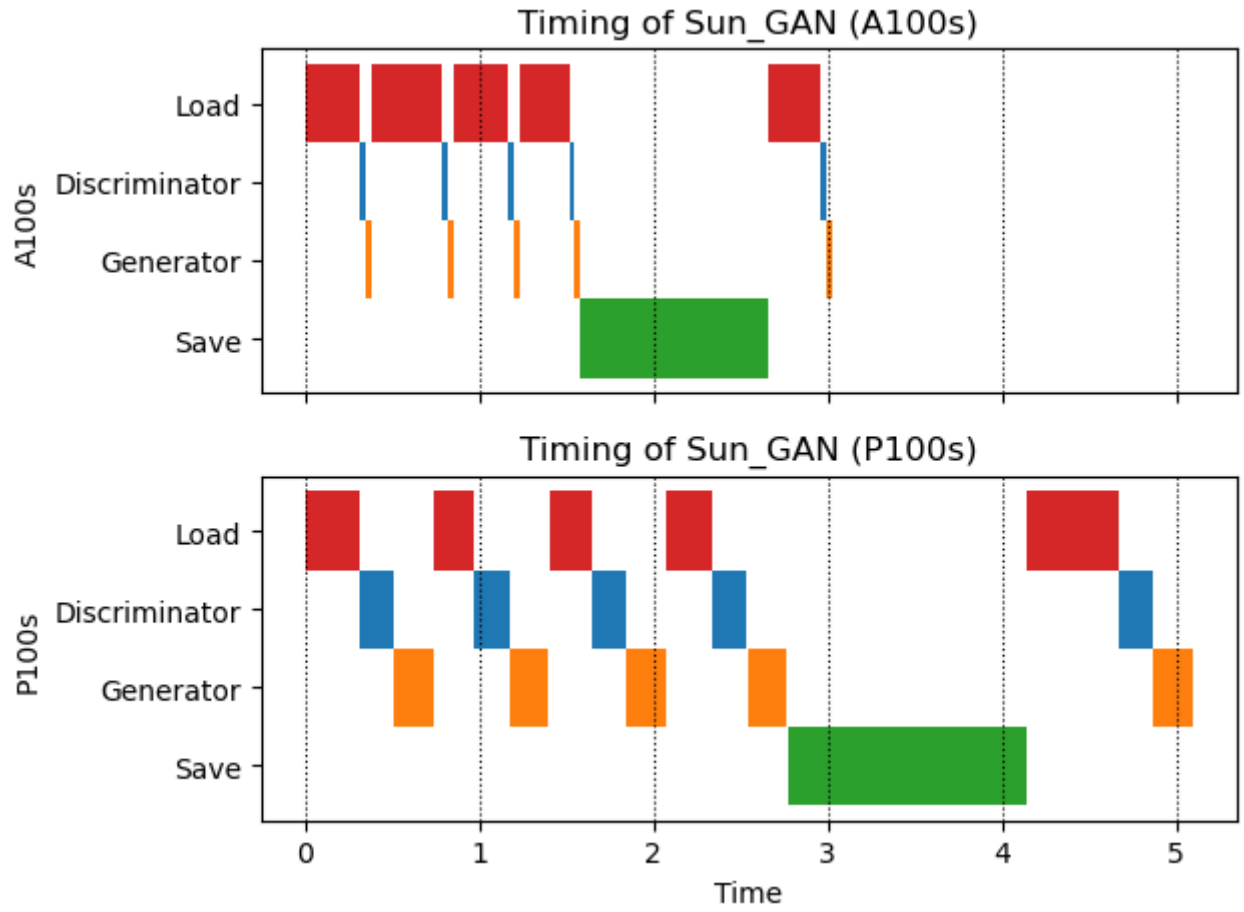
Profiling

- Running the "default" job produced the following job report

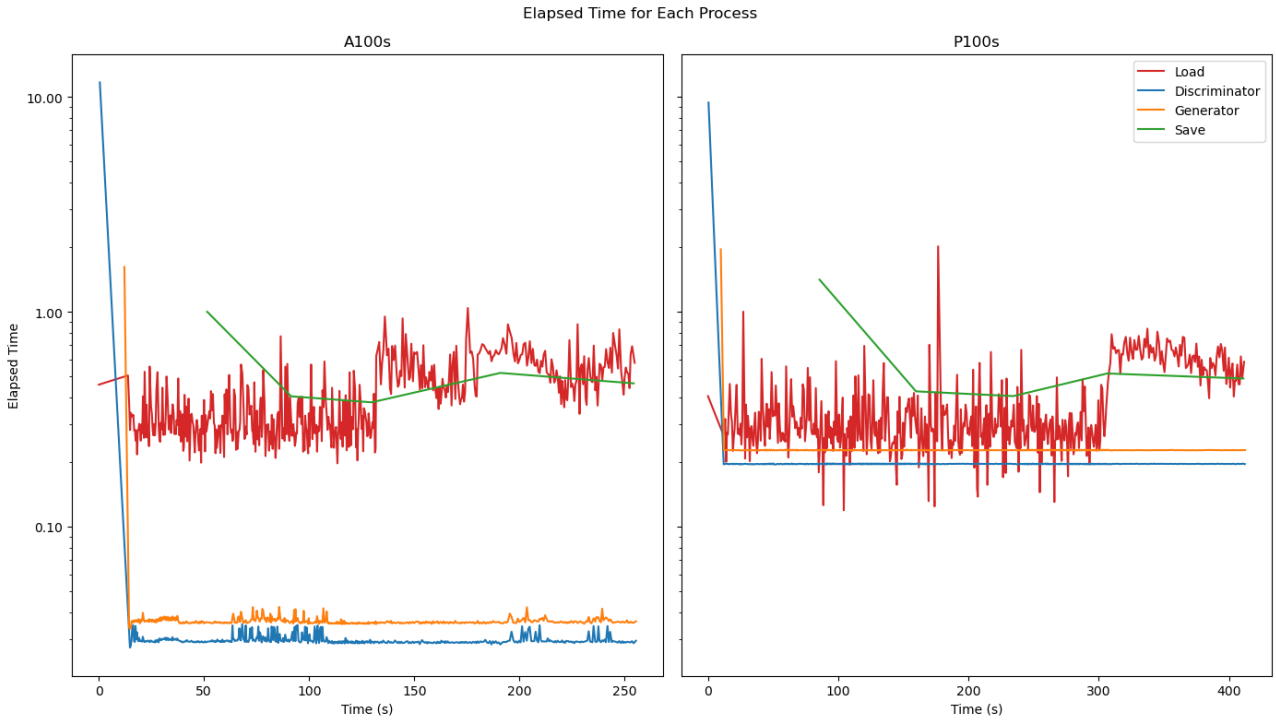
```
$ jobreport 61719952
+------------------ Job Report: 61719952 (CANCELLED) ------------------+
| Memory (RAM)  [#######             ] 35.5% (3.6 GB peak / 10 GB)     |
| CPU           [#                   ]  9.5% average                   |
| GPU           [                    ]  2.6% average                   |
| Time          [------------------> ] 92.5% (0-00:55:29 / 0-01:00:00) |
|                                                                      |
| Lustre Filesystem:                                                   |
|   Path    Total Read    Total Write    Total IOPS                    |
|   ------  ------------  -------------  ------------                  |
|   /apps   1 GB          0 B            21.2 K                        |
|   /fred   80.9 GB       2.2 MB         49.6 K                        |
|   /home   12 KB         180.3 KB       335                           |
|   OS      13.3 MB       0 B            20                            |
|                                                                      |
| Warnings:                                                            |
|   - Too much memory requested                                        |
|   - CPU usage is low                                                 |
|   - GPU usage is low                                                 |
+----------------------------------------------------------------------+
```

- The code is underutilising the GPU/CPU.
- It is heavily reading from the parallel filesystem, creating a bottleneck.
- This was confirmed by adding simple timing measurements to the code, broken down into:
    1. Reading/processing images
    2. Training the discriminator model
    3. Training the generator model
    4. Saving the model to file/disk

- We found that a significant portion of the time is spent on reading/processing images.



- We also found that using newer GPUs (A100s vs P100s) accelerates the training steps by approx x6.5
- The load times remained similar as they are dependent on the underlying parallel filesystem (Lustre).



Suggested improvements

- Establish a set of simple regression tests to aid the development cycle.
- Refactor code to use TF v2 instead of v1, and possibly Keras 3 instead of 2.

- Determine whether the code is best run with eager execution or graph mode.
- Investigate whether the custom training loop is necessary, or if something like `model.fit()` could be used.
- Optimize the input pipeline by overlapping image reading with computation.
    - Utilize TensorFlow datasets, specifically their "prefetching" functionality. See the TensorFlow Data Performance Guide.
    - We may also be able to parallelize the data reading transformation using "interleave"

Useful links

- TensorFlow 1.x vs TensorFlow 2 - Behaviors and APIs
- Migrate from TensorFlow 1.x to TensorFlow 2
- TensorFlow Eager Execution v.s. Graph (@tf.function)
- pix2pix: Image-to-image translation with a conditional GAN
- Better performance with the tf.data API
- How to use Dataset and Iterators in Tensorflow with code samples
- Writing a training loop from scratch

Resourse Assessment for future MAP proposal

- Establish a set of simple regression tests to aid the development cycle [1 week]
- Refactor code to use latest TensorFlow and Keras versions to improve maintainability and facilitate additional feature development [3 weeks]
- Investigate performance impact of different tensorflow execution modes [0.5 weeks]
- Investigate suitability of built-in training loop functions for simplifying the code base [0.5 weeks]
- Optimize the input pipeline by overlapping image reading with computation using TensorFlow datasets [3 weeks]
- [**Total time:** 8 weeks]