Assignment 1: Red/Blue movement

By Yuanqi Pang
ID: 460490911
Date: 18/04/2018

1. Problem description

The Red/Blue computation simulates two interactive flows: an n by n board is initialized so cells have one of three colors: red, white, and blue, where white is empty, red moves right, and blue moves down. (The board may be initialized with roughly 1/3 cells in read, 1/3 in white and 1/3 in blue and colors should be interleaved and spread across the board. You need to write a separate function board_init to initialize the board.) In the first half step of an iteration, any red color can move right one cell if the cell to the right is unoccupied (white); on the second half step, any blue color can move down one cell if the cell below it is unoccupied (white); the case where red vacates a cell (first half) and blue moves into it (second half) is okay. Note: any red or blue color can only move at most one cell in one iteration. Colors wraparound to the opposite side when reaching the edge. Viewing the board as perfectly overlaid with a t by t grid of tiles (i.e., t divides n, and every tile contains n/t by n/t cells), the computation terminates if any tile's colored cells are more than c% one color (blue or red).

Requirements

- Use MPI to write a solution to the Red/Blue computation.
- Assume the processes are organized as a one-dimensional linear array.
- Each process will hold a number of k rows of tiles, or $k \times n/t$ rows of cells for $k \geq 0$.
- For the purpose of load balancing, processes should have roughly the same number of rows and the difference not greater than one tile row, or n/t cell rows.
- Your program must produce correct results for nprocs being greater than or equal to one.
- Your program needs to ask for 4 user defined parameters (integers) as inputs: cell grid size n, tile grid size t, terminating threshold c, and maximum number of iterations max_iters.
- Your program needs to print out which tile (or tiles if more than one) has the colored squares more than c% one color (blue or red).
- After the parallel computation, you main program must conduct a self-checking, i.e., first perform a sequential computation using the same data set and then compare the two results.

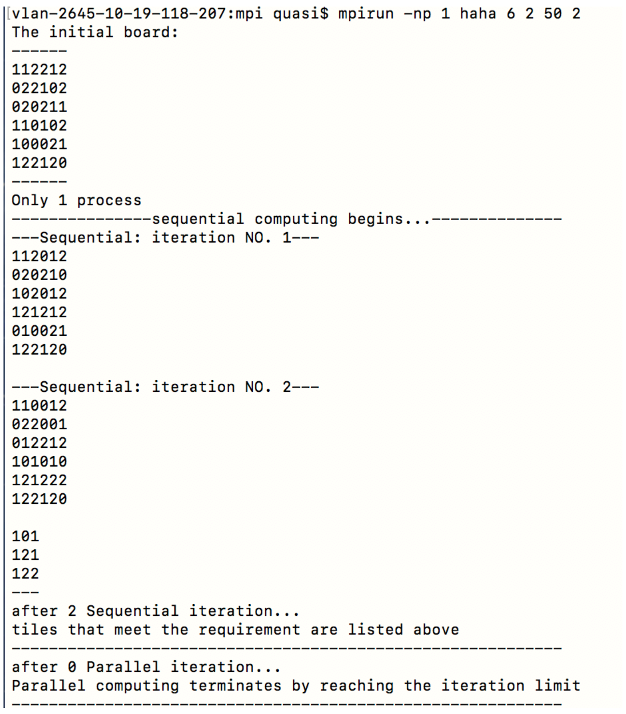
2. Algorithm design

- A board_init method is to initialize a nxn board with random 0-2 integers and print out the initial board.
- A sequential method to computing sequentially and called when there is only 1 processor or to compare with the parallel computing.
- After board initialize, the grid is separated and sent to all the processors by using the MPI send and receive method, if the tiles cannot be devided by number of procs, proc id which is lower than tile%procs is assigned 1 more tile than the rest. If the number of procs greater than tiles, each proc get 1 tile and rest do not work.
- All the procs start computing red movement and red movement do not need communication between procs. Then the blue movement, all procs send the first row to the former processor. Each processor computes the lines that do not need data from others and compute the final row with the received data.
- The board will be print out after each iteration and the output can be compared with the sequential computing
- A barrier is set for all procs to hang on until all procs finish the movement before checking.
- All the procs start computing all the color percentage in all tiles that are assigned to them. Any tile that meet the terminate requirement will be printed out.
- Any tile in any processor meeting the requirement will set the stop signal to 1. An Allreduce method is used to notify all other procs if any processor met the termination requirement

3. Implementation and Testing and Issue

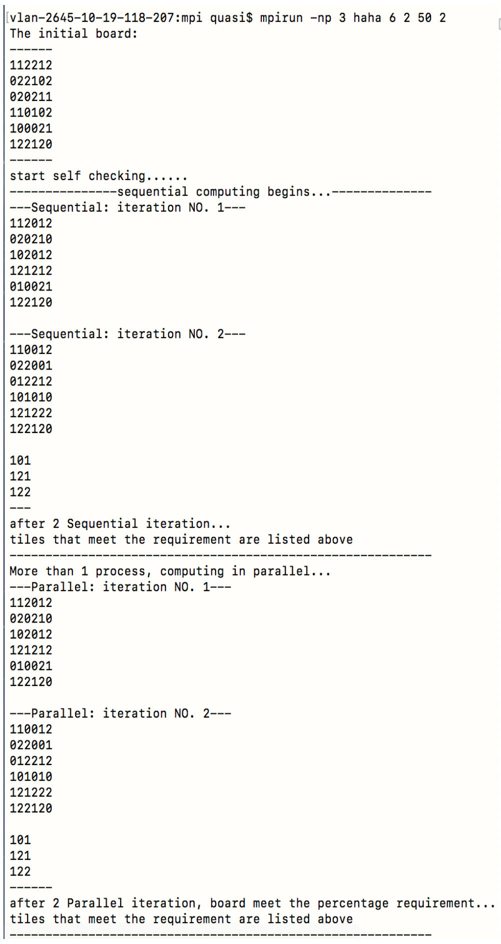
The test part is carried out on remote connection to soit-mpi-pro-1.ucc.usyd.edu.au by using the private macbook.

When there is only 1 processor, the project will carry out only the sequential computing. And there will be no sequential self-checking. Which is shown in the below screenshot.

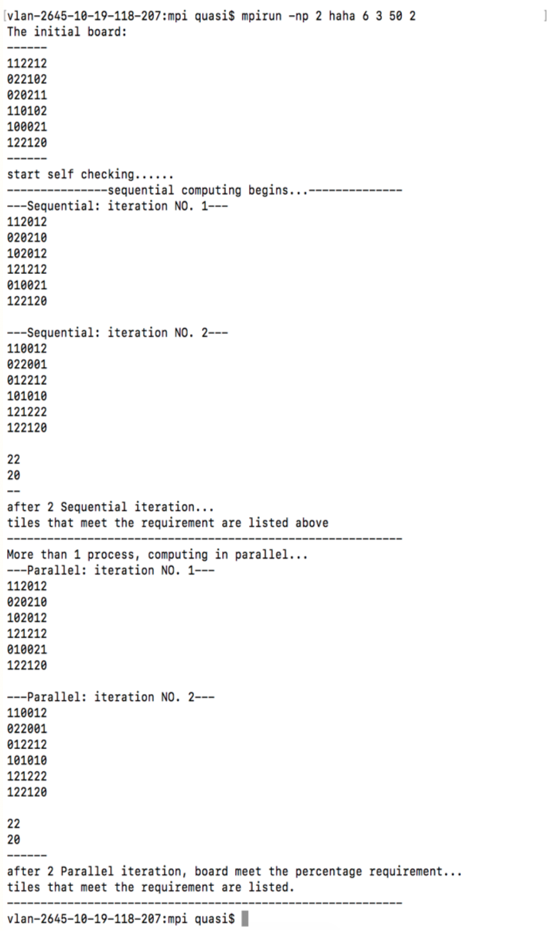


```
[vlan-2645-10-19-118-207:mpi quasi$ mpirun -np 1 haha 6 2 50 2
The initial board:
------
112212
022102
020211
110102
100021
122120
------
Only 1 process
---------------sequential computing begins...--------------
---Sequential: iteration NO. 1---
112012
020210
102012
121212
010021
122120

---Sequential: iteration NO. 2---
110012
022001
012212
101010
121222
122120

101
121
122
---
after 2 Sequential iteration...
tiles that meet the requirement are listed above
------------------------------------------------------------
after 0 Parallel iteration...
Parallel computing terminates by reaching the iteration limit
------------------------------------------------------------
```

And when numprocs greater or smaller than tile number, the project work normally, the test result. The result seems to be correct and the sequential result is the same as parallel result as shown below.

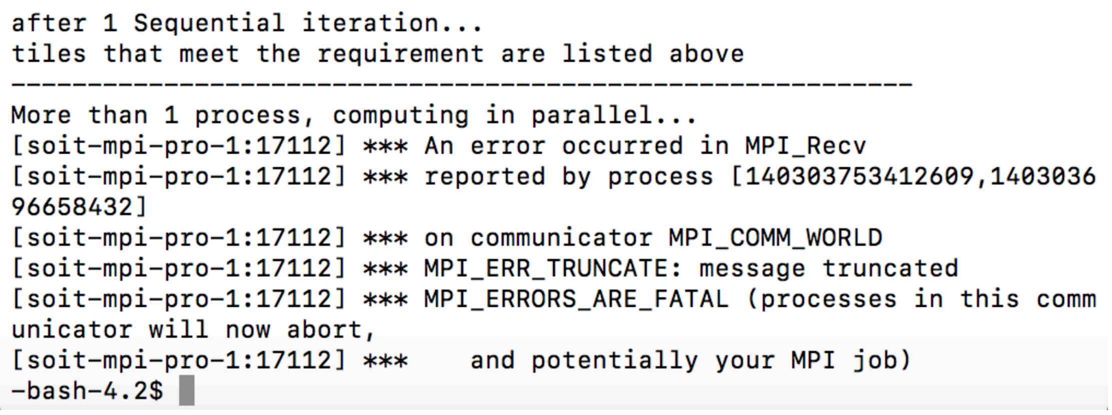


```
[vlan-2645-10-19-118-207:mpi quasi$ mpirun -np 3 haha 6 2 50 2
The initial board:
------
112212
022102
020211
110102
100021
122120
------
start self checking......
---------------sequential computing begins...--------------
---Sequential: iteration NO. 1---
112012
020210
102012
121212
010021
122120

---Sequential: iteration NO. 2---
110012
022001
012212
101010
121222
122120

101
121
122
---
after 2 Sequential iteration...
tiles that meet the requirement are listed above
------------------------------------------------------------
More than 1 process, computing in parallel...
---Parallel: iteration NO. 1---
112012
020210
102012
121212
010021
122120

---Parallel: iteration NO. 2---
110012
022001
012212
101010
121222
122120

101
121
122
------
after 2 Parallel iteration, board meet the percentage requirement...
tiles that meet the requirement are listed above
------------------------------------------------------------
```



```
[vlan-2645-10-19-118-207:mpi quasi$ mpirun -np 2 haha 6 3 50 2
The initial board:
------
112212
022102
020211
110102
100021
122120
------
start self checking......
---------------sequential computing begins...--------------
---Sequential: iteration NO. 1---
112012
020210
102012
121212
010021
122120

---Sequential: iteration NO. 2---
110012
022001
012212
101010
121222
122120

22
20
--
after 2 Sequential iteration...
tiles that meet the requirement are listed above
------------------------------------------------------------
More than 1 process, computing in parallel...
---Parallel: iteration NO. 1---
112012
020210
102012
121212
010021
122120

---Parallel: iteration NO. 2---
110012
022001
012212
101010
121222
122120

22
20
------
after 2 Parallel iteration, board meet the percentage requirement...
tiles that meet the requirement are listed.
------------------------------------------------------------
vlan-2645-10-19-118-207:mpi quasi$
```

However, the project does not go well when using mpirun -np 3 haha 10 5 50 2, this is the only input that the developer found which lead to this system crash. The system can work well with n=16 and t=8, numprocs = 3, the developer do not know the reason for this crash,The information is shown below. And the system will also go wrong when size of grid n is too big(>25), the project will blocked at parallel part when the n is too big.



```
after 1 Sequential iteration...
tiles that meet the requirement are listed above
------------------------------------------------------------
More than 1 process, computing in parallel...
[soit-mpi-pro-1:17112] *** An error occurred in MPI_Recv
[soit-mpi-pro-1:17112] *** reported by process [140303753412609,1403036
96658432]
[soit-mpi-pro-1:17112] *** on communicator MPI_COMM_WORLD
[soit-mpi-pro-1:17112] *** MPI_ERR_TRUNCATE: message truncated
[soit-mpi-pro-1:17112] *** MPI_ERRORS_ARE_FATAL (processes in this comm
unicator will now abort,
[soit-mpi-pro-1:17112] ***    and potentially your MPI job)
-bash-4.2$
```

In all the testing part, the sequential computing print the same result as the parallel computation.

4. User Manual

Open the terminal, and compile the c file by using:

```
mpicc -o xxx main.c
```

Then the project can be run using:

mpirun -np numprocs xxx [number of grids in a row: n] [number of tiles in a row: t] [terminate threshold c(70 means 70%)] [max iteration times]

For example: `mpirun -np 3 haha 6 2 50 2`

Warning:
length of grid n must be divisible by number of tile t
c should be integer from 0 to 100
numprocs can be lower, greater or equal to number of tile t.

Output:

1. The initial board
2. The sequential result (board after each iteration and the specific result of tiles that meet the requirement) for comparison and self-checking.
3. The parallel result (board after each iteration and the specific result of tiles that meet the requirement).