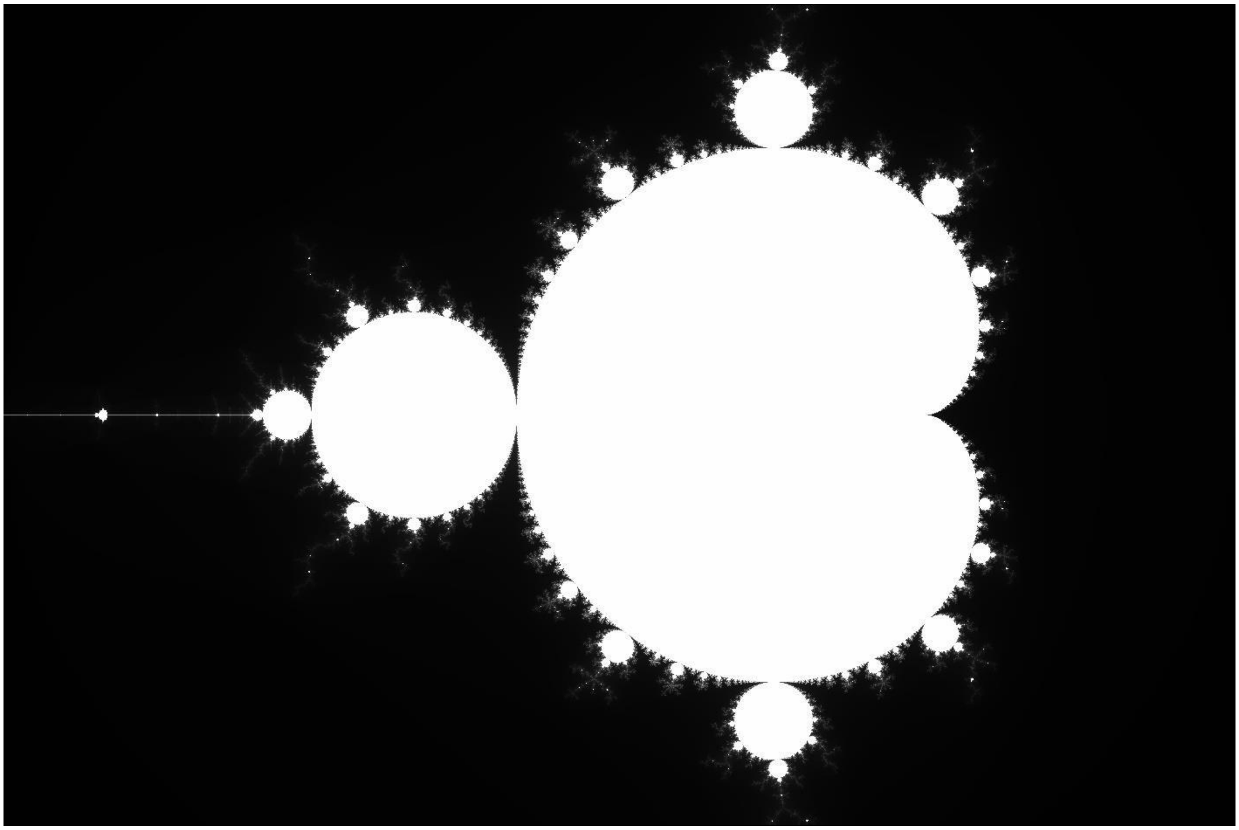
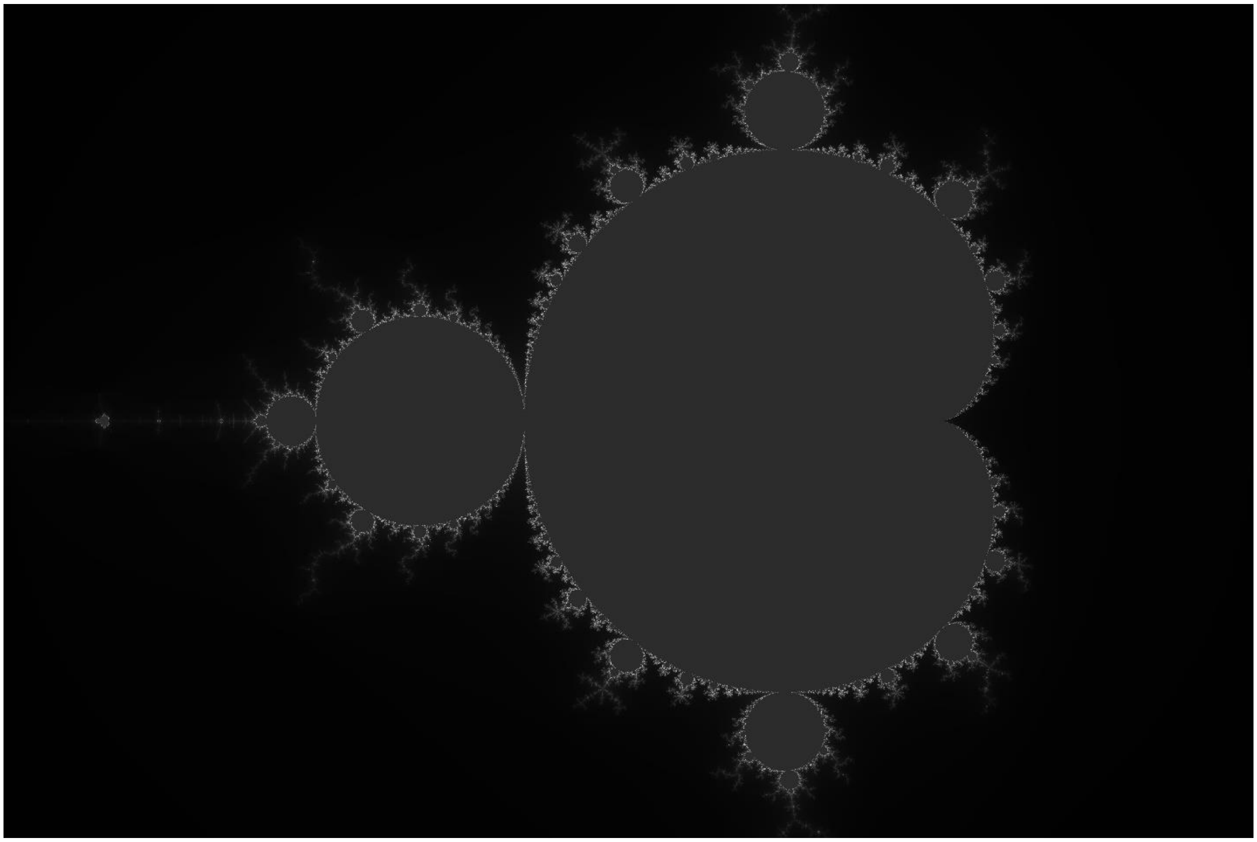
Carlos Daniel Jimenez
jimenezc1@wit.edu

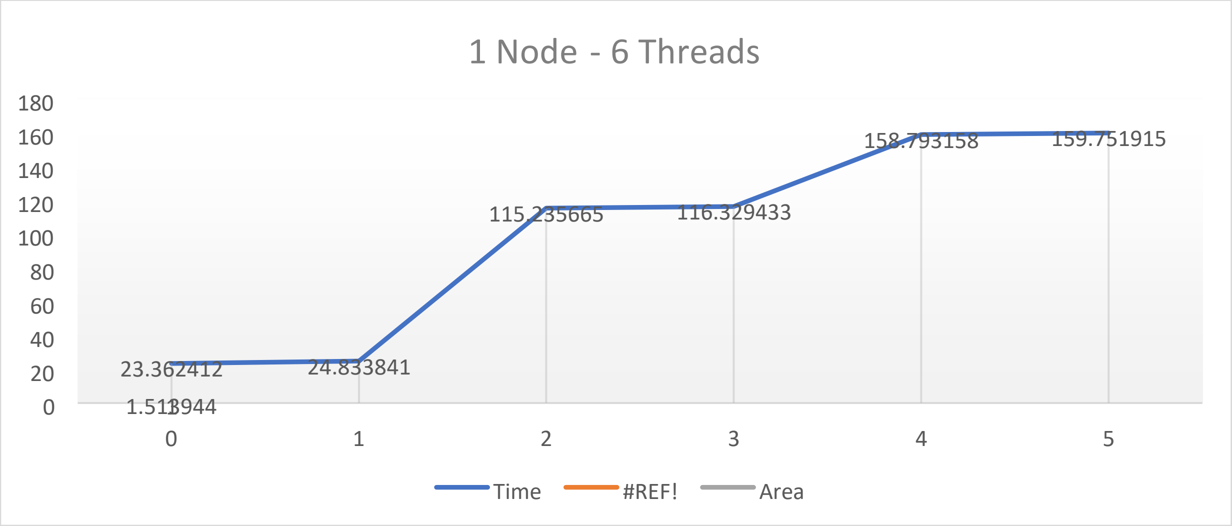
Exam Analysis

I used the OpenMP approach for this problem. However, I timed everything using MPI. Bellow you will find the final image. Also, I tried to run this a few times with different parameters. For example, I changed the number of nodes (10, 5, 1) as well as the number of threads (12, 6). And, the number of iterations (510, 300) just to see analyze the performance a little better. The first picture is the final version. I found something interesting in my analysis. When I went from 510 to 300 iterations, the image itself is not black and white. It's rather gray and black. I will attach it bellow the black and white picture.

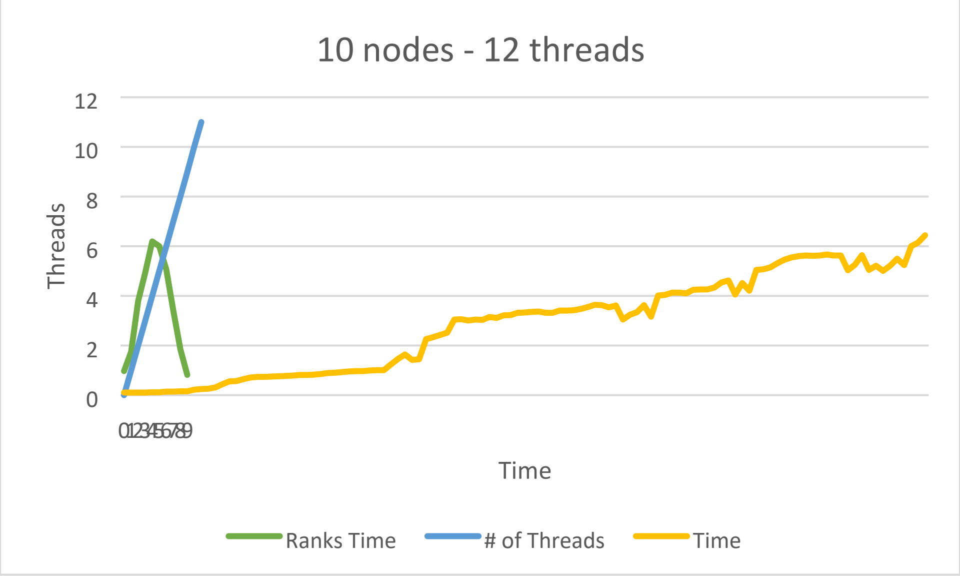


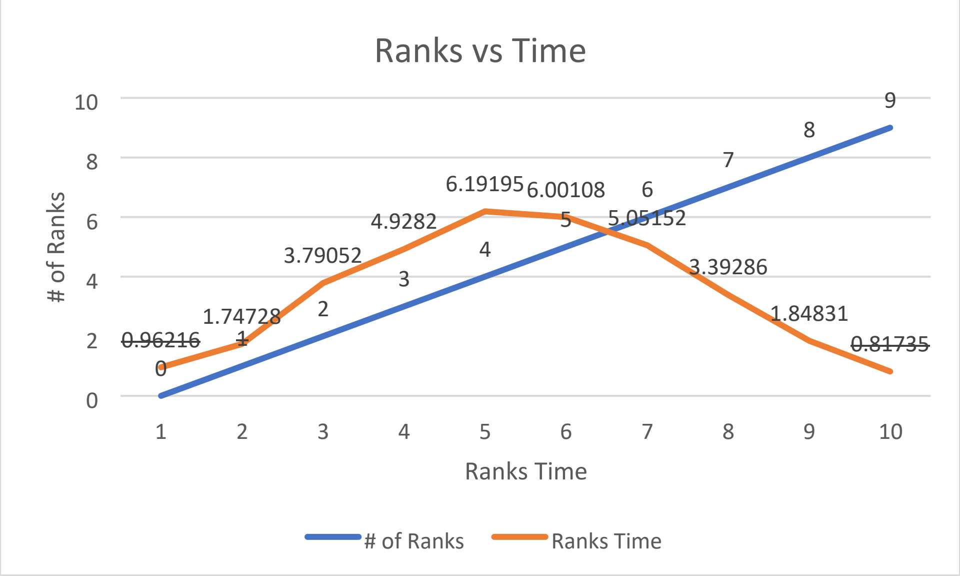


I timed my application as mentioned above, bellow you will find the graphs. In this set of graphs, the only constant is the number of iterations – which is 510. The rest various from graph to graph.

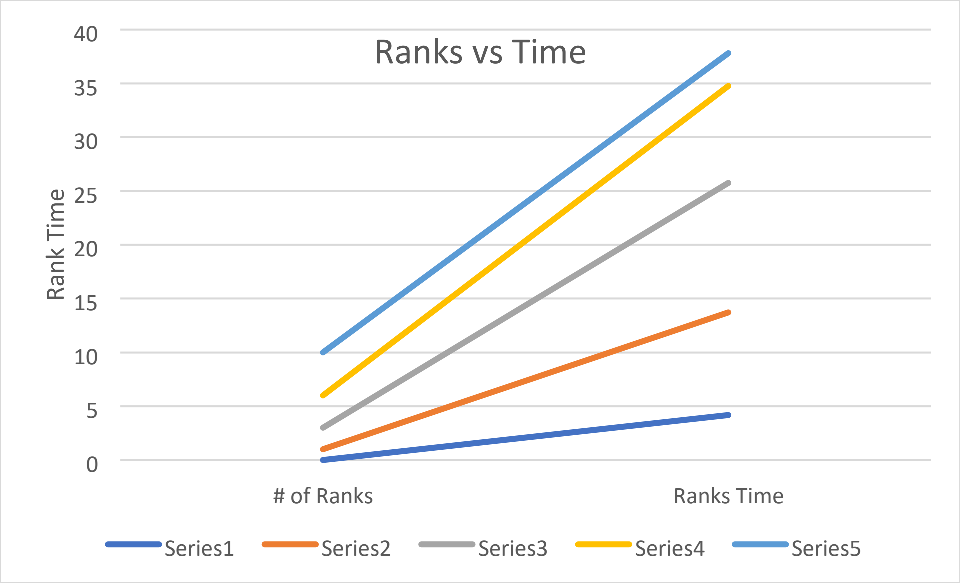


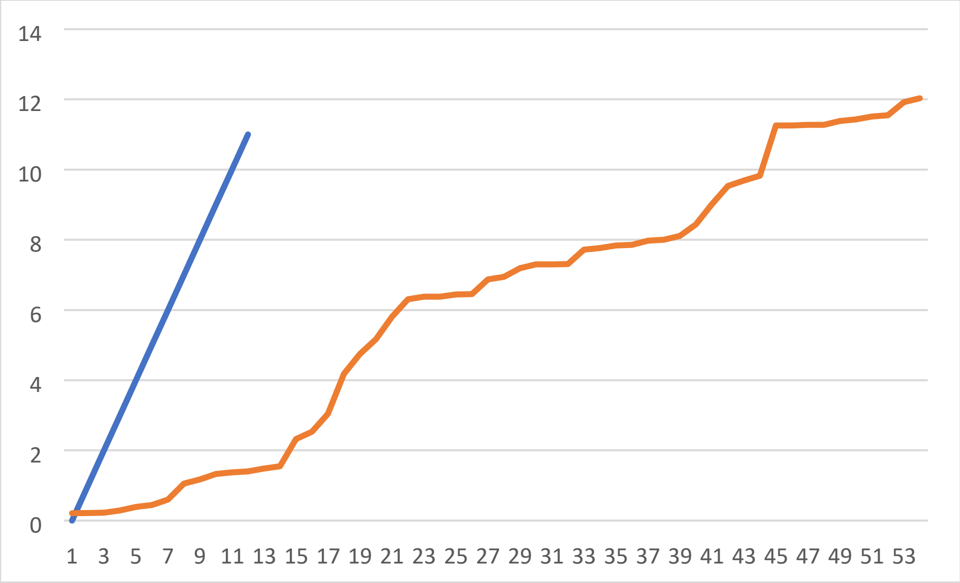
It seems like there is not a significant difference between consecutive nodes. For example, from 0-1 node, there is a very small difference. But if we add one more thread, we see a huge difference. The process repeats; 2-3 nodes, no big of a difference. But 3-4 there's a huge jump.





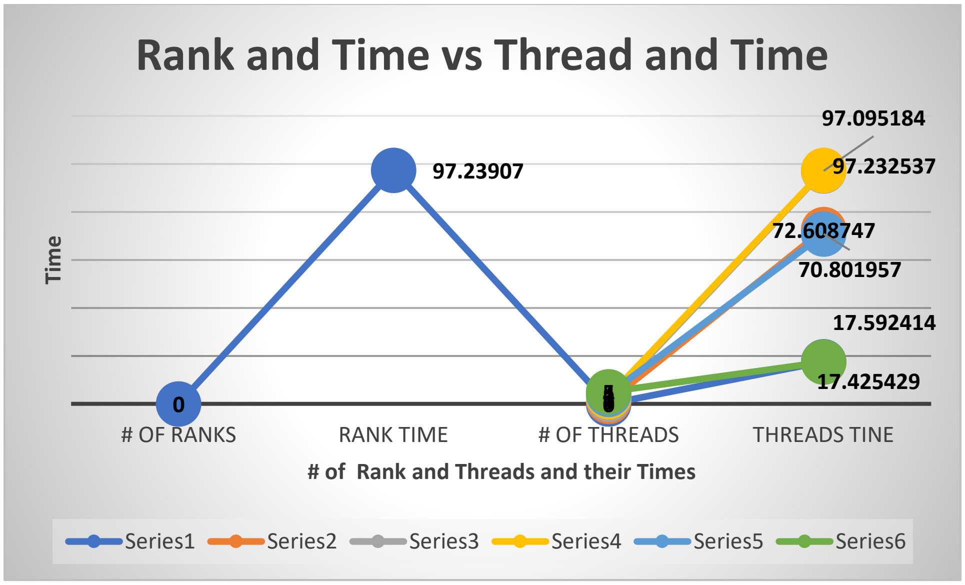
As we can see here, we have ten nodes and twelve threads – the best performance we can get. As time progresses, we can see that the performance of the threads keeps increasing (yellow line). The green and blue lines show the # of nodes and threads. I pretty much combined the second graph with the first one to see if the results will make more sense.



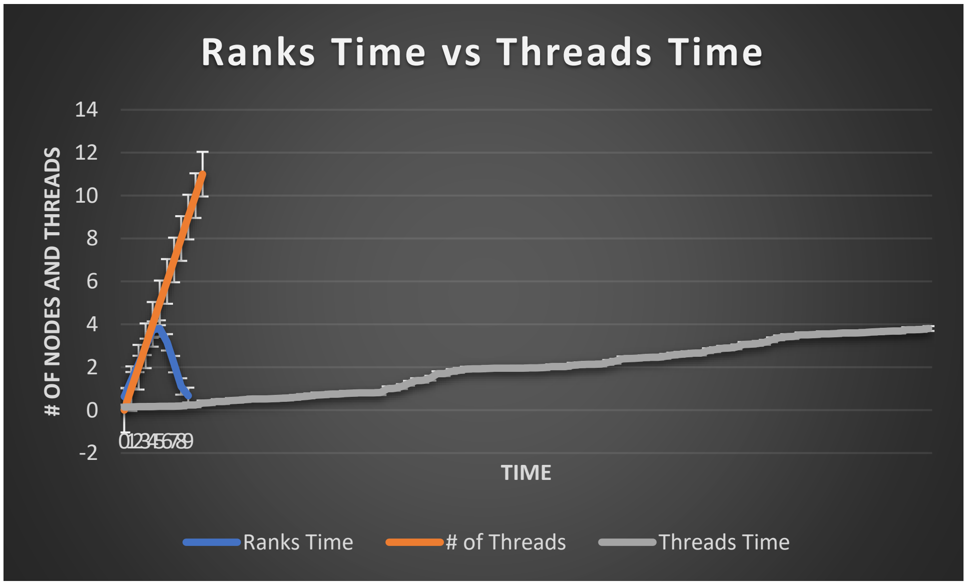


These final graphs have five nodes and twelve threads. The first graph is about ranks and ranks time. The second one is about threads and threads time. The first one shows that the more ranks we have, the fastest it will be. The second one shows the number of threads available. And, how effective it is while handling the calculations.

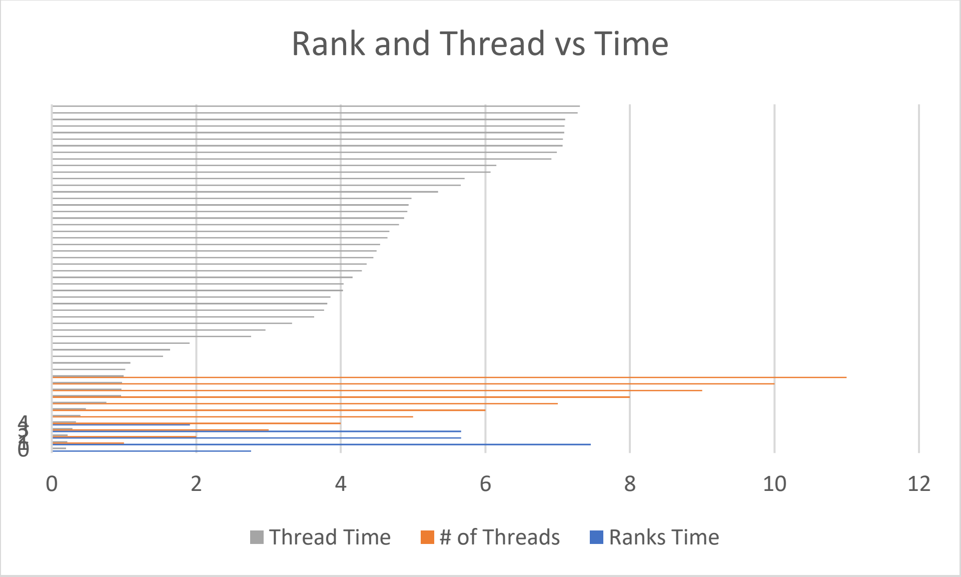
Now, this is the set of graphs containing the 300 iterations. Like the previous ones, we have the same number of nodes and threads. The only difference is the number of iterations. Here we go.



This is the worst case ever with one rank and six threads. As we can see, there are a few of the thread cases that don't improve performance. I'm fact, it is very similar to the case where we must even threads with the bare minimum increase between them. We need to bump it to +1 thread to see a huge impact.



This is the best case. Ten nodes and twelve threads. If we look closely to the gray line, we start to see a slow increase where we notice that adding more threads maybe not make much sense. Now, let us jump into the sweet spot – five nodes and twelve threads.



Our x-axis measures the thread time, # of threads and ranks time. The y-axis is all about the actual time score per each x-axis.