Programming Skills: Individual Assessment 2

B083194

November 26, 2015

Contents

1 Introduction

The aim of this project is to conduct performance analysis on the predator-prey model designed in `Coursework 2 - Development`. Key areas of exploration include: optimisation flags, investigating sources of overhead, and varying grid size and proportions. The predator-prey model was based around the population densities of hares and pumas, and the program was written in C. There are a large number of calculations computed in the program.

All performance tests were run on a 2015 i7 MacbookPro with no other programs running. The optimisation flags were run twice on a bash script which cycled through a range of optimisation flags with islands.dat (given). Both minimum and average times were recorded. The times measured for the different grid sizes and grid proportions were also run twice, recording both the minimum and average time measurements.

2 Performance Experiments

2.1 Optimisation Flags

In the absence of optimisation flags, compilers take the path of least resistance to reduce the cost of compilation. Optimisation flags aim to improve the performance of code, with different flags representing different levels of optimisation. With more optimisation, comes more compilation time, and less debugging. The `O0` flag is default if no other flag is specified, and flags with higher numbers correspond to higher levels of optimisation. The optimisation flags `Os` and `Ofast` lie somewhere between numbered optimisation flags. `Os` contains all the optimisation associated with the `O2` flag that do not reduce the size of the code, and then some more which may decrease it. The `Ofast` enables all of the `O3` optimisation with some added math optimisation. Before conducting the performance experiments, we expect `O0` (default) to be the slowest, and `Ofast` to be the fastest due to the large number of calculations within the program.

Figure 1 plots against the minimum time, rather than the average. Although they both produce similar plots, it is better to take the minimum time because it was run on a laptop which could have had something else running in the background. Minimum time is hence more likely to be reproducible. Figure 1 shows that the default optimisation flag takes much longer that its numbered counterparts, as expected. Flags `O1` through `O3` all take much less time, however `01` takes less time than the more aggressive `02` and `03` optimisation flags. Due to the number of calculations, it was estimated that `Ofast` would be quickest, however `Os` had the fastest recorded time. The second fastest was still `Ofast`, perhaps profiling will reveal why.

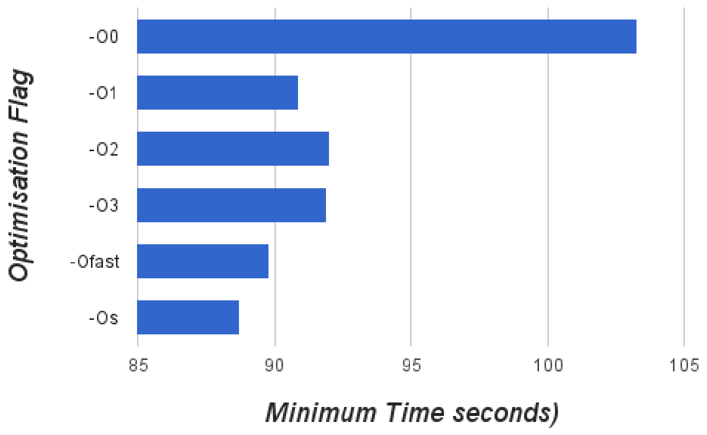
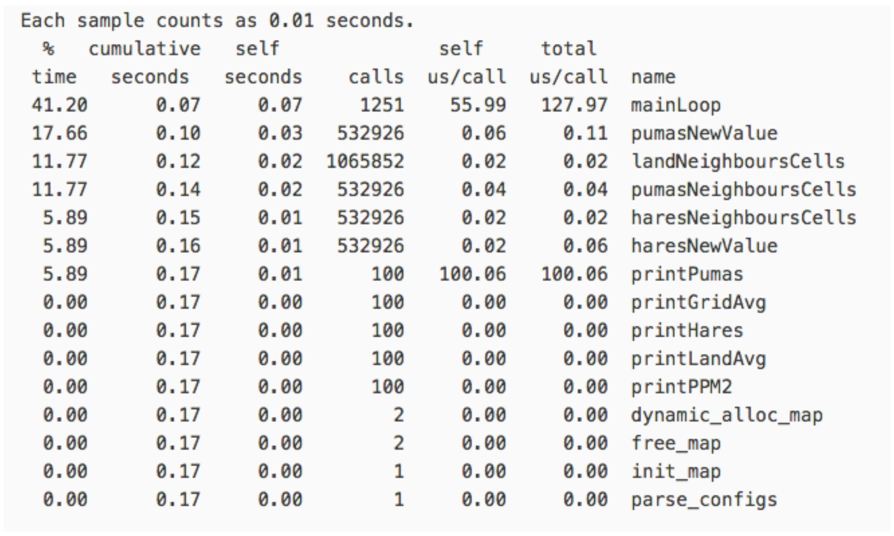


Figure 1: Plot displaying the times taken for each type of optimisation flag

2.2 Profiling

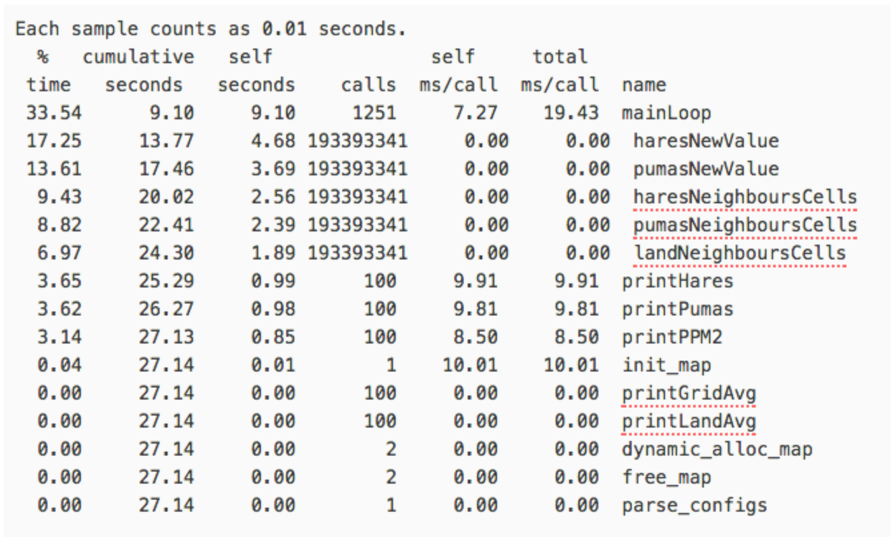
Profiling analyses which parts of a program takes how long, and how many times functions are called. This is particularly useful for assessing which parts of a program can improved to make a significant difference on the overall time. Profiling was carried out for each of the different optimisation flags and was run on small.dat (given). The output for the `O0` is given in Figure 2.



```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  us/call  us/call  name    
 41.20      0.07     0.07     1251    55.99   127.97  mainLoop
 17.66      0.10     0.03   532926     0.06     0.11  pumasNewValue
 11.77      0.12     0.02  1065852     0.02     0.02  landNeighboursCells
 11.77      0.14     0.02   532926     0.04     0.04  pumasNeighboursCells
  5.89      0.15     0.01   532926     0.02     0.02  haresNeighboursCells
  5.89      0.16     0.01   532926     0.02     0.06  haresNewValue
  5.89      0.17     0.01      100   100.06   100.06  printPumas
  0.00      0.17     0.00      100     0.00     0.00  printGridAvg
  0.00      0.17     0.00      100     0.00     0.00  printHares
  0.00      0.17     0.00      100     0.00     0.00  printLandAvg
  0.00      0.17     0.00      100     0.00     0.00  printPPM2
  0.00      0.17     0.00        2     0.00     0.00  dynamic_alloc_map
  0.00      0.17     0.00        2     0.00     0.00  free_map
  0.00      0.17     0.00        1     0.00     0.00  init_map
  0.00      0.17     0.00        1     0.00     0.00  parse_configs
```

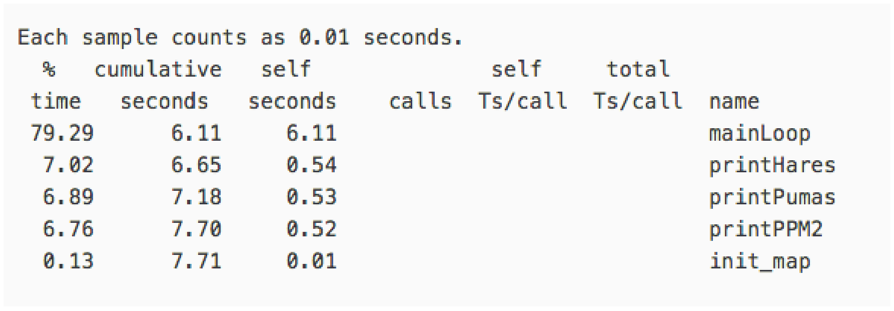
Figure 2: Profiling results of `O3` optimisation on small.dat

The first thing to stick out in the number of calls to `landNeighbourscells`. It is called twice as many times as it needs to be, and takes 11.77% of the total time. Upon inspection of the code it can be seen that `pumasNeighbourscells` and `haresNeighbourscells` were both calling on that function. To improve this, the code was edited to call the function once then pass this on to both functions, thereby reducing the number of function calls, and optimising the program. Profiling was carried out on the edited code for all used optimisation flags, except this time using islands.dat because it is a larger grid, and hence contains less variation, yielding more representative results. The results for the updated `O0` and `Ofast` are plotted in Figures 3 and 4.



```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.54      9.10     9.10     1251     7.27    19.43  mainLoop
 17.25     13.77     4.68 193393341     0.00     0.00  haresNewValue
 13.61     17.46     3.69 193393341     0.00     0.00  pumasNewValue
  9.43     20.02     2.56 193393341     0.00     0.00  haresNeighboursCells
  8.82     22.41     2.39 193393341     0.00     0.00  pumasNeighboursCells
  6.97     24.30     1.89 193393341     0.00     0.00  landNeighboursCells
  3.65     25.29     0.99      100     9.91     9.91  printHares
  3.62     26.27     0.98      100     9.81     9.81  printPumas
  3.14     27.13     0.85      100     8.50     8.50  printPPM2
  0.04     27.14     0.01        1    10.01    10.01  init_map
  0.00     27.14     0.00      100     0.00     0.00  printGridAvg
  0.00     27.14     0.00      100     0.00     0.00  printLandAvg
  0.00     27.14     0.00        2     0.00     0.00  dynamic_alloc_map
  0.00     27.14     0.00        2     0.00     0.00  free_map
  0.00     27.14     0.00        1     0.00     0.00  parse_configs
```

Figure 3: Profiling results of `O0` optimisation on islands.dat



```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 79.29      6.11     6.11                             mainLoop
  7.02      6.65     0.54                             printHares
  6.89      7.18     0.53                             printPumas
  6.76      7.70     0.52                             printPPM2
  0.13      7.71     0.01                             init_map
```

Figure 4: Profiling results of `Ofast` optimisation on islands.dat

Now the program spends a smaller proportion of time on `landNeighbourscells`. The next step is to compare the times taken for each of the different optimisation flags. The fastest time was for the `Ofast` flag as predicted, and the slowest was with `O0`. This conflicts with the performance results from 2.1. This is likely due to the profiler only measuring active time, whereas the previous timer may have measured interruptions, i.e. when work was briefly done on other functions. Another explanation could stem from conducting the tests on a laptop, with processes running in the background. The largest proportion of time for the fastest flag, `Ofast`, was spent in the `main loop`. There are many sources of overhead in this section, but the repeated calculation of water cells is an area where significant time could be shaved off through a single calculation of water cells.

2.3 Investigating The Effects From Varying Proportions of Land and water

Different proportions of land and water will yield different results for a constant sized grid. The grid used to calculate these differences has dimensions 500x500, and land proportions will range from 0% to 100%. Before computing any values, it is logical to expect grids with higher land proportions to take more time. This is because the land value must be recalculated and updated to a different value each time. However, this program calculates the value for water squares too. This means that there is a measure of time being spent doing updates, depending on whether or not this amount of time is significant will determine the gradient of Figure 5. If not much time has been spent on the water cells, then Figure 5 is expected to be close to ideal ($y = x + c$). If a significant amount of time is spent on the water cells, then the plot is expected to be sub-ideal. It should be noted that these results were all calculated with one island present. Running this test yields the results in Table 1.

Land %	Time (s)
0	25.4085
25	31.306
50	38.371
75	43.698
100	53.959

Table 1: Time measurements for increased land proportions

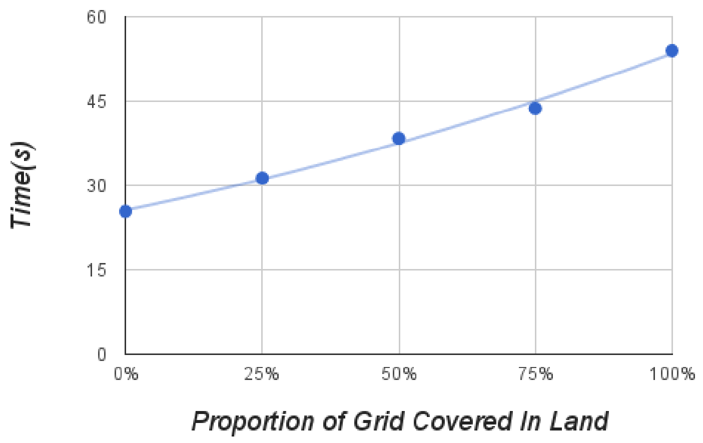


Figure 5: Time Measurements for Increased Land Proportions

Increasing the number of islands whilst keeping the land proportion constant returned approximately the same value because the same number of calculations are still taking place. Table 1 shows time taken is increasing as land % increases. This was expected, but is it linear? This is displayed in Figure 5.

Figure 5 is approximately linear, however time is increasing at a slower rate than proportion of land, indicating that a significant amount of time was spent calculating the water cells. This test yielded similar results for both larger, and smaller girds.

2.4 Investigating The Effect of Varying Grid Size

Increasing the size of a landscape, with a constant proportion of land, should increase the time taken to complete the calculations. Work should increase proportionally to time, and hence be linear. The there are three grid sizes being investigated: large, which has dimensions 1000x1000, medium, with dimensions 500x500, and small, with dimensions 50x50. All 3 grids were plotted using minimum times, and are shown below in Figure 6.

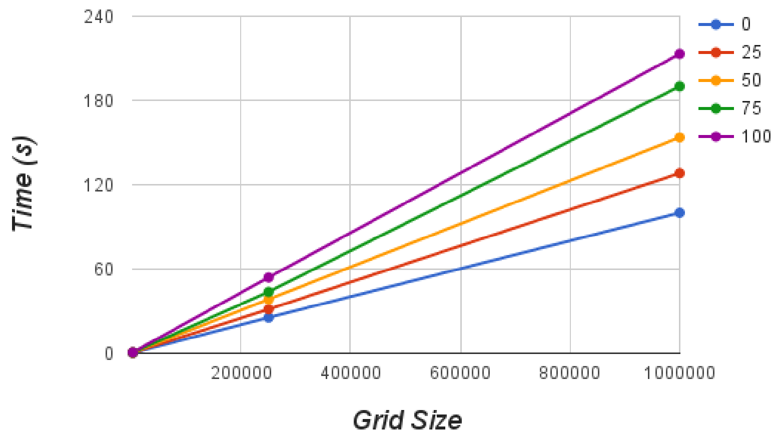


Figure 6: Time Measurements for Varying Land Sizes On Different Land Proportions

Figure 6 achieves a linear increase in time with respect to grid size as expected. The lines for different land proportions tend to diverge as grid size is increased. This is a result of the program spending more time updating the land values.

3 Conclusions

Optimisation flags can be a powerful tool in reducing the execution time of a program, however a user must beware of the associated dangers in utilising them. Higher levels of optimisation come at a price: less debugging and increased compilation time. Before applying an optimisation flag, one must carefully consider the nature of their program. One with a lot of math may work better with the `Ofast` flag, for example.

Profiling indicates where most time is being spent, and hence where most time can be saved. For the all optimisation flags, most time was spent in the main loop. Within this loop lies the calculations and updates, which take up a significant amount of time. To improve this program, the water should only be calculated once.

Increasing the size of the grid and increasing the proportion of land, both resulted in a linear increase with time. This is concurrent with our predictions, and indicates that our model is working correctly, if somewhat inefficiently. These performance tests help to identify the parts of a program which may be unnecessarily slowing it down. They also serve as a good checking method to ensure programs are working as expected.