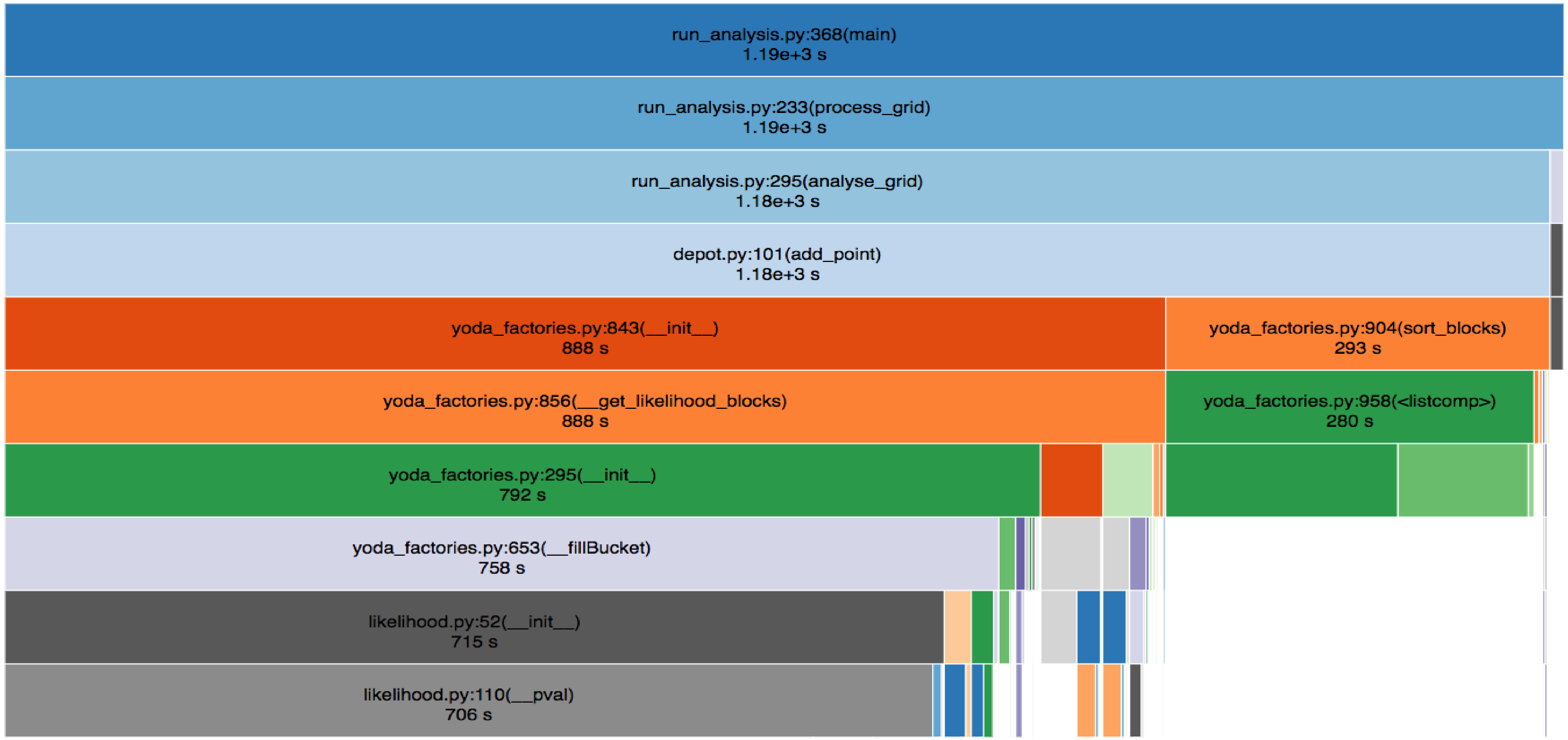
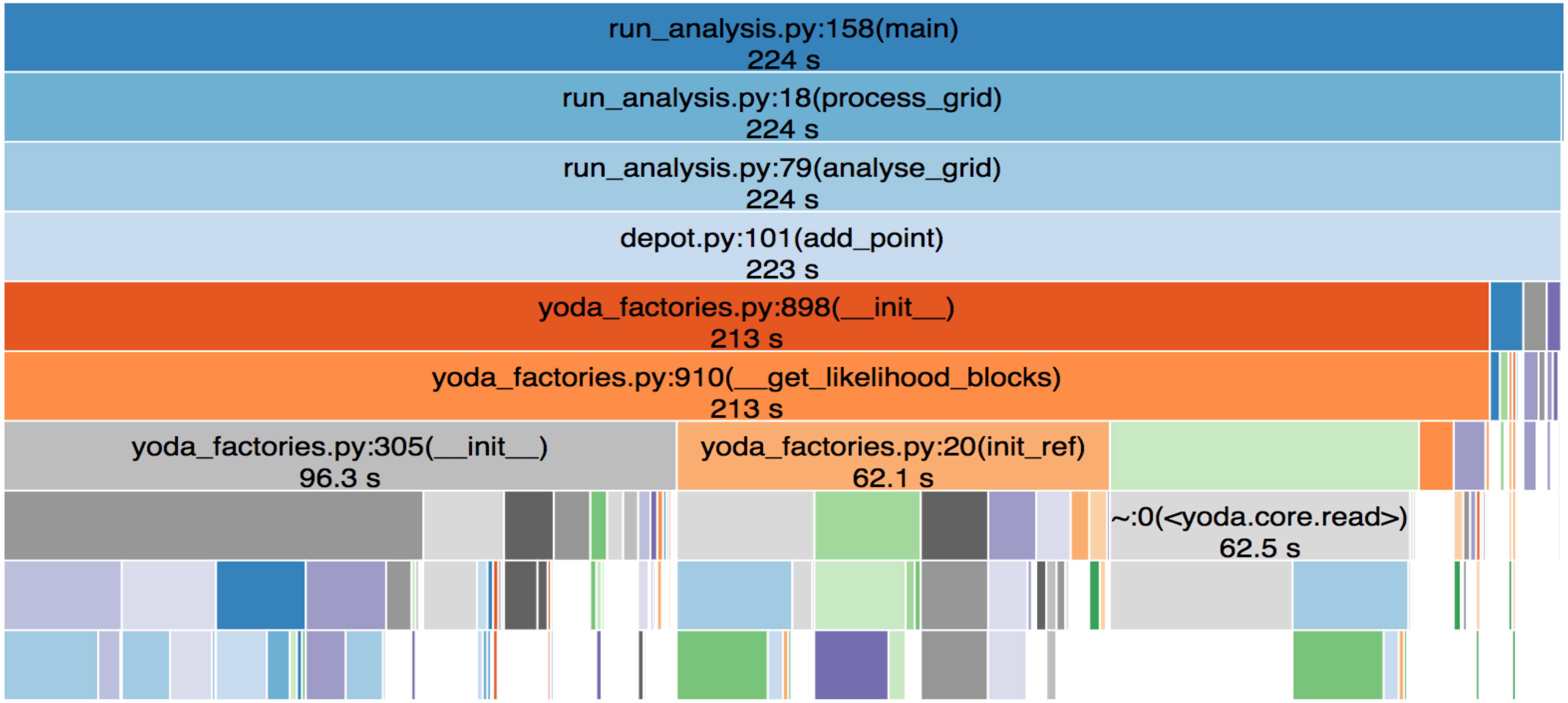
Optimising Contur

10 x 10 grid profile **before optimisation**



10 x 10 grid profile **after optimisation**



Optimisation changes performed (1)

- Two main changes performed which give speed up were in the sort blocks method and in the likelihood calculation.
- For sort blocks we had a list concatenation within the a nested for loop.
- Each call to the list concatenation bucketed likelihood blocks into pools and calculated the max CLs for each pool, each of these operations took 10^-4 seconds.
- We were calling the above method c.a. 10^6 times though for 10x10 grid giving us the runtime in the region 10^2.
- The update calculates the max CLs for each pool once outside of the nested for loop, stores the result in a dictionary and then the dictionary is called in the nested for loop. Each of these dictionary calls takes 10^-7 seconds so run time in the loop should now be of order 10^- seconds.

Optimisation changes performed (2)

- Within the likelihood calculation the main issue speed wise were calls to scipy.stats.norm.sf, the survival function used to go from the chi test stat to p value.
- Before optimisation we made over 3 million calls to this method in the 10 x 10 grid run.
- Focus of the optimisation was collecting test stats into numpy arrays and passing these numpy arrays to the method, reducing the number of calls to the method.
- Our profiling of the scipy method showed for the size of n we deal with the runtime difference between passing a single value or an array of length n to the scipy method was immaterial.
- In the after optimisation set up for the 10 x 10 grid we have between 100 and 200 calls to the scipy method. For any n x n grid with the new set up the number of calls to the scipy method will be between n and 2n .

Contur after optimisation – Further progress (1)

- In the final profile the get_likelihood_blocks method splits into three branches that give nearly all of the run time.
- In the yoda factories init call (run time 96 seconds) we spend about 40 seconds making calls to yoda to retrieve attributes (yVals, yErrs, xErrs), we could possible make run time improvements here and plan is to look at this before thesis finishes.
- There are other smaller items in the init call that we will also look at (root_n_errors function runtime 10 seconds and covariance calculation in likelihood class just under 10 seconds).

Contur after optimisation – Further progress (2)

- Other two blocks are largely focused on reading in data, a lot of time here is spent just reading yoda files (with yoda.core.read method) and there is not really anything we can do about this from a contur perspective.
- Aim is to see if there is anything else small in these two blocks that can be tidied but expectation is that out of the c.a. 100 seconds run time coming from those blocks at the moment 60-80 seconds is directly attributed to reading data which is unavoidable.