Scientific Computing 9505a
Assignment2: Matrix Computations

Due: 30 November 2022

1. **Matrix-Vector multiplication:** Start with the code posted on OWL for this (this is the one we went over in lecture). Using MPI_Wtime, calculate the time spent on the actual computation part of the algorithm (i.e. the cumulative sum of the Arow.b calculations) on each processor, collect and average the results at the end of the program and output the result. Also time on the boss processor the total time spent on the calculation excluding the setup time. What we want here is T, the total time to run (excluding the setup time), Tcalc (average calculation time per processor). Note that you can also compute a Tcomm (a communication time) from the difference of these two. The aim is to compare to what is expected (as described in the lecture notes).
   Test it with reasonable trial data for the matrix A and vector b (e.g. A=Identity, and b=(1,1,1,…) is in the code but you could also try b=(1,2,3,4…) and output showing that you get b back again. A second test with A being the mirror image of the identity would reverse the entries in b. Then test with a dense matrix A that has all nonzero entries).
   Time your code for a representative example to see how the speed on a fixed number (≤ 8) of processors depends on N (A being a NxN matrix). For sufficiently large N you should see some sort of scaling with N (Note that to determine if you have the expected scaling you can make use of the fact that if the time T scales like N^2 then a plot of log T versus log N should have a slope of 2.) Then, once you have decided what a "large N" is required to get some sort of scaling, try varying the number of processors (2 to 8 say). Compare the scaling, versus N and p, to what is expected from lecture (show plots and a discussion). Discuss the speedup and efficiency.

2. **Matrix-Matrix multiplication:** Generalize problem 1 to do matrix-matrix multiplication. Again, come up with some reasonable test data for the two matrices that makes it easy to see the veracity of the code. Again test the scaling to see if you get what is expected from the lecture discussion. Discuss the speedup and efficiency. Again, plotting results is expected here as well as a discussion of what the plots show.

3. **BLAS:** Alter the code in problem 2 to use BLAS routines for the actual multiply parts of the code (don't change the entire code, just few lines in the *local* part that multiplies 1 row by 1 column (or matrix)). In this case you can use the dot product function DDOT and it is only 1D vectors (so you do NOT need to worry about row-major or col-major order issues but NEED to use ncol for the stride). Again, do the timing and discuss any changes from what was observed in 2.

4. **Lapack** In lecture we discussed solving Poisson's equation using finite-differences and a direct solution of the resulting matrix equation using LAPACK. The code is posted on OWL. Test the code and time the runtime as a function of the linear size of

the system L (number of sites in each direction). What is the largest system you can run in a reasonable time (say 5 minutes)? How does the runtime scale with L? (Again a log-log plot may be useful here). What scaling is expected, did you get that?

<u>**What to hand in**</u>: See Assignment 1 for what is expected, I expect some plots for the scaling but it is ok to combine the plots where it makes sense to be able to compare results from different cases (eg. problem 2 and 3).