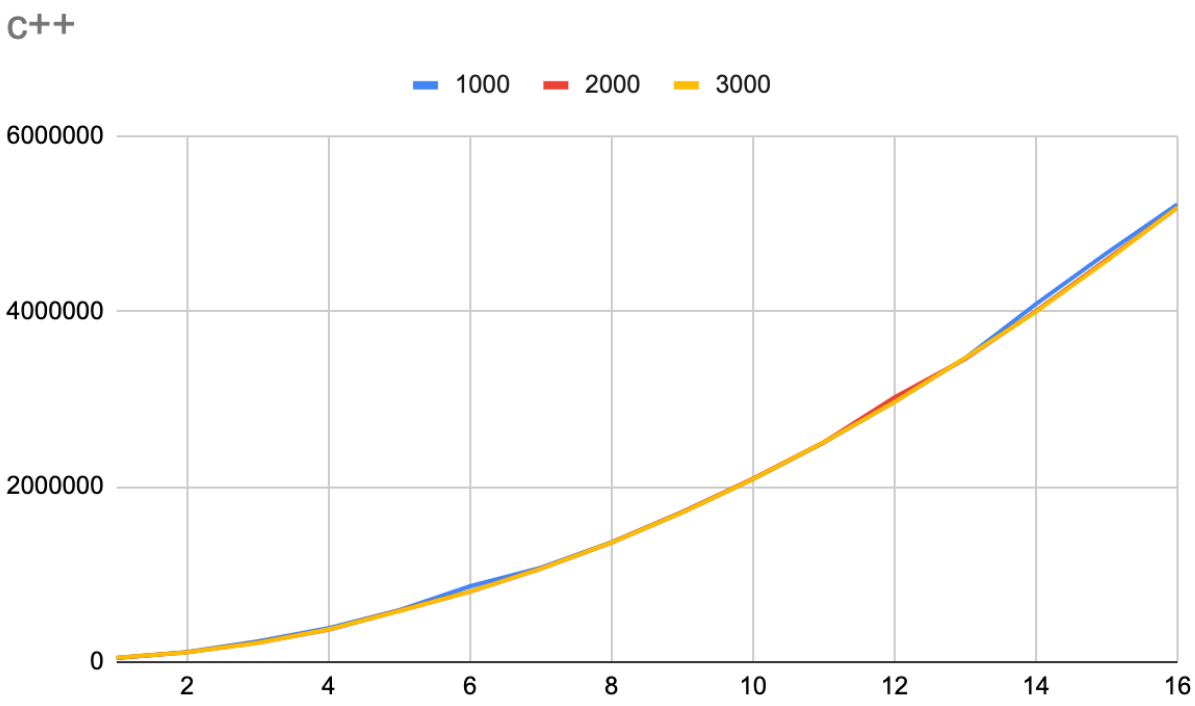
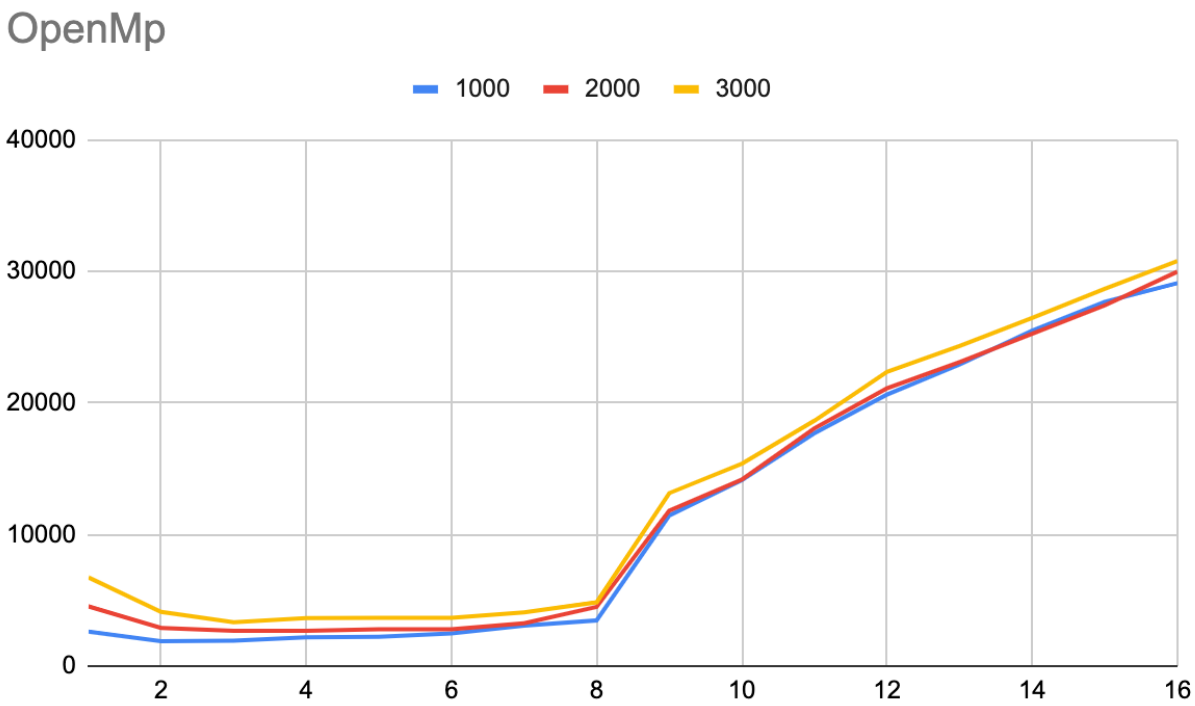
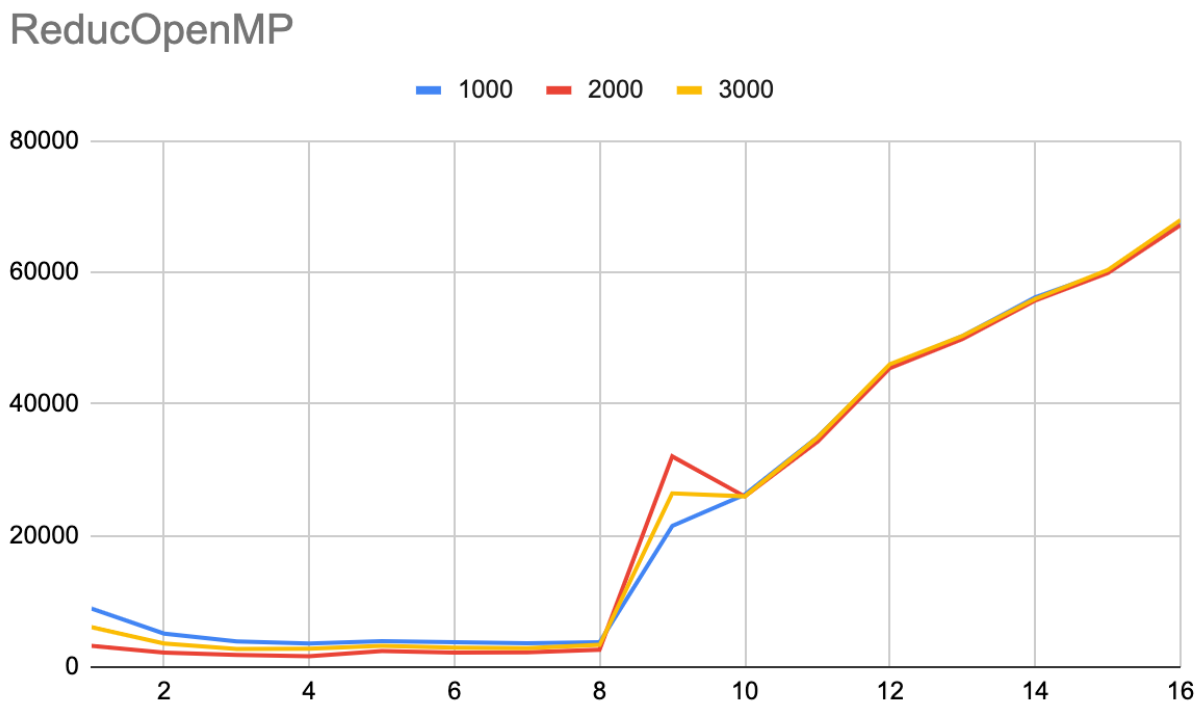
Here is the Graph for C++ with 1000, 2000,3000 in size growing from 1 to 16 threads



Here is the Graph for OpenMP with manual reduction with 1000, 2000,3000 in size growing from 1 to 16 threads



Here is the Graph for OpenMP build in reduction with 1000, 2000,3000 in size growing from 1 to 16 threads

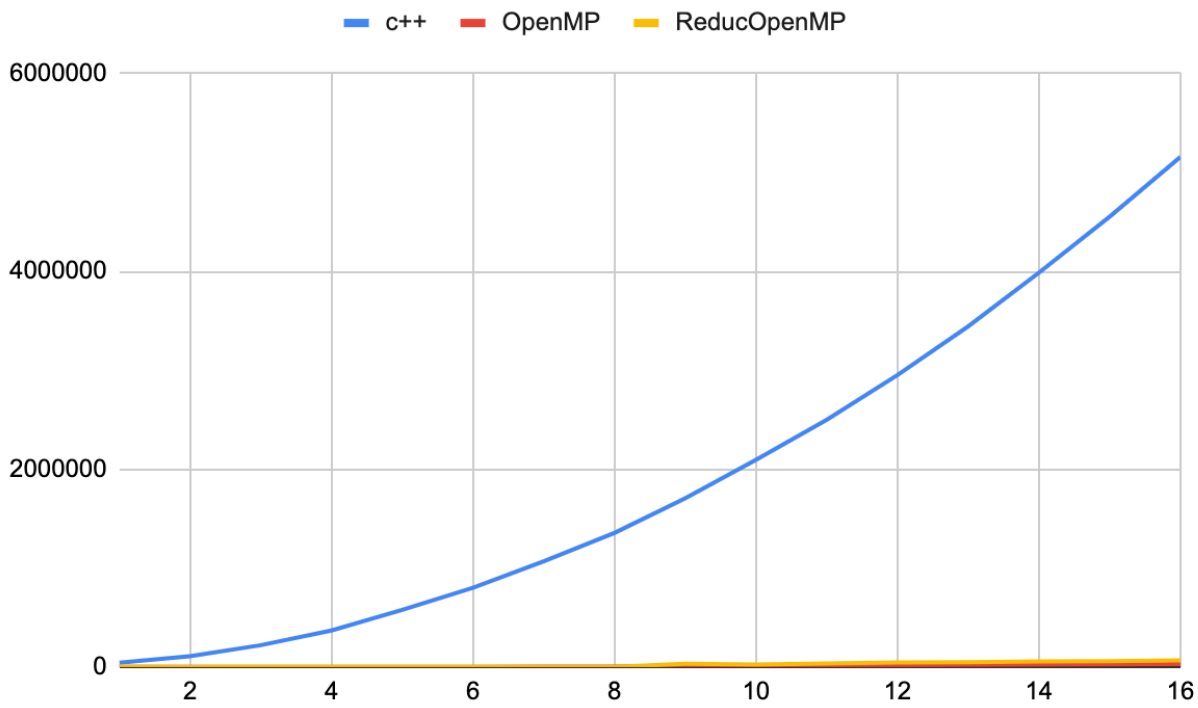


Question1:Do the curves match the ideal case? If not, why?

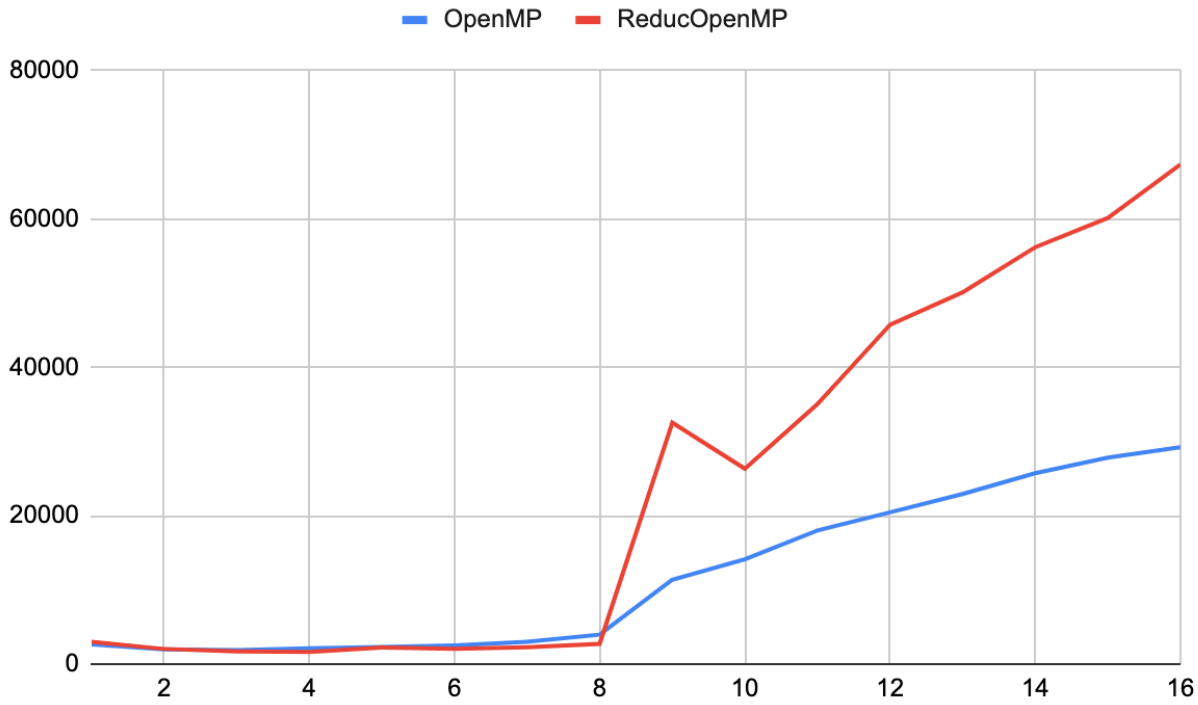
For my graphs, they do not match. The C++ makes sense on the increased time because that is not memory allocation with how I used the locks and barriers. Im not sure way with my OpenMP my custom one is faster then the build in one. Im guessing that there are some functions to help with efficiency built into OpenMP that my code is not using correctly. The time still goes up but it is a logarithmic graph. This is how it should be adding more threads.

Question2:Produce a single graph containing results from all 3 functions. Does the curve match the ideal case? If not why?

Here is the soft correlation with growing threads and size



Here is the soft correlation with growing threads and size this is the same graft as above but zoomed in to see the difference in the OpenMP



My c++ locks do not so I would need to go back and fix my locks. The lines should be logarithmic. With the graphs, you can also see a spike using the built-in reduction. I'm guessing there is an efficiency error with how my code is running the OpenMP.

Question 3:Which of the three methods above performs best on each test? Can you explain why?

As I have guessed before I guessing the built-in reduction in OpenMP should be the best when it comes to a large amount of data with more threads. But with how my code it I don't think it is growing correctly after 8 so there is a big spike in the graph.