HPC Assignment 1

Samuel Richard Bonsor

In this assignment we write a parallel program designed to split the task of multiplying two $N \times N$ matrices between N processes. We are currently unable to deal with the case when the number of processes assigned ($nproc$) is less than N. Thus, our code must be run with $nproc \geq N$. Additionally, all of the files required to run this code on cirrus are included. Simply run the compile_now executable, change the $SLURM_SUBMIT_DIR in the .slurm file to the correct directory, and then run the job.

We now detail the operation of the code. The first step is to ensure that we are working with a set of exactly N processes. To this end we first check if the number of processors has been over-allocated ($nproc > N$). In this case we employ the functions MPI_Comm_group, MPI_Group_incl, and MPI_Comm_create_group in order to split off a subset of N processes into a new communicator.

The result of this is that as long as $nproc \geq N$ we are now working with a communicator of size N. We then generate the matrix B using only the root process, and use MPI_Bcast to broadcast this matrix to the other processes.

Now, as we have N processes in the communicator we may then construct the rows of the matrix A using the idea that $i = rank + 1$. This means that the process 0 produces the $i = 1$ row of A, while process 1 produces the second, and so on. This will be important for ensuring that the resulting matrix has its' rows in the correct order. It is also worth noting here that this is an example of geometric decomposition as a parallelisation strategy as we are exploiting the geometric structure of the matrix in order to split up the required tasks.

These individual rows are then multiplied by B to produce a row of the resultant matrix.

The function MPI_Gather may then be used to return each of the rows calculated to the master process, resulting in an array of length N^2 as the receive buffer. However, the gather command places the received data in the receive buffer in rank order of the processes. Due to how we indexed the construction of rows of A the first N elements of the receive buffer correspond to the first row of the required matrix, the second N elements the second row, and so forth. The task of putting the rows in the correct order is then very simple as we can loop the row number and pick out the required N elements from the receive buffer for each row.

The matrix, AB, is then output to a text file.

I do have some ideas for how to perform the generalisation to $nproc < N$. To balance the computational load as best I can I would split the processes into two groups, based on the remainder $= N\%nproc$. The processes of rank $\{0, ..., \text{remainder} - 1\}$ would need to produce $\lceil N/nproc \rceil$ rows of A. The remaining processes would then need to produce $\lfloor N/nproc \rfloor$ rows of A. I would then be able to gather all of the data into a single array using the MPI_Gatherv function to allow the different sized outputs from the two groups to be gathered into one array. Unfortunately I was unable to produce a working code using this framework in time.