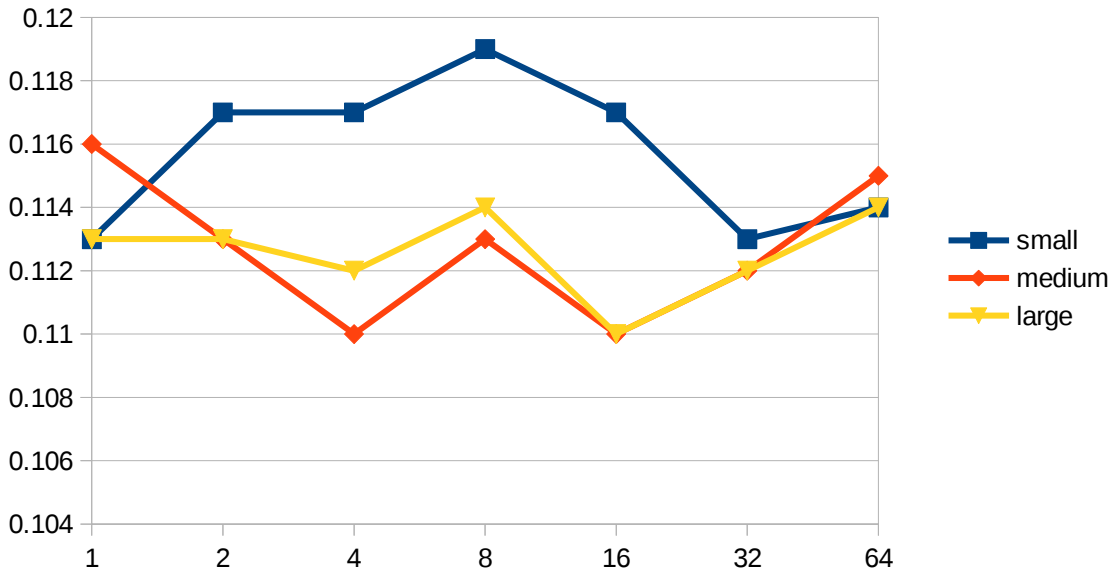
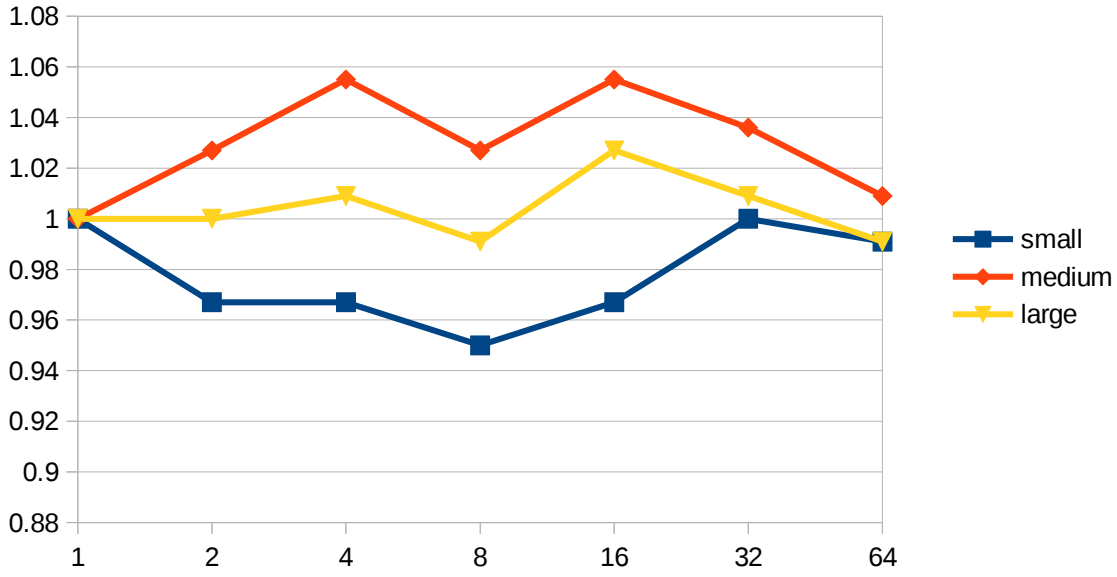


Execution Times

For my calculations I took the average of the user CPU time after running my code five times. The primary conclusion I have drawn from my graph is that the program is scalable. Not only did it do well with the larger input files, but in fact the execution time was faster with medium and large than with small. With small we can see a nice bell shape, where it looks as though the overhead of adding processes made the execution time take slightly longer before it dropped again after 16 processes. Considering that only three processes were actually computing the unknowns, I have a feeling that this had more to do with fluctuations in user CPU time (considering I got varying times each time I ran the code). For medium and large, however, the graph seems to make more sense to me. The execution time drops when the files make use of the larger number of processes before it goes back up again due to overhead of creating processes and possibly bandwidth consumption. The graph is not perfect in representing this because of random fluctuations in the execution time each time I ran the code despite the fact that these points are all averages.



Speedup

The speedup for the small input file doesn't look very good. At best we see that the speedup is 1 and this is because the overhead of communication and creating processes doesn't outweigh the benefit of parallelization with such a small amount of input data. We only see speedup over 1 with the medium and large input files because the program now has the ability to make better use of the extra processes. Here the benefit of parallelization clearly outweighs the cost of communication. We see a dip in the speedup at 8 processes due to there being more overhead once again, but it raises again at 16 processes because of the increased parallelization. This drops again at 32 and 64 processes, however, because more overhead is created with adding more processes with no increased parallelization.