These instructions should help you get started with the CS402 coursework. These instructions may be updated and so do keep an eye on the version number.

**Downloading code & supporting materials from the CS402 repository**

You can find a copy of the sequential code in the following directory:

```
/local/java/cs402/cfd/
```

Make a copy of these files in your home directory using:

```
cp -r local/java/cs402/cfd  ~
```

This will create a cfd directory in your home directory. Now compile the code by changing directory to cfd and by using the makefile (i.e. make).

The command line options can be listed by using:

```
karman --help
```

Then run the code using:

```
./karman
```

This will read in any previously saved state from karman.bin (if it exists), run the simulation for a number of time steps, and then write the output to karman.bin.

The output can be turned into an image using the bin2ppm command, i.e.

```
./bin2ppm < karman.bin > karman.ppm
```

This ppm file can be viewed with any UNIX image viewer such as xview (or kuickshow). Alternatively the ppm can be turned into a jpeg, gif or whatever using ppmtojpeg (etc.).

**Study the sequential program and familiarise yourself with the operations performed by the program; identify main data structures and parameters**

To profile the code you will need to modify the makefile so that CFLAGS has a `-pg` at the end of it. This will instruct the profiler gprof to record timing behaviour for the main routines in the code.

Once you have changed the makefile then you need to rebuild the code using:

```
make clean all
```
When you run the program it will generate profiling information to gmon.out, this can be viewed using the command:

```
gprof -bp ./karman
```

this generates a sorted list of the execution time spent in each function.

Or

```
gcc –pg filename.c –c
gcc filename.o –pg –o filename
./filename
Gprof filename > filename.gprof
```

This should give you a good idea as to where the majority of the runtime is spent. This is where you should focus your attention when thinking about parallelizing the code.

**Develop a parallel version of the program using MPI – calculate the relative speedup of your program against the sequential version**

The lectures and seminars have taught you how to program using MPI

colcopy.c is an example program that shows how to time an MPI program and send arrays of data between two processors.

At some point you will want to verify that your parallel code is producing equivalent answers to the sequential code. We have written a small utility called diffbin that will take two bin files and compare them. If there are any significant differences between the two then they may be a problem with your parallelization. For more information about this utility see:

```
./diffbin --help
```

**Generate a performance model for the application; use this to explore the performance space**

The timing behaviour of the program can be collected using MPI_Wtime. You can find out more about this by typing:

```
man MPI_Wtime
```

The performance characteristics of the network can be collected using the pingpong utility that you have been provided with. This utility must be run on exactly 2 machines that have been set up as an MPI ring (see lecture notes on MPI).

```
mpiexec -np 2 ./pingpong
```

This will print out a list of timings for sending a packet of a particular size and receiving a reply. This can be used to estimate network latency and bandwidth.