Name : Calum Lim Sheng En
ID : 27372537
Title : Assignment 2: Part A – Report & Part B

Part A - Report

1) Program Structure

This assignment requires us to make use of the Message Passing library known as OpenMPI, to construct a grid in order to simulate a wireless sensory network(WSN). The assignment specification tells us that the grid is required to be 4x5 and that the nodes within the grid can only communicate adjacently. For each node within the grid, there will be a random number assigned to them. This random number will then aid the criterion specified by the assignment specification, it states that at any node, if there are at least three or more nodes with the same random number assigned, then it counts as an event. An event can then be sent to a base station which role is to solely receive messages from the grid. Therefore, once the base station knows that there is an event, it can then write the relevant information about the event into a log file.

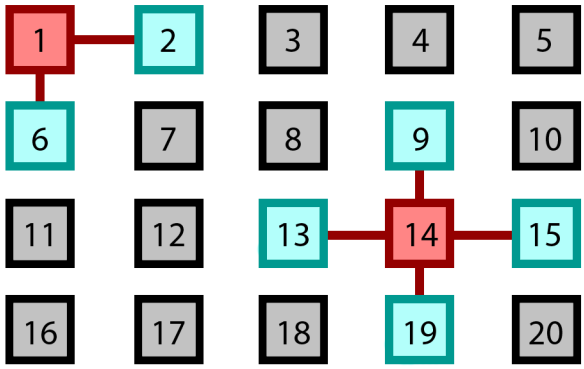

Figure 1, visual representation of the WSN grid with coloured adjacent and reference nodes.

The design of the grid is simple enough, 4 rows together with 5 columns. With each grid, each process occupies a space, therefore there will be 20 processes which will occupy the entire grid.

Sensor nodes: Each node in the grid represents both a reference node as well as an adjacent node. For example, figure 1 shows us that “1” is a reference node to “2” and “6”, but that also means that “1” can be considered as an adjacent node for “2”. This logic is then applied throughout all the nodes present in the grid. Since the node represents both a reference node as well as an adjacent node, this means that every node in the grid will have to act as both a sender and a receiver of messages in an MPI system.

Sending random values:

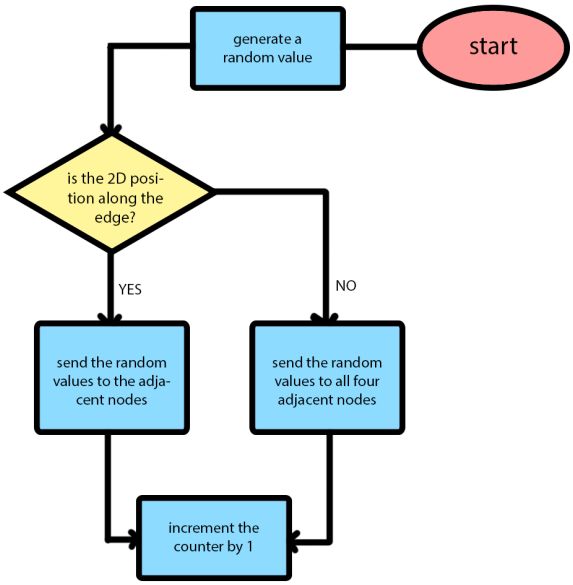

Figure 2, a flowchart demonstrating the sending procedure.

Before the node can send anything, it needs to generate a random number first. For example, "rand()%5" can be used so that the chance of 3 adjacent nodes having the same number is higher. After it generates a random number, it then checks whether it is at the edge of the grid, because there are only three or lesser adjacent nodes when the reference node is at the edge of the grid. If the reference node is at the edge of the grid, then the random numbers are only sent to three or lesser adjacent nodes depending on its position. If the reference node is not at the edge, then it sends its generated random values to all four of its adjacent nodes. After initiating all the non-blocking sending requests, the counter is then incremented for each send request.

Receiving random values:

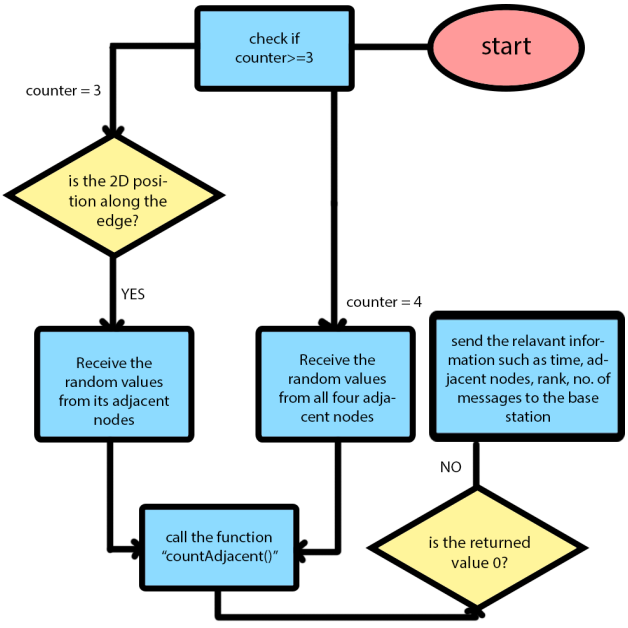

Figure 3, a flowchart demonstrating the receiving procedure.

After sending the random values, the program then checks to see if the counter is more or equals to 3. If the counter is 3 or more, that means that there are at least 3 messages sent out from a reference node, which also means that there could be a chance that this reference node will meet the criterion. To be sure that it meets the criterion, the program calls upon a predefined function called countAdjacent() which uses an array initialized with 0s to calculate the total number of occurrences for each random number. After finding the number of occurrences, the function then iterates through the array to find any occurrences which are equals to 3 or more. If there is none, then return 0; but if there is a value of 3 or more, the function then calculates all the adjacent nodes of the current reference node and stores all the relevant information into an array which will be returned. Finally, the program checks if the returned value is 0 or not. If the value returned is not 0, that means the criterion has been met and the array containing the relevant information can then be sent to the base station. After sending to the base station, the program also sends an exit message to the base station to notify the base station that the program has finished its sending procedures.

Base station:

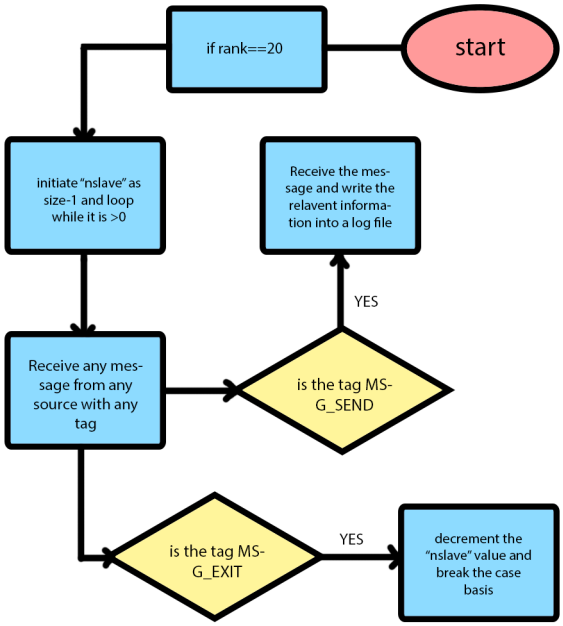

Figure 4, a flowchart demonstrating the base station's procedure.

The base station acts as a universal receiver, whenever the criterion is met for a reference node, its relevant information is sent to the base station. Therefore, the base station's role is to receive the information then write it into a log file. The program uses the master and slave concept; the base station is considered the master because it only receives. The base station receives any message from any source with any tag into a buffer, after that a switch case is initialized to check if to end or to write to the log file. The type of messages includes integers and double values.

2) Inter Process Communication Scheme

Adjacent nodes in a WSN.

For the program to check the adjacent node's boundaries, the program has to calculate the adjacent node's 2D position in order to access the grid. To do that, the program uses the formulas: position of row = rank/no. of columns, position of column = rank%no. of columns. By having the position of both row and column, the program can then check the boundaries by simply adding or subtracting 1 from the 2D position. For communication between the adjacent nodes using sending and receiving functions, the program has to obtain the adjacent nodes' 1D value (rank). To do that, another formula is used: rank position = position of column + (no. of columns/position of row); which can be similarly adjusted to obtain the adjacent nodes' 1D values. After that, messages can be sent to the adjacent nodes using their 1D positions as an argument to a non-blocking send function. Because this program uses non-blocking send to send its messages, a MPI_Waitall() function needs to be present in order to prevent a deadlock from occurring. For each non-blocking send function called in the program, there is an equal amount of blocking receiving function present. The receiving function facilitates the random value being received by the reference node, it then stores it into an array depending on its adjacent position. Other than that, the minimum number of messages that can be passed in this program is "2" which occurs when the process is at either rank "1, 5, 16, 20", because there are only 2 available adjacent nodes to exchange messages with. On the other hand, the maximum number of messages that can be passed in this program is "4" which occurs when the node has four adjacent nodes to communicate with.

Node and Base station.

This program uses the master and slave concept in order to delegate its tasks. The slave function will compute the 1D to 2D to 1D mapping, send the random numbers, receive the random numbers from the adjacent nodes, determine if there is an event, send information to the base station. On the other hand, the master function will only receive any information from the slave function and write it into a log file. The program uses MPI_Comm_split to split the master and slave function. Other than that, the number of messages that are exchanged between the reference node and the base station given an event occurrence is, "3". The first message will contain the information about the triggered event such as: the adjacent nodes which triggered the event, the rank of the reference node, and etc. The second message will contain the time stamp for when the event was triggered, this data is exchanged in the form of a float value. The last message will contain the number of requests at the time of event occurrence, which will give the program enough data to compute the number of messages exchanged up to the event and the number of nodes which triggered the event.

3) Compilation and usage instructions

OpenMPI must be installed before executing this program.
Compilation command line:

Linux:
mpicc –o wsn wsn.c
mpirun –np 21 wsn

MacOS:
mpicc –o wsn wsn.c
mpirun --mca shmem posix --oversubscribe –np 21 wsn

Cluster File:
mpicc –o wsn wsn.c
mpirun --hostfile clusterfile.dms wsn

Results from log file:

```
=========================================================
Reference Node: 13
Adjacent Nodes: (8,18,12,14)
Random Number: 3
Time Spent: [ 0.000398s ]
Messages Exchanged: 7
No. of Nodes which triggered the event: 4
Total no. of events that have occurred: 1217

=========================================================
Reference Node: 11
Adjacent Nodes: (6,16,10,12)
Random Number: 1
Time Spent: [ 0.000338s ]
Messages Exchanged: 7
No. of Nodes which triggered the event: 4
Total no. of events that have occurred: 1218

=========================================================
Reference Node: 11
Adjacent Nodes: (6,16,10,12)
Random Number: 1
Time Spent: [ 0.000005s ]
Messages Exchanged: 7
No. of Nodes which triggered the event: 4
Total no. of events that have occurred: 1219
```

These results are taken from three of the last events from the log file.

Part B

According to the paper, the hierarchical graph neuron (HGN) implements an associative memory which is scalable. Also, this scheme allows noisy patterns to be detected without the issue of crosstalking. With this approach there are many improvements to the previous attempts such as, crosstalk being resolved, scalable architecture and memory, as well as real-time application support.

A parallel processing architecture that will best suit the HGN associative memory technique could be to have a MISD implementation. This could be suitable as the MISD implementation makes use of systolic arrays where its input is inserted parallel using a series of interconnected communication networks.

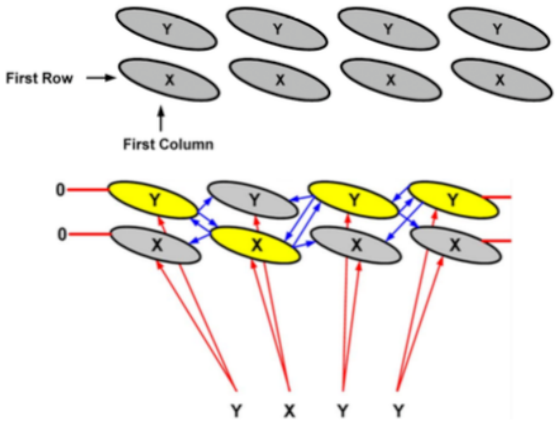

Figure 5, a representation of a GN network array and its response for an activity (Nasution & Khan, 2008).

One of the reasons that this architecture could be suitable for this network is due to the interconnectivity of each node array. This means that if a node reacts from the first matching value, then the signal is relayed to the next column in this network. Using this, the program will then determine which node will react to the next input given the matching value.

Another reason is that the MISD model, in its parallel context aids problems that are attributed or specialized by a higher degree of regularity such as recognition of patterns. This can help the normal graph-based method that does real-time pattern recognition as there can be a bottleneck when it is faced with larger pattern databases, thus increasing its computational efforts and minimizing efficiency. Another benefit for using the MISD architecture is that it utilizes a 2D array that models the processor array which is utilized by the HGN, because at the basic level, the HGN uses a 2D array for its approach.

Since the MISD architecture utilizes 2D arrays, we can assume that arrays will not increase the overall space complexity of a program but rather decrease it. From that, we can also conclude that the use of arrays will contribute positively to the efficiency and speed of completion of the process.

Lastly, another benefit of using the MISD architecture is that it uses synchronous execution which benefits the overall approach used by the HGN. In this scenario, when a value matches with a given GN's ID, it will then broadcast a report to all the other GN's that have the indexes of the row and column. Thus, all of the GNs are synchronized after each match (Nasution & Khan, 2008).

References

Nasution, B. B., & Khan, A. I. (2008). *A Hierarchical Graph Neuron Scheme for Real-Time Pattern Recognition*. Retrieved October 7, 2018, from https://ieeexplore-ieee-org.ezproxy.lib.monash.edu.au/stamp/stamp.jsp?tp=&arnumber=4359217