EC527: Lab 6

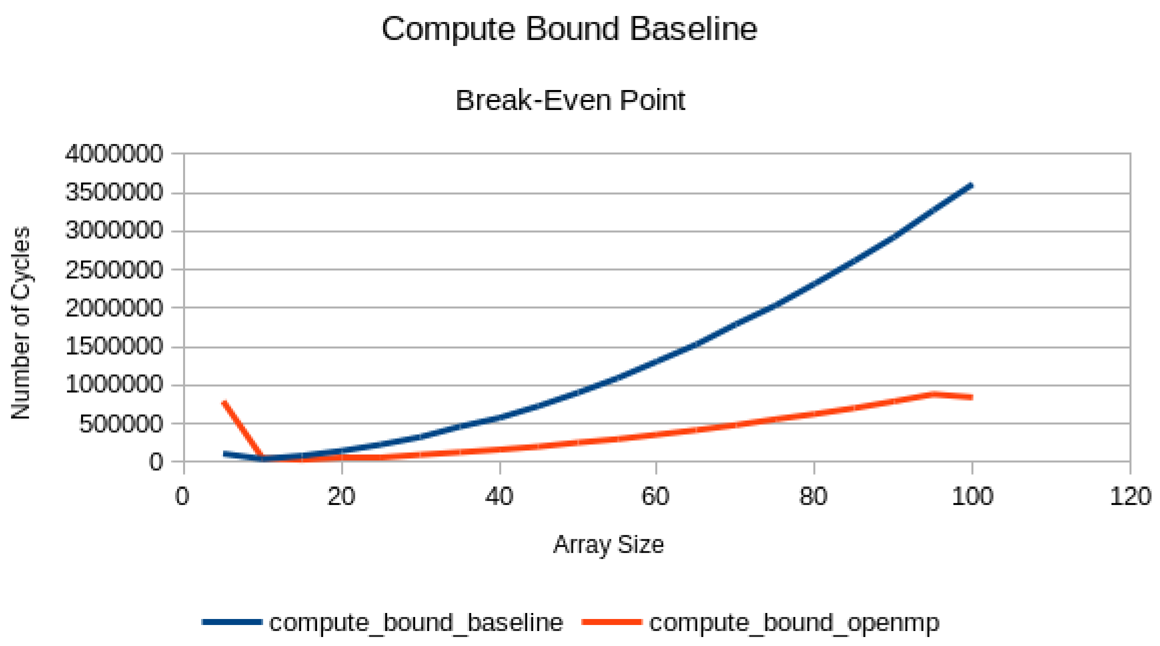
Part 1.1

Ans: **<u>Output</u>**

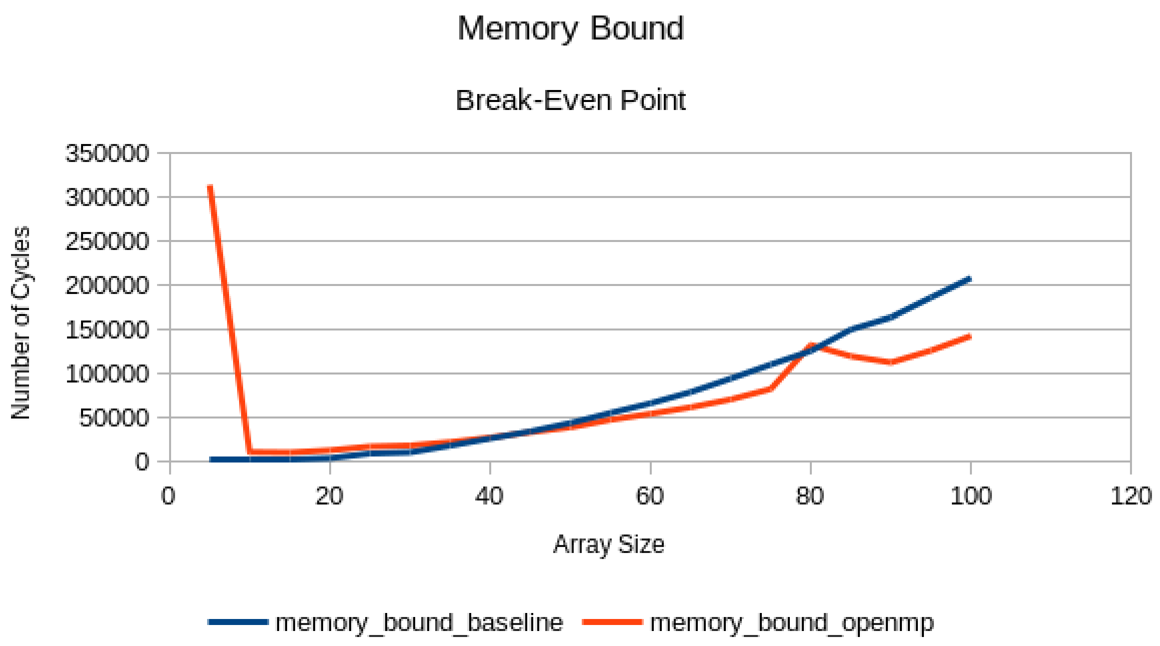
```
Hello World -- Test OMP
Heoll rld!Wo

Using Parallel for:
ldH!elWloo r
```

Part 1.2

Ans:

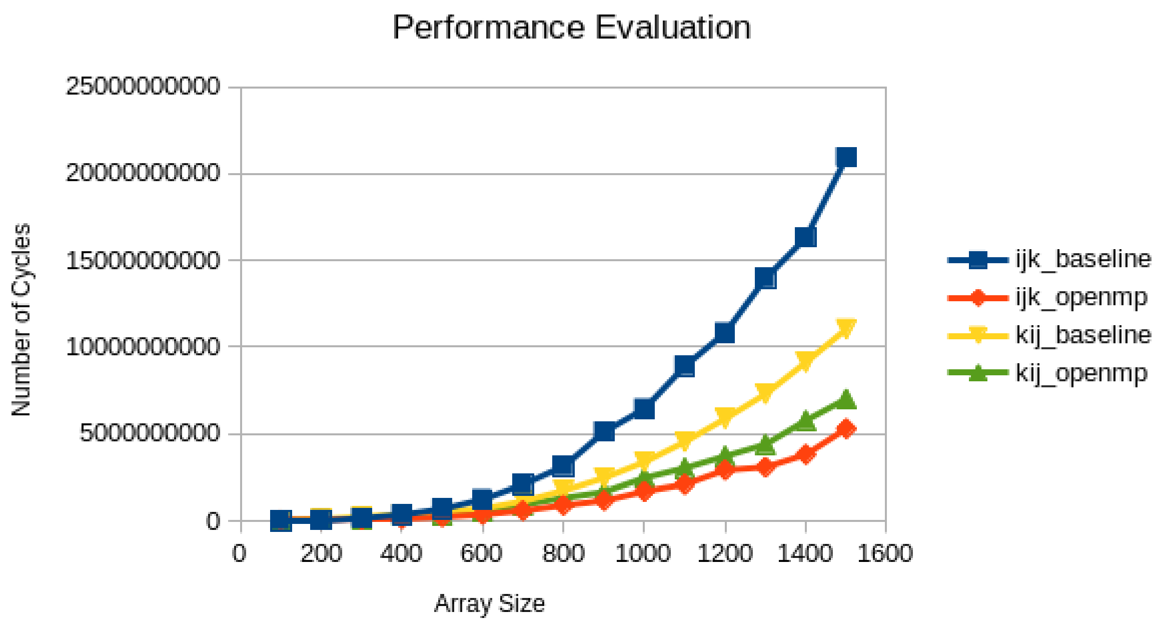


As we can see from the graph, the break-even point for the Compute-Bound Baseline is nearly at the array size of 15.



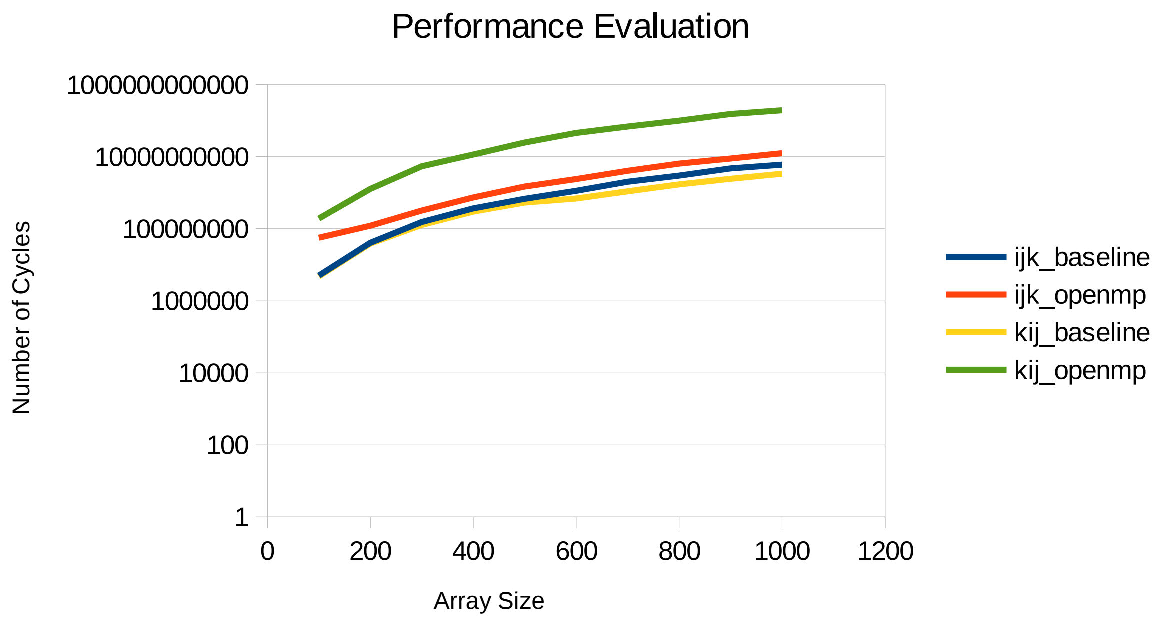
The break-even point for memory bound function is nearly at the array size of 40.

Part 1.3.
Ans: a)



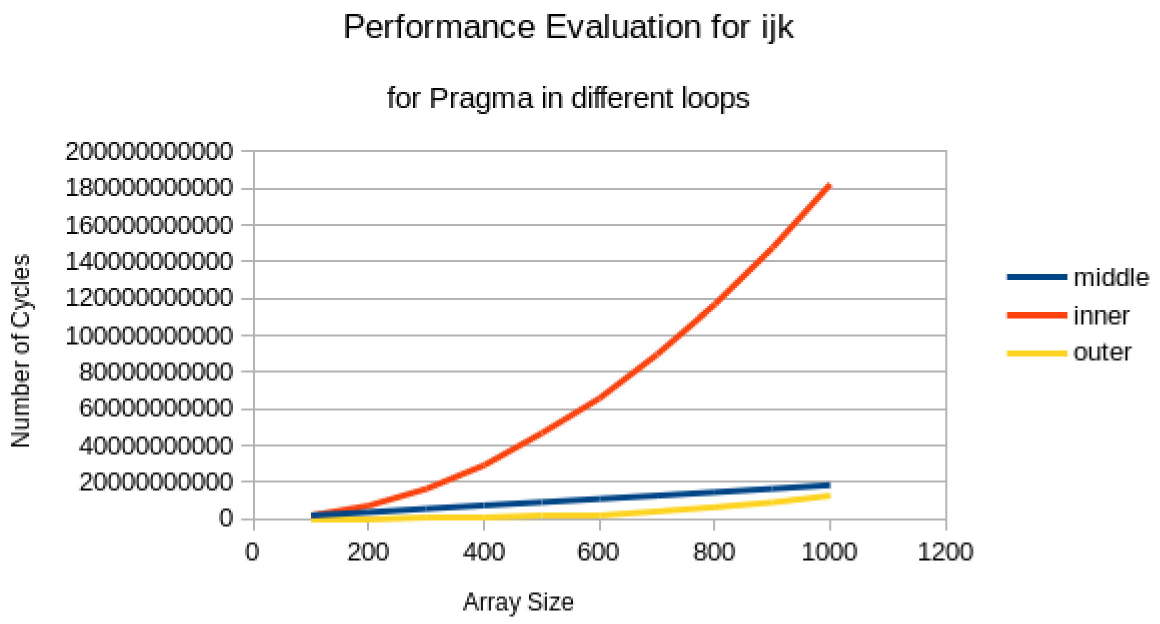
As evident from the graph, the MMM for ijk baseline case is the worse than kij_baseline case. However, when we use OpenMP, MMM for ijk performs better than kij. This is due to the fact that since each thread is performing an operation on a set of elements, the data is more likely to be present in the cache leading to relatively lesser number of misses. Hence, ijk for OpenMP performs better for kij for OpenMP.

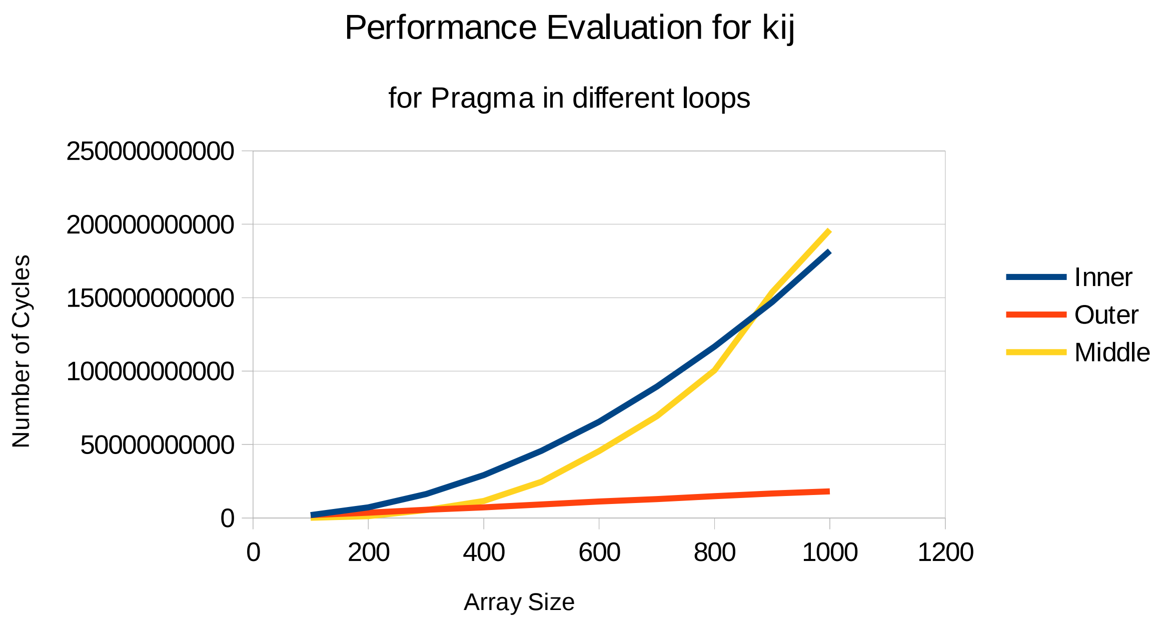
b) Performance Evaluation with loop indices "shared".



When we make previously private loop indices as shared, then each thread cannot access the loop variables separately and hence the overall operation becomes too slow. As a result, the baseline functions perform relatively better than the OpenMP functions.

c)

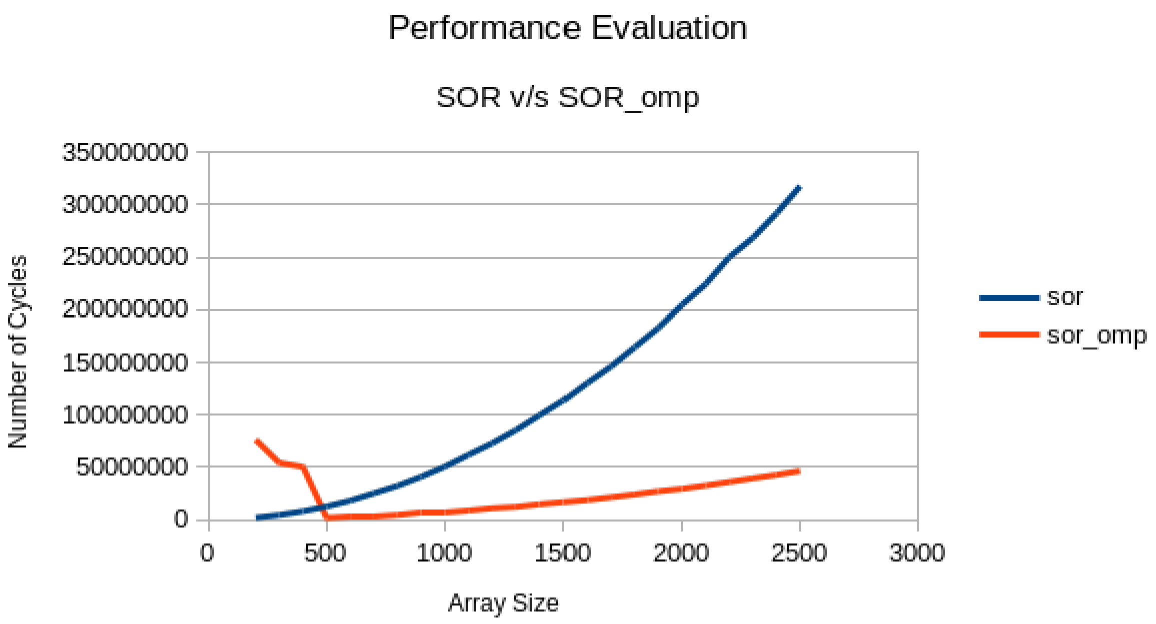




For both ijk and kij, we observe that the performance for pragma in outer loop gives the best performance, followed by middle and then inner. The reason for this is that as we keep moving the pragma to inner loops the parallelism keeps decreasing and hence the operation becomes slower.

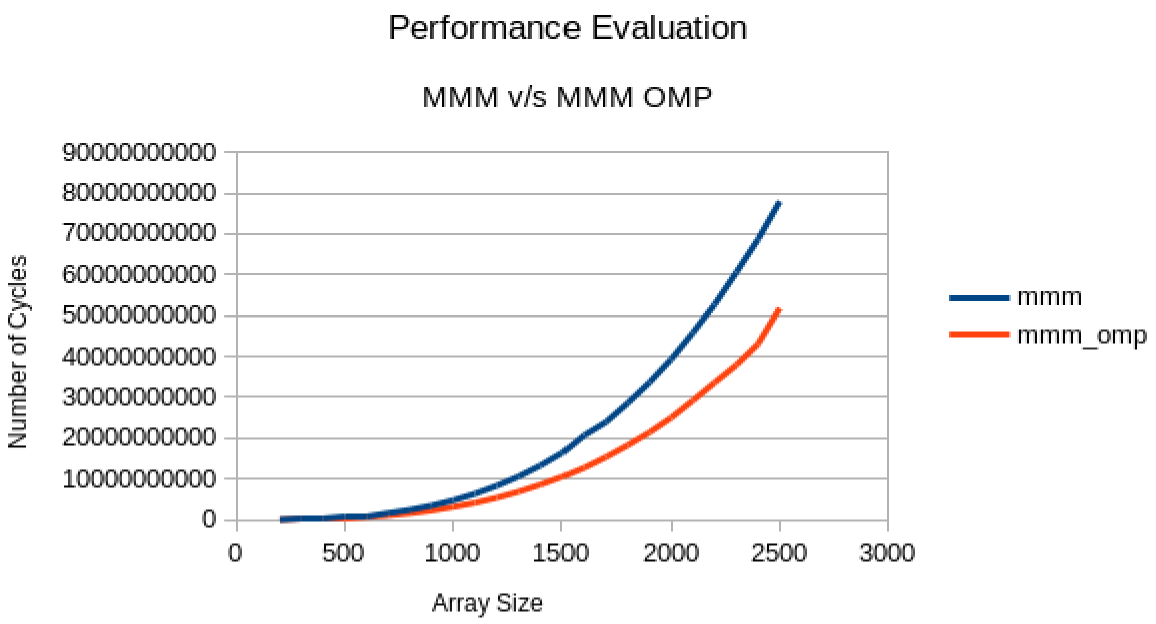
Part 2.

2.1
Ans:



We observe that on using OpenMP, we introduce a degree of parallelism in the code since the processes gets divided into threads that are operating in parallel. This reduces the number of cycles involved in the computation.

2.2
Ans:



We observe that on using OpenMP, we introduce a degree of parallelism in the code since the processes gets divided into threads that are operating in parallel. This reduces the number of cycles involved in the computation.