Simple Ways to Increase the Performance of your Python Code

Sheri Mickelson
Software Engineer, NCAR

AMS 96th Annual Meeting
Monday, January 11, 2016
CISL/TDD/ASAP Group

• Our group focuses on
  – Improving the scalability and performance of NCAR applications
  – Efficient use of accelerator technology
  – Workflow and I/O
Before you can improve the performance of your code, you need to understand your code’s performance and be able to identify the critical hotspot locations ....

This is done through profiling tools
Using cProfile and profile to capture function execution times

Both use similar interfaces and are mostly interchangeable

• cProfile
  – C extension
  – Reasonable overhead

• profile
  – Pure Python
  – Significant overhead
  – Easier to extend
Command Line Interface

Both have command line interfaces that allow users to profile their code without any code modifications

• cProfile
  – Ex: python -m cProfile -s time script.py

• profile
  – Ex: python -m profile -s time script.py
Sample output from cProfile
(profile provides similar output)

36347 function calls (35740 primitive calls) in 9.102 seconds

Ordered by: internal time

<table>
<thead>
<tr>
<th>ncalls</th>
<th>tottime</th>
<th>percall</th>
<th>cumtime</th>
<th>percall</th>
<th>filename:lineno(function)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>8.833</td>
<td>0.177</td>
<td>8.833</td>
<td>0.177</td>
<td>{_hlu.contour_map_wrap}</td>
</tr>
<tr>
<td>50</td>
<td>0.073</td>
<td>0.001</td>
<td>0.073</td>
<td>0.001</td>
<td>{_hlu.NhlDestroy}</td>
</tr>
<tr>
<td>3</td>
<td>0.030</td>
<td>0.010</td>
<td>0.112</td>
<td>0.037</td>
<td><strong>init</strong>.py:1(&lt;module&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>0.030</td>
<td>0.001</td>
<td>0.030</td>
<td>0.001</td>
<td>{_hlu.open_wks_wrap}</td>
</tr>
<tr>
<td>1</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>0.026</td>
<td>simplecomm.py:524(<strong>init</strong>)</td>
</tr>
</tbody>
</table>
More detailed timing information can be gathered with line_profiler

- Gives line by line execution time profiling
- Must be installed
  - `pip install line_profiler`
- Requires a small code modification
  - Users must add `@profile` above functions they would like to profile
Command Line Interface

Ex: kernprof.py -l -v script.py

- kernprof.py is found in the line_profiler install directory
- Use the ‘-v’ option to output the line profiling results on the terminal
- Otherwise, you can view the results by executing:
  - python -m line_profiler script.py.lprof
Sample output from line_profiler

Total time: 8.94924 s
File: better_parallel_plot_all.py
Function: create_plots at line 8

<table>
<thead>
<tr>
<th>Line #</th>
<th>Hits</th>
<th>Time</th>
<th>Per Hit</th>
<th>% Time</th>
<th>Line Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>@profile</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>def create_plots(comm):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>50</td>
<td>978</td>
<td>19.6</td>
<td>0.0</td>
<td>print &quot;plotting &quot; + v</td>
</tr>
<tr>
<td>66</td>
<td>50</td>
<td>8813906</td>
<td>176278.1</td>
<td>98.5</td>
<td>map=Ngl.contour_map(wks, var, res)</td>
</tr>
</tbody>
</table>
For more profiling information, there will be a tutorial on Profiling and Debugging tomorrow morning at 9AM.
Some tips to consider to improve performance in your own code ...
How to use parallelism within your workflow

Parallelism

What can be executed at the same time?
ASAPTools

• Contains wrappers around mpi4py functionality

• Requires:
  – MPI (any flavor)
    • This needs to be installed and in your path before proceeding with the Python installs
  – mpi4py

• Can be installed with
  – pip install [--user] ASAPTools
Steps to run a Python script in parallel with ASAPtools

• Add to the script:

1. from asaptools import simplecomm, partition
2. comm = simplecomm.create_comm(serial=False)
3. local_list =
   comm.partition(global_list, func=partition.EqualStride(), involved=True)

• Execute with ‘mpirun –n <# of tasks> python script.py’
Demo: Plot all 3D variables in a file
Other operations within ASAPTools

• Other MPI Support
  – Multiple partitioning methods
    • EqualLength, EqualStride, SortedStride, and WeightBalanced
  – Sends/Recv
  – Split Communicators
  – Rank/Master/Size Query

• Timers
Questions?

• line_profiler
  – pip install line_profiler
  – https://github.com/rkern/line_profiler.git

• ASAPTools
  – pip install [--user] ASAPTools

• mpi4py
  – pip install [--user] mpi4py
  – https://bitbucket.org/mpi4py/mpi4py

• MPI
  – Building from source instructions: https://mpi4py.scipy.org/docs/usrman/appendix.html#building-mpi-from-sources