MONASH University
Information Technology

ASSESSMENT COVER SHEET

Student ID number	27502333
Given Name	HUSSAIN SADIQ
Family name	ABUWALA

Unit Name and Code:	**FIT 3143 PARALLEL COMPUTING**		
Campus:	**MALAYSIA**		
Assignment Title:	**ASSIGNMENT - 2**		
Name of Lecturer:	Dr Vishnu Monn Bhaskaran		
Name of Tutor:	Dr Vishnu Monn Bhaskaran		
Tutorial Day and Time:	**Tueday 2.00 PM**		
Phone Number:	**+60 11-6110 8857**		
Email Address:	**hsabu1@student.monash.edu**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes NO☐ No			
Due Date:	**8 october 2018**	Date Submitted:	8 october 2018

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date)_____ Signature of lecturer/tutor _____

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature .....HUSSAIN..... Date……8/10/2018……

* delete (iii) if not applicable

The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. *Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If*

QUESTION - 1

I. Overview

WSN Description: The remote sensor organize involves 20 nodes and a base station. These nodes are masterminded in a 4 x 5 (rectangular-formed) matrix.

Separation between the contiguous nodes is kept in that capacity that these nodes can remotely impart. The contiguous nodes can trade information through unicast and communicate methods of correspondences.

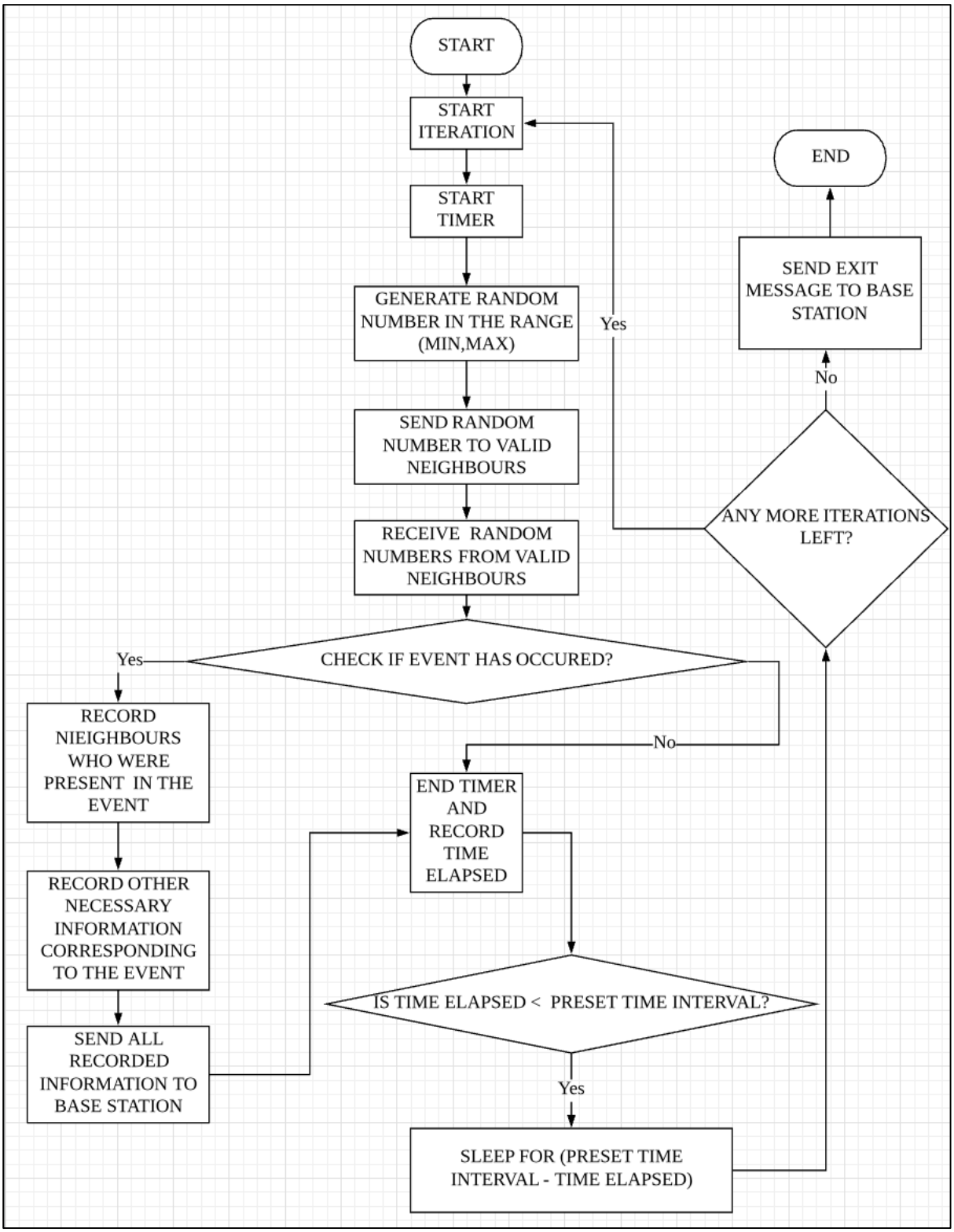
Correspondence between non-nearby nodes isn't conceivable.

Each node in the WSN can anyway autonomously trade information with the base-station (e.g. through a satellite connection). Base-station for this WSN is an extra PC that is depended with the undertaking of information from all the 20 nodes in the WSN

Event Detection Criterion: all together for an event to be recorded by the WSN, no less than three contiguous nodes, to a reference node, should at the same time report their initiations to the base-station. The base station at that point gathers all the occasion reports and composes these to its log document.

The objective is to find a communication scheme that minimizes the number of messages sent to the base station while satisfying the event detection criteria.

II. How the system works from the point of view of a particular node (NOT THE BASE STATION) in the system



III. Explanation of the above flowchart

Firstly, there are in total 21 MPI processes where 20 of them are nodes and one of them is the base station. The flowchart above shows how any of the 20 nodes would work.

I have visualized the 20 nodes as a two dimensional grid as shown below:

RANK 1 NODE	RANK 2 NODE	RANK 3 NODE	RANK 4 NODE	RANK 5 NODE
RANK 6 NODE	RANK 7 NODE	RANK 8 NODE	RANK 9 NODE	RANK 10 NODE
RANK 11 NODE	RANK 12 NODE	RANK 13 NODE	RANK 14 NODE	RANK 15 NODE
RANK 16 NODE	RANK 17 NODE	RANK 18 NODE	RANK 19 NODE	RANK 20 NODE

There are few parameters which will be predefined before running the program:

- **Upper Bound** – this is the maximum value the random number can take.
- **Lower Bound** – this is the minimum value the random number can take.
- **Total Iterations** – the total number of iteration each node will run through.
- **Preset Time** – the minimum time for which each iteration.
- **Time Elapsed** – the actual time which took for a particular iteration.

1. At the start of each iteration, the **start time** is recorded using **MPI_Wtime ()**
2. Each node generates a random number in the range:

   (Lower Bound <= N <= Upper Bound)

3. Each node sends its random number to only those neighbors who themselves have at least three neighbors. These neighbors are known as valid neighbors because only the valid neighbors can have an event. **This is one strategy I used to minimize messages being passed.**
4. Each node also receives random numbers from its neighbors only if the receiving node is itself a valid neighbor for its neighbors.
5. For send and receive, I am using **MPI_Isend ()** and **MPI_Irecv (). I used this over MPI_Send () and MPI_Recv (),** because I wanted to avoid a deadlock situation. The way my program is structured, each node always sends first and then receives. If I used blocking send and receive, each node would send and wait for the other node to receive the message. But the other node is also sending first and waiting for others to receive. This is how my program would go into a deadlock. On the other hand, as I am using non-blocking, i had to make sure that the message is sent and received. For that I used **MPI_Waitall ()** which basically blocks the program until messages are sent and received.
6. After every node receive random numbers from its respective valid neighbors, then I check if an event has occurred or not.
7. Checking of the event is rather straightforward, I just check if there are at least three random numbers which are equal and record the respective nodes associated with each random number.

8. If there is an event, each node record 4 information:

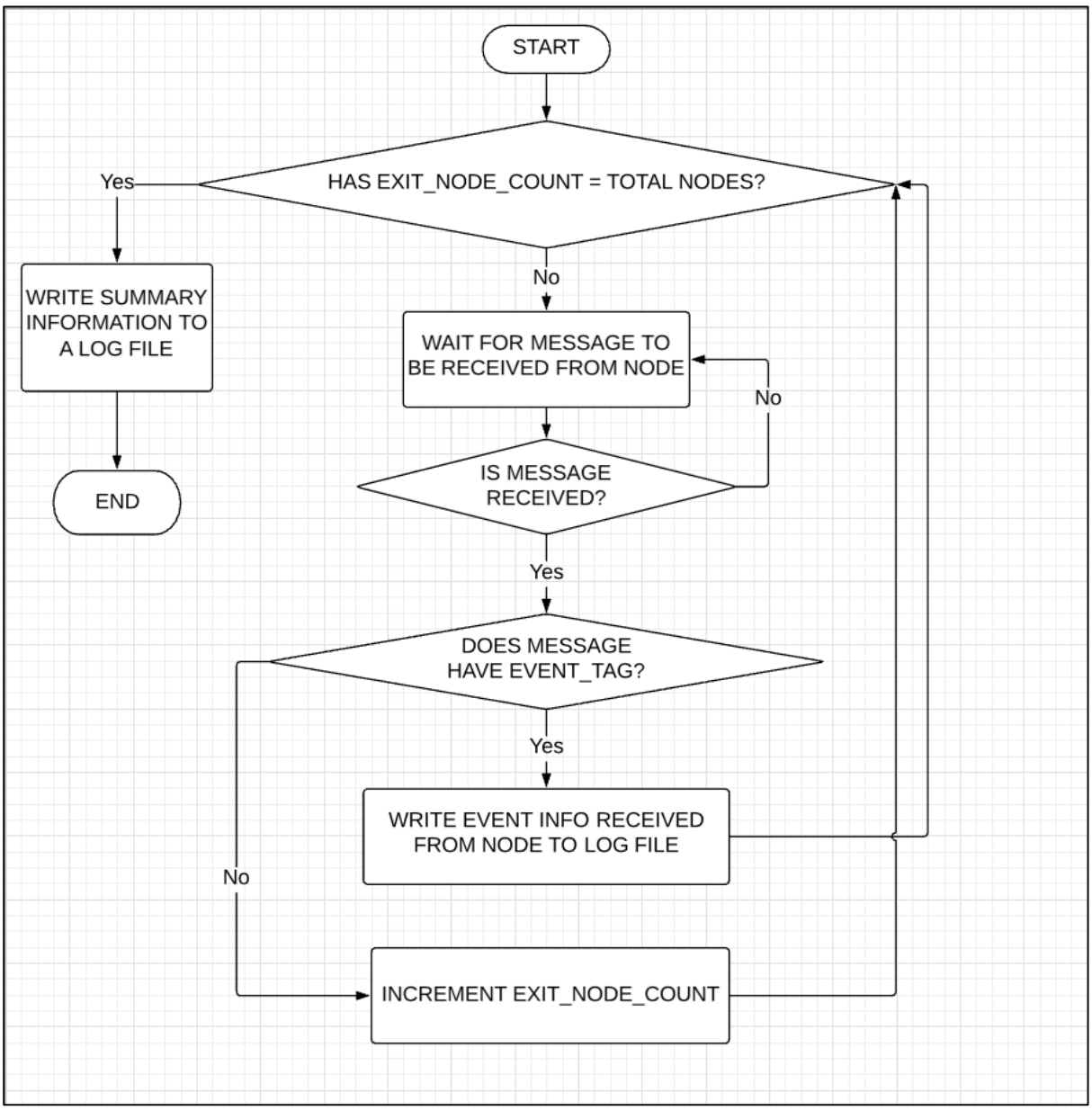
    a. **Iteration Number** – at which iteration the event occurred
    b. **Rank** – rank of the node at which the event occurred.
    c. **Messages Sent** – total number of messages passed during that iteration to adjacent nodes.
    d. **Adjacent Nodes Involved** – all the adjacent nodes which generated the same random number to make the event happen.
    e. **Time Stamp of the event**

9. All of this information is packed into a **Struct** datatype and sent to the base station. Rather than sending each piece of information separately using MPI_Send which would have resulted in 4 messages being sent to the base station, each node only sends one message to the base station using the **Struct** datatype. **This is the second strategy I used to minimize messages being passed.**
10. For sending to the base station, I am using MPI_Send which is a blocking type communication. I am using this because there is no chance of any deadlock to occur because the base station will always be available to receive from all nodes until and unless all nodes exit.
11. If no event occurred, then all the work for the current iteration has ended, so MPI_Wtime () is called to record the **end time**.
12. Then the time elapsed is calculated using the formula below:

    Time Elapsed = **End Time** – **Start Time**

13. If Time Elapsed is less than the Preset Time, then the program is brought to a sleep for (Preset Time – Time Elapsed) seconds.
14. If all iterations are over, each node sends an exit message to the base station with an **EXIT TAG**.

IV. How the system works from the point of view of the base station



V. Explanation of the above flowchart

There is one base station which has a rank of 0. The base station keeps track of the following information:

- **Exit Count** – total number of node's which have exited already.
- **Total Events** – total number of events that have occurred.
- **Total Messages sent to Base Station Sent due to events**
- **Total Messages sent to Base Station due to exiting of nodes**
- **Total number of messages exchanged between adjacent nodes**

1. Base Station can receive two types of messages from nodes, one messages is regarding an event and another message is regarding a node exiting. Message regarding an event has an **"EVENT TAG = 0"** and message regarding exit has an **"EXIT TAG = 1".**
2. For receiving on the base station side, I am using a normal blocking receive because there is no chance of deadlock.
3. If the message is regarding an event, then the base station does three things:

    a. Increments **"Total Events"** counter by 1.
    b. Increments the "**Total Messages sent to Base Station Sent due to event"** counter by one.
    c. Writes the following information to the log file:

        i. Iteration number.
        ii. Reference node where the event has occurred.
        iii. Adjacent nodes involved in the event.
        iv. Time Stamp of the event

4. If the message is regarding an exit of a node, then the base station does two things:
    a. Increments "**Exit Count"** counter by one**.**
    b. Increments **"Total number of messages exchanged between adjacent nodes"** counter**.**
5. **I**f all nodes have not exited, then the same steps are repeated from 1.
6. If all nodes have exited, then the base station writes the following summary information to the log file:

    a. Total number of events that occurred
    b. Total Messages sent to Base Station Sent due to events
    c. Total Messages sent to Base Station due to exiting of nodes
    d. Total number of messages exchanged between adjacent nodes

VI. Statistics regarding count of messages being passed in the system

RANK 1 NODE (Sends 2 messages to Rank 6 and Rank 2)	***RANK 2 NODE*** (Sends 2 messages to Rank 3 and Rank 7)	***RANK 3 NODE*** (Sends 3 messages to Rank 2 and Rank 4 and Rank 8)	***RANK 4 NODE*** (Sends 2 messages to Rank 3 and Rank 9)	***RANK 5 NODE*** (Sends 2 messages to Rank 4 and Rank 10)
RANK 6 NODE (Sends 2 messages to Rank 7 and Rank 11)	***RANK 7 NODE*** (Sends 4 messages to Rank 2, Rank 6, Rank 8 and Rank 12)	***RANK 8 NODE*** (Sends 4 messages to Rank 3, Rank 7, Rank 13 and Rank 9)	***RANK 9 NODE*** (Sends 4 messages to Rank 4, Rank 8, Rank 10 and Rank 14)	***RANK 10 NODE*** (Sends 2 messages to Rank 9 and Rank 15)
RANK 11 NODE (Sends 2 messages to Rank 6 and Rank 12)	***RANK 12 NODE*** (Sends 4 messages to Rank 7, Rank 11, Rank 13 and Rank 17)	***RANK 13 NODE*** (Sends 4 messages to Rank 8, Rank 14, Rank 12 and Rank 18)	***RANK 14 NODE*** (Sends 4 messages to Rank 9, Rank 19, Rank 13 and Rank 15)	***RANK 15 NODE*** (Sends 2 messages to Rank 10 and Rank 14)
RANK 16 NODE (Sends 2 messages to Rank 11 and Rank 17)	***RANK 17 NODE*** (Sends 2 messages to Rank 12 and Rank 18)	***RANK 18 NODE*** (Sends 3 messages to Rank 13 and Rank 17 and Rank 19)	***RANK 19 NODE*** (Sends 2 messages to Rank 14 and Rank 18)	***RANK 20 NODE*** (Sends 2 messages to Rank 15 and Rank 19)

Let's say there are in total "N" number of iterations. For each iteration "i" in the system, the following facts are true for my program:

1. The number of messages that each WSN node sends can be seen from the table above.
2. Total number of messages sent between WSN nodes can be summed up from the table above to get the total count to be equal to **54** messages.

Hence for "N" iterations, total number of messages exchanged between WSN nodes are equal to N * 54 = **54N** messages.

Let's say there are **"X"** number of events that happened in a certain simulation run, then the total number of messages sent by all the nodes to the base station would be **"X"**. Hence for each event only **"1"** message is sent to the base station.

Total Number of exit messages sent for any simulation run is always equal to the number of WSN nodes in the system, which in our case is 20. So each WSN node upon exiting sends "1" exit message to the base station. So in total, **"20"** exit messages are sent to the base station.

Therefore, for a particular simulation run consisting of "N" iteration's, "X" number of events occurring, 20 WSN nodes:

Total number of messages passed through the system = 54N + X + 20,

- **54N** messages are passed between WSN nodes
- **X + 20** messages are between WSN nodes and base station.

VII. MPI_FUNCTIONS USED

MPI FUNCTIONS USED	REASON FOR USING
MPI_Init ()	For initializing the MPI environment
MPI_Comm_size ()	To get the size of the communicator
MPI_Comm_rank ()	To get the rank of the process
MPI_Wtime ()	To get the elapsed time on the calling processor.
MPI_Waitall ()	To ensure all messages have been sent and received for non-blocking communication.
MPI_Send (), MPI_Recv ()	For sending and receiving messages using blocking scheme.
MPI_Isend (), MPI_Irecv ()	For sending and receiving messages using non-blocking scheme.
MPI_Finalize ()	To safely exit the MPI environment.
MPI_Type_create_struct ()	To create custom datatype so that I can sent Struct.
MPI_Type_commit ()	To finalize the creation of the newly created datatype before using it in communication.

QUESTION - 2

I. Brief overview of associative memory and HGN

HGN is made out of layers of GN systems which are organized in a pyramid-like structure. The base layer relates to size of the pattern to be used in the pattern-recognition example. The size of the pattern is equivalent to the quantity of components in the pattern. Every component might be doled out a settled number of conceivable qualities from the pattern domain.

HGN calculation is like the straightforward GN calculation, except for the various leveled structure (hierarchical). The advantage of having this structure is that the best GN hubs can give oversight to the base layer, and in this manner kill the cross-talk. Each layer of the HGN goes about as a straightforward GN coordinate with the distinction that the higher layers (all layers aside from the base layer) would store the match esteems as p(index_left, index_middle, index_right).

In **[1]**, HGN is implemented using associative memory.

Associated memory is a kind of PC memory from which things might be recovered by matching some piece of their content, as opposed to by determining their address

To recover a word from associative memory, a search key (or descriptor) must be introduced that represents specific values of all or a portion of the bits of the word. This key is compared in parallel and the comparing lock or label bits of all stored words, and all words matching this key are motioned to be accessible.

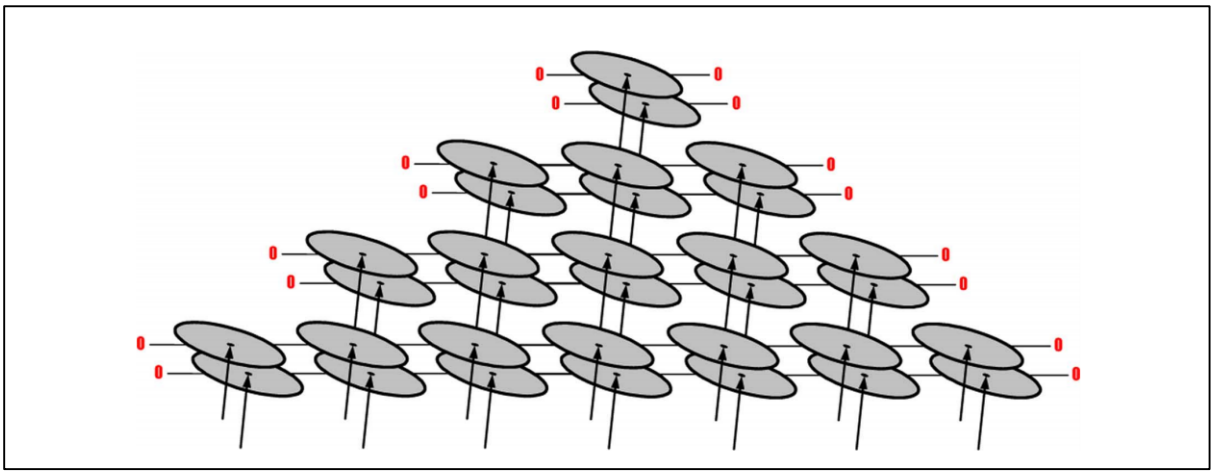
II. Pseudocode

This is a high level pseudocode describing how the HGN communication scheme works. Some of the specific details regarding the communication is skipped from the pseudocode.

```
layer = base_layer

while(layer != top_layer){

    for column_i_GNs in base_layer:
        data = stimulator.create_data()         //p(value,command)
        broadcast(data,column_i_GNs)

    mydata;
    previous_column_recv_data;
    next_column_recv_data;
    for GN in base_layer:
        recv(mydata,stimulator)
        prev_column_GN = GN.prev_column()
        next_column_GN = GN.next_column()
        send(mydata,prev_column)
        send(mydata,next_column)
        recv(previous_column_recv_data,prev_column_GN)
        recv(next_column_recv_data,next_column_GN)

    bias_entry = calculate_bias_entry(previous_column_recv_data,next_column_recv_data,
    mydata)

    if(bias_entry exists in associative memory){

        //GNs in base layer read index of bias

    }

    else{

        if(command is to store){
            //store bias in associative memory
        }
    }

    for GN in active_GNs:

        data = (GN.row,GN.bias_index,GN.command)
        my_column = GN.column
        broadcast(data,my_column)

    for GN in active_GNs:

        data = (GN.level,GN.column,GN.bian_index)
        send(data,stimulator)

    layer = layer.next()
}
```

III. Suggested Parallel Architecture and the reasons behind it

I would suggest using a SIMD architecture which uses a distributed associative memory. Firstly, graph neurons in HGN are visualized as having the following structure:



It is seen that all the neurons have a pyramid style connectivity.

1. As learnt from the week-10 lecture, we can use a SIMD architecture where processors can be arranged in a pyramid style manner. Here the graph neurons can basically be represented as each processor. Each processor is connected in a pyramid-style manner
2. As seen from the pseudocode above on how the HGN communication works, each neuron needs to store a bias entry in their memory. That's why I have chosen to use distributed memory where each processor will have its own local memory.
3. I have chosen the local memory to be associate because associative memory works very fast in practice for searching or look-up of some data.
4. Also as seen from the pseudocode above, neurons need to communicate with other neurons. This can be achieved using a network which is a key component of a distributed memory system.
5. Also using a SIMD architecture, our single instruction will be the "send" instruction and the multiple data will be the bias entry of each neuron. So using the SIMD architecture, each neuron can in parallel send their bias entry to their neighbours through networks.

References

[1] B. Nasution and A. Khan, "A Hierarchical Graph Neuron Scheme for Real-Time Pattern Recognition", *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 212-229, 2008.