Profiling and optimizing RAM and CPU in RMG-Py
Richard West
“Premature optimization is the root of all evil.”

- Donald Knuth, Turing Award Lecture 1974
“Premature optimization is the root of all evil.”
- Donald Knuth, Turing Award Lecture 1974

Not an excuse to write bad code!

1. Get it working
2. Get it working right
3. Test it’s working right
4. Profile it
5. Improve the slow parts, then go to 3
How Python does memory management

Reference counting...
  ...with a garbage collector.

Memory Allocator is in cPython...
  ...and in the OS.

Every object has a dictionary of attributes...
  ...unless you use __slots__ or Cython.
What makes Python slow?

Attribute lookups!

```python
for species1 in long_list:
    for species2 in long_list:
        distance = (species1.molecule[0].atom[0].x -
                    species2.molecule[0].atom[0].x)

for species1 in long_list:
    x1 = species1.molecule[0].atom[0].x
    for species2 in long_list:
        distance = x1 - species2.molecule[0].atom[0].x

newlist = []
for word in oldlist:
    newlist.append(word.upper())

upper = str.upper()
newlist = []
append = newlist.append
for word in oldlist:
    append(upper(word))
```
How to profile RMG-Py for CPU use

$ python rmg.py -p hexadiene/input.py

MODEL GENERATION COMPLETED
The final model core has 9 species and 8 reactions
The final model edge has 140 species and 317 reactions

RMG execution terminated at Wed Jan 22 14:42:49 2014

Profiling Data

Sorted by internal time

Wed Jan 22 14:42:49 2014 /Users/rwest/Code/RMG-Py/examples/rmg/1,3-hexadiene/RMG.profile

25196711 function calls (24439766 primitive calls) in 46.146 seconds

Ordered by: internal time
List reduced from 2468 to 25 due to restriction <25>

<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>477031/34</td>
<td>5.251</td>
<td>0.000</td>
<td>10.022</td>
<td>0.295</td>
<td>rules.py:427(fillRulesByAveragingUp)</td>
</tr>
<tr>
<td>100802</td>
<td>1.812</td>
<td>0.000</td>
<td>1.831</td>
<td>0.000</td>
<td>statmechfit.py:332(hinderedRotor_heatCapacity)</td>
</tr>
<tr>
<td>2</td>
<td>1.668</td>
<td>0.834</td>
<td>1.671</td>
<td>0.835</td>
<td>{Pickle.dump}</td>
</tr>
<tr>
<td>100802</td>
<td>1.563</td>
<td>0.000</td>
<td>1.563</td>
<td>0.000</td>
<td>statmechfit.py:357(hinderedRotor_d_heatCapacity_d_barr)</td>
</tr>
<tr>
<td>475605</td>
<td>1.549</td>
<td>0.000</td>
<td>1.549</td>
<td>0.000</td>
<td>base.py:1105(getAllCombinations)</td>
</tr>
<tr>
<td>440</td>
<td>1.422</td>
<td>0.003</td>
<td>5.334</td>
<td>0.012</td>
<td>{rmgpy.pdep.msc.applyModifiedStrongCollisionMethod}</td>
</tr>
<tr>
<td>594135</td>
<td>1.300</td>
<td>0.000</td>
<td>1.300</td>
<td>0.000</td>
<td>{method 'isSubgraphIsomorphic' of 'rmgpy.molecule.molecule.Molecule' objects}</td>
</tr>
<tr>
<td>11767</td>
<td>1.252</td>
<td>0.000</td>
<td>1.577</td>
<td>0.000</td>
<td>adjlist.py:51(fromAdjacencyList)</td>
</tr>
<tr>
<td>482343</td>
<td>1.242</td>
<td>0.000</td>
<td>1.582</td>
<td>0.000</td>
<td>rules.py:404(getAllRules)</td>
</tr>
<tr>
<td>440</td>
<td>1.240</td>
<td>0.003</td>
<td>1.240</td>
<td>0.003</td>
<td>{method 'generateCollisionMatrix' of 'rmgpy.pdep.configuration.Configuration' objects}</td>
</tr>
<tr>
<td>25919</td>
<td>1.104</td>
<td>0.000</td>
<td>1.104</td>
<td>0.000</td>
<td>{method 'copy' of 'rmgpy.molecule.molecule.Molecule' objects}</td>
</tr>
<tr>
<td>32</td>
<td>1.101</td>
<td>0.034</td>
<td>8.042</td>
<td>0.251</td>
<td>{method 'solve' of 'nudgedPOED' objects}</td>
</tr>
</tbody>
</table>
How to profile RMG-Py for memory use

Just run it

statistics.xls
How to profile RMG-Py for memory use

Just run it
How to profile RMG-Py for memory use

Take your pick of toolkits, libraries, etc.

- Guppy
- Meliae
- Pympler
$ python rmg.py -m example/input.py
Snapshots statistics

2 core 5 edge species 13 seconds snapshot at 00:00:13.42

Total virtual memory assigned to the program at that time was 2.58 GB, which includes 12.14 KB profiling overhead. The ClassTracker tracked 4.66 MB in total. The measurable objects including code objects but excluding overhead have a total size of 0 B.

<table>
<thead>
<tr>
<th>Class</th>
<th>Instance #</th>
<th>Total</th>
<th>Average size</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmgpy.rmg.model.CoreEdgeReactionModel</td>
<td>1</td>
<td>4.66 MB</td>
<td>4.66 MB</td>
<td>0.18%</td>
</tr>
<tr>
<td>rmgpy.rmg.model.Species</td>
<td>7</td>
<td>3.12 KB</td>
<td>456 B</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

3 core 8 edge species 15 seconds snapshot at 00:00:14.94

Total virtual memory assigned to the program at that time was 2.61 GB, which includes 21.41 KB profiling overhead. The ClassTracker tracked 6.87 MB in total. The measurable objects including code objects but excluding overhead have a total size of 0 B.

<table>
<thead>
<tr>
<th>Class</th>
<th>Instance #</th>
<th>Total</th>
<th>Average size</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmgpy.rmg.model.CoreEdgeReactionModel</td>
<td>1</td>
<td>6.87 MB</td>
<td>6.87 MB</td>
<td>0.26%</td>
</tr>
<tr>
<td>rmgpy.rmg.model.Species</td>
<td>11</td>
<td>4.87 KB</td>
<td>453 B</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

4 core 8 edge species 16 seconds snapshot at 00:00:16.02

Total virtual memory assigned to the program at that time was 3.66 GB, which includes 46.01 KB profiling overhead. The ClassTracker tracked 7.66 MB in total. The measurable objects including code objects but excluding overhead have a total size of 0 B.

<table>
<thead>
<tr>
<th>Class</th>
<th>Instance #</th>
<th>Total</th>
<th>Average size</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmgpy.rmg.model.CoreEdgeReactionModel</td>
<td>1</td>
<td>7.66 MB</td>
<td>7.66 MB</td>
<td>0.26%</td>
</tr>
<tr>
<td>rmgpy.rmg.model.Species</td>
<td>11</td>
<td>4.87 KB</td>
<td>453 B</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Other resources

SlideShare.net

 e.g. http://www.slideshare.net/g3_nittala/profiling-and-optimization-14511764

Google.com

 e.g. “Python memory profiling”