N-Neuron Simulation Using Multiprocessor Cluster

Derara Senay Shanka(✉)

College of Engineering and Technology, Electrical and Computer Engineering Department, Bule Hora University, Bule Hora, Ethiopia

Abstract. A growing number of research effort has been made to make the Simulation of the astonishing biophysical activities of the massively interconnected organic neurons on various computational platforms such as Field programmable Gate Arrays (FPGAs) and General Purpose Graphics Processing Unit (GP-GPU). Nonetheless, to make efficient and realistic simulation of this biophysical activities, a considerable amount of processing power and memory space is incurred. Although computational platforms come with an advanced solutions such as high performance cluster, and super computers, they come with high establishment cost. Hence, biologically realistic and computationally efficient simulation on consumer level cluster of microprocessor nodes would be helpful, as they give a room for a better performance of compute intensive applications in a relatively cheaper cost as compared to dedicated High performance computing facilities. Accordingly, in this work, large scale simulation of spiking neural network (SNN) on a cluster of 8 physical cores enabled with hyper threading (16 logical cores) is presented. The neural network is composed of the biologically plausible and computationally efficient Izhikevich single neuron model. To improve the performance of the simulation and effectively exploit the computational capacity of the cluster, we have used two parallel programming techniques: distributed parallel programming using Message Passing Interface (MPI) library and distributed shard (hybrid) parallel programming using MPI in tandem with Open Multi-Processing (OpenMP) library. Moreover, to harness the combined memory and computation power of the cluster the neurons were distributed across the nodes using static load balancing mechanism. Hence, we were able to simulate up to 160,000 neurons and 3.2M synapses connection per neuron. Performance evaluation for different configuration of the SNN with a purely MPI and Hybrid Parallelization method was presented. Our performance result show that for 160K neurons with 200 synapses connections, using purely MPI parallelization with 16 MPI processes the sequential simulation has improved by 43.12% and using the hybrid parallel programming the sequential simulation has improved by 69.58%. Hence, comparing the performance results the hybrid parallelization approach demonstrated to be a good programming solution for simulation of SNNs on a cluster of consumer level multiprocessors.

Keywords: Spiking neural network · MPI · OpenMP · Hybrid parallelization

M. L. Berihun (Ed.): ICAST 2021, LNICST 411, pp. 343–367, 2022.
https://doi.org/10.1007/978-3-030-93709-6_23

1 Introduction

The mammalian cerebral cortex, the largest portion of the organic brain, contains a massive interconnection of information processing element called neuron. It is now understood that these neurons incorporate computational and memory units as well. Moreover, a single neuron is connected to as many as 1000–10,000 to other neurons, creating massively parallel, and extensively interconnected network of biological neurons.

It has long been since an effort was being reported to model the computational power of this biological phenomena. From the earliest McCulloch Pitts neuron based network of artificial neurons to the latest biologically realistic spiking neural networks (SNN). Be that as it may, the need to have biologically accurate simulation is computationally demanding. This fact have further led researchers to crave in finding a way to realize an efficient large-scale simulation of biologically plausible neural networks both on software and hardware platforms.

Advancements in the computational efficiency of the present day multiprocessor computers and the increasing understanding of computational principles of the organic brain in the past decades have led to an ever growing efforts in simulating the physiological activities of biological neuronal networks using various computational platforms. For instance, R. Ananthanarayanan and et al. [1] simulated 55 million neurons with a 442 billion synapses on a BlueGene/L computer which has more than 32,000 cores with a reasonable time.

Inspired by the astounding physiological activity of network of biological neural networks there is a proliferated effort to mimic this activity using a variety of mathematical models on a range of computational platforms both on software development approaches and specialized hardware architectures like Field Programmable Gate Array (FPGA) and General Purpose Graphics processing Unit (GP-GPU).

In line of this effort, there exists a tradeoff between realizing biologically realistic simulations and computational cost, where such simulations require a considerable amount of processing power and memory space. Conventional processors do not have enough parallelism and memory bandwidth for real-time simulation of SNNs with large number of neuron population. Modern parallel architectures (such as clusters, supercomputers, or high-performance processors) promise powerful alternatives for speeding spiking neural network simulation, but incur high price cost to establish them. Therefore, simulation on consumer level cluster of microprocessors would be helpful and relatively less expensive than high performance facilities.

This research work seeks to speed up the simulation using message passing interface (MPI) library and hybrid MPI + Open Multi-processing (OpenMP) library and measure and compare the performance of the simulation on different neural network configuration. Moreover, Performance Measurement and comparison of the simulation on varying number of MPI processes and OpenMP threads.

In the ensuing section, we present a brief discussion of literatures related with other works done by manifold researchers on large-scale simulation of SNN on various computational platforms in general and on multiprocessor cluster in particular followed by the theoretical background of how biological neural networks function from

computational neuroscience perspective. Three prominent mathematical model of spiking neurons are discussed in brief. The methodology section discusses regarding parallel programming fundamentals on shared memory and distributed shared memory computers. Moreover the experimental setup of the simulation is also presented.

2 Literature Review

Simulation of how biological neural networks function in organic brain has been an interest of study for decades. The effort to efficiently make large scale simulation of spiking neural networks has a long history that spans from hardware implementations to the more popular distributed compute cluster implementations. A growing body of literatures exist regarding the various approaches followed by different research groups. These approaches, particularly parallel simulations, are broadly categorized as hardware and software based implementations. In this section, we present notable works demonstrated using these platforms.

2.1 Literatures on Simulation of SNN Using Hardware Platforms

Parallel simulation of SNNs on different hardware architectures and platforms have saw three major approaches. For instance in [2], digital signal processing (DSP) accelerator based simulation was reported. On the other hand in [3], using one category of very large scale-integration(VLSI) technology, application-specific integrated circuit (ASIC), 512k neurons with up to 104 synaptic connections was demonstrated. Furthermore, field programmable arrays (FPGAs) have been a major class of hardware-based approach that are used to significantly accelerate SNN algorithms. For example two notable works [4] & [5] can be mentioned.

2.2 Literatures on Simulation of SNN Using Software Implementations

Several SNN software simulators have been developed using different programming languages which can be categorized based on the computational platform in which they are implemented. These could be on a single dedicated computer or a network of distributed computers. In addition, graphics card enabled computers and supercomputing facilities are used for large scale simulations. The paragraph that right follows goes through prominent researches made on the mentioned platforms. Starting from the earliest HH-neuron based SNN simulation environments like GENESIS [6] and NEURON [7], several sequential SNN software packages written in C++ and Python programming languages have been developed, and are currently in a widespread use.

The advent and ubiquity of multi-core computers have drawn the attention of a growing number of computational neuroscientists aiming to harness the increased processing power of these machines. Accordingly, a plethora of software based neural network simulators that are made on a parallel computers have been presented in various times. For instance, PGENESIS [6]: the parallel version of GENESIS and a parallel version of NEURON [8] have been reported. Simulation efforts that came later one have targeted multiprocessor computers. One quintessential example of such

parallel software technologies for large-scale SNN simulation works on multiprocessor computers is Neural Simulation Tool (NEST) [9].

Another category of work that fall under software-based neural network simulation package are the one that were made on a cluster of distributed multi-processor computers. Accordingly, the research work presented in [10], proposed coarse grained nature of parallelism in which a groups of neurons are mapped on to different processors and made to be solved independently and hence in parallel. This approach was tested and have resulted a 10x speed up using a cluster of 16 processors and the authors pointed out that further speed up can be achieved.

In 2007, Plesser and his colleagues [11], were able to map NEST simulation software on a cluster of multiprocessor computers. A hybrid programming which combines multi-threading and distributed parallel computing techniques using MPI and Pthread APIs respectively was introduced to the simulation. An event driven benchmark simulation of a network composed of 12,500 leaky-integrate and fire neuron model on a cluster of four Sun $\times$ 4100 computer nodes with a capability of dual core AMD Opteron processors. In this work, the neural network was partitioned and distributed over the "virtual processes", which are a combination of MPI process and Pthread threads in each processes. They have used maximum of 20 processes and 4 threads in each process. Eventually the performance of the simulation was evaluated and a super linear scaling up to 16 "virtual processes" was observed regardless of the order of combination of MPI processes and Pthread threads in the hybrid penalization and up to 8 "virtual processes" in the purely MPI implementation. Moreover, they've reported that the hybrid parallel programming has outperformed the purely MPI based distributed simulation.

Another biophysically realistic parallel simulation effort that uses two parallel programming APIs was reported in a paper [12] by Jingzhen Hu in 2012. In this work more than 100,000 Hodgkin-Huxley neurons were simulated using distributed memory parallel programming using MPI API and distributed shared memory parallel programming using MPI together with OpenMp API on a multiprocessor cluster of 52 nodes each with 32 processors. Due to the parallel strategy employed, it was reported that the execution time of the simulation has shown an improvement of 10% using a hybrid approach than the purely MPI implementation. Moreover, it was reported that the hybrid approach was 31$\times$ faster than the serial implementation when used on 32 processors.

Another distributed simulation of SNN that exhibit time locked spiking behavior was demonstrated in 2014 [13]. This simulation was made on cluster made of up to 128 processors running at 2.5 GHz. The neural network was arranged as a bi-dimensional column of Izhikevich neurons and this neurons were distributed to the processors in the cluster using a collective communication MPI primitive. In this paper, the simulation performance of the network varied from 6.6 Giga synapses down to 200k synapses was demonstrated to be scalable and fast for the maximum number of synapses used. Performance evaluation by varying the number of MPI processes for a fixed network size and fixing the number of MPI processes for a varying number of neurons and synapses was made. A speed-up of 41.5 was reported for a simulation of 204M synapses on a cluster of 128 cores. The authors have attributed the deviation from the

theoretically expected speed up of 128 to the communication cost between the processors in the cluster.

Ever since NVIDIA introduced the first graphics processing unit (GPUs) in to the main stream computing arena at the end of the 20thcentury, it has got the attention of a sizeable number of researchers from the computing community for the implementation of computationally demanding applications. The reason for this is its flexibility for parallel implementations. So much so that, several GPU-based simulation of SNN were reported at various times. In the paragraphs that follow, we present popular and notable studies under this category.

A. K. Fidjeland and M. P. Shanahan [14], demonstrated biologically realistic large-scale simulation of a network of Izhikevich spiking neurons on GPU. In this study, they were able to made real-time simulation of around 55,000 neurons with 1000 synapses per neuron. Similarly in [15], 4,096 Izhikevich neurons were simulated on an NVIDIA GTX260 GPU that resulted a real-time performance with a 9 times speedup compared to a CPU-based simulation.In 2009 T. B. Vekterli [16], has showed parallelization of SNNs both on CPU and GPU. The method involves parallelization of SNN model using OpenMp implementation of up to 100,000 neurons on CPU and 70,000 neurons for GPU Computer Unified Device Architecture (CUDA) implementation each with synaptic connection of 100, 200 and 300 every time the network simulation was made to run. In this work, the OpenMp implementation have exhibited a speed ups of nearly 2x with four threads over the single threaded results. On the other hand, the CUDA implementation have proved to be the fastest alternative in the simulation process.

Another domain of simulation is employed on super computers in big research facilities like IBM research center using software-based techniques. For instance, IBM C2 simulator [1] demonstrated a rat-scale cortical simulation with a 55 Million neurons and 442Billion synapses using a Blue-Gene supercomputer having more than 32K processors. Such computational devices are due to their cost to make unfortunately the cost and development time make these approaches impractical for general purpose, large-scale simulations.

Although the idea of performing large scale neuron simulation is not and different techniques and successes have been reported, the prior works that exist on the simulation of spiking neural network on multiprocessor CPU cluster use the integrate and fire neuron model to simulate the neural dynamics. Using IF neuron model as a base for the simulation of large scale SNN is computationally efficient than the HH model but lacks biological realism. Another important and distinguishing issue with our work is that, previous works on distributed parallel computation use either Pure Message passing Interface/interaction or MPI with POSIX thread (Pthreads). In this work, we follow a hybrid parallel programming model that brings both a multithreading with OpenMP library and distributed MPI on polychronous Izikevich spiking neuron. Furthermore, the simulation effort targets a cluster consumer level multiprocessor clusters than costly high performance compute nodes.

3 Biological Background

Neurons which are the elementary building block of the cognitive system have different shapes and form, but they share common anatomical appearance and physiological activities. This specialized nerve cells constitute mainly the central nervous system (CNS) of animals, particularly the primary information processing "machine", the brain. Their number ranges from handful in invertebrates to 100 billons in human brain.

A biological neuron has four main parts (see Fig. 1) viz. dendrite, an axon, synapse and soma. Dendrite is a "tree" like structure in which a neuron receives an electro-chemical signal from other neurons where connection is established.

Soma is the cell body of a neuron where protein and other important substances are produced. It is also a place where electrochemical progress occurs. Moreover, electrical pulses are generated and incoming electrical signals are processed by the soma. An axon is a cable like extension where electrochemical pulse signal called action potential is sent through. In addition, it has a protective fatty white substance called myelin sheath which serves as an electrical insulation. Synapse is a narrow gap between one neuron's axon terminal and another neuron's dendrite. Individual neuron is "connected" to 1000–10,000 other neurons through synapse. It is also a place where the neuron passes its signal to other neurons by a means of special chemical substance called neurotransmitters.

The information processing and communication between these interconnected neurons is done by a means of electrochemical pulse signal called "spikes".

3.1 Spiking Neural Networks

In order to aid our understanding of the complexities and information processing capability of biological neural networks, different efforts have been made so far for so long to represent it at various levels of abstraction. Among them, the conventional Artificial Neural Networks (ANNs) are the popular and most widely used one for a variety of applications including real world problems. Since it was first introduced back in 1943 by McCulloch-Pitts [17], ANNs have evolved in to three generations along the past seven decades, distinguishing their computational unit from one generation to the other. For instance, the first generation of ANNs were based on applying a non-linear function (threshold or perceptron) to a weighted sum of inputs.

Experimental results from neurobiology: information processing in biological neural system depends on the timing of action potential (spike) of a neuron, have led to an emergence of Spiking Neural Networks (SNNs) that fall under the third generation of ANNs which take into account the spiking nature of real neurons and are able to encode spatial-temporal information into both spike timing and spiking rates. This makes SNNs more biologically plausible in mimicking the physiological activity of biological neural networks than previous generations [18]. Moreover, the information processing ability in general and cognitive capability specifically in organic brain is achieved through the weighted and complex interconnection of SNNs.

3.1.1 Single Neuron Model

In Computational models can be a paramount tool for comprehending the complex properties and activities that happen in organic brain. Several models exist that characterize neural dynamics of an individual biological neuron in multiple levels of abstraction. Here we present only the three most popular and widely used classes of computational single neuron modeling of neural dynamics starting from the oldest Hodgkin Huxley model to the recent Izhikevich model.

Detail discussion of this models and review of various mathematical models is beyond the scope of this research work. For further read, see [19].

3.1.1.1 Hodgkin-Huxley Neuron Model

Hodgkin-Huxley (HH) model [20], is a pioneering model in describing ionic processes and voltage dependent conductance during the generation and propagation of action potential, which was studied by Hodgkin and Huxley on a giant squid axon in 1952. It explains how a neuron responds by producing various spiking patterns to current stimuli. The model uses four differential equations and tens of parameters that describe the corresponding physiological measurements. The parameters introduced are biologically meaningful and measurable. Thus, it is considered to be the most biologically plausible model. However, the number of ordinary differential equations (ODE) used in the model incur high computational cost for simulation purpose using different computational platforms. For this reason, only a few number of neurons can be simulated using HH model [19].

3.1.1.2 Integrate-And-Fire Neuron Model

Despite the fact that HH model is biologically plausible, it incurs a considerable computational cost. Moreover, it is difficult to implement it on hardware devices. Therefore, more simplified models are needed. In Integrate- and-fire (IF) model [21] the four dimensional HH model is reduced to a two-dimensional model. As its very name suggests, this model proposes two steps that abstract the neural dynamics. The first stage is the integration stage, where by inputs are summed up. The next stage that rightly follows is the firing phase, in which the neuron fires if the summed value exceeds a predetermined threshold and reset the membrane potential. After the firing state the neuron enters into a refractory condition.

Because LIF models are simple to implement and easy to analyze they are the most widely used Spiking neuron models for a wide range of applications, however, IF model fails short to exhibit all neural dynamics and literatures so conclude that this model is not good enough for computer simulation if we are concerned of biologically realistic simulation [22].

3.1.1.3 Izhikevich Neuron Model

In a paper presented in 2003 [23], Izhikevich proposed a new model which is almost as accurate as the highly detailed Hodgkin-Huxley neuron model and also which has an efficient computational cost.

The Izhikevich (here after called “IZ”) neuron model is based on the knowledge of bifurcation mechanisms [3], which enables to reduce HH neuron model in to a two-dimensional system of first order ordinary differential equations (FODEs).

Accordingly, IZ neurons are represented by the following expressions:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{1}$$

$$u' = a(bv - u) \tag{2}$$

With the auxiliary after-spike resetting.

$$\text{If } v \geq 30\,\text{mV}, \text{then} \{u^v \leftarrow^{\leftarrow} u^c + d \tag{3}$$

Where v is the membrane potential, u is the recovery variable where the membrane potential is adjusted to. The dimensionless parameters a and b represent the sensitivity of the recovery variable and the time scale of the recovery variable respectively.

Represents the after-spike reset value of the membrane potential v. The common value of c = –65“m” V. The parameter d describes the value in which the recovery variable u is update after a particular spike has occurred.

The firing state of the neuron is determined if the membrane potential exceeds over a certain threshold, in this case + 30 mV. After reaching this apex, the membrane voltage and the recovery variable are reset according to Eq. (3).

By varying the values of a and d various spiking patterns that correspond to a particular type of neuron can be produced. For instance, for excitatory Regular spiking (RS) neurons, which are a typical neuron that reside in the cerebral cortex, we use the values a = 0.02 and d = 8. In addition, for inhibitory Fast Spiking (FS) cortical neurons we use the a = 0.1 and d = 2.

Two important factors must be taken in to consideration when choosing single neuron model for SNN implementation. That is the model has to biologically realistic by yield a reach set of firing patterns as exhibited in biological neurons and it has to be computationally simple and efficient when being implemented on various computational platforms [19]. Therefore, the models are scrutinized in light of the above two metrics when choosing a particular model for large scale simulation. In light of the above to criteria, HH model is biologically plausible yet computationally prohibitive. IF and its variant models are computationally simple and efficient but, fall short of being biologically plausible [4]. In the contrary, IZ single neuron model is both biologically realistic and computationally efficient. Therefore, we have so chosen it for this research work.

3.2 Learning in Spiking Neural Network

Any neural network gains functionality through learning or training function. Learning in conventional ANN is performed through back propagation or gradient descent in which errors are propagated backwards in the network. However, the learning and memory capabilities of the organic brain are substrate essentially on the plasticity of synapses (weight/conductance modification) between pre-synaptic and post-synaptic neurons. The first study regarding synaptic weight modification depending on the relative spiking of pre-synaptic and post- synaptic neurons was introduced by Hebb in 1949 [24]. In this paper, Hebb formulated a synapses weight adjustment which states: a synapse weight should be strengthened if a pre-synaptic neuron repeatedly takes part in firing before a postsynaptic neuron fires. This scenario is apparently dubbed as long-

term potentiation (LTP). Nonetheless, Hebb didn't addressed about synaptic weakening. Since then, a detailed explanation of the mechanism in which synaptic weight modification occur has been developed. One commonly used unsupervised Hebbian learning approach which uses spike timing information to set the synaptic weights for SNNs is called spike time dependent plasticity (STDP) [25]. STDP can be viewed as a more quantitative form of Hebbian learning. It emphasizes the importance of causality in synaptic strengthening or weakening; the relative spike timing of pre-synaptic and post-synaptic neurons leads to changing synaptic weight of the neurons [26]. Whenever STDP is invoked the synapses weight is either depreciates or potentiates. This phenomenon is hailed as long-term potentiation (LTP) and long-term depression (LTD), of synaptic weight value, respectively.

Another main characteristic of synaptic connection is the axonal conduction delay. After a pre-synaptic neuron has fired and before this spike signal reaches to the point of post synaptic neuron, a certain amount of time elapses. H.A. Swadlow [27] have reported regarding this study; the value of an axonal conduction delay in mammalian cortex depends on the neuron type and their location and this value ranges between 0.1 ms to 44 ms. Hence, when modeling dynamics of spiking neurons, a conduction delay has to be considered. For this reason, unlike conventional ANNs that discard axon conduction delay, when modeling biological SNNs it is essential to consider the time required for an action potential to travel from its initiation of presynaptic neuron to the axon terminals of the post-synaptic neuron. This feature makes possible the generation of stable, and time locked spatio-temporal neural firing patterns [28].

Regarding inhibitory neuron to allow the inhibitory connections to provide sufficient inhibition to their postsynaptic neurons, all their M synapses have a fixed conduction delay of 1 ms. As in the work in [23], for our network simulation the axonal conduction delay is as low as 1ms and as large as 20 ms.

One of the issues that determine the overall behavior of neural network simulation is the network structure. In this work, we have adopted the neural network structure used in [28]. Unlike the conventional feed forward neural network, this neural network structure involves neurons which are randomly connected, with axonal conduction delay and having polychromous property, which is briefly discussed in the ensuing section.

4 Parallel Programming Fundamentals

Parallel programming is a programming paradigm in which large and compute intensive problem is divided in to multiple smaller simultaneously executed tasks. This model can be implemented through one of the following ways; shared memory parallelism, distributed shared memory parallelism, and hybrid: shared-distributed memory parallelism. Different software technologies have been introduced to aid the implementation of these methods. The following section discusses briefly about these methods.

4.1 Shared Memory Parallelization

The architecture of recent decades consumer-level multicore computers have more than one processor inside each individual core. This feature makes them suitable for speeding up computations by concurrently executing independent tasks on these processors. There are different hardware and software approaches that are known to realize parallelization in a multiprocessor computer. One is multithreading parallel programming model: generation of multiple threads and assigning tasks to this threads for simultaneous execution.

A typical process; an instance of program execution in many of operating systems (OS) available have a single master thread in which instructions are executed sequentially. This thread has its own program counter to keep track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history. To execute a computational task single threaded processes tasks a lot of resource than multiple threaded process. This is because, the creation of multiple threads provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Threads are "light weight" processes. Threads created in the same process share the same address space.

4.1.1 Open Multi-processing (OpenMP)

OpenMP is the de facto industry standard API for parallel programming paradigm which targets shared memory multiprocessors [29]. It provides a set of compiler directives, run-time library functions and environment variables that enables the Transformation of a sequential source code written in FORTRAN, C, or C++ programming language in to a parallel application. This feature makes it easy to implement compared to other parallel programming APIs like Pthread for multithreaded parallel programming. It is essentially based on fork-and-joint parallelism model, where threads in a particular process are categorized as master and slave. In this model, program execution commences as a single process or thread which is hailed as master thread and continues the execution sequentially. When "Parallel" parallelization directive is encountered in the sequential source code, a block of slave threads are forked/created under this master thread and continue the program execution in a parallelized manner until the end of the parallel region. When this team of threads finish execution of job, they join back together again and only a single thread continues.

4.2 Distributed Parallel Programming Model

In distributed memory parallel programming paradigm, a program runs as a separate, independent process with a private memory can be considered as separate serial programs. Parallelization for distributed memory computer systems is done by distribution of computational task and data on the processors through message passing programming model. This processes communicate via message passing programming model: a coordinated communication between processes inside a network of computer nodes.

In the message passing model, resource on a networked computers have their own local memory and they communicate and coordinate to execute a set of tasks through exchange of data by sending and receiving messages between the nodes. Among the

various message passing software approaches that run on a distributed compute nodes, Message passing Interface (MPI) is the popular and widely used one. In message passing model processes in compute nodes share data and coordinate/synchronize their operation.

4.2.1 Message Passing Interface

Message passing Interface (MPI) [30] is a message-passing library interface specification used to handle data communication between distributed shared memory computers. Currently, it is the *defacto* industry standard API for distributed parallel programming. Essentially, it comes with different library functions which allow coordinated message passing between distributed nodes. This includes a protocol and semantic specifications for how its features behave in any implementation. Depending on the type of the operating system a process contains a program counter and address spaces. In addition, a single process can have multiple threads sharing a common address spaces. MPI parallel programming model enables a communication between separate and distribute processes. In an MPI application processes that are allowed to communicate with each other do the communication through a default MPI communicator function. Processes that are bound in a communicator will have a unique identifier called rank, numbered from 0 to n–1. By default in an MPI program, initially all the processes belong to the MPI_COMM_WORLD communicator.

MPI API offers two communication schemes namely, point to point communication and collective communication. Point-to-point communication scheme involves exclusive sending and receiving of messages between two processes. Conversely, in collective communication scheme all processes do part-take in passing message from one processes to all processes in a more efficient manner. For instance data can be scattered to all other processes using a single communication routine, MPI_scatter with reduces the communication overhead had it been used in a point-to-point communication. Among open source MPI implementations, LAM-MPI [31] and MPICH [32] are the most widespread use ones, where the former is for Unix-based systems and the later for Windows systems. Hence, for our simulation, we have used MPICH that is based on MPI 3.1 standard.

4.3 Hybrid Parallel Programming Model

It is intuitive to assume employing both shared and distributed parallel programming models to work in tandem for a performance improvement of a computer program at hand. Such a programming approach is hailed as a hybrid/mixed mode parallel programming model. It is the use of two distinct parallel programming models in a complimentary way for a cluster of multiprocessor computer nodes. Unlike pure multithreading parallelization approach inside a shared memory programming node, message passing between nodes will also occur in this model.

Through this hybrid programming model, multiple threads are forked inside each process in each multiprocessor node while MPI API controls the communication between this processes.

This scenario is shown as below in Fig. 5, how a two-dimensional array is decomposed in two four processes and three threads in individual process.

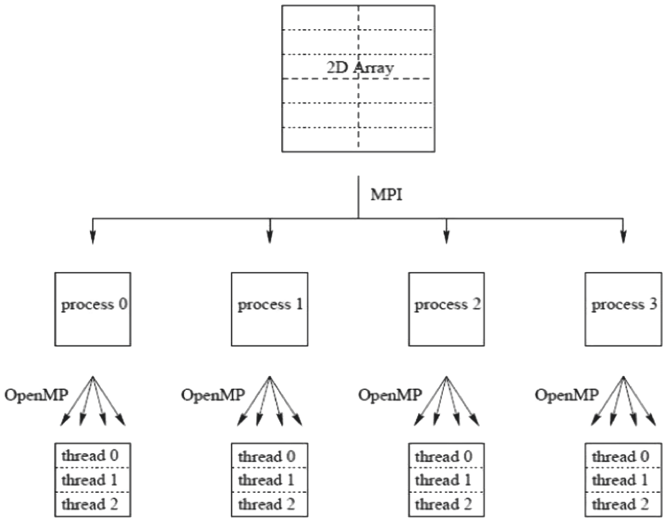

Fig. 1. Hybrid MPI OpenMp integration model [33]

5 Methodology

Different techniques have been proposed and implemented as for the parallel simulation is concerned. For instance, a paper by A. Jahnke and et al. [28] States that parallel simulation of SNN can be realized through three approaches namely, neuron parallelism (n-parallel), synapses parallelism (s-parallel), pattern parallelism (p-parallel). In n-parallel approach each neuron are mapped to a processing element and associated computations are processed in parallel while their corresponding computations with synapses state update are executed serially. In s-parallel approach the synapses state update of each neuron is executed using parallel processing element. As individual neuron will have more than one synapses connections, this approach gives an additional opportunity for parallelization. The third approach, p-parallel approach uses the above two approaches in tandem i.e. when the neuron state update is required the process is executed in a parallel fashion so will be the case for synapses state updates. In this work we have used the n-parallel approach for the pure MPI parallelization and p-parallel approach for the hybrid MPI +OpenMp parallelization.

Moreover, another perspective about simulation techniques of SNN can have a different categories. According to [34], two class of methods do exist in SNN simulation viz. time-step driven, and event-driven.

In time-step driven simulation the state variable of individual neuron in the SNN is updated at every discrete time step. On the contrary, in an event - driven simulation the state of SNN is updated only at the instant of spike occurrence at a particular neuron. The later gives a speed up advantage as update to the neural network only occurs when event/spike is emitted or arrives. Moreover, it suits the IZ single neuron model used in this research work. The mixed mode simulation approach amalgamates the above two methods. The important feature of this method is that, it renders an opportunity to use

the advantages of both time-step driven simulation and event driven simulation. Therefore, in our simulation we have adopted this approach for the reason mentioned just above.

The real-time simulation value for both network construction and simulation phase of the purely serial execution and parallel execution for simulating 10s of model time was recorded. To keep the stochastic nature of neuron connection, the simulation were run five times with different random number seed and the average of five run was taken at the end for each number of varying synapses and neuron value.

An effort to get the average sequential execution time for a large problem size faces a hindrance due to both high memory requirement and time restrictions. For instance, for synapses value of 100 it was possible to measure average execution time of only 120,000 neurons on a single process. Moreover, it was also impossible to get an average sequential execution time for neuron size of more than 80,000 with 200 synapse connection. Consequently, to get the speed up gain for exceeding network sizes, we need to extrapolate based on the value we got for lesser problem sizes. Execution time for sequential simulation that exceeds running capability of a single processor can be found by extrapolating the result gained from less work load [35]. The extrapolation was done using a second order polynomial extrapolation as it takes in to consideration the entire data points of the simulations average runtime before the compute intensive network size.

5.1 MPI Based Parallel Simulation

The SNN simulation passes through two main steps: network construction and actual simulation steps. This section outlines the steps followed to make a solely MPI based parallel implementation. The original MATLAB/C++ algorithm used in [28] passes through two main stages namely; network construction and network simulation. The following paragraph discusses the steps used to map this two stages on to a cluster of multiprocessors using MPI API. Before making the actual network simulation, a network of spiking neuron is constructed under network construction section using the steps mentioned below.

Step 1:- Array of N neuron holding the index of i^{th} individual neuron were distributed equally with static load balancing in mind among the MPI processes using the MPI_scatter routine. In this step the number of neuron population was based on the 4:1 ratio of excitatory and inhibitory neuron type of the total neuron population (i.e. 80% are excitatory neuron and 20% are inhibitory neuron in each node) Ne = N*0.8 and Ni = N*0.2.

Step 2:- For each N neuron, M number of synapses which resides in the i index holding the index of the post-synapses neuron j projecting from neuron i is distributed among the MPI processes in the cluster in which neuron j will exist [36]. In this way the data structure that contains a neuron and synapses that it is post-synaptic do exist in on the same processes rank.

Step 3:- Initialization of neural dynamics parameters a and d.

For Ne: a = 0.02 and d = 8, For Ni: a = 0.1 and d = 2. This determines the behavior of each neurons.

Step 4:- A call to the MPI_all_to_all collective communication routine will create awareness about a connection from the incoming axon of neuron N will be created. Impliedly, this will establish the connection between neurons that reside in different process. In addition, excitatory neurons (Ne) connect to other model processes in compute nodes share data and coordinate/synchronize their operation. Excitatory (Ne) and inhibitory neurons (Ni) but, inhibitory neurons only connect to excitatory neurons.

Step 5:- Initial synaptic weight (s) of 6.0 mV is assigned for every excitatory connections and –5.0 mV for every inhibitory connections. The synaptic weight derivative; the value the synapse weight to be modified (*sd*) is also set to zero at this stage.

Step 6:- For each neuron in same MPI process an axonal conduction delay (D) in a range from 1ms-20ms was assigned with a uniform distribution of M/D axon conduction delay table. For inhibitory neuron an axonal conduction delay of 1ms was assigned.

Step 7:- For each neuron in the same MPI process an initial value of the activity variables like the membrane potential v = 65 mV and recovery variable u = 0.02*v was assigned according to the IZ neuron model. This will make neurons to wait in the reset state until a triggering signal reaches and makes it to spike. In addition, the STDP functions LTP and LTD were made to be zero at this stage.

The network simulation Stage can be broken down in to a category of steps:
For each simulation second:

Step 1:- Reset Input step:
For every neuron N the value of input to the neurons is reset to zero.

Step 2:- Spike generation step:
For a randomly picked neuron thalamic input current with a value of 20 mV is supplied.

Step 3:- Spike detection step:
The membrane potential, v, of individual neuron is checked whether it has reached at its apex vt, which is set to cause a spike at greater than or equal to the value of vt.
If any neuron have fired during this time step (i.e. $v \geq vt$), then:
The membrane potential value is re-set to parameter c, membrane recovery is incremented by parameter d,
The time step associated with this neuron is updated and the neuron index is added to the list of fired neurons.
STDP variables of the neuron; LTP and LTD are reset to their maximum values of 0.1 and 0.12 respectively, according to the STDP rule.
Synapse derivative sd connections with the neuron are potentiated according to STDP LTP according to the time signature the last spike has occurred relative to the current firing.

Step 4:- Spike Exchange between processes:
All neurons that have fired in the D previous time steps are processed.
Each MPI process containing the fired neuron broadcasts spike input to other MPI processes holding the post synaptic neurons.

The total current introduced by the sent input spike to the target neuron are calculated and stored to be used for the neuron state update stage.

Step 5:- Neuron state update step:

For each individual neuron in the subset of each MPI process.

Membrane potential v is updated according to IZ model numerically using a forward Euler method based on input from previous simulation stage.

Step 6:- Synapses state update step:

For every Ne, their synapses s and their derivative Sd is updated as:

$$s \leftarrow s + 0.001 + Sd \text{ and } Sd = 0.9 * Sd \tag{4}$$

The constants 0.01 represents activity independent increase of synaptic weight and 0.9 a simulation parameter as suggested in [35].

If $s \geq 10$ mV, then $s \leftarrow 10$ mV

If $s < 10$ mV, then $s \leftarrow 0$ mV

5.1.1 Spike Exchange between Processes

Each MPI processes communicate by sending and receiving spike messages. In order to make a spike exchange between MPI processes, address event representation (AER) scheme which was first rolled out for communication in neuromorphic chips [36] was adopted. In this mechanism a spike message from neuron N will be represented by the neuron id of spike origin and the time signature of the spike that has occurred. Hence, Information about all the neuron that have fired in the same simulation time step is gathered by MPI processes using MPI_All_to_all communication routine and distributed to all the neuron that they have established a connection.the parallel region. When this team of threads finish execution of job, they join back together again and only a single thread continues.

The single neuron model's FODE is not suitable for use in a computer program. Therefore, the ODE was approximated using a numerical method that fits to a computer program. Among the available numerical integration methods for ODE, forward Euler's method was used because it demands less number of iterations unlike its counterpart backward Euler's Method. Moreover, in our simulation, in order to update the membrane potential value we have used four iterations and 0.1 time step for the sake of minimizing the numerical error incurred due to the method we followed.

5.1.2 Hybrid MPI+OpenMP Parallelization Simulation

In the purely MPI based simulation there are portion of the SNN simulation were we can take advantage of multithreaded parallel programming. In the network construction/initialization phase(initialization of neuron dynamic parameters, synapses and synapses weight derivatives) and in the simulation phase it contains for loops including neuron and synapses state updates which can be executed in a parallel fashion on more than one thread. Consequently, an additional loop level parallelization is introduced inside the pure MPI based implementation. As it was mentioned above in Sect. 4.3.1, MPI API supports a multithreaded parallel programming.

In order to use OpenMp API to parallelize the source code, the integrated development environment (IDE) used for source code compilation and editing has to be enabled to support it. The IDE used in this work, visual studio 2013, supports OpenMp 2.0 hence was enabled for multithreaded execution.

In this approach a omp_set_num_threads routine will be called to specify the number of threads to be used in each processes. In this regard, among the available 4 logical processors in side each the cluster, a thread number of 2, and 4 are forked in each simulation experiment with in each MPI process of combination for 8 and 4 MPI processes respectively.

To ensure work load balance between threads in each process dynamic work scheduling was incorporated using dynamic clause (#pragma omp parallel for schedule (dynamic, 1). This scheduling mechanism is a good way to reduce work load imbalance between OpenMP threads created in MPI processes.

To get the execution time of both the network construction, and the actual simulation phases on the master node, an MPI routine, MPI_wtime was used.

5.2 Experimental Setup

To achieve efficient parallelization and scale up in simulation time of SNN with a considerable amount of neurons and synapses connections, we have used a cluster of nodes with multiprocessors where the nodes are connected in separate private network.

The simulation was conducted on a 4-node cluster, each of which are configured with Intel core i3 3.30 GHz processors enabled with hyper threading and having 2 GB RAM. MPICH API was installed on each nodes in the cluster and the processes were managed using MPICH process manager.

8-Port 10/100Mb fast Ethernet switch which enables a communication between the connected nodes in the network was used. This communication between the computer nodes was made using Transmission Control protocol (TCP)/Internet Protocol (IP) protocol version 4. The server management software we used was Microsoft windows server 2012 standard edition. In order to have a network of computers to function suitably for our experiment, the following services were installed and configured with an appropriate setting on it; Active Directory Domain Service (ADDS) for role management of nodes, DNS (Domain Name System) for resolving host names and to group host computers into domains, Network File System (NFS) for sharing files between nodes in the cluster network and DHCP (Domain Host Configuration protocol) to dynamically assign IP addresses to the nodes that exist in the cluster network. The head node serve as a domain controller for the cluster and slave nodes are made to join the domain. The issue of a proper mechanism of process mapping among the multiprocessor nodes and the communication between the nodes in performance is important. In this research work maximum of one process is created per logical processors in each node in the cluster and a thread number of 2 and 4 created in each process employed in the cluster. The main reason in doing so is that this mechanism enables a better communication control among the individual process in the nodes.

6 Result and Discussion

Pursuant to the methodologies followed in the previous section, this section presents the performance results of the simulation of N-spiking neural network using two parallelization methods on a multiprocessor cluster. Moreover, we elucidate the observations made based on these results.

One of the goals of using parallel programming and multiprocessor cluster in an application is performance improvement. The two important quantities that allow to make performance evaluation are speedup and efficiency.

A speed up gain from using n number of processing elements can be gained as:

$$\text{Efficiency} = (\text{Speed up (n)})/(\text{No. of processors}) \tag{5}$$

In this regards, the results of both purely MPI and hybridized MPI + OpenMp implementation is presented accordingly below.

6.1 MPI Implementation

The following paragraphs entail the results from solely MPI implementation of the simulation of different SNN configuration on a multiprocessor cluster. In order to observe the performance of the cluster implementation we have calculated the speed up based on the average execution time of the simulation both for the serial and parallel implementation on different network sizes by varying number of MPI processes.

The following figure depicts the speed up gain from MPI processes of 2, 4, 8 and 16 when used in a network of increasing number of neurons of 20k, 40k, 80k and 160k with 100 synapses connections per each neuron.

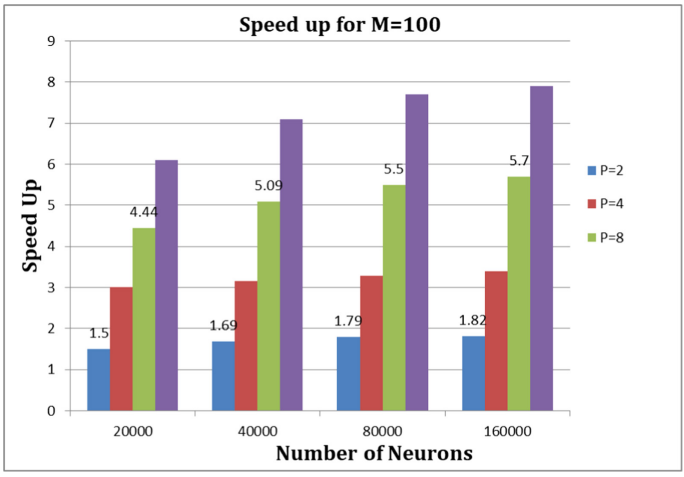

Fig. 2. Speed up of scaling number of neurons and increasing MPI processes

$$\text{Speedup}(n) = Ts/Tp \tag{6}$$

Where, Ts is the sequential execution time and Tp is the execution time for n number of processing elements

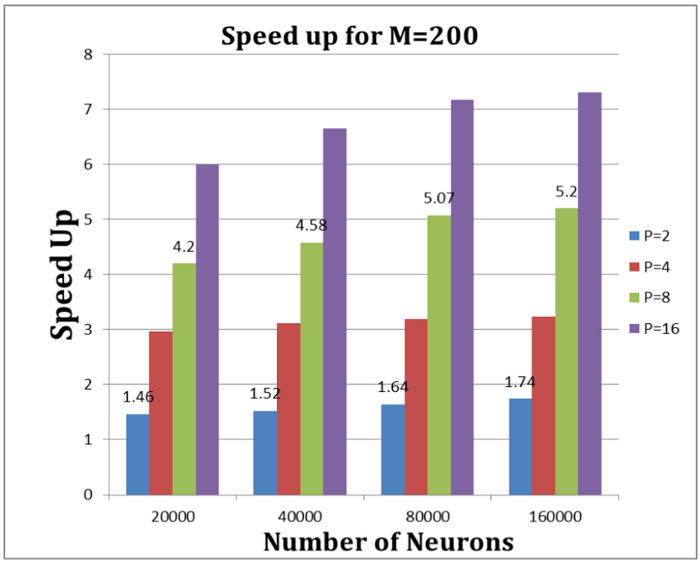

Fig. 3. Speed up of scaling number of neuron and MPI 200 Synapses

The efficiency of a parallel implementation can be calculated as follow:

Figure 2 shows the speed up gain for MPI processes of 2, 4, 8 and 16 when used in a network of increasing number of neurons of 20k, 40k, 80k and160k with 200 synapses connections per each neuron.

A parallel program is considered as scalable if the speed up increases with increasing number of parallel processing elements. Accordingly, for an ideal speed up of a parallel application using four processes will be four but, most applications exhibit sub linear speedup. In this regard, we could observe from Figs. 7 and 8 that the speed up of both network configuration scales as the number of MPI process increases. But a better scalability was observed in an increased number of neuron and when a lesser number of MPI processes are used. For instance, the speed up for 2 MPI processes on 20k neuron with 100 synapses was 1.5 while for 160k neuron with 100 synapses was 1.82, which is closer to the ideal speed up. Similarly, the speed up for 2 MPI processes on 20k neuron with 200 synapses is 1.46 and on 160k neuron with 200 synapses is 1.74.

As illustrated in both figures we could observe that increasing the number of synapses from 100 to 200 while keeping the neuron population fixed in the network makes the overall speed up to decrease. For example the speed up for N = 80,000 has

decreased from 5.5 to 5.07 when 8 MPI processes are used. The prime reason for this would be the bandwidth and memory requirement due to increased connection per neuron so that more spike messages has to traverse from one MPI processes to another.

Figure 3 illustrates reduction in average execution time using different number MPI process implementation on varying number of neurons with 100 synapses connections per neuron. To use this approach. This saturation issue in scalability could be addressed by introducing threads in each processes Fig. 4 shows a strong scaling evaluation the effect of increasing synapses connection per neuron. In this regard, the speed up for network size that contains 160k neurons with 100 and 200 synapses connection for scaling MPI processes is presented.

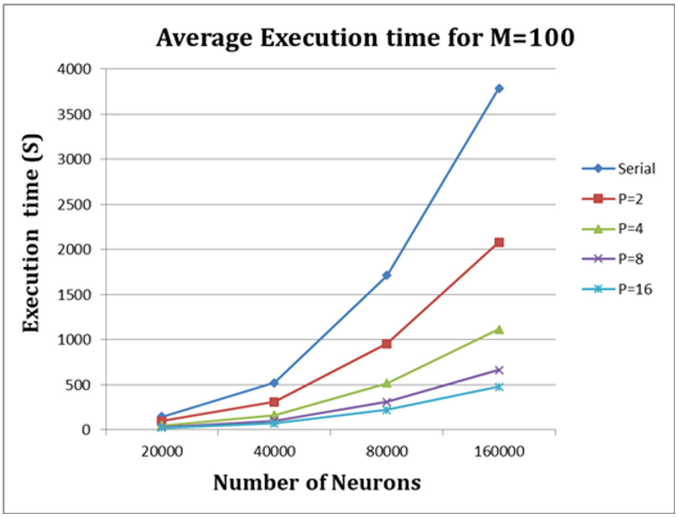

Fig. 4. Average execution time in seconds for increasing number of number of neurons simulated under different MPI processes

We can observe from the above figure that a significant decrease in execution time of the simulation for 2 and 4 MPI process. Even though the average execution time beyond 8MPI process does not decreased as per the theoretically expected level, the time saved for running the simulation still does worth to use this approach. This saturation issue in scalability could be addressed by introducing threads in each processes.

Figure 4 shows a strong scaling evaluation the effect of increasing synapses connection per neuron. In this regard, the speed up for network size that contains 160k neurons with 100 and 200 synapses connection for scaling MPI processes is presented.

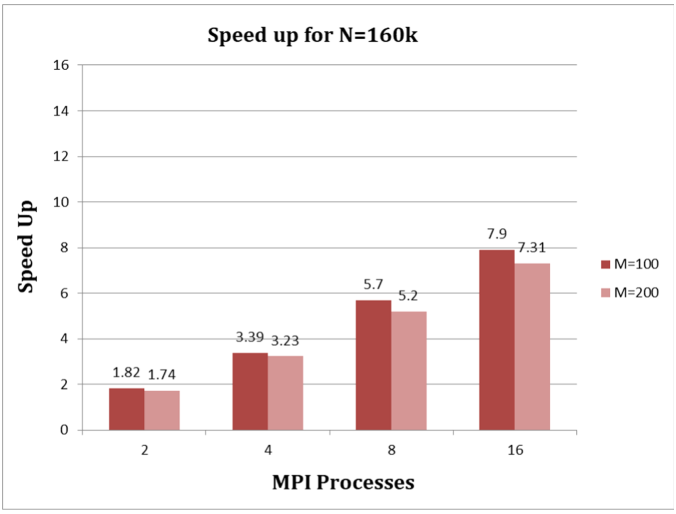

Fig. 5. Speed up for N = 160K neuron with M = 100 and M = 200.

Accordingly, a speed up gain from the MPI implementation for 160,000 neurons with 100 and 200 synapses values. The speed up scales very good for a less number of process. But, as the synapses per neuron increases from 100 to 200 and the number MPI processes increases the speed up scalability decreases more noticeably.

6.2 Hybrid MPI+OpenMP Implementation

Below we present performance evaluation for the hybrid MPI +OpenMP implementation for a network size of 160K, neurons with 100 and 200 synapses values per each neuron. In this experiment the number of OpenMP threads per process is limited by the number of cores in each nodes. Figure 11 shows the speed up of the simulation for various combination of MPI processes and OpenMp threads (Fig. 6).

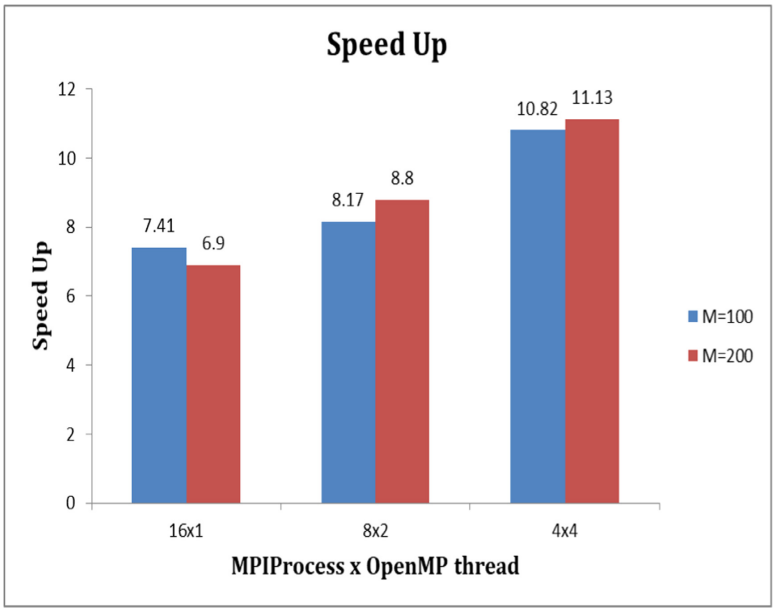

Fig. 6. Speed up for N = 160K neuron with M = 100 and M = 200

Figure 7 depicts the efficiency of a hybrid parallel programming for a network size of 160K neuron with 100 and 200 synapses connections in each neuron using a hybrid parallelization strategy for varying number of MPI processes and OpenMp threads in each process.

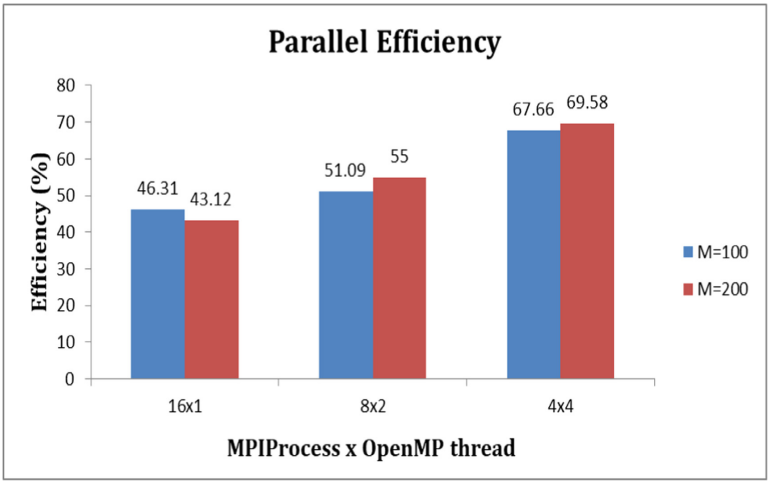

Fig. 7. Efficiency comparison for N = 160k with M = 100 and 200

From the figure we can observe that the hybrid parallel programming approach has improved the efficiency of the purely MPI implementation for 160k neuron with 100 synapses by 4.69% when 2 OpenMP threads were used in 8 MPI process and by 21.35 % when 4 OpenMP threads were used in 4 MPI processes.

Similarly, the efficiency of purely MPI implementation for 160k with 200 synapses was raised by 11.88% when 2 OpenMP threads are used in 8 MPI processes and by 26.46% when 4 OpenMP threads were used inside the 4 MPI processes. As it can be noticed from the results the speed up for all the performance measurement of the parallel implementations is below the theoretically expected value which is also dubbed as sup linear scaling. The usual suspect for such issues is the communication overhead of the MPI processes and OpenMp threads.

Better performance was observed in the hybrid parallel programming approach when 4 MPI process was used in tandem with 4 number of threads. Moreover, a significant drop in average execution time was observed in increased number of threads.

Figure 13 shows the average execution time of the simulation on sequential and hybridized parallel execution. As the result shows the execution time decreases significantly when threads along with MPI processes used on the sequential execution.

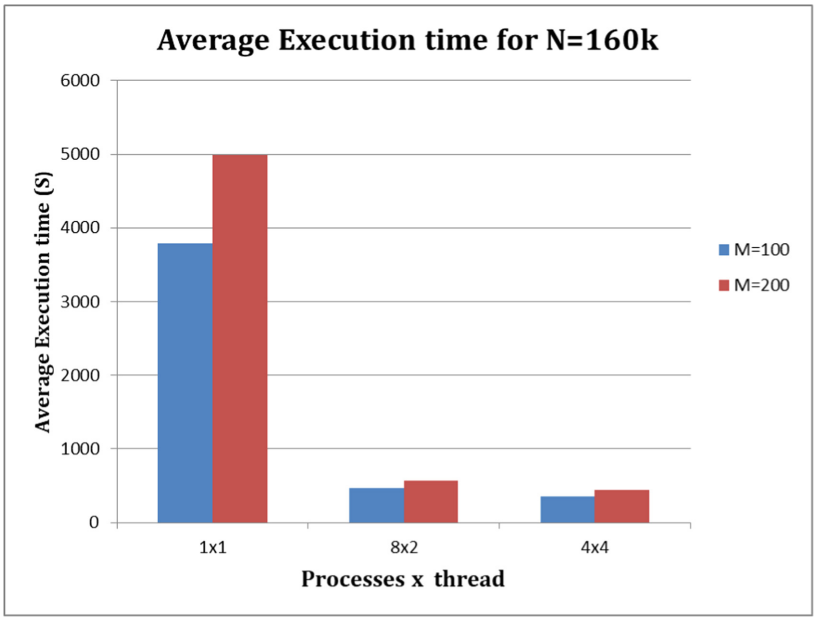

Fig. 8. Average execution time of serial and hybrid implementation

7 Conclusion and Recommendations

In this research work, we have demonstrated biologically realistic large scale simulation of spiking N-neurons on a multiprocessor cluster using distributed memory and distributed shared memory parallel programming through two APIs viz. MPI and OpenMP. The original simulation effort done on a single processor was mapped on a consumer level cluster of multiprocessors with the relatively cheaper price cost for making such compute intensive simulations using supercomputing facilities or specialized hardware platforms.

The performance of our simulation was analyzed for different configurations of the SNN, first for MPI based implementation then followed by a Hybrid MPI +OpenMp based implementation is presented in terms of speed up or efficiency for varying arrangement of MPI processes and OpenMp threads.

From the experimental results the purely MPI based implementation scales good and demonstrated performance gains until 8 MPI processes. Beyond 8 MPI processes the scalability suffers mainly from communication overhead and by the virtue of the fact that the actual processors used are logical processors. But still it does worth to go for the solely MPI based simulation than the sequential simulation as it has allowed more number of neuron and synapse connections.

The scalability issue in the purely MPI based simulation with increasing MPI processes is addressed using Hybridized parallel programming approach. Furthermore, our performance evaluation shows that the hybrid programming out performs the purely distributed programming for the simulation of SNNs. For 160K neuron with 100 synapses and 200 synapses the best performance was gained at 4 thread with 4 MPI processes. Hence, it is an ideal candidate for a multiprocessor cluster based SNN simulation.

The achievements reported in this paper can be further enhanced by developing an easy to use tool set can be developed on the top of the implementation. Moreover, further analysis on memory consumption of both parallel programming approaches can be conducted. Additionally, future research might consider employing different load balancing mechanisms particularly when using purely MPI implementation.

References

1. Ananthanarayanan, R., Modha, D.S: Anatomy of a cortical simulator. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, p. 3 (2007)
2. Wolff, C., Hartmann, G., Rückert, U.: ParSPIKE-a parallel DSPaccelerator for dynamic simulation of large spiking neural networks. In: MicroNeuro 1999 Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, pp. 324–331 (1999)
3. Jahnke, A., Roth, U., Klar, H.: A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN). In: Proceedings of Fifth International Conference on Microelectronics for Neural Networks, pp. 232–237 (1996)

4. Cheung, K., Schultz, S.R., Leong, P.H.: A parallel spiking neural network simulator. In: Proceedings of International Conference on Field-Programmable Technology (FPT), pp. 247–254 (2009)
5. Thomas, D.B., Luk, W.: FPGA accelerated simulation of biologically plausible spiking neural networks. In: 17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009 FCCM 2009, pp. 45–52 (2009)
6. Bower, J.M., Beeman, D.: The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System, 2nd edn. Springer Science & Business Media, New York (2003)
7. Hines, M.L., Carnevale, N.T.: The NEURON simulation environment. Neural Comput. **9**(6), 1179–1209 (1997)
8. Migliore, M., Cannia, C., Lytton, W., Markram, H., Hines, M.L.: Parallel network simulations with NEURON. J. Comput. Neurosci. **21**(2), 119–129 (2006)
9. Diesmann, M., Gewaltig, M.-O.: NEST: an environment for neural systems simulations. Forsch. Wisschenschaftliches Rechn. Beitr. Zum Heinz-Billing-Preis **58**, 43–70 (2001)
10. Thomas, E.A.: A parallel algorithm for simulation of large neural networks. J. Neurosci. Methods **98**(2), 123–134 (2000)
11. Plesser, H.E., Eppler, J.M., Morrison, A., Diesmann, M., Gewaltig, M.-O.: Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 672–681. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74466-5_71
12. Hu, J.: BIOPHYSICALLY ACCURATE BRAIN MODELING AND SIMULATION USING. A&M University, Texas (2012)
13. Paolucci, P.S., et al.: Distributed simulation of polychronous and plastic spiking neural networks: strong and weak scaling of a representative miniapplication benchmark executed on a small-scale commodity cluster (2013). arXiv Prepr. arXiv13108478
14. Fidjeland, A.K., Shanahan, M.P.: Accelerated simulation of spiking neural networks using GPUs. In: The International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2010)
15. Yudanov, D, Shaaban, M, Melton, R, Reznik, L.: GPU-based simulation of spiking neural networks with real-time performance & high accuracy. In: IJCNN, pp. 1–8 (2010)
16. Vekterli, T.B.: Parallelization of artificial spiking neural networks on the CPU and GPU. Master's Research, Norwegian University of Science and Technology, Department of Computer and Information Science, Trondheim (2009)
17. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biol. **52**(1–2), 99–115 (1990)
18. Paugam-Moisy, H.: Spiking neuron networks: a survey. Rapp. Tech. RR-11 IDIAP Martigny Switz (2006)
19. Izhikevich, E.M.: Which model to use for cortical spiking neurons? IEEE Trans. Neural Netw. **15**(5), 1063–1070 (2004)
20. Hodgkin, A.L., Huxley, A.F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. **117**(4), 500–544 (1952)
21. Burkitt, A.N.: A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. Biol. Cybern. **95**(1), 1–19 (2006). https://doi.org/10.1007/s00422-006-0068-6
22. Feng, J.: Is the integrate-and-fire model good enough?—a review. Neural Netw. **14**(6), 955–975 (2001)
23. Izhikevich, E.M., et al.: Simple model of spiking neurons. IEEE Trans. Neural Netw. **14**(6), 1569–1572 (2003)
24. Hebb, D.O.: The Organization of Behavior: A Neuropsychological Theory. Wiley, Hoboken (1949)

25. Song, S., Miller, K.D., Abbott, L.F.: Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nat. Neurosci. **3**(9), 919–926 (2000)
26. Bi, G., Poo, M.: Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. J. Neurosci. **18**(24), 10464–10472 (1998)
27. Swadlow, H.A.: Physiological properties of individual cerebral axons studied in vivo for as long as one year. J. Neurophysiol. **54**(5), 1346–1362 (1985)
28. Izhikevich, E.M.: Polychronization: computation with spikes. Neural Comput. **18**(2), 245–282 (2006)
29. Dagum, L., Enon, R.: OpenMP: an industry standard API for sharedmemory programming. Comput. Sci. Eng. IEEE **5**(1), 46–55 (1998)
30. Message Passing Interface (MPI) Forum Home Page. http://www.mpi-forum.org/. Accessed 19 May 2016
31. Burns, G., Daoud, R., Vaigl, J.: LAM: An open cluster environment for MPI. Proc. Supercomput. Symp. **94**, 379–386 (1994)
32. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Comput. **22**(6), 789–828 (1996)
33. Smith, L.A.: Mixed mode MPI/OpenMP programming. UK High-End Computing Technological Report, pp. 1–25 (2000)
34. Brette, R., et al.: Simulation of networks of spiking neurons: a review of tools and strategies. J. Comput. Neurosci. **23**(3), 349–398 (2007)
35. de Velde, E.F.V.: Concurrent Scientific Computing. Springer Science & Business Media, Berlin (2013)
36. Merolla, P.A., Arthur, J.V., Shi, B.E., Boahen, K.A.: "Expandable networks for neuromorphic chips. IEEE Trans.Circuits Syst. Regul. Pap. **54**(2), 301–311 (2007)