# Assignment 4 - Matrix Data Processing

## Aims

- Revise and develop knowledge of the programming fundamentals using the Massage Passing Interface (MPI) platform.
- Implement a complex algorithm using a distributed computing architecture.
- Explore the complexities of task abstraction for computation by multiple machines.

## The Problem

The problem is to process the data in a matrix by replacing each coordinate value with the sum of its neighbours. For example: A 3 x 3 matrix:

```
1 2 1
2 3 1
1 0 1
```

The corresponding processed matrix:

```
7 8 6
7 9 7
5 8 5
```

A basic algorithm for achieving this involves the following:

**Step 1:** Read in the N x N matrix e.g.

```
1 2 1
2 3 1
1 0 1
```

**Step 2:** Supersize it to N+2 x N+2 by adding zeros around the borders:

```
0  0  0  0  0
0  1  2  1  0
0  2  3  1  0
0  1  0  1  0
0  0  0  0  0
```

Now all inner coordinates (the coordinates of the original matrix) have a full set of neighbours.

**Step 3:** Calculate the processed matrix by looking at each of the interior elements and adding up the values of its neighbours. Note: Each inner coordinate has 8 neighbours.

# You Task

Write a parallel program in C and MPI to process a square data matrix. There should be 1 clone process for each row of the data matrix. The main process reads the original matrix file and creates a supersized version. It sends 3 consecutive rows of the supersized matrix to each other process. That is, process *i*, will compute row *i* of the result matrix. It will require 3 rows (rows *i-1*, *i* and *i+1* of the supersized matrix), because it needs to check neighbours. All but the main process send back their computed row to the main process to write to a matrix file.

Matrix filenames and dimension should be provided to the program as command line arguments.

Submit your source code and makefile.

# Hints and Tips

- In order to implement this program, you should review the code from lecture 17:

Lecture 17 Examples

- These program store and retrieve matrices stored as files.

| Function/Item | Purpose |
|---|---|
| mkIdentityMatrix | makes an identity matrix of a given file name and size |
| mkRandomMatrix | makes a random matrix of a given file name and size |
| getMatrix | display the contents of a matrix of given filename and size |
| matrix.c | various functions for reading and writing matrix values |

- If you intend to leave debugging outputs in your code, make sure you include a debug mode using a macro switch. (I don't want to see large sections of commented-out code)

- Construct a full-strength error checking function for processing the return values of systems calls and library functions. (i don't want to see repetitive error checking code - Code once and reuse)
- Make sure your makefile has a *clean* target to remove all binary files for an easy rebuild.
- Build your program on `bourbaki` or a node of the cluster - not `turing` (MPI is not installed there).
- Use the `submit` program on `turing`. It is very easy (and foolproof) to submit directories rather than submit the assignment file by file.

## Tentative Marking Scheme

| item | Marks |
|---|---|
| ***The makefile*** | .... |
| targets | [/1] |
| uses -Wall | [/1] |
| compiles without warnings etc. | [/1] |
| ***The Program*** | .... |
| Creating the matrix | [/4] |
| Output | [/6] |
| Correct and Efficient Algorithm | [/8] |
| Error Checking | [/2] |
| Doesn't use hardwired constants | [/1] |
| Consistent Use of Good Style | [/1] |
| ***Total*** | 25 Marks |