

Scientific Computing

Cellular Automaton Exercise

EPSRC

NERC SCIENCE OF THE ENVIRONMENT

 **archer**

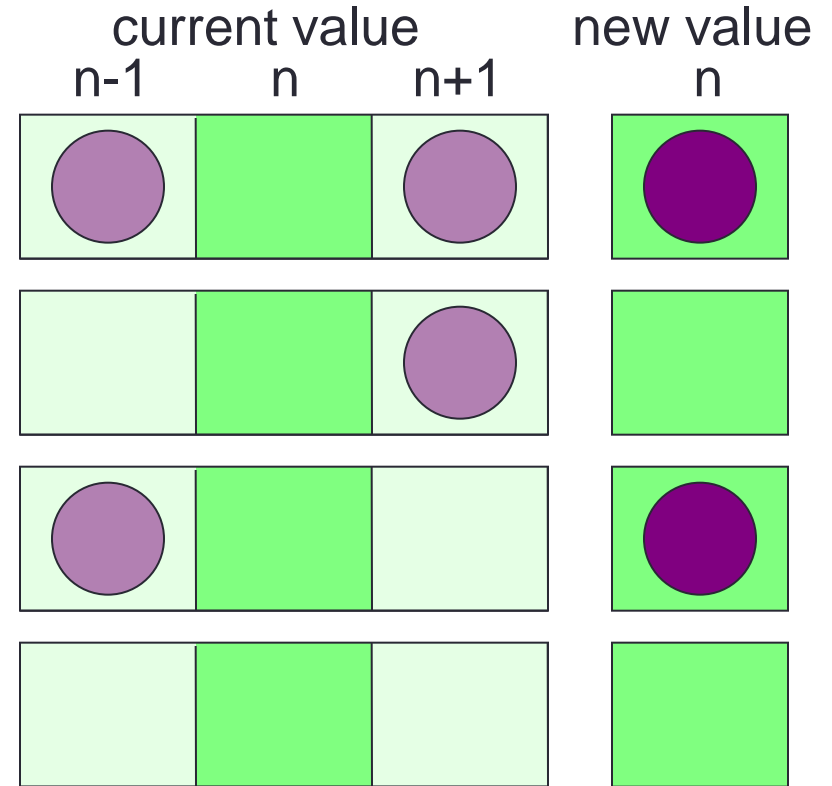
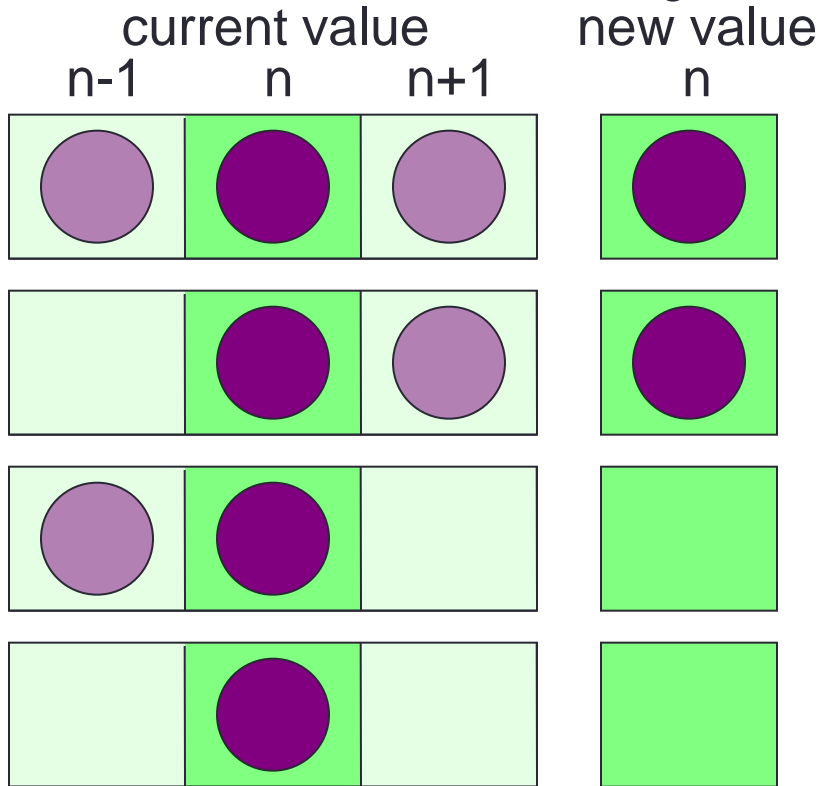
CRAY
THE SUPERCOMPUTER COMPANY

epcc



Traffic simulation

- Update rules depend on:
 - state of cell
 - state of nearest neighbours in both directions



State Table

- If $R^t(i) = 0$, then $R^{t+1}(i)$ is given by:

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

- If $R^t(i) = 1$, then $R^{t+1}(i)$ is given by:

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

Pseudo Code

```
declare arrays old(i) and new(i), i = 0,1,...,N,N+1
initialise old(i) for i = 1,2,...,N-1,N (eg randomly)
loop over iterations
  set old(0) = old(N) and set old(N+1) = old(1)
  loop over i = 1,...,N
    if old(i) = 1
      if old(i+1) = 1 then new(i) = 1 else new(i) = 0
    if old(i) = 0
      if old(i-1) = 1 then new(i) = 1 else new(i) = 0
  end loop over i
  set old(i) = new(i) for i = 1,2,...,N-1,N
end loop over iterations
```



Threads Parallelisation

- Load balance not an issue
 - updates take equal computation regardless of state of road
 - split the road into equal pieces of size N/P
- For each piece
 - rule for cell i depends on cells $i-1$ and $i+1$
 - can parallelise as we are updating new array based on old
- Synchronisation required
 - to ensure threads do not start until boundary data is updated
 - to produce a global sum of the number of cars that move
 - to ensure that all threads have finished before next iteration

Shared Variables Parallelisation

serial: initialise `old(i)` for $i = 1, 2, \dots, N-1, N$

serial: loop over iterations

serial: set `old(0) = old(N)` and set `old(N+1) = old(1)`

parallel: loop over $i = 1, \dots, N$

 if `old(i) = 1`

 if `old(i+1) = 1` then ...

 if `old(i) = 0`

 if `old(i-1) = 1` then ...

 end loop over i

synchronise

parallel: set `old(i) = new(i)` for $i = 1, 2, \dots, N-1, N$

synchronise

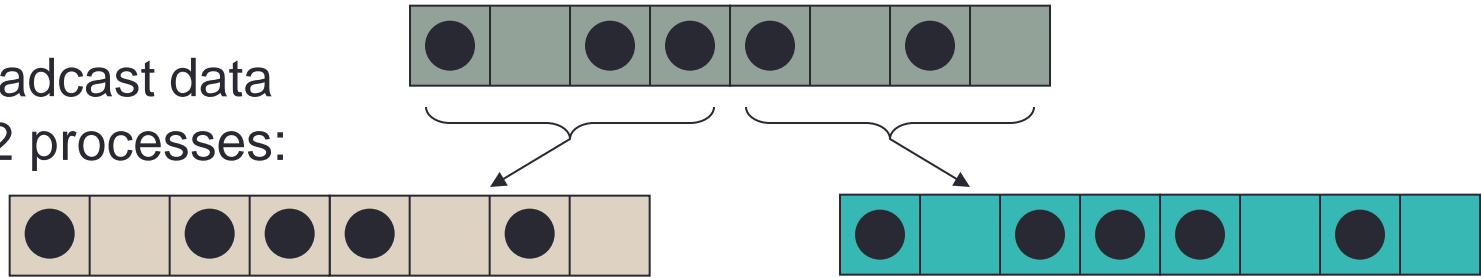
end loop over iterations

- private: i ; shared: `old`, `new`, N
 - reduction operation to compute number of moves



Message-Passing Strategy (1)

Broadcast data
to 2 processes:



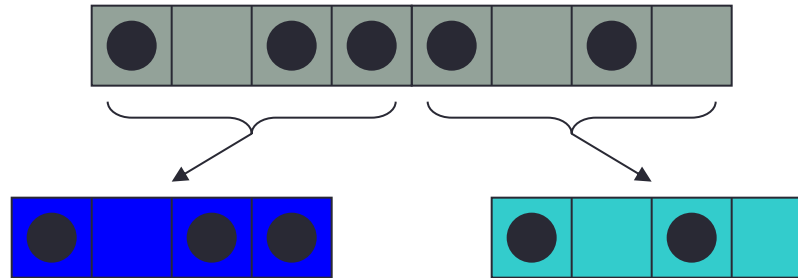
Split calculation
between 2 processes:



- Globally resynchronise all data after each move
 - a **replicated data** strategy
- Every process stores the entire state of the calculation
 - e.g. any process can compute total number of moves

Parallelisation Strategy (2)

Scatter data
between 2 processes:
distributed data strategy



- Internal cells can be updated independently.
- Must communicate with neighbouring processes to update edge cells.
- Sum local number of moves on each process to obtain total number of moves at each iteration.

Split calculation
between 2 processes:



- Each process must know which part of roadway it is updating.
- Synchronise at completion of each iteration and obtain total number of moves.

Parallelisation

- Load balance not an issue
 - updates take equal computation regardless of state of road
 - split the road into equal pieces of size N/P
- For each piece
 - rule for cell i depends on cells $i-1$ and $i+1$
 - the $N/P - 2$ interior cells can be updated independently in parallel
 - however, the edge cells are updated by other processors
 - similar to having separate rules for boundary conditions
- Communications required
 - to get value of edge cells from other processors
 - to produce a global sum of the number of cars that move

Message Passing Parallelisation

