MONASH University

# ASSIGNMENT COVER SHEET

| | | | |
|---|---|---|---|
| **Student's name** | (Family name)<br>Ong | (Given names)<br>Jing Wei | |
| **ID number** | 32909764 | **E-mail** | jong0074@student.monash.edu |
| **Unit code & name** | FIT3143 Parallel Computing | **Unit code** | FIT3143 |

| | |
|---|---|
| **Title of assignment** | **Distributed Wireless Sensor Network** |
| **Lecturer/tutor** | Vishnu Monn, Leong Shu Min |
| **Is this an authorised group assignment?** ☐ **Yes** ☒ **No**<br>If this submission is a group assignment, each student must attach their own signed cover sheet to the assignment. | |
| **Has any part of this assignment been previously submitted as part of another unit/course?** ☐ **Yes** ☒ **No** | |
| **Tutorial/laboratory day & time** | Laboratory 05, 4pm-6pm |
| **Due date** 13 October 2023 | **Date submitted** |

All work must be submitted by the due date. If an extension of time to submit work is required, a Special Consideration Application (In-semester Assessment Task) must be submitted.

**Has an extension been approved?** **Yes** ☐ **No** ☐ **If yes, please give the new submission date ……./……/…….**

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

**Student Statement:**

- I have read the University's Student Academic Integrity Policy and the University's Student Academic Integrity: Managing Plagiarism and Collusion Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Assessment and Academic Integrity Policy
- I have taken proper care of safeguarding this work and made all reasonable effort to ensure it could not be copied.
- I acknowledge that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  i. provide to another member of faculty; and/or
  ii. submit it to a plagiarism checking service; and/or
  iii. submit it to a plagiarism checking service 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.

*Jingwei* *10 October 23*

*Signature* ..... *Date*.....

**Privacy Statement**

The information on this form is collected for the primary purpose of assessing your assignment. 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 you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

MONASH University

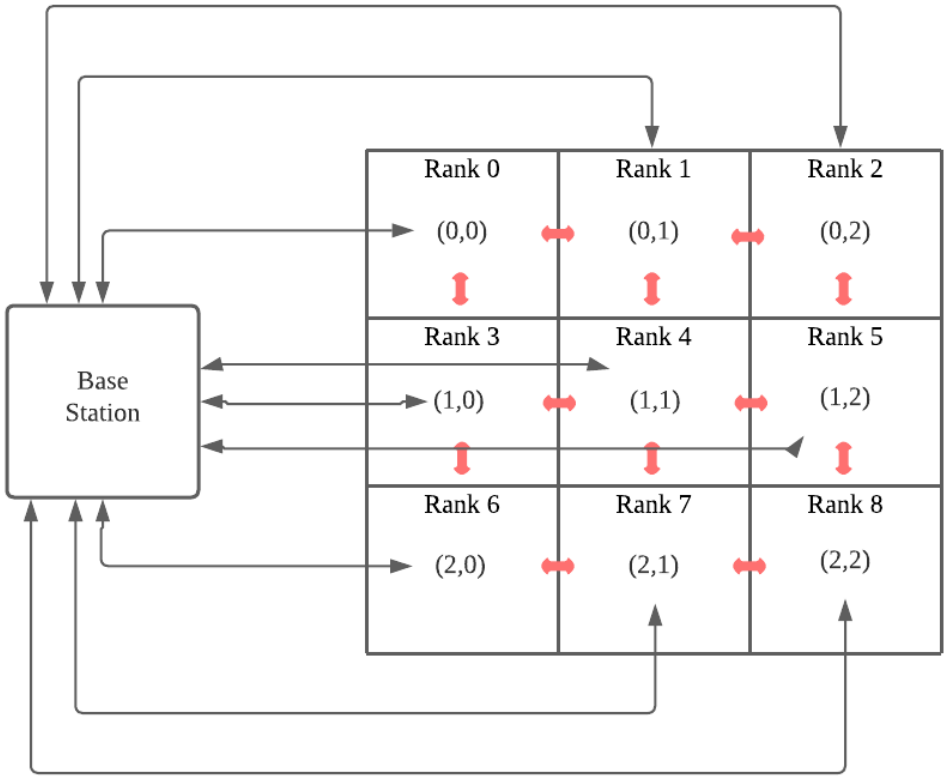
**FIT3143 Semester 2, 2023**
**Assignment 2 – Report**
***Title – Specify your tile based on the Assignment Specifications***

Include the word count here (for Sections A to C): 1489

## A. Methodology

In this assignment, the simulated wireless sensor network of a set of interconnected EV charging nodes is implemented with a virtual topology in the form of a 2D Cartesian grid using the Message Passing Interface (MPI) and POSIX threads for multi-threading within each charging node. The simulation offers a practical and scalable solution for monitoring and managing EV charging ports in real-time.

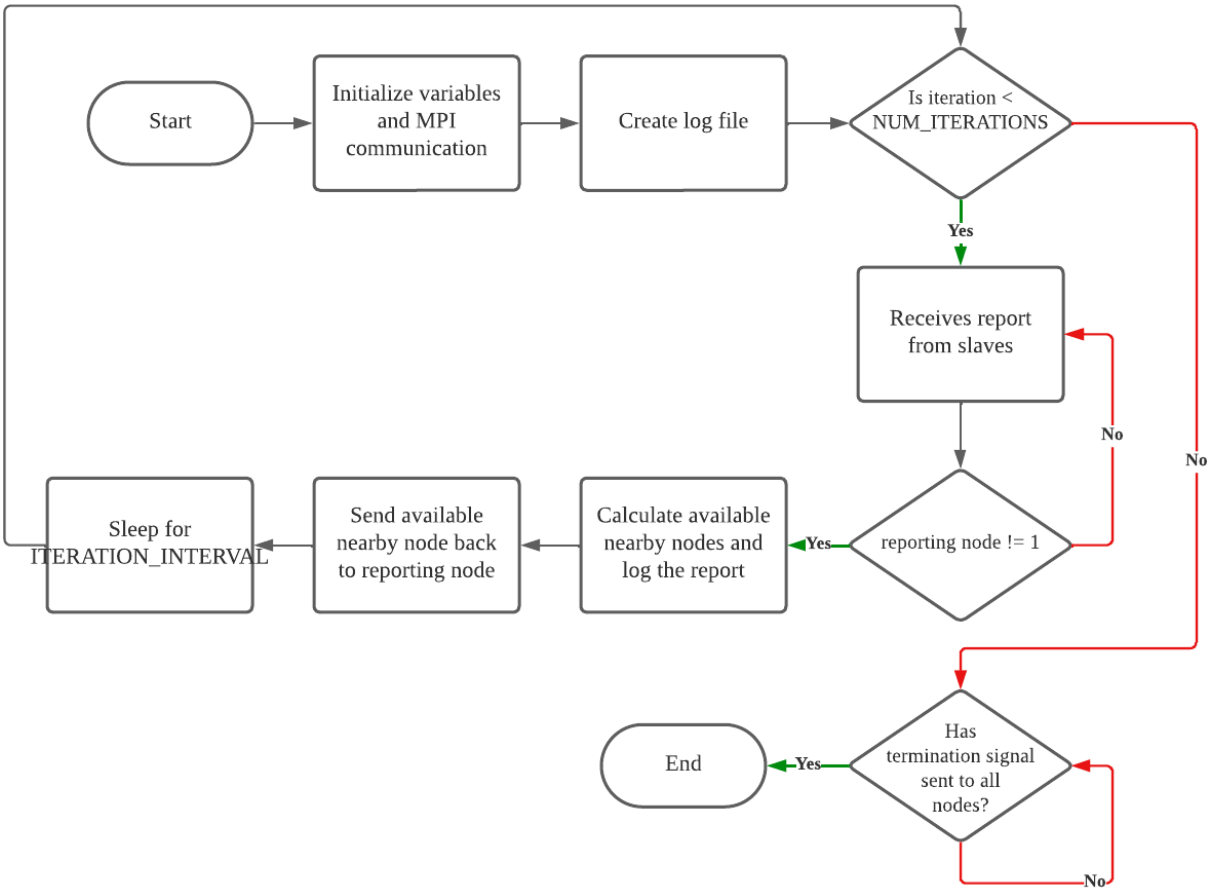
The diagram below shows how the base station communicates with each of the EV charging nodes:




In the main function, we divide the process in MPI_COMM_WORLD into two sets, which are the master_io (base station) and the slaves_io (EV charging node) using MPI_Comm_split. We are going to investigate these two subtasks below in detail.

## 1. Base Station

The base station serves as the central control unit in the wireless sensor network simulation as it receives reports from the EV charging node and checks for the nearest available neighbour nodes for that reporting node. The flow of how the base station works is shown in the flow chart below:




a. Initialization of MPI environment

Base station initializes the MPI environment, retrieves its rank, and sets up communication channel. It also opens a log file for recording simulation data.

```
function master_io(master_comm, comm) {
        Initialize file mFile
        Initialize buf
        Sprint(buf, “BaseStation_log.txt”)
        Open mFile with “w” mode
        MPI_Comm_size(master_comm, &size)
        Set up communication parameters
        Create necessary variables and data structures
        …
```

*}*

b. Data collection

***for*** *each iteration in 0 to NUM_ITERATIONS:*
***for*** *each 'i' from 1 to size:*
*MPI_Recv() receive NodeReport data from slaves into node_report[i-1]*

In each iteration, the base station receives reports from all EV charging nodes using MPI_Recv. These reports include information about the charging nodes' status, such as availability and the number of port values in that node. The neighbour nodes and nearby nodes of that reporting node are calculated as follows:

Calculation for neighbour nodes:

```
int top = (node_report[i].coordx > 0) ? reporting_node - ncols : -100;
int bottom = (node_report[i].coordx < nrows - 1) ? reporting_node + ncols : -100;
int left = (node_report[i].coordy > 0) ? reporting_node - 1 : -100;
int right = (node_report[i].coordy < ncols - 1) ? reporting_node + 1 : -100;
```

For example, we calculate the top neighbour by checking if the x-coordinate of the reporting node is greater than 0. If it is greater than 0, then we subtract the reporting node rank from the number of columns to get the rank of neighbour nodes.

```
neighbour_coords[j][0] = j / ncols;
neighbour_coords[j][1] = j % ncols;
```

This calculation is used to compute 2D coordinates (i,j) from a 1D rank index.

Calculation for nearby nodes:

```
int nearby_nodes[8] = {
    top-ncols, top-1, top+1, left-1, right+1, bottom-1, bottom+1, bottom+ncols
};
```

We take the neighbour nodes that we calculated above and compute for the nearby nodes.

```
int cx = nearby_nodes[k] / ncols;
int cy = nearby_nodes[k] % ncols;
if (cx >= 0 && cx < nrows && cy >= 0 && cy < ncols)
```

Before logging the nearby nodes data into the log file, we check that if the nearby nodes are correctly calculated.

c. Logging

The base station records the information received from the EV charging node, including the number of iterations, timestamps, neighbour nodes, and nearby nodes in a log file.

d. Communication

MPI communication function such as 'MPI_Isend' is used to send back the available nearby nodes to that reporting node.

*MPI_Isend(&available_nearby_nodes, 1, MPI_INT, reporting_node, 0, MPI_COMM_WORLD, &req)*

e. Termination Signal

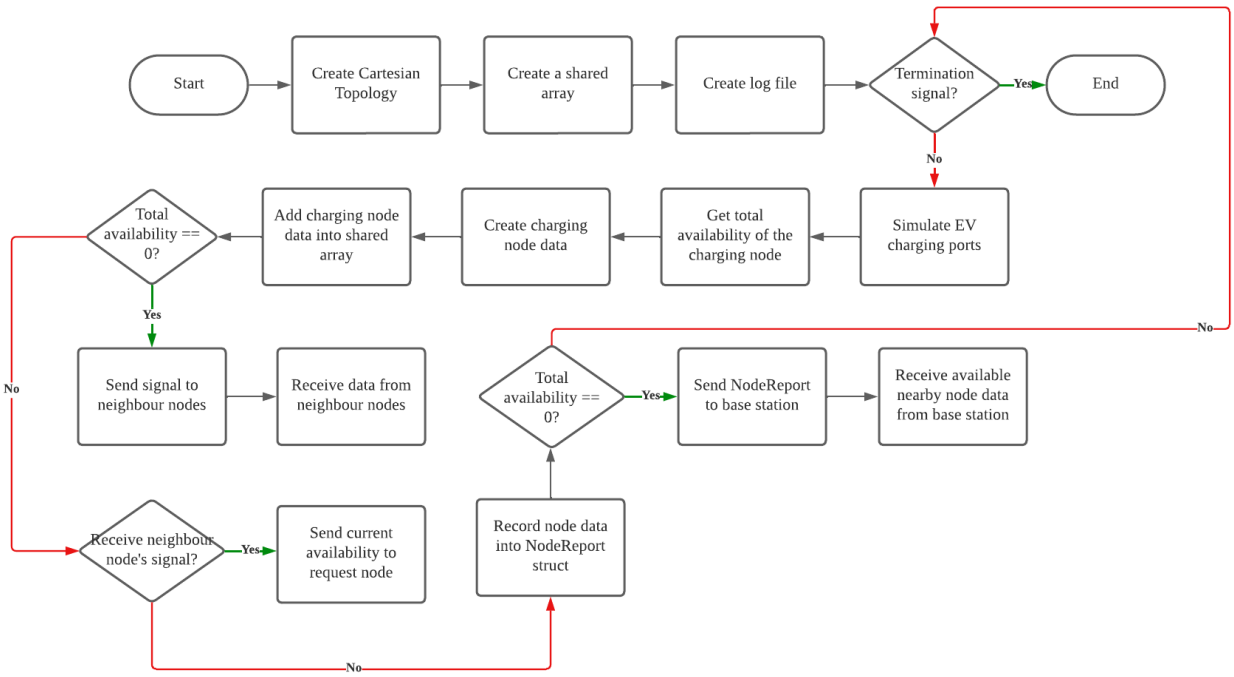
'MPI_Isend' is used to send the termination signal to all the EV charging nodes after a fixed number of iterations when the simulation completes.

*Initialize termination_signal to 1*
***for** each 'node_rank' from 1 to size-1:*
*MPI_Isend(&termination_signal, 1, MPI_INT, node_rank, 0, MPI_COMM_WORLD, &send_request[node_rank-1])*
*Wait for all send requests to complete*

MPI_Isend() is used as it is a non-blocking send operation, it allows us to continue executing other code while the message is being sent in the background as to improve the overall performance and responsiveness of the program.

## 2. EV Charging Nodes

EV charging nodes represent wireless sensor network nodes that provide charging services to electric vehicles. Each node runs a simulation of charging ports and communicates with its neighbouring nodes. The flow chart below outlines how the EV charging node works:



a. Initialization and setup

Charging nodes initialise MPI, identify their ranks, and set up communication channels within the Cartesian grid topology. It also creates threads to simulate the behaviour of individual charging ports using Pthreads. Each charging node opens a log file to record its data.

```
function slave_io(master_comm, comm2D) {
    my_rank, size = MPI_Comm_rank(comm2D), MPI_Comm_size(comm2D)
    create a file sFile for logging

    Create a 2D Cartesian Topology;
    ndims=2, reorder=1, ierr=0
    define dimensions and wrap_around array
    create cart_comm with MPI_Cart_create

    determine process coordinates in the Cartesian communicator:
    MPI_Cart_coords(cart_comm, my_rank, ndims, coord)

    Determine neighbour processes:
    MPI_Cart_shift(cart_comm, SHIFT_ROW, DISP, nbr_i_lo, nbr_i_hi)
    MPI_Cart_shift(cart_comm, SHIFT_COL, DISP, nbr_j_lo, nbr_j_hi)
    Create an action_list with neighbour ranks

    Initialize stationData and open sFile for writing
    …
}
```

Cartesian topology abstracts away the underlying complexities of managing communication between processes. It is used in slaves as it provides a natural and logical way to represent the relationships between the EV charging nodes. It allows processes to identify their neighbours based on their grid coordinates. This is essential when processes need to communicate or share data with nearby neighbours. It simplifies the communication between neighbouring nodes as we can easily determine the rank and coordinates of the neighbour node by using MPI communication functions such as MPI_Cart_coords and MPI_Cart_shift.

b. Charging port simulation

'Pthread_create' calls the 'simulateChargingPort' function. This function simulates the availability status of charging ports, where 0 or 1 indicate in-use or available. The port availability is updated periodically with each port running in its own thread. 'pthread_mutex_lock' is to ensure thread safety when updating the port availability.

```
pthread_create(&ports[i].tid, NULL, simulateChargingPort, &ports[i])

function simulateChargingPort(port) {
    lock the port's mutex
    port.availability = randomly_choose(0 or 1)
    unlock the port's mutex
}
```

Using POXIS threads to stimulate charging ports as it allows charging ports to operate independently of each other, and each port can simulate its availability updates

simultaneously. By using pthreads, we can achieve parallelism, which improves the efficiency and realism of the simulation. The mutex is used to lock and unlock the port data when updating its availability. This ensures that multiple threads do not access and modify the port data simultaneously. Thread safety is to prevent race conditions.

c. Data collection

Each charging node collects data about its own total availability and the availability of its neighbouring nodes. It will send a signal to the neighbouring nodes once the total available port in that node is 0, which means that all the ports are full. Hence, for every node, they also needed to constantly receive signals from the neighbouring nodes. When the neighbour node receives the signal, it will send back its availability to that request node. The data on the node's availability in each iteration is stored in a circular array for recording.

```
function slave_io(master_comm, comm2D) {
    …
    Initialize ChargingNodeData struct
    Calculate total availability of charging ports
    Store the node data into the circular array

    If totalAvailability is 0:
            Set request_flag as 1
            Loop over action_list:
                    Initiate non-blocking sends with MPI_Isend to neighbour processes
                    Initiate non-blocking receives with MPI_Irecv from neighbour
                    processes
            Wait for all non-blocking sends to complete

            Loop over received data:
                    If the availability received is not -1, record neighbour availability

    Loop over action_list:
            Use MPI_Iprobe to check for incoming signals from neighbour
            If received incoming signals:
                    Initiate non-blocking receives with MPI_Irecv
                    Initiate non-blocking receives with MPI_Isend to send the total
                    availability
    …
}
```

MPI_Isend and MPI_Irecv are used here as we can initiate multiple communication operations simultaneously, reducing the time it takes for the data to be exchanged between processes. Next, non-blocking communication provides flexibility in terms of handling incoming and outgoing messages. Our programme can continue to check for other messages while the communication operations are in progress. This can lead to more efficient and responsive simulations.

d. Reporting

After getting the total number of available ports in that node, if all ports are full, the node will send a signal to neighbouring nodes and get their availability. Hence, for every node, they also needed to constantly receive signals from the neighbouring nodes.

Once they receive the signal, the neighbour nodes will send back its availability. Next, the charging node with full ports also needs to send a report to the base station.

```
function slave_io(master_comm, comm2D) {
    …
    Create a NodeReport struct with local data
    If totalAvailability is 0:
            Send NodeReport to base station with MPI_Send
            Receive nearby node data from base station with MPI_Irecv
    …
}
```

```
typedef struct {
    int reporting_node;
    time_t time;
    int coordx;
    int coordy;
    int num_port;
    int port_available;
    int neighbours[4];
} NodeReport;
```

Screenshot above is the NodeReport struct where:

| | |
|---|---|
| Reporting_node | : the rank of that node |
| Time | : the time when the node is sent |
| Coordx | : x-coordinate of that node |
| Coordy | : y-coordinate of that node |
| Num_port | : total number of ports in that node |
| Port_available | : total number of available ports in that node |
| Neighbours | : array which stores the neighbour rank |

e. Termination signal

Charging nodes continuously check for a termination signal sent by the base station. If received, they clean up and terminate the simulation process.

```
function slave_io(master_comm, comm2D) {
    …
    Terminate charging ports and join the threads
    Test for termination signal with MPI_Test
    …
}
```

MONASH University

## 3. Results Tabulation

Tested on local machine:

Scenario 1: Test with different number of iterations in base station

| Number of iterations | Grid Size | Number of Ports | Number of reported messages |
|---|---|---|---|
| 2 | 3x3 | 3 | 2 |
| 3 | 3x3 | 3 | 2 |
| 4 | 3x3 | 3 | 1 |

Screenshot of base station log in 3 iterations:

```
--------------------------------------------------------------------------------
Iteration           : 0
Logged time         : Sun Oct 15 22:40:17 2023

Alert reported time : Sun Oct 15 22:40:23 2023

Reporting Node     Coord     Port Value     Available Port
0                  (0,0)     3              0

Adjacent Node      Coord     Port Value     Available Port
1                  (0,1)     3              1
3                  (1,0)     3              1

Nearby Nodes       Coord
2                   (0,2)
2                   (0,2)
4                   (1,1)
6                   (2,0)

--------------------------------------------------------------------------------
Iteration           : 1
Logged time         : Sun Oct 15 22:40:17 2023

Alert reported time : Sun Oct 15 22:40:26 2023

Reporting Node     Coord     Port Value     Available Port
1                  (0,1)     3              0

Adjacent Node      Coord     Port Value     Available Port
0                  (0,0)     3              1
2                  (0,1)     3              2
4                  (1,0)     3              3

Nearby Nodes       Coord
3                   (1,0)
3                   (1,0)
5                   (1,2)
7                   (2,1)
```

Scenario 2: Test with different number of ports in base station

| Number of ports | Grid Size | Number of iterations | Number of reported messages |
|---|---|---|---|
| 3 | 3x3 | 3 | 2 |
| 6 | 3x3 | 3 | 0 |

| | | | |
|---|---|---|---|
| 9 | 3x3 | 3 | 0 |

Screenshot of charging node 0 log file with 6 ports:

```
Global rank: 0. Cart rank: 0. Coord: (0, 0). Left: -2. Right: 1. Top: -2. Bottom: 3

Time: 1697386347, Availability: 4
Time: 1697386350, Availability: 5
Time: 1697386353, Availability: 5
Time: 1697386356, Availability: 6
```

Screenshot of charging node 1 log file with 9 ports:

```
Global rank: 1. Cart rank: 1. Coord: (0, 1). Left: 0. Right: 2. Top: -2. Bottom: 4

Time: 1697386481, Availability: 4
Time: 1697386484, Availability: 6
Time: 1697386487, Availability: 9
Time: 1697386490, Availability: 7
```

Tested on CAAS:

Scenario 1: Test with different number of grid size

| Grid Size | Number of ports | Number of iterations | Number of reported messages |
|---|---|---|---|
| 2x2 | 3 | 3 | 2 |
| 3x3 | 3 | 3 | 2 |
| 4x4 | 3 | 3 | 1 |

## 4. Analysis & Discussion

Test with different number of iterations:

The hypothesis here is that the communication time is directly proportional to the number of iterations, if other factors remain constant.

From the results tabulation above, we can see that when we test with different number of iterations, it will result in different communication time. The communication time is expected to increase as the number of iterations increases. This is because each iteration involves a set of communications between charging nodes and base station. In each iteration, charging nodes report their status and request information from the base station, which adds to the communication overhead.

Some issues might include message congestion or message processing at base station. As with higher number of iterations, there might be a higher volume of messages exchanged, leading to congestion, and longer queuing time for messages. Next, the base station may take longer time to process incoming messages.

Test with different number of ports:

The hypothesis here is that the number of report messages will decrease as the number of ports increases.

From the results tabulation above, we can see that as the number of ports increases, the likelihood of all ports being in full use concurrently decreases, leading to fewer report messages. This suggests that the availability of more charging node has an impact on the frequency of reports generated by the charging nodes.

Test with different grid size:

The hypothesis here is that the greater the grid size, the longer the communication time.

From the results tabulation above, we can see that grid size had an impact on the system performance. As the grid size increases, the number of charging nodes and communication links also increases, leading to longer communication time, since we need to go through all the charging nodes and compute for their neighbour nodes.

Comparison between running on single computer and across CAAS

CAAS took longer execution time than running on a single computer due to the increased grid size.