CMPE 655 Assignment 1
Compartmental Hodgkin-Huxley Neuron Model, Parallel Implementation with MPI

Robert Relyea
Submitted: October 21, 2016

Professor: Dr. Muhammad Shaaban
TAs: Akshay Yembarwar
Mohit Sathawane

Abstract

The purpose of this assignment was to observe and compare the sequential and parallel performances of a large computational problem. Using a given sequential implementation of the Compartmental Hodgkin-Huxley Neuron Model, an MPI parallel implementation was created by exploiting data parallelism inherent in the model. To compare the performance of both implementations, many simulations were performed with varying input parameters for the amount of dendrites and compartments per dendrite. The results from each simulation were logged in data files and plotted in graphs. Each simulation was run on the RIT Computer Engineering research computing cluster.

Design Methodology

In the Compartmental Hodgkin-Huxley Neuron Model, the current injected into the soma from an individual dendrite is calculated independently from all other dendrites. This allowed the dendrite current calculations to be performed simultaneously while still evaluating to the same total current injected into the soma. Once this property was established, a method for assigning a number of dendrites to an individual processing node was devised.

The sequential implementation for the neuron model was designed to calculate multiple dendrite currents in a single process. This was easily adapted for a parallel implementation where an individual process may need to calculate multiple dendrite currents. The number of dendrites that each process needs to calculate was found by dividing the total number of dendrites in the simulation by the number of processes. Any remainder from this division was then distributed amongst the process to ensure every dendrite was assigned. These assignments were distributed amongst all processes by the “master process” denoted with an MPI rank of zero.

For the current calculation to be performed, some information about the state of the soma was required. The electric potential of the soma directly relates to the current injected by each dendrite which means each process required the soma potential for each dendrite step. The master process sends this information to all the other processes to cover this dependence.

Once all the currents were calculated by each process, the master process received all the dendrite currents from the worker processes. The sum of these currents is used to find the new soma electric potential. This electric potential is distributed back to the worker processes by the master process and the cycle begins again.

The master process was responsible for any strictly sequential calculations such as calculating the new soma electric potential and outputting the simulation data to the data file.

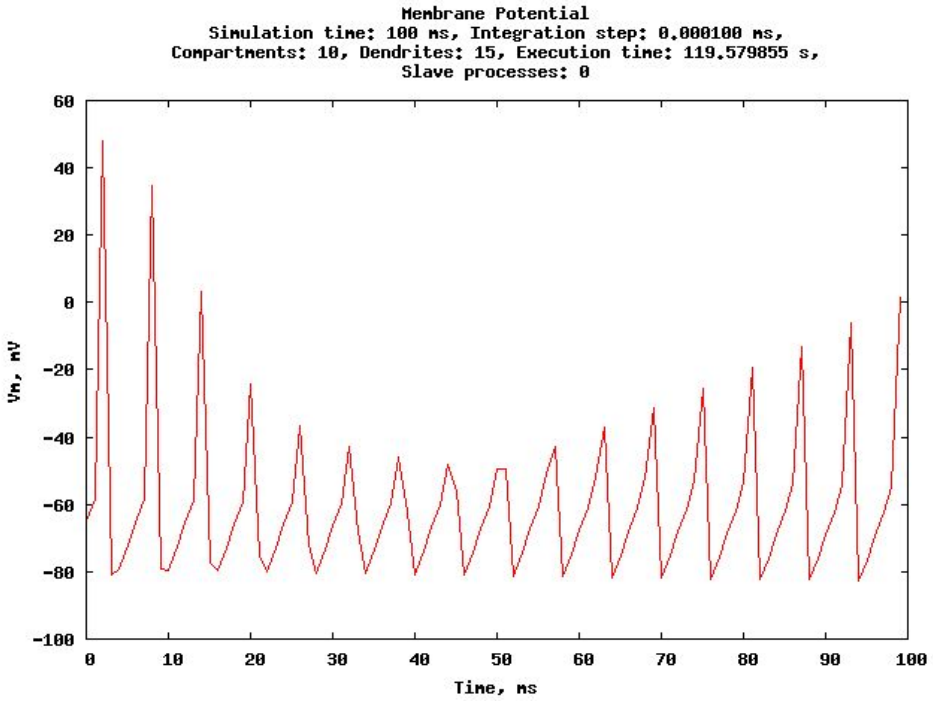
Results/Data Analysis

Several different simulations were run to show the benefits and drawbacks of parallel execution over sequential. The first set of simulations shown in Table 1 demonstrate the effect of dendrite length (number of compartments) on the execution time.

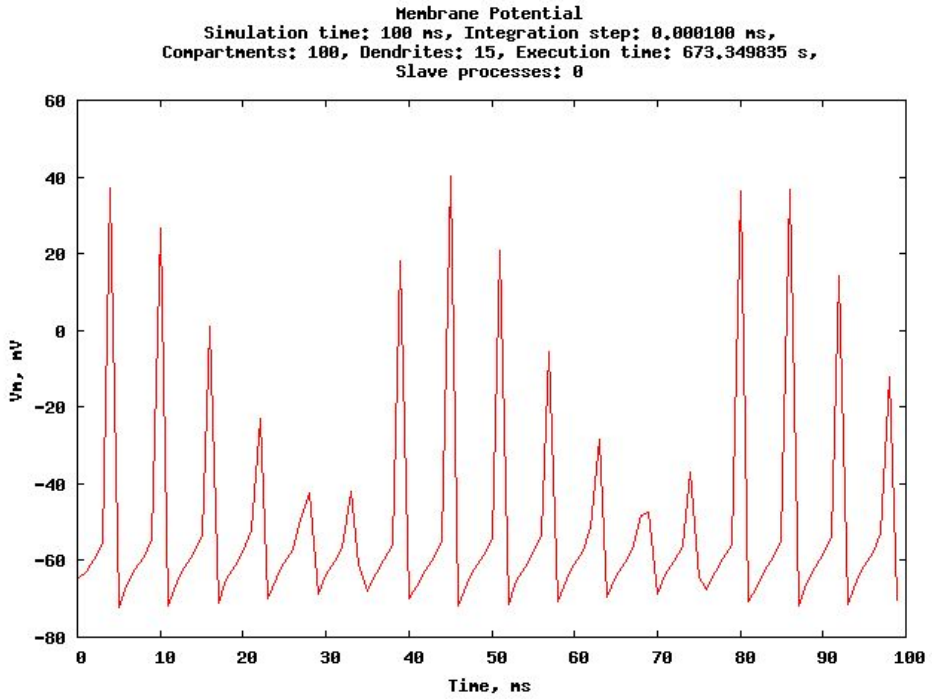
Question Set 1				
Number of Slaves	Dendrites	Compartments	Execution Time (sec)	Speedup
0, sequential	15	10	119.579855	1
5 (mpirun –np 6)	15	10	41.177727	2.903993584
12 (mpirun –np 13)	15	10	204.377184	0.5850939555
0, sequential	15	100	673.349835	1
5 (mpirun –np 6)	15	100	267.06365	2.521308441
12 (mpirun –np 13)	15	100	301.498639	2.233342868
0, sequential	15	1000	6795.628095	1
5 (mpirun –np 6)	15	1000	2623.391385	2.590398114
12 (mpirun –np 13)	15	1000	1746.057456	3.891984237

Table 1: Effect of Dendrite Length on Execution Time

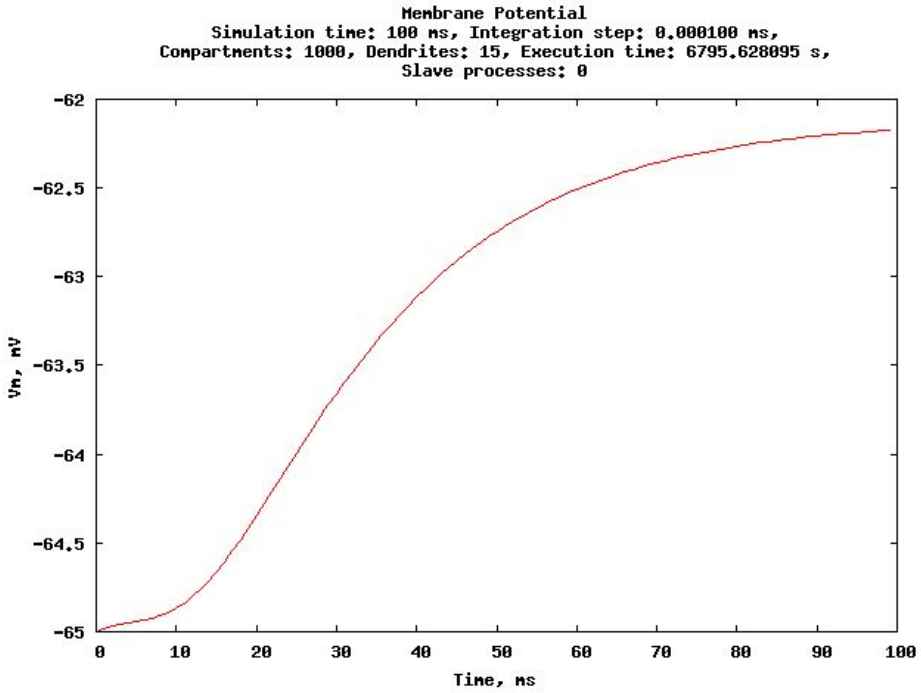
Most of the parallel executions shown in Table 1 demonstrate a considerable speedup over the sequential executions. This speedup results from simultaneously calculating the dendrite currents. One exception to this trend is the simulation of 15 dendrites with 10 compartments over 13 processes which had an execution time significantly longer than the single process simulation. This is likely due to the greater communication overheads incurred with a large amount of processes not being offset by the simultaneous execution performance gains. These communication overheads are the same reason for the parallel speedups being less than the number of processes (ideal speedup.) The resulting graphs for soma potential generated from the simulation data are shown in Graphs 1, 2 and 3.



Graph 1: Soma Membrane Potential Simulation for 15 Dendrites with 10 Compartments.



Graph 2: Soma Membrane Potential Simulation for 15 Dendrites with 100 Compartments.



Graph 3: Soma Membrane Potential Simulation for 15 Dendrites with 1000 Compartments.

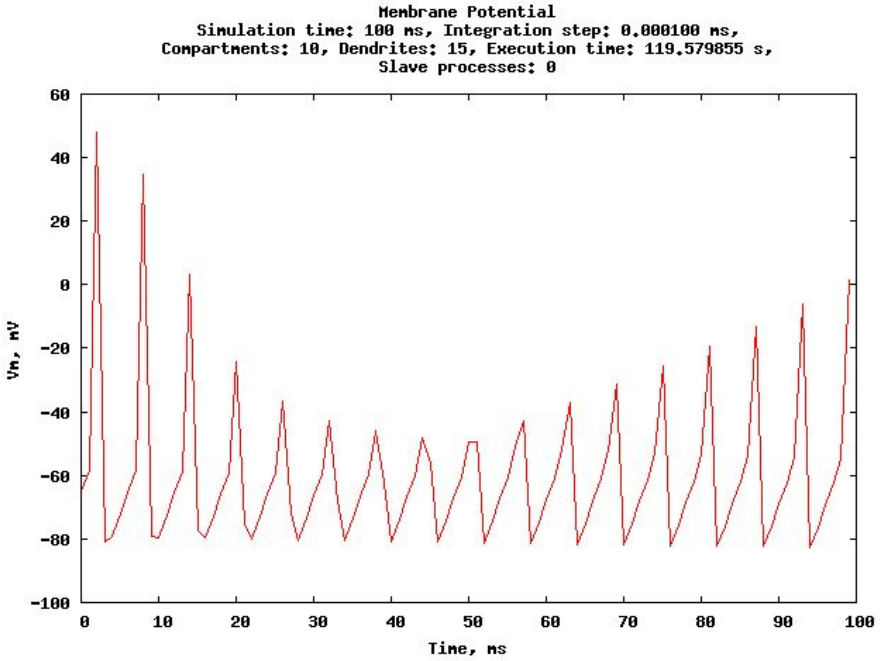
The large spiking patterns from the simulations shown in Graphs 1 and 2 become more frequent with more compartments. Graph 3 shows a smoother increase in soma potential with greater compartment counts.

Table 2 shows simulations similar to Table 1 but with different dendrite counts rather than different compartment counts. The goal of these simulations was to show the effect varying numbers of dendrites had on execution time.

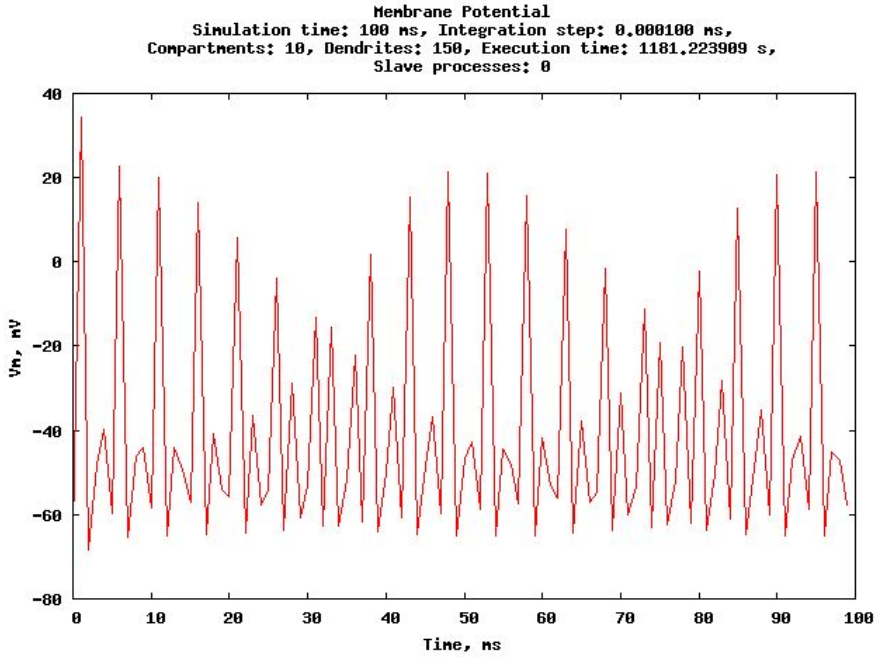
Question Set 2				
Number of Slaves	Dendrites	Compartments	Execution Time (sec)	Speedup
0, sequential	15	10	119.579855	1
5 (mpirun –np 6)	15	10	41.177727	2.903993584
12 (mpirun –np 13)	15	10	204.377184	0.5850939555
0, sequential	150	10	1181.223909	1
5 (mpirun –np 6)	150	10	387.515003	3.048201747
12 (mpirun –np 13)	150	10	440.017847	2.684490907
0, sequential	1500	10	10878.79307	1
5 (mpirun –np 6)	1500	10	2874.383075	3.784740162
12 (mpirun –np 13)	1500	10	1618.947542	6.719669899

Table 2: Effect of Dendrite Count on Execution Time.

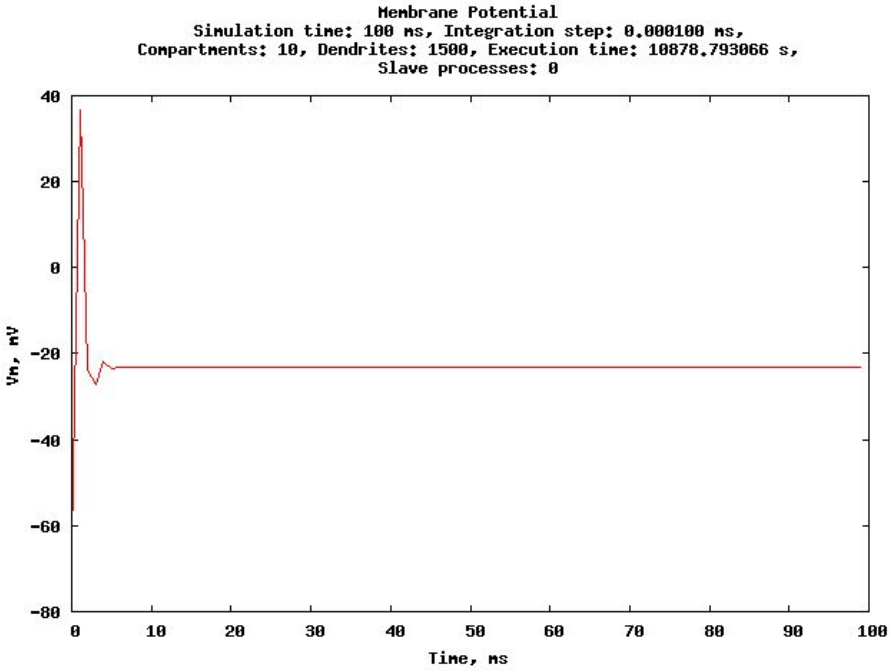
The results shown in Table 2 follow a similar trend to the results in Table 1. The parallel executions demonstrate performance gains over the sequential executions with the exception of the first 13 process simulation. Increasing the dendrite count results in greater speedups than increasing the compartment count in these simulations. This is likely due to the decreased complexity of the calculations performed by the neuron model when there are fewer compartments per dendrite. This allows the parallel executions to iterate through each dendrite quickly. The resulting graphs for soma potential generated from the simulation data are shown in Graphs 4, 5 and 6.



Graph 4: Soma Membrane Potential Simulation for 15 Dendrites with 10 Compartments.



Graph 5: Soma Membrane Potential Simulation for 150 Dendrites with 10 Compartments.



Graph 6: Soma Membrane Potential Simulation for 1500 Dendrites with 10 Compartments.

The frequency of potential spikes increases as the number of dendrites increases as shown by Graphs 4 and 5. When we get to greater dendrite values, the potential has an initial spike and then levels out at a constant value as shown by Graph 6.

Table 3 shows simulations with varying amounts of dendrites on 11 processes. The amount of dendrites were chosen to show the effects of load balancing on execution time.

Question Set 3			
Number of Slaves	Dendrites	Compartments	Execution Time (sec)
10 (mpirun –np 11)	30	100	277.015905
10 (mpirun –np 11)	33	100	277.950147
10 (mpirun –np 11)	36	100	378.074303

Table 3: Effect of Load Balancing on Execution Time with 11 Processes.

Each process is assigned a certain number of dendrites to simulate based on the number of processes and the total number of dendrites. In the case shown in Table 3, there are 11 processes that split the load of dendrites. The first simulation covered 30 dendrites on 11 processes. 30 does not divide evenly over 11 so some processes are assigned additional dendrites which causes an increase in execution time. In this scenario, 8 processes are assigned 3 dendrites and the other 3 processes handle only 2 dendrites. The second entry has a similar execution time to the first entry even though there are more dendrites. This is due to the number of dendrites dividing evenly over the number of processes. In the second scenario, 33 dendrites are split into 11 groups of 3 for a perfect dendrite load balance. Since the largest group assigned to an individual processor for the second scenario was equal to that of the first scenario the execution time was similar. The last scenario in Table 3 increases the dendrite count again from 33 to 36. This results in a load imbalance where some processes are simulating 3 dendrites and others are simulating 4. The processes that are simulating 4 bring the execution time up.

The last group of simulations shown in Table 4 also demonstrate the effect of load balancing on execution time by varying dendrite count as well as compartment count.

Question Set 4			
Number of Slaves	Dendrites	Compartments	Execution Time (sec)
5 (mpirun –np 6)	5	1000	877.209112
5 (mpirun –np 6)	6	1000	877.475622
5 (mpirun –np 6)	7	1000	1638.390429
5 (mpirun –np 6)	1499	10	2900.952165
5 (mpirun –np 6)	1500	10	2874.383075
5 (mpirun –np 6)	1501	10	2889.255777

Table 4: Effect of Load Balancing on Execution Time with 6 Processes.

The first three entries in Table 4 are similar to the entries in Table 3 where the dendrite count is relatively low. The last three entries in Table 4 were much greater than the first three. Even though the load does not perfectly balance in the fourth and fifth scenarios the execution time for the large dendrite count simulations were very similar. This was due to the sheer scale of the

problem being evaluated. As the problem size increased, the noticeable effect of load imbalancing decreased.

Conclusion

By exploiting data parallelism in the Compartmental Hodgkin-Huxley Neuron Model overall positive performance gains were observed. The size of the problem was determined by two parameters: dendrite count and compartments per dendrite. When the size of the problem was large, parallel speedup greater than 1 over sequential execution was observed. The speedups were successfully achieved due to simultaneous execution of independent data operations inherent in the Compartmental Hodgkin-Huxley Neuron Model. The resulting speedups from these simulations demonstrate the ability of parallel execution to yield performance gains for problems that contain data parallelism.