PARALLELISING A SMALL WORLD NEURAL NETWORK MODEL FOR GENERATION OF DATA USING THE MPI AND PTHREADS LIBRARY

A REPORT BY :
AVYAV KUMAR SINGH
2013A7PS084G

THE HODGKIN HUXLEY MODEL

The Hodgkin–Huxley model, or conductance-based model, is a mathematical model that describes how action potentials in neurons are initiated and propagated. It is a set of nonlinear differential equations that approximates the electrical characteristics of excitable cells such as neurons and cardiac myocytes, and hence it is a continuous time model, unlike the Rulkov map for example. The typical Hodgkin–Huxley model treats each component of an excitable cell as an electrical element (as shown in the figure). The lipid bilayer is represented as a capacitance (C_m). Voltage-gated ion channels are represented by electrical conductances (g_n, where n is the specific ion channel) that depend on both voltage and time. Leak channels are represented by linear conductances (g_L). The electrochemical gradients driving the flow of ions are represented by voltage sources (E_n) whose voltages are determined by the ratio of the intra- and extracellular concentrations of the ionic species of interest. Finally, ion pumps are represented by current sources (I_p). The membrane potential is denoted by V_m. Mathematically, the current flowing through the lipid bilayer is written as

$$I_c = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t}$$

and the current through a given ion channel is the product

$$I_i = g_i(V_m - V_i)$$

where V_i is the reversal potential of the *i*-th ion channel. Thus, for a cell with sodium and potassium channels, the total current through the membrane is given by:

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + g_K(V_m - V_K) + g_{Na}(V_m - V_{Na}) + g_l(V_m - V_l),$$

where *I* is the total membrane current per unit area, C_m is the membrane capacitance per unit area, g_K and g_{Na} are the potassium and sodium conductances per unit area, respectively, V_K and V_{Na} are the potassium and sodium reversal potentials, respectively, and g_l and V_l are the leak conductance per unit area and leak reversal potential, respectively. The time dependent elements of this equation are V_m, g_{Na}, and g_K, where the last two conductances depend explicitly on voltage as well.

Using a series of voltage clamp experiments and by varying extracellular sodium and potassium concentrations, Hodgkin and Huxley developed a model in which the properties of an excitable cell are described by a set of four ordinary differential equations. Together with the equation for the total current mentioned above, these are:

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_K) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

where *I* is the current per unit area, and and α_i are rate β_i constants for the *i*-th ion channel, which depend on voltage but not time. is the maximal value of the conductance $\bar{g}$. *n*, *m*, and *h* are dimensionless quantities between 0 and 1 that are associated with potassium channel activation, sodium channel activation, and sodium

channel inactivation, respectively. For $p = (n, m, h)$, α_p and β_p take the form

$$\alpha_p(V_m) = p_\infty(V_m)/\tau_p$$
$$\beta_p(V_m) = (1 - p_\infty(V_m))/\tau_p.$$

p_∞ and $(1 - p_\infty)$ are the steady state values for activation and inactivation, respectively, and are usually represented by Boltzmann equations as functions of V_m. In the original paper by Hodgkin and Huxley, the functions α and β are given by

$$\alpha_n(V_m) = \frac{0.01(V_m - 10)}{\exp\left(\frac{V_m - 10}{10}\right) - 1} \qquad \alpha_m(V_m) = \frac{0.1(V_m - 25)}{\exp\left(\frac{V_m - 25}{10}\right) - 1} \qquad \alpha_h(V_m) = 0.07\exp\left(\frac{V_m}{20}\right)$$
$$\beta_n(V_m) = 0.125\exp\left(\frac{V_m}{80}\right) \qquad \beta_m(V_m) = 4\exp\left(\frac{V_m}{18}\right) \qquad \beta_h(V_m) = \frac{1}{\exp\left(\frac{V_m - 30}{10}\right) + 1}$$

While in many current software programs, Hodgkin–Huxley type models generalize α and β to

$$\frac{A_p(V_m - B_p)}{\exp\left(\frac{V_m - B_p}{C_p}\right) - D_p}.$$

WHAT THE CODE DOES

This code generates a small world network with N nodes with k average connections per node. It then simulates N stochastic HH neurons connected in such a topology using the 4th order Runge - Kutta method for seconds. It calculates Mean ISI (Inter Spike Interval), Firing Correlation, Avg. Firing Correlation etc. The parallelisation process is implemented via POSIX standard-compliant threads and MPI (Message Passing Interface) framework for generating parallel processes of different trials, for different rates of coupling strengths on each core.

Methodology

- Each discrete value of coupling strength is accompanied by 10 trials, with the coupling strength values decided by the user.
- Each trial will launch a new process rooted from the coupling strength process and each process will take place on one core.
- In effect, each parent process (representing a discrete value of coupling strength) will have 10 trial processes.
- It will continue running till the trial process ends.
- This way, with a limited number of cores, a new trial (or effectively a new process) is launched only when a currently executing trial ends.
- This limits competition for resources if, say, two trials are executed simultaneously.

File Organisation

The important files which are generated at the end of the process are the following:

- FCDegreeVaryingD(100,8) (0.9; 0.6).txt
- FiringCoherenceVaryingDs(100,8) (0.9;0.6).txt
- ISI_no_of_connections(100,8) (0.9,0.6).txt
- MeanISI(100,8) (0.9,0.6).txt

The above files are tabulated for the whole number of coupling strength and trial values. However, if these are computed by different trials, then their final values will be scrambled if they are being written in the same file. Hence, for each trial, different files must be made.

For the file **ISI_no_of_connections(100,8) (0.9,0.6).txt**, the values written are calculated for a particular value of coupling strength. These would be calculated into a file for the whole parent process, and finally parsed into a main file.

The files **FiringCoherenceVaryingDs(100,8) (0.9;0.6).txt, MeanISI(100,8) (0.9,0.6).txt** and **FCDegreeVaryingD(100,8) (0.9; 0.6).txt** are computed again for each discrete value of coupling strength. These files' writing will be expedited by the main process.

The time taken is bound by **ceiling(<total_trials>/<number of cores>) + additional time taken for file parsing** (negligible).

Each trial has numerous file I/O operations associated with it. Hence, usage of threads is vital to provide some amount of concurrency.

Control Flow Cycle

Process launched for each value of coupling strength. It launches a process for a discrete value of a trial
Communication via message passing for each trial process

Total processes:

(number of firing coherence values)*(number of trials)

Internal flow within a trial

Synchronised via threads internally

- 093 – 163 - Thread section
- 163 – 192 - Sequential
- 197 – 254 - Thread section
- 254 – 300 - Sequential
- 301 – 358 - Thread section
- 360 – 418 - Sequential
- 422 – 427 - Thread section
- 424 – 441 - Thread section
- 445 – 452 - Thread section
- 453 – 516 - Sequential
- 518 – 544 - Thread section
- 546 – 547 - Thread section
- 548 – 575 - Sequential
- 579 – 588 - Thread section
- 588 – 596 - Sequential
- 599 – 637 - Thread section
- 637 – 824 - Sequential

HIERARCHY OF CONTROL

Hosts are the nodes used for the working of all the processes. The hosts used in the system are bits-goa, n0 and n1.

The code is launched under the following control mechanism

Process 0 of bits-goa acts as the scheduling core which enables it to manage each coupling task as it is completed. Each coupling task consists of 10 trials (each mapped to a separate core) and a secondary scheduler which manages these trials. As soon as the coupling task is completed, a signal is sent to the scheduling core which maps another coupling process to the recently evacuated cores.

The relations of the processes are given as - 0 is the scheduling process

(11n + 1) is the coupling task process

(11n + 2 ... 11n + 11) is the trial process

A list of all the files used for the coding of the neural network -

KV.c	- mathematical function
Kcoupling.c	- mathematical function
Kcoupling1.c	- mathematical function
Kh.c	- mathematical function
Km.c	- mathematical function
Kn.c	- mathematical function
clusteringcoefficient.c	- mathematical function
functions.h	- header file (all decls. of functions)
gaussrandh.c	- mathematical function
gaussrandm.c	- mathematical function
gaussrandn.c	- mathematical function
meanshortest.c	- mathematical function
randomnumbergenerator.c	- generates random numbers
small-world-network-HH-model-temp.c	- top level file; combines everything
variables.c	- contains declarations of variables
variables.h	- header file (all decls. of variables)
declare_init_files.c	- used to intialise coupling files
declare_trial_files.c	- used to initialise trial files
alloc_2d_double.c	- allocates 2d contiguous mem. of type double
alloc_2d_float.c	- allocates 2d contiguous memory of type float
alloc_2d_int.c	- allocates 2d contiguous memory of type int

A list of scripts used -

construct_directories.c	- constructs folders for separated files
construct_rankfile.c	- makes a rankfile which maps process to core
merge_files.c	- appends the output of all files into one file
worker_file.c	- top level script; starts the process

The process also generates a large number of text files. All these files are stored in separate folders, hence there is no scrambled output as a result of race conditions. At the end of the program, these files are merged to give a complete output.

A list of text files generated -

On a coupling task level -

FiringCoherenceVaryingDs(100,8) (0.9; 0.6).txt **captures fir coh. / diff. coupling strengths**
FCDegreeVaryingD(100,8) (0.9; 0.6).txt **captures total degree of the network**
ISI_no_connections(100,8) (0.9; 0.6).txt **captures the number of connections / ISI**
MeanISI(100,8) (0.9; 0.6).txt **calculates mean ISI + standard deviation**

On a trial level -

V_vs_T.txt	- generates temporary data
Spiketime.txt	- generates temporary data
Y.txt	- generates temporary data
VT.txt	- generates temporary data
RewiredMatrix.txt	- generates temporary data
Randomnumbers.txt	- generates temporary data
NoOfConnections.txt	- generates temporary data
m_noise.txt	- generates temporary data
mvalues.txt	- generates temporary data
hvalues.txt	- generates temporary data
nvalues.txt	- generates temporary data

Note - All the coupling level files are merged for the final output, which consists of the same files.

Guidelines and Restrictions -

To launch the code, cd into the directory where the executables are present. Contents should be -

```
$ ls
worker_file          cleanup               contruct_directories hostfile                  merge_files
```

Run the following command

```
$ ./worker_file <d_min> <d_max> <d_step>
```

arguments are -
d_min - The minimum coupling strength value for the code d_max - The maximum coupling strength value for the code d_step - The step intervals between the minimum and maximum value

The following must be observed -

Always ensure that d_min >= 0.001 and d_step >= 0.001
There must be an arithmetic progression between d_min and d_max with d_step as the common difference.

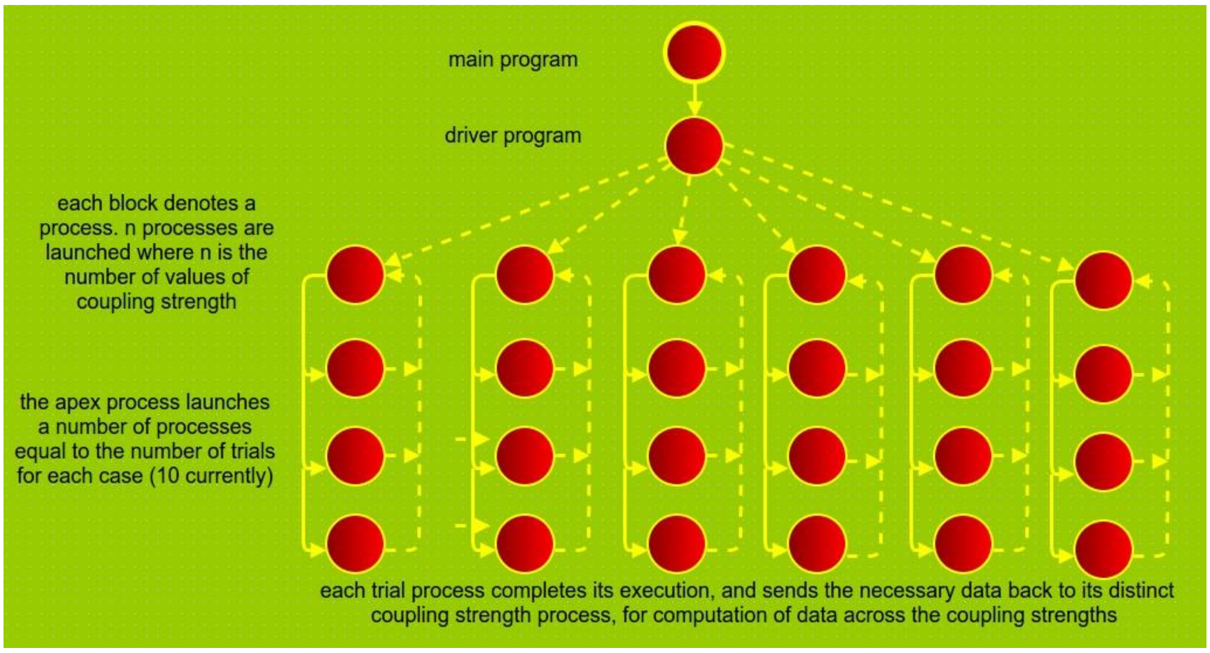
The number of processes generated will always [(d_min - d_max)/d_step + 1]. It is strictly advised to keep the number of processes generated under 400. Greater than 400 processes will lead to massive oversubscribing, which will result in a very slow outcome or the process crashing.

After the process has been completed, run

```
$ ./cleanup
```

to get rid of files and temporary folders generated. ALL THE TEXT FILES WILL BE DELETED. It is advised to have a backup in case the result needs to be stored somewhere. While the process is running, it is advised not to cd into the directory where it is running, or try to launch any text file. This might lead to the program crashing.

The diagram of the processes can be shown as


main program
driver program
each block denotes a process. n processes are launched where n is the number of values of coupling strength
the apex process launches a number of processes equal to the number of trials for each case (10 currently)
each trial process completes its execution, and sends the necessary data back to its distinct coupling strength process, for computation of data across the coupling strengths

COMPRARISONS OF THE TIME TAKEN BY THE SERIAL AND PARALLEL CODE

For the serial code -

```
real    743m51.209s    user
691m13.735s            sys
35m7.096s
```

For the parallel code -

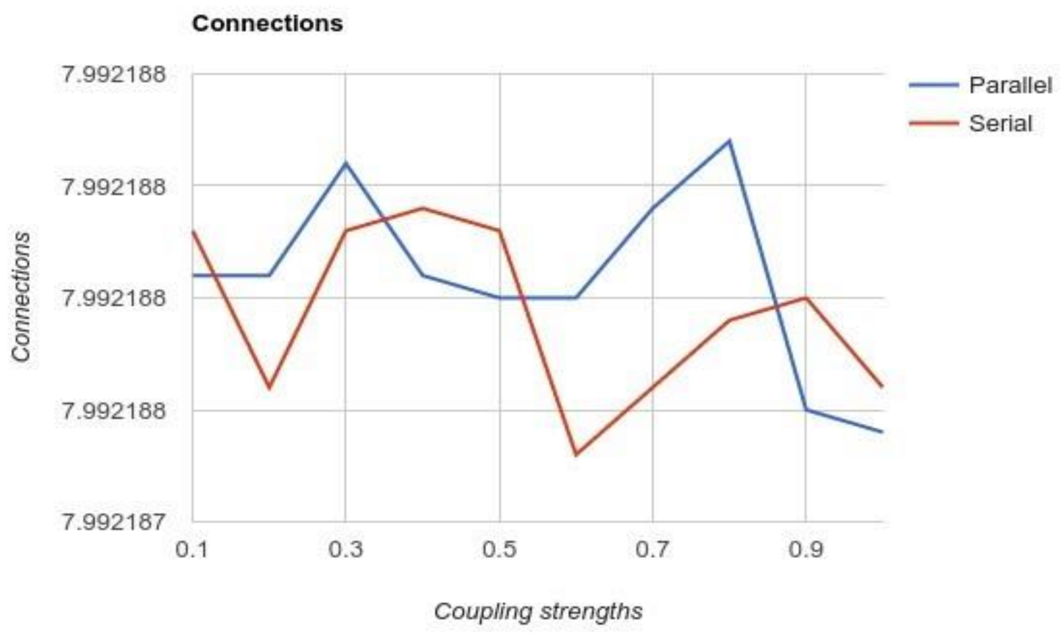
```
real    4m18.074s     user
27m8.659s             sys
17m43.414s
```

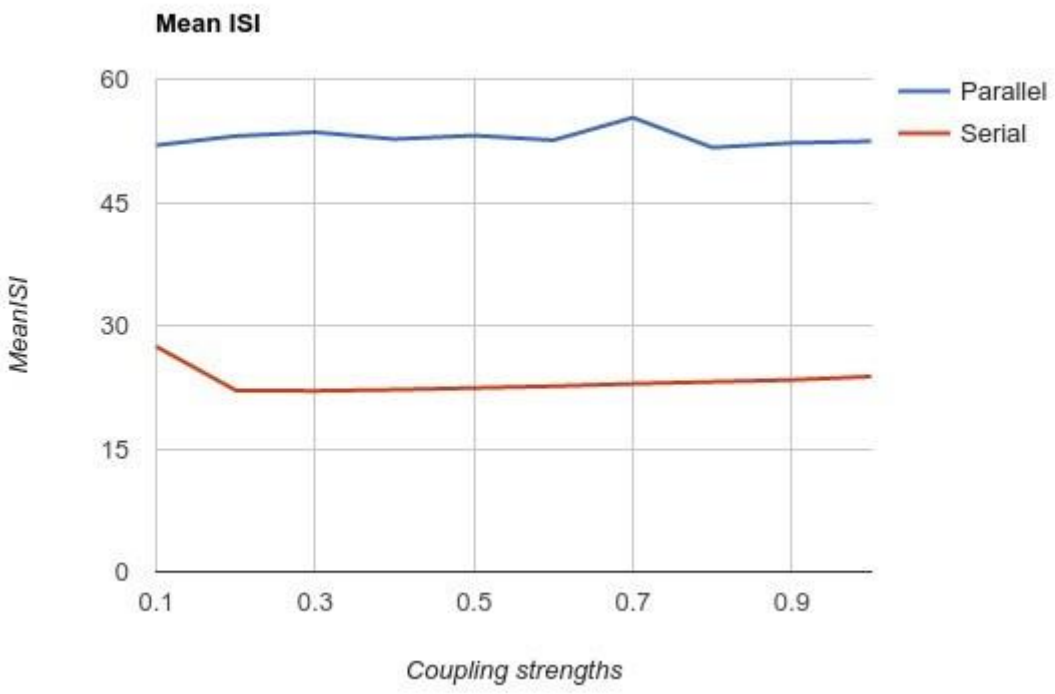
Improvement Factor = ($T_{real,serial}$ / $T_{real, parallel}$)

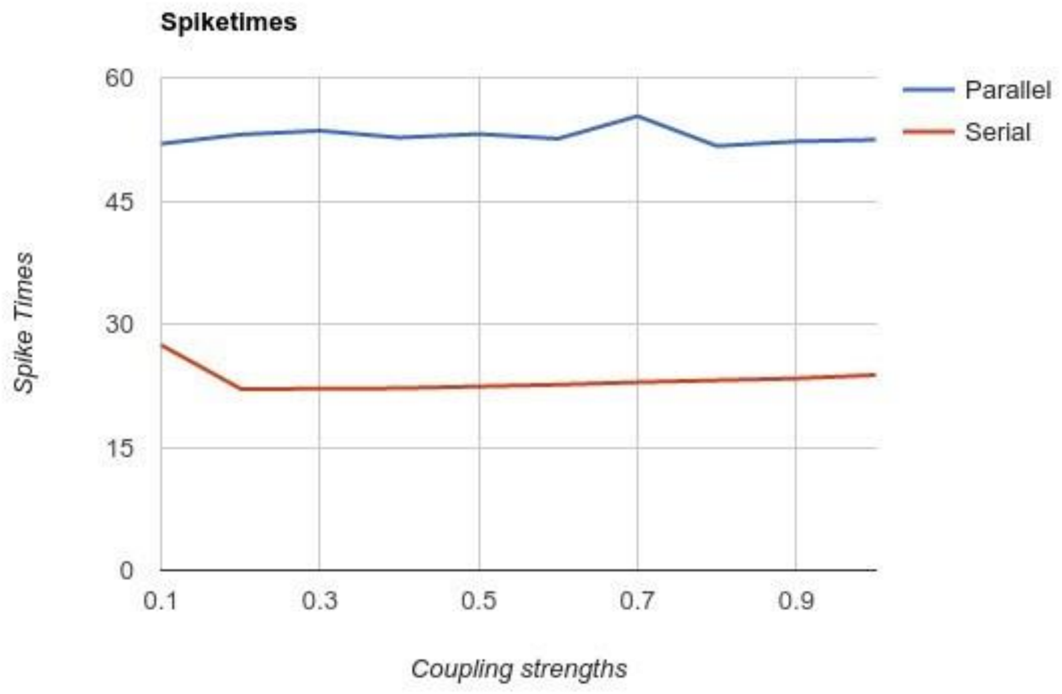
= (44,631.209 / 258.074)

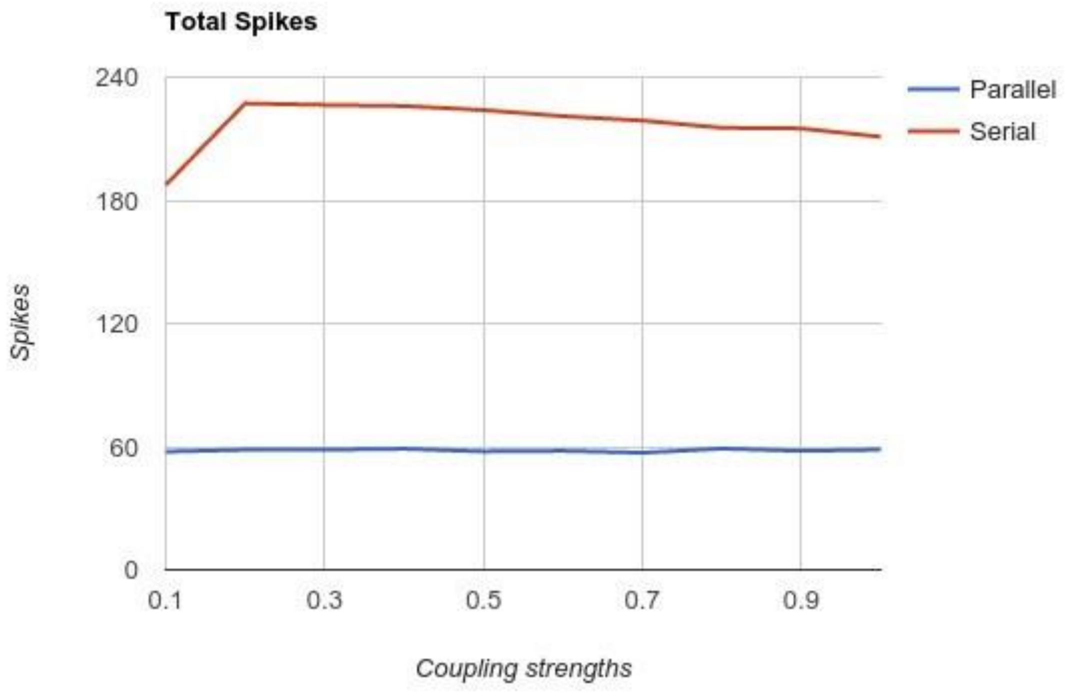
= 172.9396

COMPRARISONS BETWEEN SETS OF INTERMEDIATE DATA GENERATED

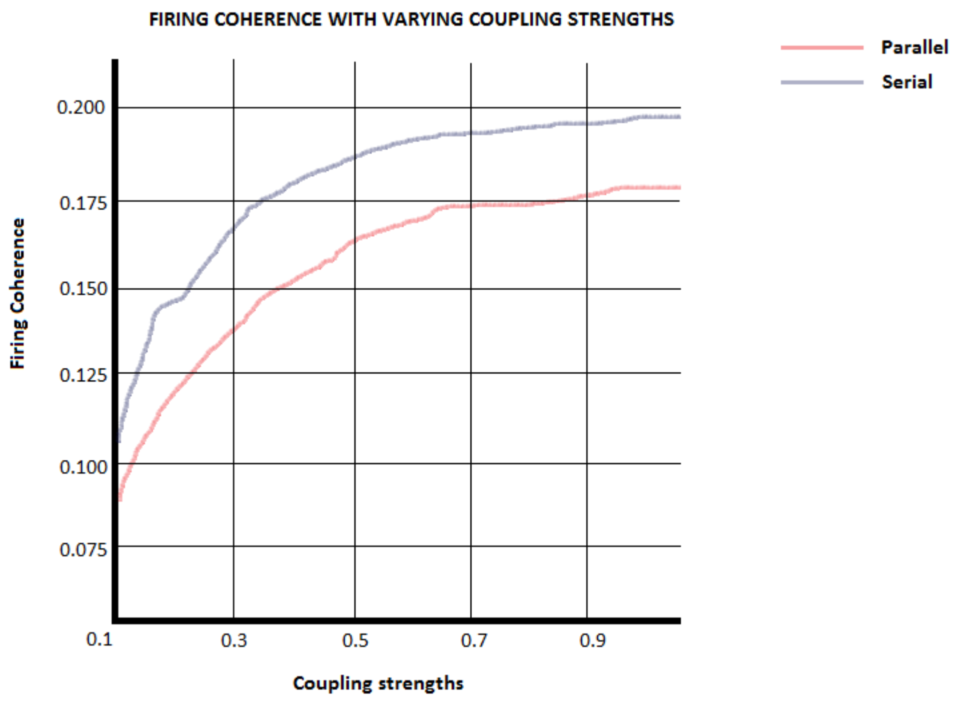





Spiketimes
60
45
30
15
0
Spike Times
0.1
0.3
0.5
0.7
0.9
Coupling strengths
Parallel
Serial


Total Spikes
240
180
120
60
0
Spikes
0.1
0.3
0.5
0.7
0.9
Coupling strengths
Parallel
Serial

FINAL DATA GENERATED – ANALYSIS AND RECOMMENDATIONS



As can be inferred, the trend is more/less similar for all the graphs.

It might be noticed that the number of spikes generated in a parallel version is lesser than that in a serial version. Hence, the ISI of the parallel code is much larger than that of a serial version, as there would be more time intervals between two intervals in parallel code as compared to serial code.

On examination of the **spiketime.txt** file for both the serial and parallel code, it was found that the spike times for serial code had higher variance. The spike times are derived from the usage of a random variable. In C, random variables are generated by the means of a clock which is initialised to 0 when the program starts. Since there are multiple processes running, it looks like there is a conflict wherein if two processes calculate the random variable at the same time – they get the same value as it is an output dependent on CPU clock cycles – an outcome common to both processes.

There looks to be a difference in the randomising function wherein the pattern of random values generated is not followed for multi-threaded process vs a single threaded process.

Recommendation – Use a randomising function which is independent of compute clock cycles.