# PS3 Questions - 583 Only

Written By: EJ Rainville, Spring 2021

## Efficiency

10. If you double the problem size for matrix-matrix product, how many more operations will matrix-matrix product perform?

Each matrix multiplication takes N^3 operations to compute. Therefore, by doubling the problem size the number of operations required is increased by a factor of 8. An example of this would be if you start with a problem size of 128 this should be computed in 2,097,152 operations however a problem of double this size (problem size of 256) it will take 16,777,216 operations to compute.

11. If the time to perform a given operation is constant, when you double the problem size for matrix-matrix product, how much more time will be required to compute the answer?

Assuming that the time to perform a given operation is constant and since we know that doubling the problem size causes an increase in operations by a factor of 8, we would expect that the time required to compute a matrix-matrix multiplication would increase also by a factor of 8.

12. What ratio did you see when doubling the problem size when mmtime called `mult_0`? (Hint, it may be larger than what pure operation count would predict.) Explain.

When doubling the problem size using 'mult_0' (no optimizations) we find that the ratio of the larger problem to the smaller problem is approximately 10.3 between a problem size of 512 and 256. When using larger problem sizes, we find that the ratio between the larger and the smaller problem is 15.2 between a problem size of 1024 and 512. In these two cases, we see that the ratio between the larger problem and the smaller problem is larger than the expected ratio of 8 which is most likely due to time being spent retreiving data from memory or not having instructions pipelined efficiently into the processor thus leading to larger times.

13. What raio did you see when doubling the problem size when mmtime called `mult_3`? Was this the same for `mult_0`? Referring to the function in amath583.cpp, what optimizations are implemented and what kinds of performance benefits might they provide?

When doubling the problem size using 'mult_3' (no optimizations) we find that the ratio of the larger problem to the smaller problem is approximately 10.6 between a problem size of 512 and 256. When using larger problem sizes, we find that the ratio between the larger and the smaller problem is 8.56 between a problem size of 1024 and 512. In these two cases we see that the smaller problem has a higher ratio of 10.6 which is most likely due to the fact that both execution times were relatively short so the difference between the two was less pronouced but for the larger problem the ratio was 8.56 which is much closer to the expected ratio of 8 with a bit of extra time that could be due to time spent accessing data from memory or lack of instructions readily available.

14. (Extra credit.)  Try also with `mult_1` and `mult_2`.

When doubling the problem size using 'mult_1' (some optimizations) we find that the ratio of the larger problem to the smaller problem is approximately 8.55 between a problem size of 512 and 256. When using larger problem sizes, we find that the ratio between the larger and the smaller problem is 14.8 between a problem size of 1024 and 512. Here we see that the smaller problem has a ratio closer to the expected 8 while the larger problem has a much larger ratio. This could be due to the larger problem not being fully optimized and the time it takes to compute is much larger due to having to access data from memory multiple times thus slowing the program down. When doubling the problem size using 'mult_2' (moderate optimizations) we find that the ratio of the larger problem to the smaller problem is approximately 10.2 between a problem size of 512 and 256. When using larger problem sizes, we find that the ratio between the larger and the smaller problem is 16.3 between a problem size of 1024 and 512.