Report on workshop 2 exercise 2

The code `amair_matmul.cpp` carries out Exercise 2 in the case where the number of processes employed is equal to the number of rows of the matrix A. The number of processes to be used is set to 3 in the `amair_matmul.slurm` file using the command:

```
#SBATCH --tasks-per-node = 3.
```

The first step of `amair_matmul.cpp` is to initialise the integer variables `rank` and `size`. Each process is identified by its `rank`, with `size` giving the total number of processes. Before initialising MPI, the matrix A is defined explicitly with `float Amatrix[3][3] = ...`, therefore meaning that all processes see the same matrix A. The 3 by 3 solution matrix D is also initiated prior to initialising MPI, along with the 9 entry vector named `Gathered_data`. The `Gathered_data` vector is where the vectors c of length 3, from each process, are stored after executing the `MPI_Gather` function. Since all 3 processes are to be used on this task, the default communicator of `MPI_COMM_WORLD` was selected. MPI was then initialised and a matrix `Bmatrix`, a vector `cvector` and a vector `Arow` were defined as `float` variables of the appropriate dimensions. An `if` command was then used to tell the the root process with `rank` 0 to fill the entries of its `Bmatrix` in accordance with the definition of B. Following this, the broadcast function

```
MPI_Bcast(&Bmatrix, 9, MPI_FLOAT, 0, MPI_COMM_WORLD)
```

was used to send the filled `Bmatrix` to processes 1 and 2. The first entry in `MPI_Bcast` is the reference to `Bmatrix`, the second entry shows that the 9 entries of B are being sent to each process, the third entry states that the matrix is filled with floating point numbers, the fourth entry states that the source process for the broadcast is the process with `rank` 0 and the final entry states the communicator being used. After starting the broadcast an `MPI_Barrier(MPI_COMM_WORLD)` ensures that the broadcast is complete, before each process defines its respective row of A and vector c. A further `MPI_Barrier(MPI_COMM_WORLD)` makes sure each process has fully computed its `cvector` before they are all gathered by process 0. Data was gathered using the function:

```
MPI_Gather(&cvector, 3, MPI_FLOAT, Gathered_data, 3, MPI_FLOAT, 0,
           MPI_COMM_WORLD).
```

The 1st entry is the reference to the c vector from each process, the 2nd entry says that each process will send 3 data, the 3rd says that floating point numbers are being sent. The 4th, 5th and 6th entries indicate respectively that the received data will be stored in `Gathered_data`, that 3 data are being received and that the data are floating point numbers. The 7th entry states that the data are gathered to the root process 0 and the last entry states the communicator. Once `MPI_Gather` is invoked, an `MPI_Barrier(MPI_COMM_WORLD)` command makes sure that all data has been received by process 0. After this an `if` statement is used to instruct process 0 to assemble the matrix product $D = AB$ from the c vectors that are consecutively stored in its version of `Gathered_data`. Thus completing the task.