3.1

Q1

Before profiling, the functions we think will be important to optimize are:

- ‘heapsort’, because sort functions are generally heavily used and it is relatively memory inefficient when compared to quicksort.
- ‘post_place_sync’ seems like a high level function with many nested loops, this suggests that the complexity of the algorithm is quite high
- ‘alloc_matrix4’ will likely be filling up a lot of space in memory as it has so many malloc’s, it also has many for loops. Freeing up that much space in memory will be slow.
- ‘alloc_sblock_pattern_lookup’ uses a ton of malloc commands and will also be filling up lots of space in memory from the looks of it. It also has many nested for loops including a five-deep nested for loop.
- ‘get_average_opin_delay’ contains three nested for loops, which will be computationally intensive

3.3

Q2

Option Flags	User Time (s)	Speed-up
-O3	5.37	1x
-O2	4.44	1.21x
-Os	3.87	1.39x
gcov -g	2.02	2.65x
-g	1.74	3.09x
gprof -g	1.73	3.10x

Q3
'-O3' is the slowest, because it is working to reach the highest optimization level. This requires more compilation time as the algorithm is more complex.

Q4
'gprof -g' is the fastest because it does not actually optimize any of the code, only adding code for debug report, thus requiring far less compilation time.

Q5
'gprof -g' is faster than 'gcov -g' because the former has a simpler function set.A simpler function set requires less time to be compiled.

3.4

Q6

Parallelism	Elapsed Time (s)	Speed-up
1	6.66	1
2	4.926	1.35x
4	2.68	2.49x
8	1.87	3.57x

The maximum parallelism was 8, because the processor has 8 cores. This only results in a 3.57x speed up because though parallelism is being used, the compilation process can only be subdivided into so many pieces. In the ideal parallel case, it could have a speed up closer to 8x.

3.5

Q7

Option Flags	Size (Bytes)	Relative size
gcov -g	1012200	3.60x
gprof -g	751168	2.67x
-g	747176	2.65x
-O3	379648	1.35x
-O2	333952	1.19x
-Os	281528	1x

Q8

Os is the smallest, because the compilation option forces gcc to optimize for total program size.

Q9

Gcov is largest, because it profiles all source lines and probably has more overhead for its functions.

Q10

Gprof produces a smaller result because it adds less overhead to the program. This is because it uses simpler functions than gcov.

3.6

Q11

Option Flags	User time (s)	Speed-up
gcov -g	2.94	1.17x
gprof -g	3.43	1x
-g	2.81	1.22x
-O3	1.18	2.92x
-O2	1.27	2.71x
-Os	1.37	2.51x

Q12

Gprof was the slowest, because it interrupts the program every few milliseconds to measure the code which adds an overhead to runtime.

Q13

O3 was the fast, because it the highest version of optimization that was compiled for this test set.

Q14

Gcov was faster, because it doesn't have the same of added overhead to measure. Makes sense since gcov simply keeps tracks of how many times it "passes" a line in the source code whereas gcov keeps track of time it has been within a function.

3.7

Q15

Top 5 functions when run with "-g" flag(s)

Function Name	Percentage
comp_delta_td_cost	17.38
comp_td_point_to_point_delay	16.49

find_affected_nets	11.35
try_swap	10.64
get_seg_start	7.80

Top 5 functions when run with "-O2" flag(s)

Function Name	Percentage
try_swap	39.43
comp_td_point_to_point_delay	17.07
get_non_updateable_bb	15.04
get_net_cost	7.32
get_seg_start	6.50

Top 5 functions when run with "-O3" flag(s)

Function Name	Percentage
try_swap	61.02
comp_td_point_to_point_delay	22.03
get_bb_from_scratch	5.08
label_wire_muxes	4.24
update_bb	1.69

Q16

In -O3, the function try_swap increased its percentage of the total runtime by 5.73x (from 10.64% to 61.02%). This can be explained by the fact that comp_delta_td_cost and find_affected_nets were optimized as inlined functions and therefore their runtime was added to that of try_swap.

Q17

The second function in the -O3 gprof is called in multiple functions, including try_swap and therefore the compiler decided that it would not make sense to inline the function in multiple different places within the code.

3.8

Q18

Running with the "-g" flag produced a total of 2001 instructions whereas running with the "-O3" flag produced a total of 762 instructions, meaning the total number of instructions was reduced by about 2.3 times.

Q19

Running with the "-g" flag had a cumulative runtime of 2.32 seconds whereas running with the "-O3" had a cumulative runtime of 1.11 seconds, meaning the total speedup of the update_bb function was around 2.09 times.
The speedup and the code reduction ratios were fairly similar in size (around 2 times), with the increase in speed being slightly less than the decrease in number of instructions. This could because the optimized version had instructions that on average took more clock cycles to run that the original version's set of instructions did.

3.9

Q20

For the loops with the function, the order in which that focus should be given to optimize was decided as:

1. For loop @ line 1377: this for loop had the most number of calls according to gcov and from first glance what looked to be the largest and most time consuming inner loop of all 5 loops.
2. For loop @ line 1512: also had a large number of calls, but nearly as big of a inner loop the for loop described above.
3. For loop @ 1473: called far fewer times that the for loop above but had more instructions that the for loop below
4. For loop @ 1279: least import for loop to optimize of the bunch, few instructions and called the least
5. While loop @ 1312: no need to be optimized, because it is never called. If there were fixed pins for the fpga then it could be optimized, but otherwise not a code section to focus on.

3.10 (BONUS)

Q21

To optimize the code and decrease the average user time to an average of 1.16 seconds (from 1.18), we preformed three modifications to the code base:

1. On line 1976, we commented out the initialization of delay_source_to_sink since its declaration to zero wasn't necessary.
2. On line 1361, we commented out the initialization of delta_c since it's declaration to zero wasn't necessary
3. On line 1279 in place.c, we expanded out the for loop since the variable num_types was set to the constant NUM_TYPES_USED which was defined as 3.