# Message-Passing Programming
# Cellular Automaton Exercise

## Aim

The aim of this exercise is to design serial and parallel algorithms to implement a simple cellular automaton which attempts to model traffic flow.

## The Model

The simulation box is a line of $N$ cells numbered $1, 2, \ldots, N-1, N$ (the road) which can each only have two values: 1 if a car is present on that section of road; 0 if it is empty. We will denote the value of cell number $i$ at time $t$ by $R^t(i)$.

The *update rules* for each time step are very simple:

- if the space in front of a car is empty then it moves forward one cell;
- otherwise it stays where it is.

Remember that we proceed through a series of iterations where the values of the cells at time $t+1$ depend entirely on the values at time $t$. In other words you should be able to apply the update rules to the cells at time $t$ in any order and you should still get the same values at $t+1$.

After each iteartion we compute the *average velocity* of the cars. This is calculated as the number of cars that move in an iteration divided by the total number of cars, and is a value between 0 and 1.

We use *periodic boundary conditions*, ie when a car moves off the right-hand-side of the road it reappears on the left, and vice-versa (this is the same as saying we are simulating a roundabout rather than a straight section of road). To do this, we identify $R^t(0)$ (the cell to the left of cell number 1) with $R^t(N)$ and $R^t(N+1)$ (the cell to the right of cell number $N$) with $R^t(1)$.

## 1 Rules for Cellular Automaton

It should be fairly clear that, when we update to a new timestep, the new value $R^{t+1}(i)$ depends on its old value at time $t$, and also on the old values of the neighbours. In other words, $R^{t+1}(i)$ depends on the three values $R^t(i-1)$, $R^t(i)$ and $R^t(i+1)$.

You should fill in Table 1 and Table 2 to get the explicit form of the update rules for this cellular automaton. For convenience, we have split the 8 possible cases into two tables, one for an initially empty cell $R^t(i) = 0$, and the other for a full cell $R^t(i) = 1$. Writing things out in this explicit form you can see that there are in fact 256 possible 1D cellular automata (each corresponding to different update rules), but very few of them are in fact of any interest!

Having written the update rules, you should now write a piece of pseudo-code that implements them for a road of length $N$. The easiest approach is to have two arrays, one corresponding to the old values and one to the new values, and simply to loop through the elements and update the new array based on the old one. You should also consider how best to implement the periodic boundary conditions.

|  | $R^t(i-1)=0$ | $R^t(i-1)=1$ |
|---|---|---|
| $R^t(i+1)=0$ |  |  |
| $R^t(i+1)=1$ |  |  |

Table 1: Values of $R^{t+1}(i)$ if $R^t(i)=0$

|  | $R^t(i-1)=0$ | $R^t(i-1)=1$ |
|---|---|---|
| $R^t(i+1)=0$ |  |  |
| $R^t(i+1)=1$ |  |  |

Table 2: Value of $R^{t+1}(i)$ if $R^t(i)=1$

## 2 Parallelisation

Examine the pseudo-code that you have just written and consider how you would split up the calculation among multiple processors. Points to consider include:

- What parts of the calculation could be done independently and what parts would require some form of cooperation (eg synchronisation or communication)?
- What is the best way to handle any communications in your parallel code?
- Is it possible to implement both the boundary conditions and the communications using the same basic approach?

Remember also that you not only have to update the cars at each generation, but you also have to compute the total number of cars that move in order to calculate the velocity.

## 3 Message Passing Implementation

How would you implement your parallel scheme using a message-passing programming model on a distributed-memory parallel machine? Points to consider include:

- How would you divide the data amongst the different processes?
- When is communications required?
- How would you calculate the velocity at each timestep?

Write some pseudo-code to illustrate your solution, assuming that you have access to simple routines to send and receive data between processes. You should consider two cases:

1. sending data is done *asynchronously* like sending a letter;
2. sending data is done *synchronously* like making a phone call.

Be careful to avoid deadlock, particularly in the second case.

If you prefer to think of an analogy, imagine that you and a friend want to simulate the traffic model by hand on the blackboards in your offices. You decide to split the calculation between you, and luckily your offices are relatively nearby. Unfortunately, the sound insulation is very good and you don't have a phone, so the only way to communicate is to go and speak to each other. You each can also only work in your own office as the blackboards are only large enough to hold half of the road. How would you organise the calculation? Try and come up with a scheme that minimises the number of times you have to get up and leave your office. How would you generalise your algorithm for three, four or eight people?