TRIUMF Research & Development #48

Memory Profiler

```
Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
   920    236.8 MiB    236.8 MiB           1   @profile
   921                                         def GenerateMultiMuonSample_h5(avg_mu_per_ev=2.5, sigma_time_offset=21.2):
   922                                             """
   923                                             Inputs:
   924                                              avg_mu_per_ev == Poisson distribution mean for number of muons in each spill
   925                                              sigma_time_offset == Width of spill (Gaussian) in nanoseconds
   926                                             """
   927    236.8 MiB      0.0 MiB           1       files = ['event998.npz']
   928
   929                                             # Remove whitespace
   930    236.8 MiB      0.0 MiB           4       files = [x.strip() for x in files]
   931
   932                                             # Check that files were provided
   933    236.8 MiB      0.0 MiB           1       if len(files) == 0:
   934                                                 raise ValueError("No files provided!!")
   935    236.8 MiB      0.0 MiB           1       print("Merging " + str(len(files)) + " files")
   936
   937                                             # Start merging
   938
   939    249.7 MiB     12.9 MiB           1       num_nonzero_events, nonzero_event_indexes = count_events(files)
```

Figure 1 Output of the memory profiler

The memory profiler uses memory-profiler 0.58.0, which is meant to be used for python to analyze the file for its memory usages.

Memory profiler was coded in to provide overall memory usage of the code to diagnose the memory error some of the project members were experiencing. This helped diagnose some of the issues and was also requested by the project clients for future uses.

Simplified Code for GenerateMultiMuonSample_h5()

```
1071        #h5_file.close()
1072        print("Finished")
1073
1074
1075    # In[ ]:
1076
1077
1078    GenerateMultiMuonSample_h5(avg_mu_per_ev=2.5, sigma_time_offset=21.2)
```

Figure 2 Original code containing over 1000 lines

```
247  pr.disable()
248  s = io.StringIO()
249  ps = pstats.Stats(pr, stream=s).sort_stats('tottime')
250  ps.print_stats()
251
252  with open('test.txt', 'w+') as f:
253      f.write(s.getvalue())
```

Figure 3 Simplified code only containing 253 lines

Simplified and cleaned the code that creates Multi Muon Sample to reduce overall run time from over 1078 lines to 253 lines of code. All the code that is not related to running the GenereateMultiMuonSample_h5() is removed.

Original code contains a lot of other functions that are not required to execute GenerateMultiMuonSample_h5() and it causes issues when attempting to debug the said function. Also, for better readability, code is reformatted to better suit the compact need of the modified code.

Code Profiling

```
 9  import cProfile
10  import re
11  import pstats
12  import io
```

Figure 4 Imports required to run cProfiler

```
244  pr = cProfile.Profile()
245  pr.enable()
246  GenerateMultiMuonSample_h5(avg_mu_per_ev=2.5, sigma_time_offset=21.2)
247  pr.disable()
248  s = io.StringIO()
249  ps = pstats.Stats(pr, stream=s).sort_stats('tottime')
250  ps.print_stats()
251
252  with open('test.txt', 'w+') as f:
253      f.write(s.getvalue())
```

Figure 5 Entirety of the cProfile code in the file

cProfile is used to capture the code execution time for each individual functions to check which of the functions is causing the code to be slow. This is useful for optimizing the code so that it can run faster, as even on the Cedar Canada network, which has powerful hardware, it can take hours to fully execute the entirety of the npz file.

Profiler is lightweight so it will not impact code speed in any way. It is also easy to configure, allowing future code that gets added to be included easily as well.

Figure 6 profile.bat code

```
C:\Users\willi\Desktop\BCIT\ISSP\TriumfCNN>profile.bat GenerateMultiMuonSample.py

C:\Users\willi\Desktop\BCIT\ISSP\TriumfCNN>python -m cProfile GenerateMultiMuonSample.py
R= 370.0958251953125 H= 1034.8980712890625
min_x= -370.0958251953125 max_x= 370.0958251953125 diameter= 740.191650390625
min_z= -370.0958251953125 max_z= 370.0958251953125 diameter= 740.191650390625
min_y= -517.4490356445312 max_y= 517.4490356445312 height= 1034.8980712890625
Merging 1 files
Counting Events
10
GenMapping
processing output entry  0  with  5  muons
         867352 function calls (858267 primitive calls) in 6.416 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <__array_function__ internals>:2(<module>)
        6    0.000    0.000    0.000    0.000 <__array_function__ internals>:2(all)
       30    0.000    0.000    0.000    0.000 <__array_function__ internals>:2(any)
      252    0.000    0.000    0.000    0.000 <__array_function__ internals>:2(can_cast)
```

Figure 7 Input and output of profile.bat being used

If the user wishes to run it on the command line and output to a command line as well, profile .bat can be used to call GenerateMultiMuonSample.py with cProfile enabled to profile the entire file.

Bug fixes

IndexError

Figure 8 Index # Error being encountered at the end of the execution

Another bug we encountered was the one involving index # is out of bounds for axis # with size #. This was a bug caused by the problem in the count_events() function which was meant to count the number of events that have 1 or more hits. This bug was worked on by me and another teammate, who eventually fixed the error with assistance.

Research & Documentation

Cedar Canada Network and Singularity

```
[jseo41@cedar1 machine_learning]$ ls -la
total 138
drwxrws---+  5 pdeperio rpp-blairt2k 4096 Nov 16 09:13 .
drwxrws---+ 98 tanaka   rpp-blairt2k 4096 Nov 15 22:17 ..
drwxrws---+  2 pdeperio rpp-blairt2k 4096 Nov 30 15:42 containers
drwxrws---+ 12 prouse   rpp-blairt2k 4096 Nov 28 13:23 data
-rwxrwx---+  1 pdeperio rpp-blairt2k 1002 Oct 22 14:23 enter_container.sh
-rwxrwx---+  1 pdeperio rpp-blairt2k 1410 Oct 22 14:19 jupyter_job.sh
-rw-rw----+  1 pdeperio rpp-blairt2k 1940 Nov 16 09:16 log-jupyter-pdeperio-55497557.txt
drwxrws---+  9 prouse   rpp-blairt2k 4096 Nov 25 15:29 production_software
-rwxrwx---+  1 pdeperio rpp-blairt2k 1065 May  3  2019 start_jupyternotebook.sh
[jseo41@cedar1 machine_learning]$ pwd
/home/jseo41/projects/rpp-blairt2k/machine_learning
[jseo41@cedar1 machine_learning]$
```

Figure 9 Location of the singularity to be used as virtual environment

Researched methods of using the Cedar Computer network, which is provided by the project client. This task included simple tasks such as connecting to the Cedar network itself, retrieving files from the network, transferring file to the network, running python virtual environment to run our code on the network and other tasks.

cProfile Research and Documentations

```
- cProfile
ncalls
    Specifies the number of times the code was called

tottime
    Total time spent on executing this function and it's sub functions

percall
    tottime divided by ncalls to calculate time spent on each calls average

cumtime
    Cumulative Time spent on the function including sub fuctions AND recursive functions

percall
    cumtime divided by primitive calls

Profile funciton is tied to pr variable, which is used to start and stop profiler capture to specify which function to capture, in this
case, is GenerateMultiMuonSample_h5. After the capture, the output is saved as test.txt, which can be used to see which code is causing
the delay
```

Figure 10 cProfile Documentation on the README

For use in the code profiling, cProfile was chosen based on its lightweight code and ease of use. I have done research and basic documentation on how to use and read cProfile output. This includes the variables in the output file of the profiler, multiple methods to run the profiler, and how to edit the profiler to include other codes that could be included in the future.

Document functions that use zlib.Decompress function

This function on its own takes up most of the code execution time. Research into the code is needed to find where in the code this function occurs. Possible code is the decompression of the npz file while being opened.

NumPy File reading

NumPy is rather efficient when it comes to loading large datasets. This is always going to be slow but it can still be improved by using memmap (https://numpy.org/doc/stable/reference/generated/numpy.memmap.html)

Another way to speed up NumPy File loading is using h5py, which I believe is already being used to load multimuonfile during GenerateMultiMuonSample.py

Method 'sort' of 'numpy.ndarray' objects

I believe this is another part of NumPy file reading and needs to be optimized.

Next Steps

More in-depth research into other functions that slow the code down should be researched, as of now running the entire event file can take hours.

Currently, the primary source of delay is SumEvent(), which needs it's file access optimized.

Next function that should be optimized is sorting of the numpty array and pickle load.

External Links

Trello - https://trello.com/b/Tc1VAhX6/group-048

WatChMaL Resources - https://www.watchmal.org/