Multiple Processor Systems CMPE – 665

Project Assignment 1

Vignesh Govindarajulu Kothandapani

Univ ID : 831001854

October 30, 2014.

Abstract:

This project is aimed at parallelizing the sequential implementation of the Hodgkin-Huxley neuron model. Here the parallelization is implemented using the message passing interface (MPI) model using the MPI_Send() and MPI_Recv() functions. MPI operates on master-slave relationship where the processor with rank 0 is the master and rest all are slaves, the values calculated by the slaves are sent to the master which calculates the final value by summing up the values received. Over here, the parallel code execution time is in-turn compared with the sequential code execution time and the speedup is calculated.

Design Methodology:

The project requirement states that, if the number of dendrites is equal to the number of slaves, then the required computation will be parallelized only by the slaves. In other cases, computations will be parallelized in both the slaves and master.

In the first case mentioned above each dendrite is processed by a separate processor, this is achieved by replacing the dendrite variable with the rank of the processor being used and the results are send to the master processor to calculate the final values.

In the latter case, each processor including the master is assigned a set of dendrites i.e. the value obtained after dividing the number of dendrites with the number of processors. The remaining dendrites are processed by the master. The intermediate values are sent to the master processor when the values are accumulated to obtain the final output.

Here initially the updated values of y [0] are sent in to all the slaves before the somaparam[2] calculation. So here basically the MPI_Send and MPI_Recv commands are used for sending and receiving the y [0] and somaparam[2] values.

Analysis and Results:

The first table shows the effect on excecution time of the program with the increase in number of compartments for the same number of dendrites.

Note: The execution time is not exactly correct as it is required to be due to cluster problems.

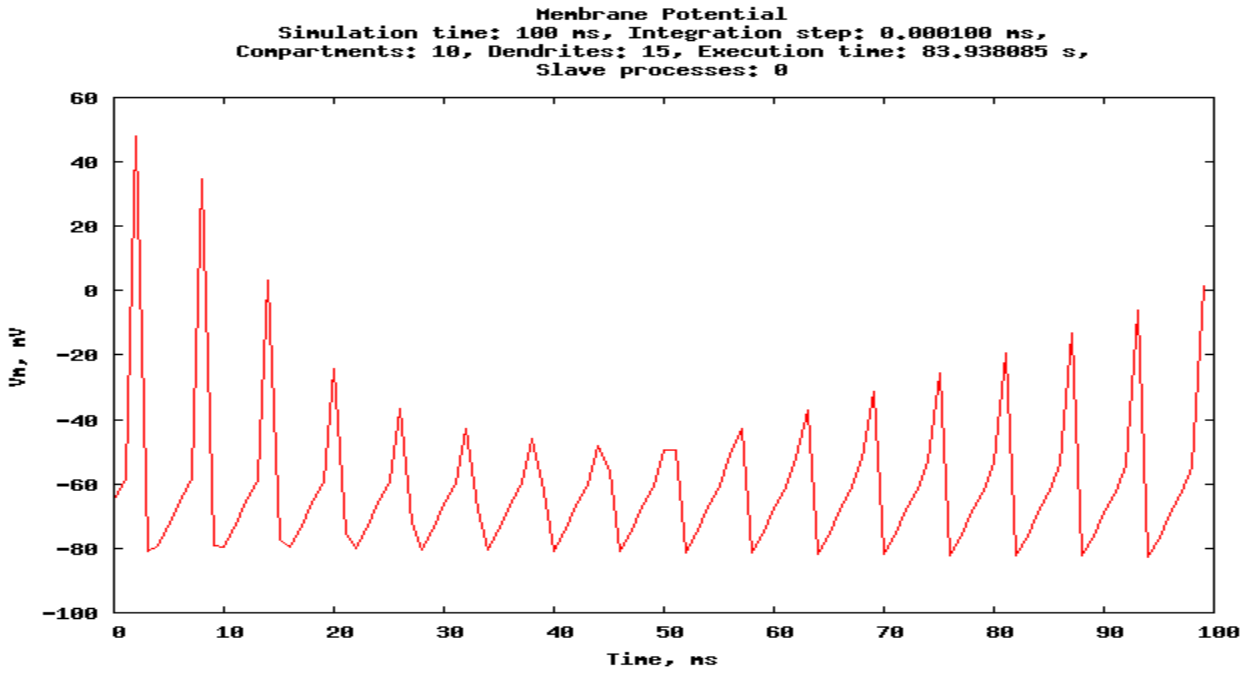
Answers related to question 1:

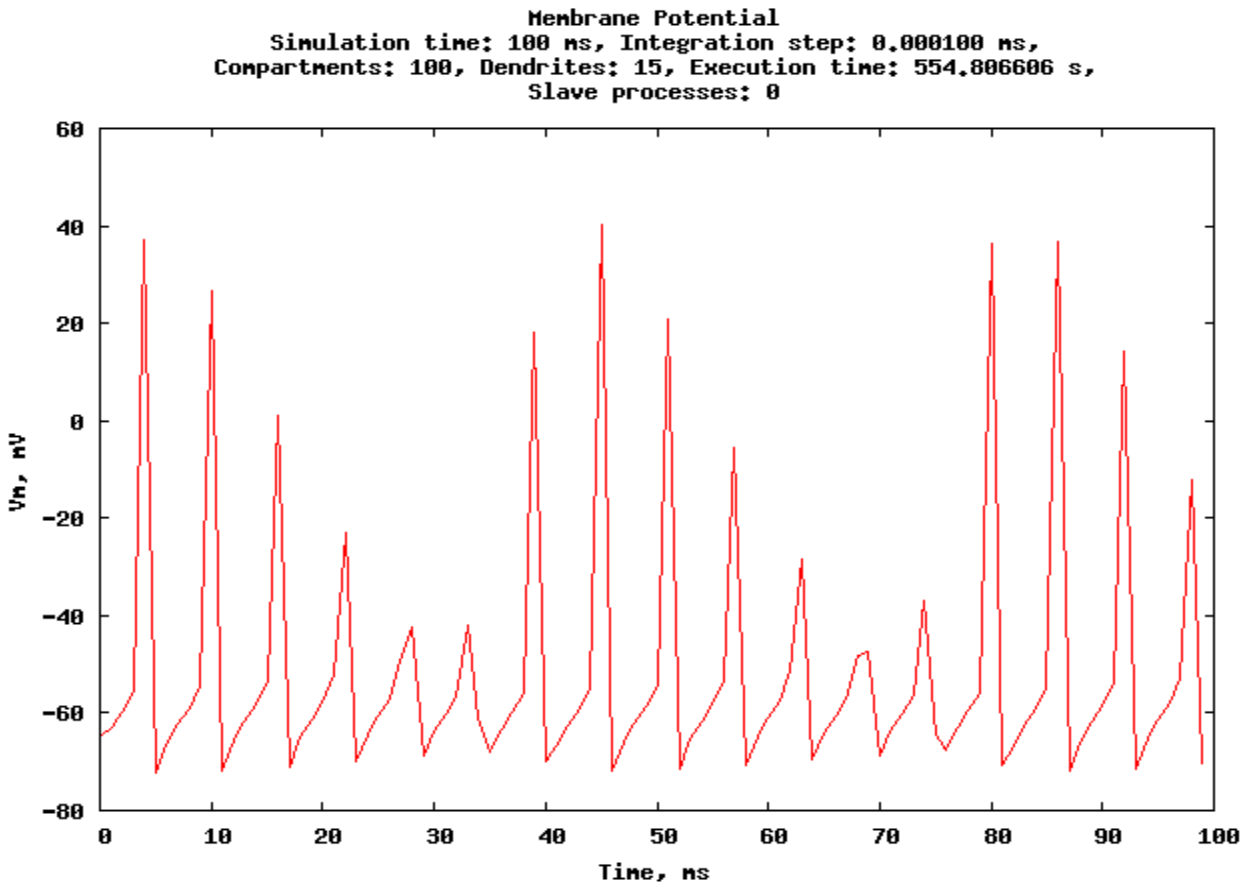
The first table shows the effect on execution time of the program with the increase in number of dendrites for the same number of compartments.

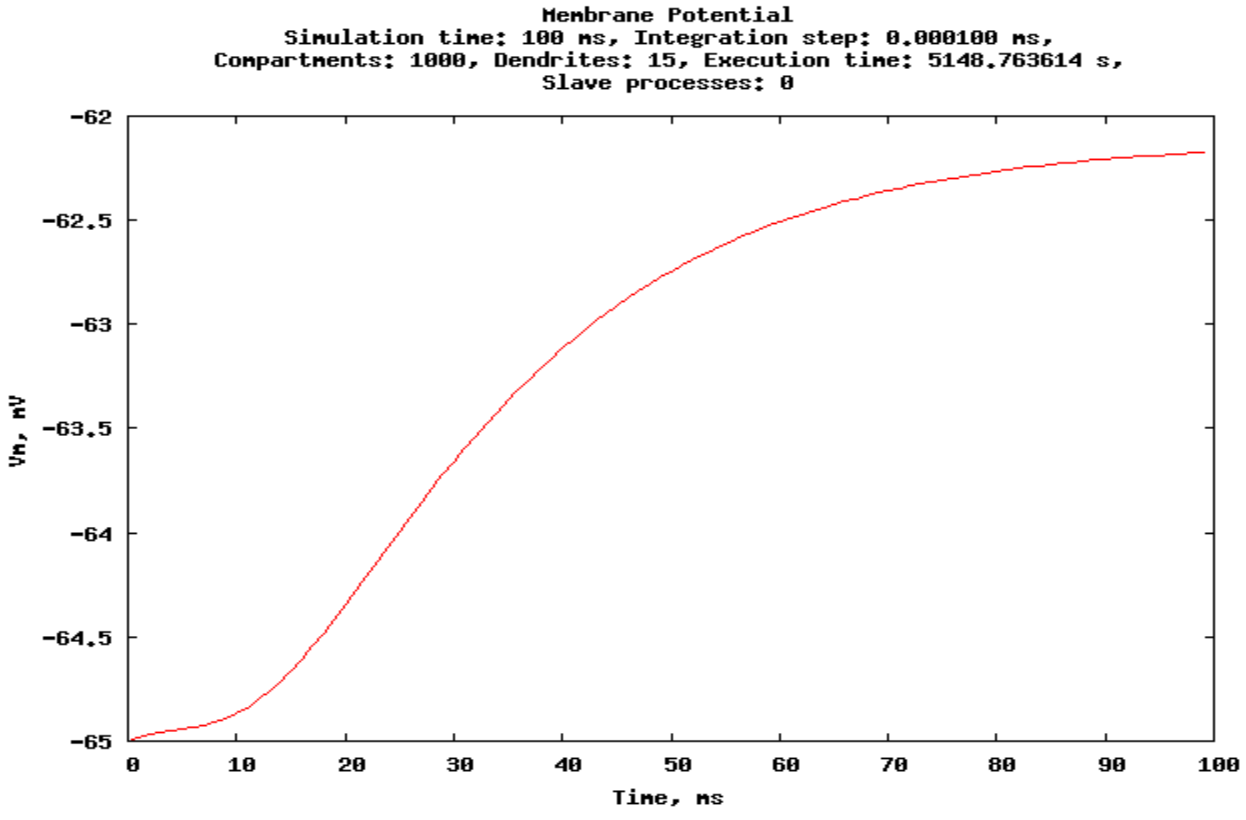
Number of Slaves	Dendrites	Compartments	Exec Time (sec)	Speedup
0, sequential	15	10	83.938085	1
5 (mpirun –np 6)	15	10	5584.553265	
12 (mpirun –np 13)	15	10	1224.858790	
0, sequential	15	100	554.806606	1
5 (mpirun –np 6)	15	100	825.032842	
12 (mpirun –np 13)	15	100	2867.215186	
0, sequential	15	1000	5148.763614	1
5 (mpirun –np 6)	15	1000	N/A	
12 (mpirun –np 13)	15	1000	13605.141941	

Table 1: Effect of Dendrite Length on Computational Load

Graphs:




Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 100, Dendrites: 15, Execution time: 554.806606 s,
Slave processes: 0
Vn, mV
Time, ms
60
40
20
0
-20
-40
-60
-80
0
10
20
30
40
50
60
70
80
90
100


Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 1000, Dendrites: 15, Execution time: 5148.763614 s,
Slave processes: 0
Vn, mV
Time, ms
-62
-62.5
-63
-63.5
-64
-64.5
-65
0
10
20
30
40
50
60
70
80
90
100

What effects do you observe from the parallel implementation? Explain why these effects are present.

In this case the results are flawed, but in general case the parallel implementation of a code executes faster than the sequential part. As the dendrites are divided among different processors results are obtained faster. Speedup might be reduced if there is large communication overhead.

Why is speedup not the same as the number of parallel processes?

The speedup is not equal to the number of processors because of the communication overhead present.

What can you tell about the spiking patterns of the *soma* graphs relative to the dendrite length?

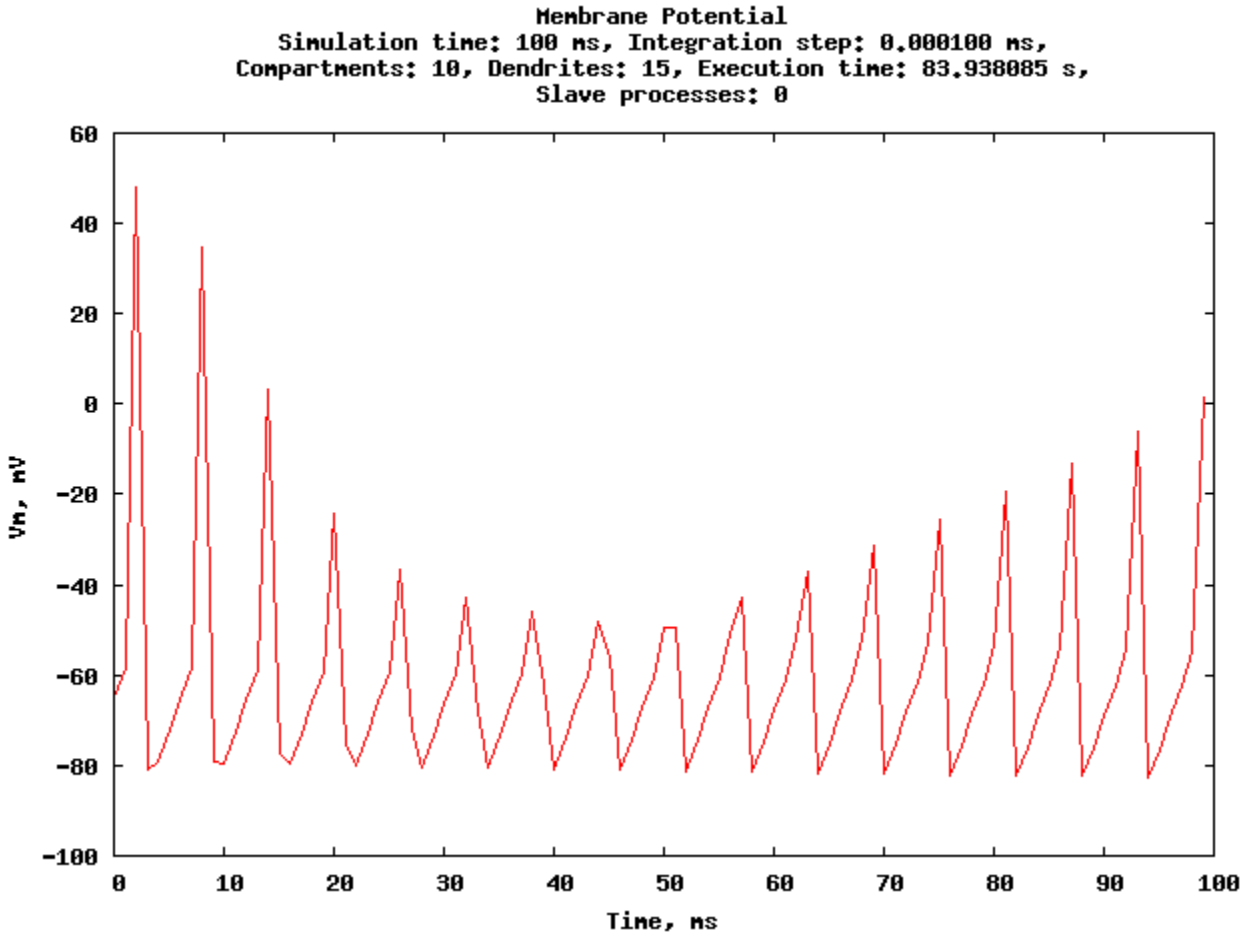
The spiking turns out to be in exponential form with the increase in the dendrite length.

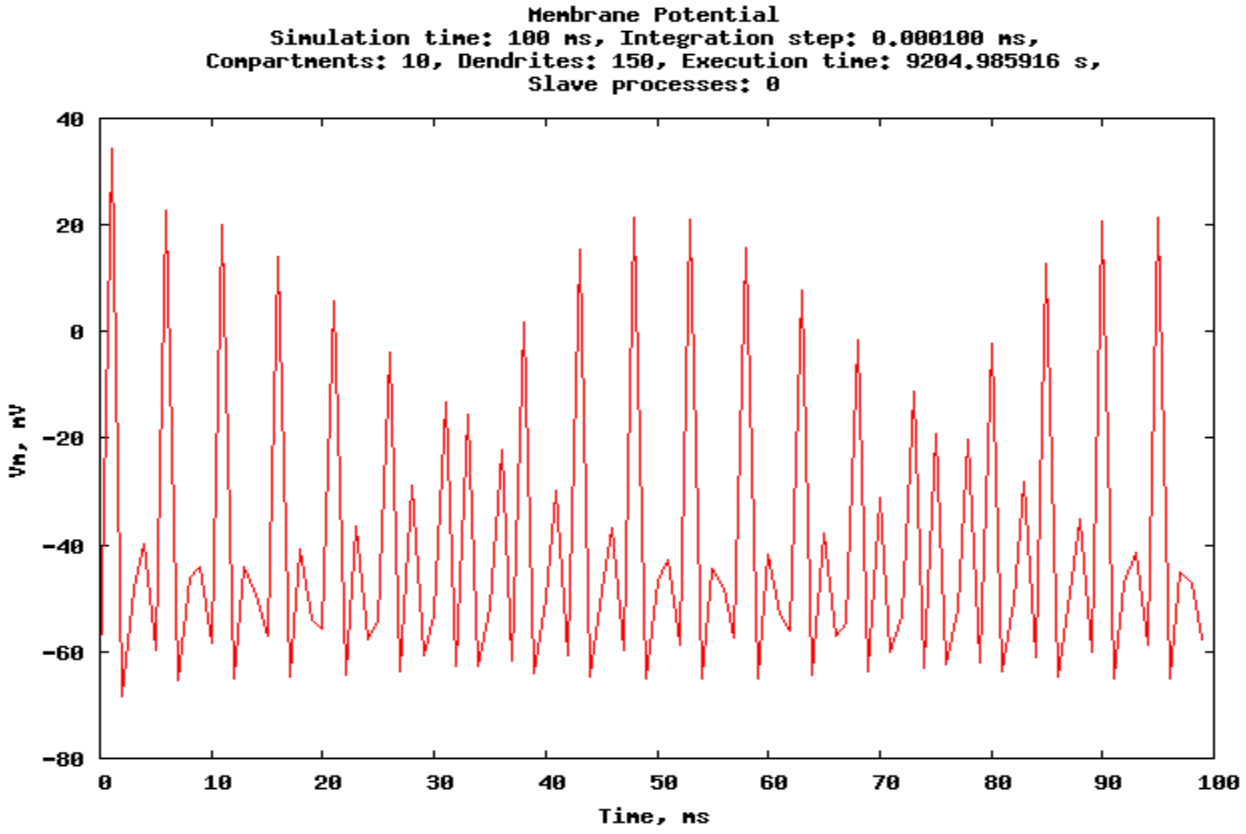
Answers related to question 2:

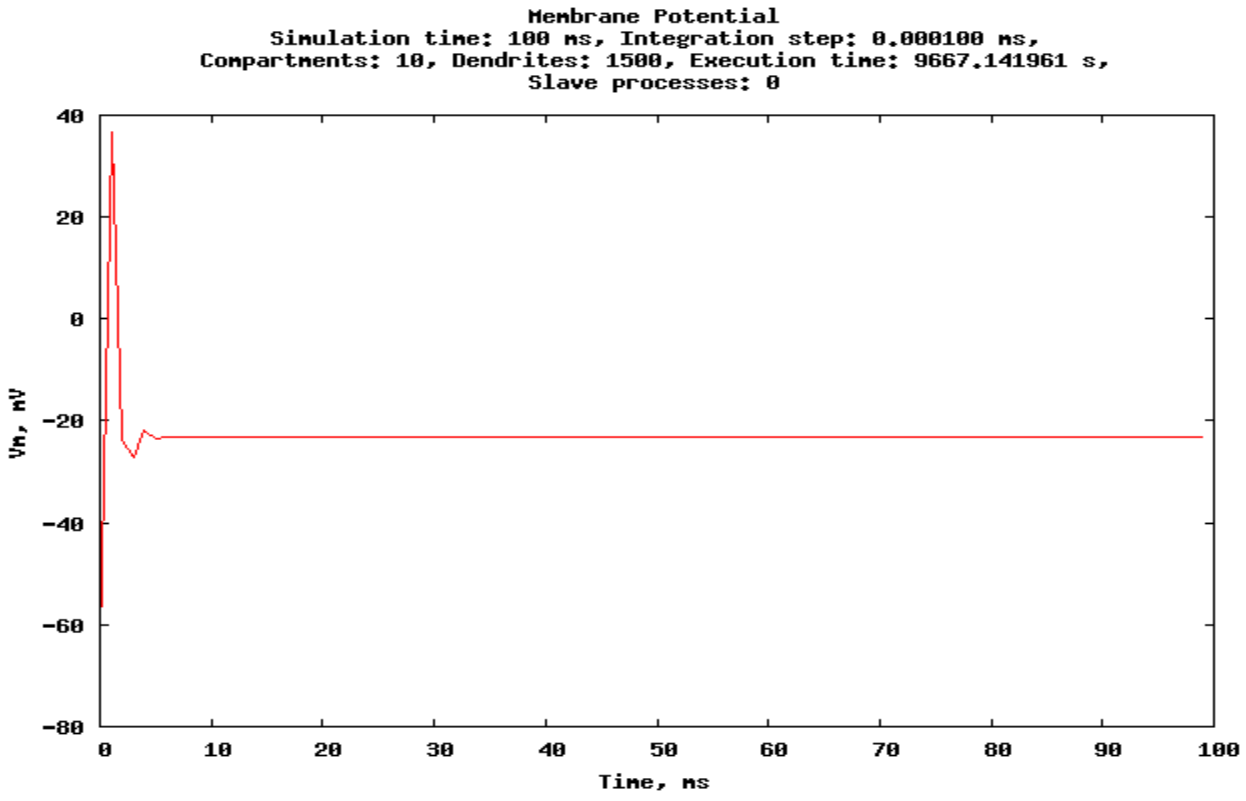
Number of Slaves	Dendrites	Compartments	Exec Time (sec)	Speedup
0, sequential	15	10	83.938085	1
5 (mpirun –np 6)	15	10	5584.553265	
12 (mpirun –np 13)	15	10	1224.858790	
0, sequential	150	10	1008.726178	1
5 (mpirun –np 6)	150	10	9204.985916	
12 (mpirun –np 13)	150	10	1235.479761	
0, sequential	1500	10	9667.141961	1
5 (mpirun –np 6)	1500	10	9116.699011	
12 (mpirun –np 13)	1500	10	N/A	

Table 2: Effect of Number of Dendrites on Computational Load

Graphs:




Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 10, Dendrites: 150, Execution time: 9204.985916 s,
Slave processes: 0
Vm, mV
Time, ms
40
20
0
-20
-40
-60
-80
0
10
20
30
40
50
60
70
80
90
100


Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 10, Dendrites: 1500, Execution time: 9667.141961 s,
Slave processes: 0
Vm, mV
Time, ms
40
20
0
-20
-40
-60
-80
0
10
20
30
40
50
60
70
80
90
100

What effects do you observe from your parallel implementation? Explain why these effects are present.

In this case the results are flawed, but in general case the parallel implementation of a code executes faster than the sequential part. As the dendrites are divided among different processors results are obtained faster. Large number of dendrites increases the level of parallelization and better results are obtained.

What similarities/differences do you see when comparing to the data from Table 1? Explain.

The increase in the number of denrites or number of compartments or both result in the increase of execution time.

What can you tell about the spiking patterns of the *soma* relative to the number of dendrites? Compare to the data obtained from the experiments used to generate Table 1.

The spiking reduces with the increase in the number of dendrites for the same number of compartments. With higher number of dendrites, the output almost comes flat.

Answers related to question 3:

The third table shows the effect on execution time of the program with load imbalance:

#	Number of Slaves	Dendrites	Compartments	Exec Time (sec)
1	10 (mpirun –np 11)	30	100	N/A
2	10 (mpirun –np 11)	33	100	N/A
3	10 (mpirun –np 11)	36	100	N/A

Table 3: Test for Load Imbalance

Note : No graphs generated due to cluster issues.

Compare and comment on the execution time of experiment #1 and #3 relative to #2 in Table 3. What does this tell you about how effective your program is at balancing computational load?

From the table it is clear that #2 is load balanced as compared to #1 and #3. Thus the speedup obtained for the second would be higher as compared to the other two.

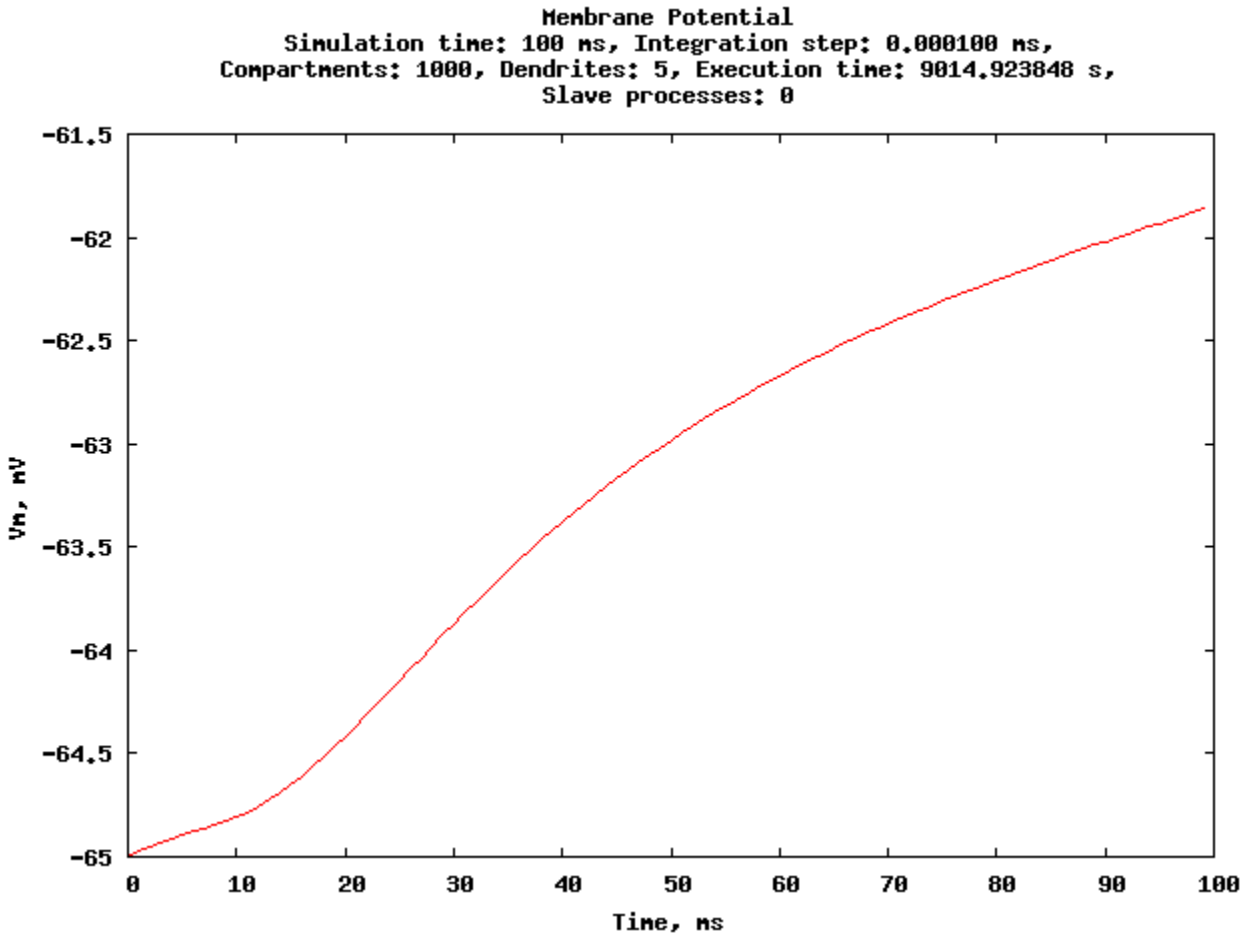
Answers related to question 4:

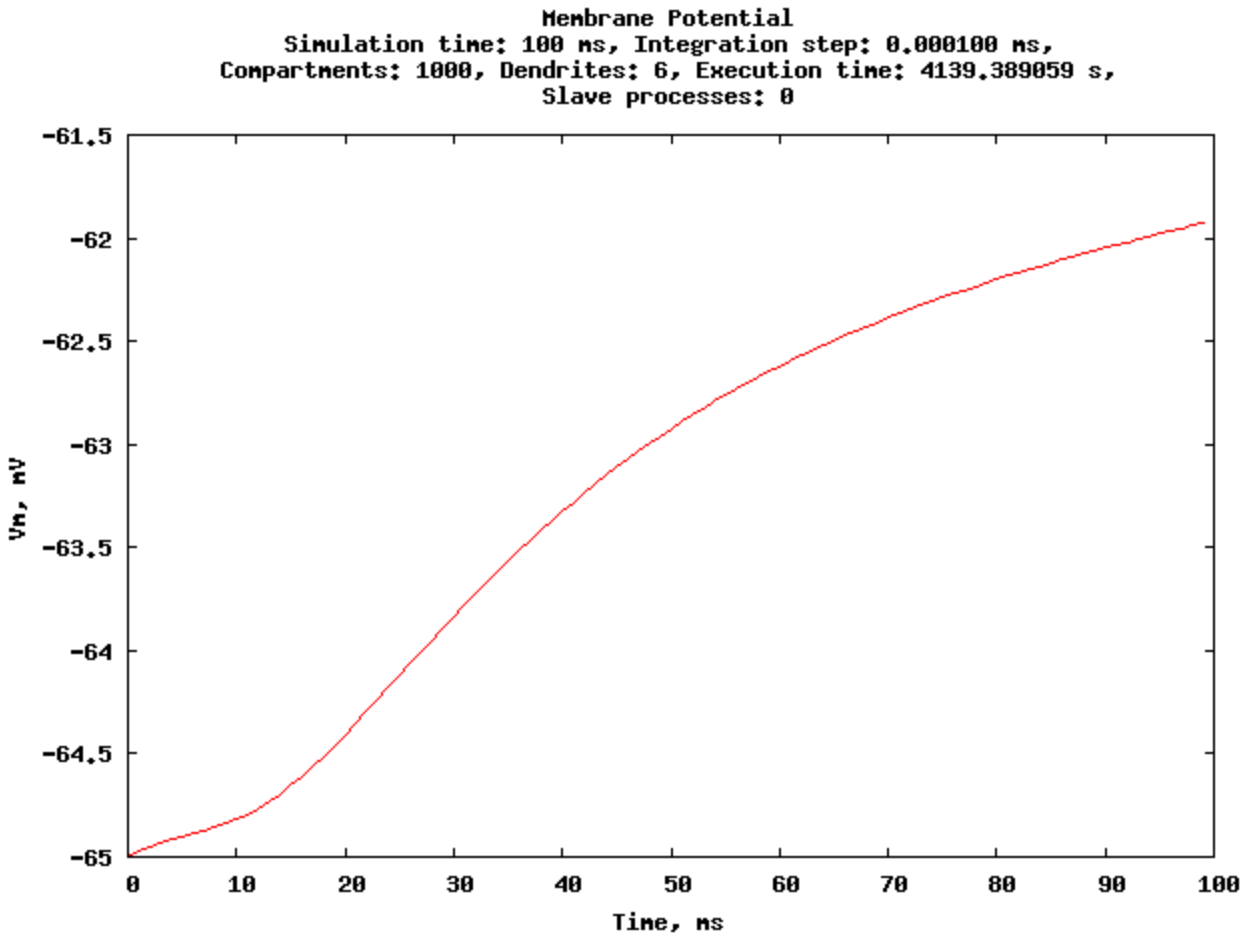
The fourth table shows the effect on execution time of the program with load imbalance:

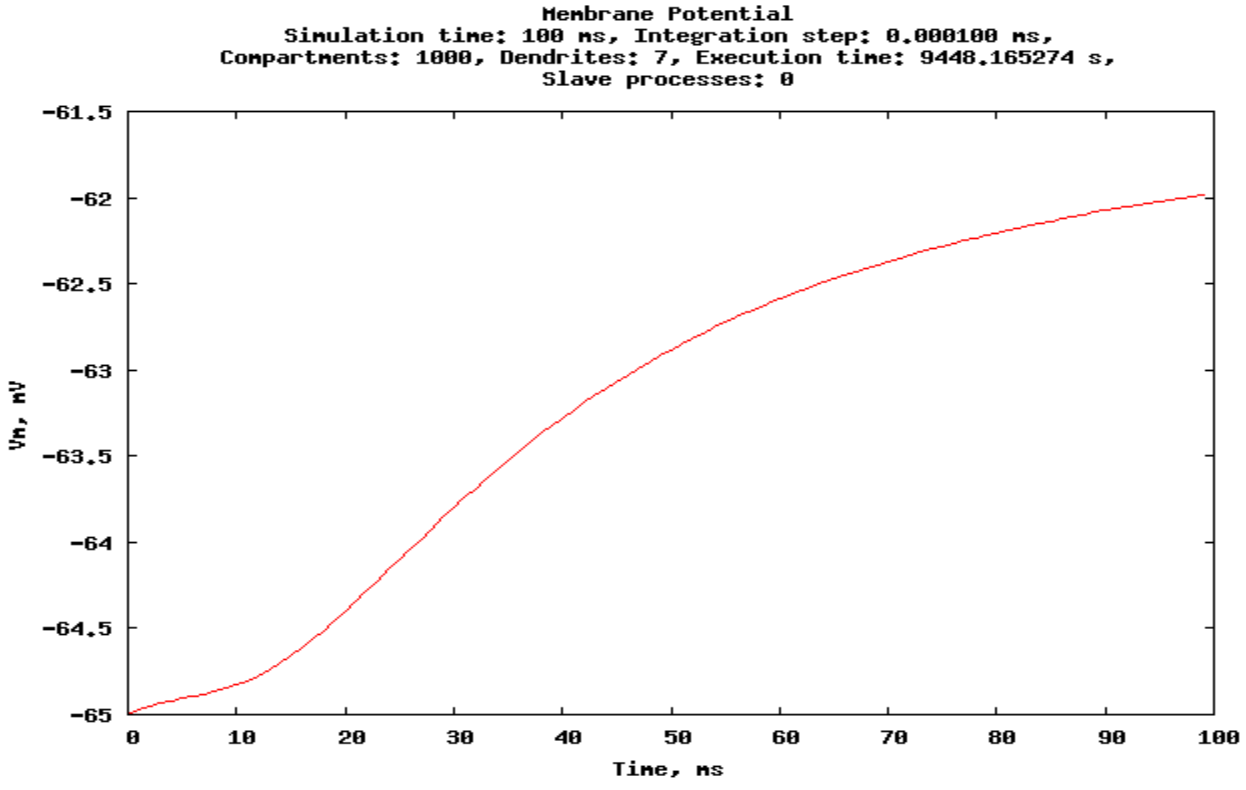
#	Number of Slaves	Dendrites	Compartments	Exec Time (sec)
1	5 (mpirun –np 6)	5	1000	9014.923848
2	5 (mpirun –np 6)	6	1000	4139.389059
3	5 (mpirun –np 6)	7	1000	9448.153335
4	5 (mpirun –np 6)	1997	10	N/A
5	5 (mpirun –np 6)	1998	10	N/A
6	5 (mpirun –np 6)	1999	10	N/A

Table 4: Effect of Load Imbalance

Graphs:




Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 1000, Dendrites: 6, Execution time: 4139.389059 s,
Slave processes: 0
-61.5
-62
-62.5
-63
-63.5
-64
-64.5
-65
Vm, mV
0
10
20
30
40
50
60
70
80
90
100
Time, ms


Membrane Potential
Simulation time: 100 ms, Integration step: 0.000100 ms,
Compartments: 1000, Dendrites: 7, Execution time: 9448.165274 s,
Slave processes: 0
-61.5
-62
-62.5
-63
-63.5
-64
-64.5
-65
Vm, mV
0
10
20
30
40
50
60
70
80
90
100
Time, ms

Compare and contrast the effect of load imbalance in experiments #1-2 and experiments #3-4 in Table 4. You may find it useful to compare in terms of % of execution time. How would you improve the worst case among these results?

#1-2 load balancing is good, so should result in better speedup , whereas the 3 and 4 the load is not perfectly balanced so speedup would be less. The worst case can be specially handled by optimizing the load in a balanced way among the diff processes.

Conclusion:

From the results obtained from the above execution, it is clearly understood that merely parallelizing the program wouldn't result in the increase in the speedup. The program should be modified in a way that there is good load balance among the processors as well as the delay due to synchronization and communication should be as minimum as possible.