CMPE-655 Exercise 1

Parallel Implementation of Compartmental Hodgkin-Huxley Neuron Model with MPI

Atticus Russell & Liam Sullivan & Sydale John Aye
Performed: 03/20/2023
Submitted: 3/23/2023

TA: Daniel Breslawski

Lecture Section: 01
Professor: Dr. Shaaban

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: _____

Abstract

This exercise explored the effects of parallelization and varying program parameters on the performance of a provided computer program which sequentially modeled neurons using the Hodgkin-Huxley model. The program was modified to implement parallelization using the Message Passing Interface (MPI) in areas of the model with inherent data parallelism. The performance of the model was quantified by recording the program's execution time when run using varying numbers of processors, compartments, and dendrites. These measurements were analyzed, and the speedup obtained through parallel processing, and the effect of various parameters on the model were calculated. All measurements of the program were conducted on RIT's Scheduled Processing on Research Computing (SPORC) Research Cluster. Execution of the parallelized simulation using six processors yielded a minimum speedup of 2.3 and a maximum speedup of approximately 3.9, when compared to sequential execution of the program. Execution using 13 processors yielded a minimum speedup of 2.7, and a maximum of 8.9. The variation in speedup is attributable to varying the number of dendrites and compartments in the neuron model; increasing the number of dendrites in the simulation by 100 times more than doubled the speedup yielded by the use of 13 processors. In general, increasing either the number of dendrites or compartments in the model, and thereby increasing the size of the problem, increased the speedup gained through parallelization. This effect was particularly emphasized with dendrites. This is because increasing these parameters increased the size of the section of the program with inherent data parallelism, while parallelization overhead remained constant for a certain number of processors.

Design Methodology

A program written in C was provided that sequentially simulated the behavior of a neuron using the Hodgkin-Huxley model. This program was parallelized during this exercise by modifying the sequential code to employ the Message Passing Interface (MPI) library, specifically the OpenMPI 1.2 implementation, for efficient communication and data distribution between multiple processes. Initially, MPI was set up, and the required communicators were established to determine the number of processes and the rank of each process. The process with rank zero was determined to be the "master process", and was responsible for executing all parts of the program that remained sequential, such as distributing work between processes and file I/O.

In the provided model of a neuron, the current injected into the soma is calculated individually for each dendrite before it is summed to find the total soma current. This step was identified as parallelizable. Domain decomposition played a crucial role in dividing the dendrites evenly among the available processes, ensuring that each process was assigned a specific range of dendrites to handle based on its rank and the total number of processes. The master process calculated the number of dendrites for each process, by dividing the number of dendrites by the number of processes. Any dendrites remaining after this division were then delegated by the master process between other processes.

To maintain the accuracy of the simulation, the MPI library facilitated inter-process com-

munications, addressing data dependencies between dendrites by using functions such as MPI_Send and MPI_Recv to transfer data between the master and slave processes. The master process used these functions to provide each slave process with updated information regarding the electrical potential of the soma at each step of the simulation, which was needed to find the dendrite currents. The master process aggregated partial results from each process to update the soma electrical potential, which was used to update the worker processes, and this process was repeated for the duration of the simulation.

Simple bash scripts were written to configure SLURM, the job scheduler for the SPORC cluster, to run the neuron simulation with varying numbers of dendrites, number of compartments (dendrite length), and number of processors assigned to the simulation. Data files and graphs were generated for each run of the simulation.

Results and Analysis

Four sets of simulation runs with varying parameters were performed, each to observe the effect of a different variable on the program's performance. The parameters for the first group of simulations were chosen to reveal the effect of dendrite length (the number of compartments per dendrite) on computational load. The results of these simulations are shown in Table 1.

Table 1: Effect of Dendrite Length on Computational Load

Number of slaves	**Dendrites**	**Compartments**	**Exec Time (sec)**	**Speedup**
"0, sequential"	15	10	40.425675	1
5 (srun -n 6)	15	10	17.484443	2.302
12 (srun -n 13)	15	10	14.357417	2.816
"0, sequential"	15	100	248.626913	1
5 (srun -n 6)	15	100	84.338491	2.948
12 (srun -n 13)	15	100	58.933703	4.218
"0, sequential"	15	1000	2345.657705	1
5 (srun -n 6)	15	1000	756.185078	3.102
12 (srun -n 13)	15	1000	504.265572	4.652

The largest effect of parallelizing the program is a reduction in execution time, as shown in Table 1. Additionally, the speedup realized from each additional processor used increased with the length of the dendrites. This is because the proportion of parallelizable work in the program compared to the fixed amount of sequential work and limited amount of overhead increased with the number of compartments per dendrite. The speedup is not the same as the number of parallel processes because it is limited by the sequential portion of the program, and the overhead inherent in distributing the work between multiple processes. During this set of simulations, the spiking pattern in the graphs of soma potential was observed to be affected by dendrite length, as shown in Figures 1, 2 and 3.

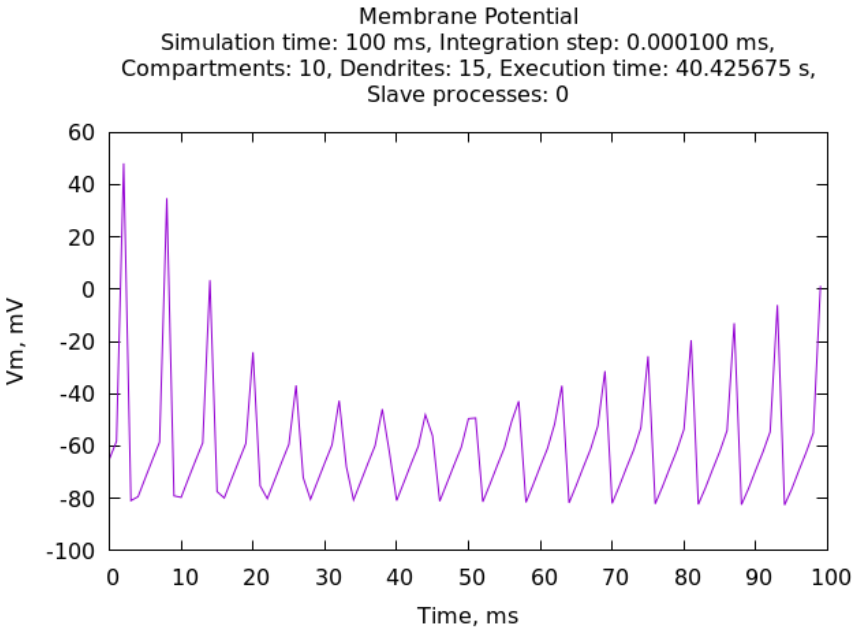

Figure 1: *Soma Potential vs Time for Dendrites with 10 Compartments*

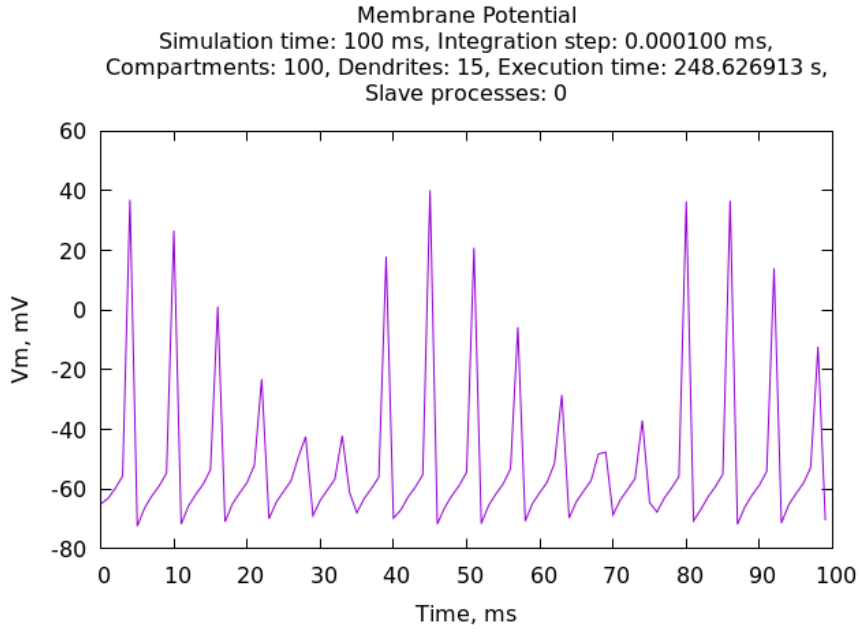

Figure 2: *Soma Potential vs Time for Dendrites with 100 Compartments*

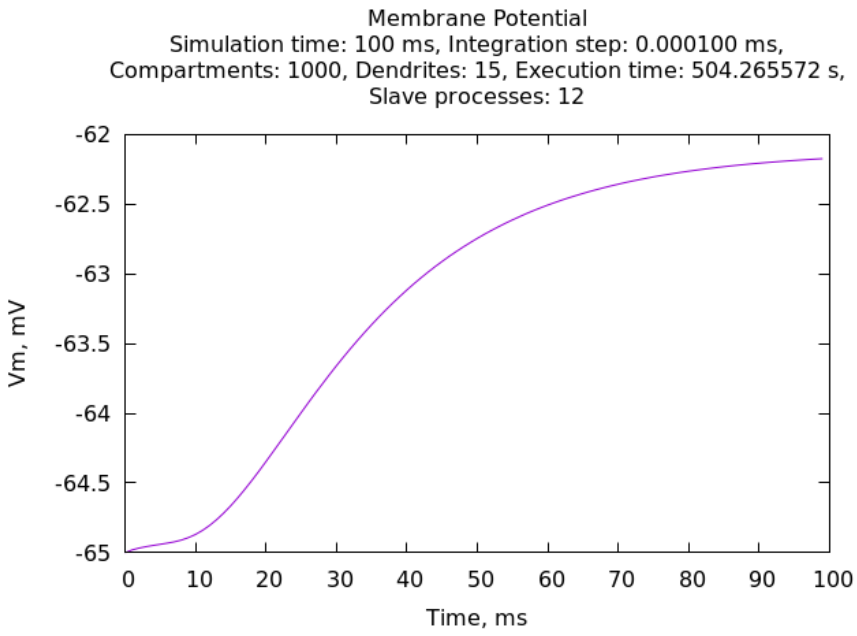

Figure 3: *Soma Potential vs Time for Dendrites with 1000 Compartments*

The differences between the pattern of soma potential in Figures 1, 2 and 3, shows that the spiking pattern of the soma potential relative to time ceases to exist at a certain dendrite length between 100 and 1000 compartments.

The next set of simulations was run with the intent to observe the effect of the number of dendrites on computational load. To achieve this, the number of compartments in the model was constant, and the number of dendrites was varied. The results of these simulations are shown in Table 2.

Table 2: Effect of Number of Dendrites on Computational Load

Number of slaves	Dendrites	Compartments	Exec Time (sec)	Speedup
"0, sequential"	15	10	41.083955	1
5 (srun -n 6)	15	10	14.832247	2.77
12 (srun -n 13)	15	10	12.54813	3.274
"0, sequential"	150	10	389.023796	1
5 (srun -n 6)	150	10	102.630555	3.791
12 (srun -n 13)	150	10	51.317353	7.58
"0, sequential"	1500	10	3732.72096	1
5 (srun -n 6)	1500	10	984.075078	3.793
12 (srun -n 13)	1500	10	427.777333	8.96

The largest effect of parallelizing the program is a reduction in execution time as the number of processors increases, as shown in Table 2. Additionally, the speedup realized from each additional processor used increased with the number of dendrites. This is similar to the

effect in Table 1 of increasing the length of the dendrites. The underlying reason for the similarity is that both increasing the length of the dendrites and increasing the number of dendrites increase the portion of the program that is spent working on a parallelizable task, and therefore can benefit from the increased number of processors. Increasing the number of dendrites increases the speedup realized from each additional processor to an even greater degree than increasing the length of the dendrites. The maximum speedup recorded was therefore greater in Table 2 than in Table 1.During this set of simulations, the spiking pattern in the graphs of soma potential was observed to be affected by the number of dendrites, as shown in Figures 4, 5 and 6.

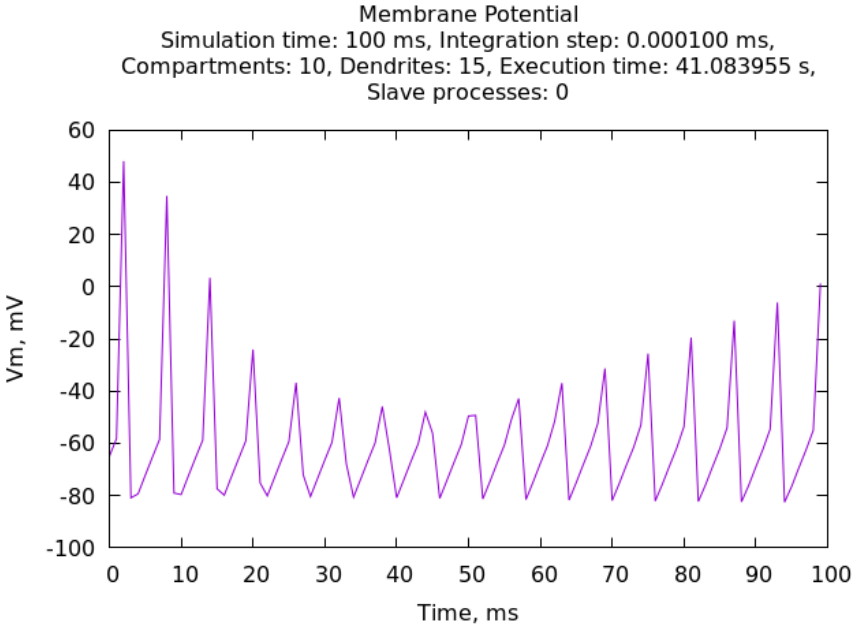

Figure 4: *Soma Potential vs Time for 15 Dendrites*

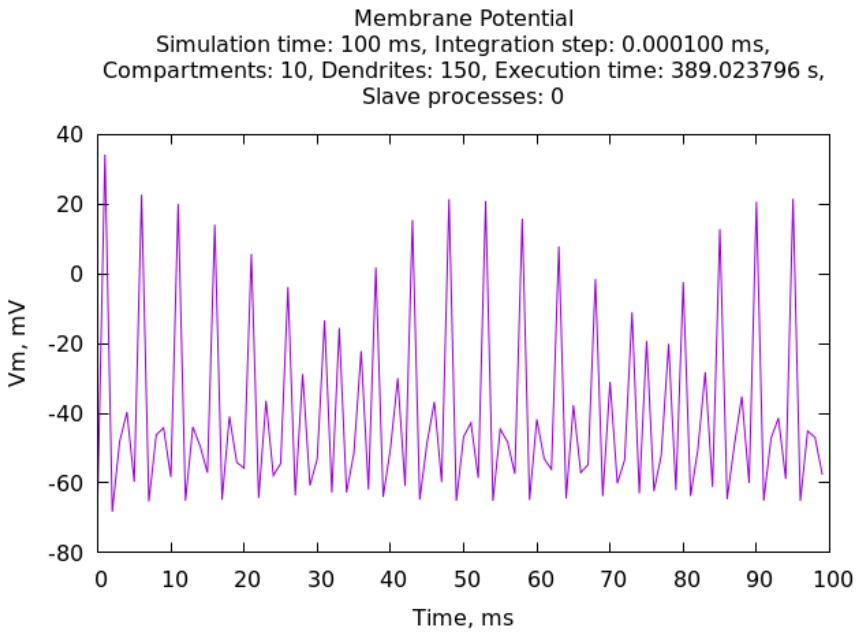

Figure 5: *Soma Potential vs Time for 150 Dendrites*

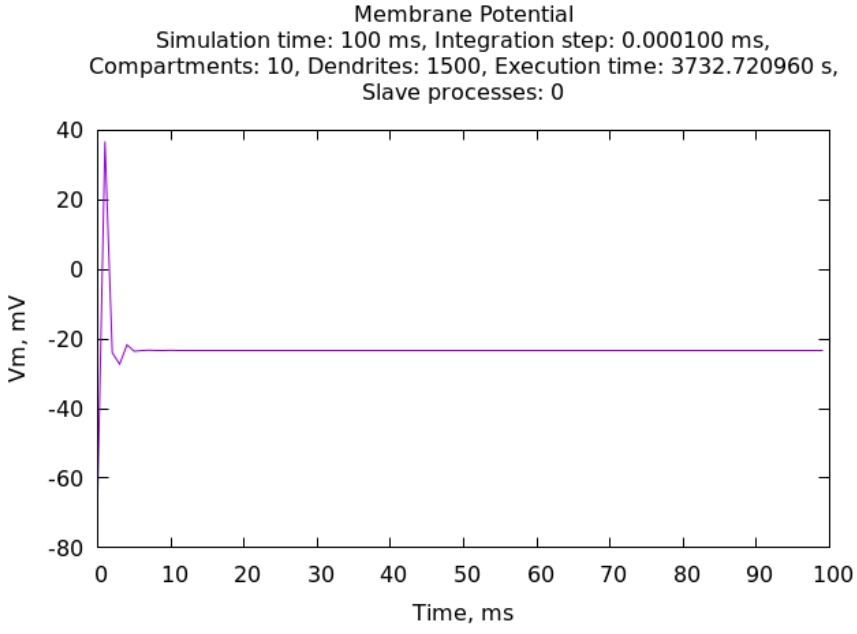

Figure 6: *Soma Potential vs Time for 150 Dendrites*

The differences between the pattern of soma potential in Figures 4, 5 and 6, shows that the spiking pattern of the soma potential relative to time ceases to exist at a certain number of dendrites between 150 and 1500. This is a similar end result to that observed in Figures 4, 5 and 6 from the first set of simulations, however the pattern of spikes are quite different.

From Figure 4 to Figure 5, the frequency that the spikes in soma potential occur with increases, which not true between Figure 1 and Figure 2. Additionally, while both Figure 3 and Figure 6 show the spiking pattern dissipating at a certain dendrite length and dendrite number, respectively, Figure 6 shows a single spike that follows the pattern before the soma potential stays constant, whereas Figure 3 shows no spikes and only a small gradual increase in soma potential.

The third set of simulations was conducted with the goal of observing the effect of load imbalance on execution time. The results of these simulations are shown in Table 3.

Table 3: Test for Load Imbalance

#	Number of slaves	Dendrites	Compartments	Exec Time (sec)
1	10 (srun -n 11)	30	100	86.171065
2	10 (srun -n 11)	33	100	87.918258
3	10 (srun -n 11)	36	100	127.711476

The approximately 45% increase in execution time shown in Table 3 between Simulation #2 and Simulation #3, which differ only in three dendrites, is extremely large, especially when compared to the approximately 2% increase in execution time between Simulation #1 and Simulation #2, which are also separated by three dendrites. This significant difference is caused by the load balancing in the program: the number of dendrites assigned per processes is based on the total number of dendrites divided evenly by the number of processes, and remaining dendrites are then distributed between processes. In all of these simulations, the number of processes is 11. In Simulation #1, 30 divided by 11 yields two dendrites split evenly between each process, and eight dendrites remaining which are also distributed to processes. This means the most dendrites assigned to any one process in Simulation #1 is three. For Simulation #2, 33 divides by 11 evenly to three, which means that each process handles at most three dendrites. In Simulation #3, 36 divides by 11 evenly three times and three dendrites remain to be distributed. These dendrites are allocated to processes, and mean that some processes are assigned four dendrites. The same maximum of three dendrites in any process between Simulations #1 and #2 explains the small difference in execution time. The larger maximum of four dendrites in some processes in Simulation #3 explains the large discrepancy in execution time between simulation runs. These results mean that the program is only moderately effective at balancing computational load, because it cannot distribute computations relating to a single dendrite across multiple processes.

The final group of simulations were conducted to explore the effect of dendrite length on load balancing. The results from these simulations are shown in Table 4.

Table 4: Effect of Dendrite Length on Load Imbalance

#	Number of slaves	Dendrites	Compartments	Exec Time (sec)
1	5 (srun -n 6)	5	1000	251.832346
2	5 (srun -n 6)	6	1000	249.414926
3	5 (srun -n 6)	7	1000	495.756016
4	5 (srun -n 6)	1499	10	855.861892
5	5 (srun -n 6)	1500	10	855.64765
6	5 (srun -n 6)	1501	10	858.559609

Simulations #1 through #3 shown in Table 4 were conducted with very long dendrites. Between Simulations #2 to #3, there is an approximately 97% increase in execution time. This is another case, like in Table 3, where the number of dendrites does not divide evenly by the number of processes, and some processes are required to compute values for an additional dendrite before execution of the program can conclude. This same uneven distribution of dendrites occurs between Simulation #5 and Simulation #6, however these simulations were conducted with dendrites 100 times shorter than #1-#3, and a much greater number of dendrites. The shorter dendrite length and larger amount of dendrites means that there is only an 0.3% increase in execution time from #5 to #6. This is because when each dendrite is shorter, it takes up less of the total execution time, so the effects on execution time of a number of dendrites that are not able to be evenly distributed between processes is much smaller.

Conclusion

In this study, the Hodgkin-Huxley neuron model was parallelized using the Message Passing Interface (MPI) to explore the effects of parallelization and varying program parameters on the performance of the provided computer program. The performance of the model was quantified by recording the program's execution time when run using varying numbers of processors, compartments, and dendrites. The results demonstrated that the parallelized program's execution time was significantly reduced as the number of processors increased. The speedup obtained through parallel processing increased with the number of dendrites and compartments in the neuron model, emphasizing the benefit of parallelization for larger problems. The largest speedup observed was 8.96 times faster when using 13 processors, compared to sequential execution. Furthermore, the spiking pattern of soma potential was affected by the number of dendrites and compartments, and ceased to exist at certain levels of these parameters. The exercise confirmed the potential of parallel processing in speeding up the execution time of the Hodgkin-Huxley neuron model, particularly when dealing with large numbers of dendrites or compartments. This finding is crucial for researchers and developers working on large-scale neuron simulations, as it highlights the importance of optimizing parallelization strategies to maximize computational efficiency. Future work could focus on optimizing load balancing and exploring other parallelization techniques, such as GPU computing, to further enhance the performance of neuron simulations.