**ASSIGNMENT COVER SHEET**



|  |  |  |  |
| --- | --- | --- | --- |
| **Student’s name** | Austin Sing Jet | Wong | |
| **ID number** | 32933975 | **E-mail** | awon0070@student.monash.edu |
| **Unit code & name** | FIT3143 Parallel Computing | **Unit code** | FIT3143 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Title of assignment** | **Assignment – 2 *(You may include an appropriate title as per your report)*** | | |
| **Lecturer/tutor** | Dr. Vishnu Monn | | |
| **Is this an authorised group assignment?  Yes**  **No**  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** | | Thursday/8.00 a.m. | |
| **Due date 16/10/2023** | | | **Date submitted 18/10/2023** |

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)](https://www.monash.edu/students/admin/assessments/extensions-special-consideration) must be submitted.

yes

**Has an extension been approved? Yes No If yes, please give the new submission date …18.…/.10.…./…2023….**



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](https://www.monash.edu/students/study-support/academic-integrity) and the University’s [Student Academic Integrity: Managing Plagiarism and Collusion Procedures](https://publicpolicydms.monash.edu/Monash/documents/1935762). * I understand the consequences of engaging in plagiarism and collusion as described in [Assessment and Academic Integrity Policy](https://publicpolicydms.monash.edu/Monash/documents/1935748) * 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:  1. provide to another member of faculty; and/or 2. submit it to a plagiarism checking service; and/or 3. 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.   *Signature ...................................................................................................... Date……17/10/2023………………………………* |
| **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](mailto:privacyofficer@adm.monash.edu.au) |

**FIT3143 Assignment 2, 2023**

*.*

1. **User Guide**

**User Guide:**

mpicc  : make EVSimulation

mpirun : make run n={argument1} x={argument2} y={argument3}

1. **Methodology**

The EV Simulation has been simulated by the Message Passing Interface which has been substituted as a network connection between the Base Server Station and Electrical Charging Station, which Electrical Charging Stations has been arranged as grid cartesian which, each Charging Station got its own respective Cartesian coordinates ranging from (0,0) to (rows , columns) with rows \* cols numbers of EV Charging Station which could be done with MPI Topology from the standard MPI Library.

`MPI\_Comm\_Split` has been made to split the communication of Base Station and EV Station while the new communicator only allows the communication of neighbouring EV Station and only through MPI\_COMM\_WORLD to communicate with the base station, which set up the spec of the assignment, each communication has its own respective rank for each process, of which the global rank of the last ranks is acting as Base Station while others is EV Station within its own defined function.

The approach of mine is making it as synchronised as possible for all processors, which When listening to indefinite incoming message, using ` IProbe() `, which allows it to listen to a certain Message during that time, which is stable but used unnecessarily as it loops by a counter. But after receiving, there is a preprocessing step to send back for responding to the request from sources.

**For EV Station:**

**Charging Ports Simulation:**

Each EV Station’s Ports has simulated by threads with using #omp and that each

Port will be used to generate car which by checking the ports by checking the id of the ports, which the car will left when the battery is above 90% and each turns, it will charge by 20% percent, soon, the car charged up and left, the car ID will be -1, which means it is available and be occupied, if a car is generated, the port will be recorded as occupied.

**Communication between EV Station:**

The communication only starts after reporting the availability of charging ports Simulation which only reports the current availability of charging ports. In each iteration there will be listening to the passing message from EV Station.

However, it will be receiving the message after checking messages from its neighbour EVS using `IProbe()` which it is placed inside a counter while loop and send the respective index of the neighbour in the  source's neighbour which ease to find the index of the respective node when sending back to the source’s node. Once finished the first stage of communication, the information, (the neighbour's index from node sources and the current availability of the node), will be sent back to the initial senders and the availability will be stored inside the respective index of the array specifically for the availability of the neighbouring nodes. Each EV Station will be sending to each Neighbour one time only during the two communications period.

then if there is available, it will report back the Neighbours’ ID and its respective availability during that period, and if there is no more than one or (two for showcasing purposes) availability of Neighbouring EV Stations, it will need to send the Alert Log to the Base Station. which it is synchronised using MPI Barrier of the communicator (EVS Station Comm)

**For Base Station:**

#omp has been used to spawn another threads to act as the communicator in base station while the other is used to do the analysis of the nearest node and record all the Alert Log that has been received.

**Alerting Base Station between EVS Station:**

After reaching a period (MPI\_BARRIER(COMM\_WORLD)) after communication between EVS Station, Base Station will start listening through `IProbe()` from any EV Station and it will listen and to any event as long as it is from EVS Station, which will capture the Alert Period that was sent by any EV Stations.

**Analysis of Base Station:**

After it had done the receiving Alert Logs from EV Station, another Threads will simulate the analysis of the receiving message and use an array to check whether there is any message received from the last 2 period(iteration) with each index storing the message of the last 2 periods, which it would search from Base Station Database, storing each coordinate of each EV Station and its neighbour. With this information, we can find the neighbours’ neighbours much more efficiently.

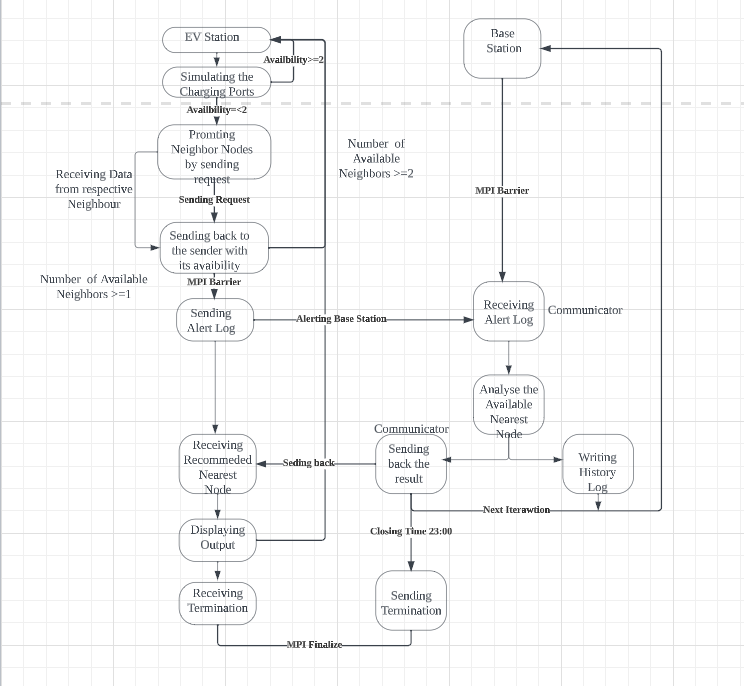
 If there are 3 incoming messages in this predefined period, it will store inside the array for each Array[ID][12], (since in 2D, the nearest node here is the neighbour’s neighbours, which it could be maximum of 4 neighbours, then each neighbour exclude to the node that requested it, so 4x4-4=12 of nearest nodes), which it would be necessary to append and  write  Log  into Output File and send the data back to the one that sent .

**Recommended Nearby EV Station back to the Requester:**

While writing the file in another time, another thread would be ongoing to send back the respective requester and send back the available (Ports>1) as there is no more changes of information as both of the threads will be only reading value of the information to write the Logs and sending back the information. Once EV Station finished receiving, it will report all the nearby EV Station that has been reported back to the sender,

**Termination of EV Station and Base Station:**

After a certain time, for my case during the system time of 23:00 pm, the base station will send a specific tag TERMINATION to the EV Station.  Once finished, EVS Station will listen inside the first while loop and once it receives the tag TERMINATION , it will immediately return(terminate) the program. Once Base Station finishes Sending, it will also be terminated following with other programs running EV Station.



FlowChart: Communication between EV Station and  Base Station

Pseudocode: Base Station and EV Slave

Charging Station (EVStation, baseStation) {

    while (!termination) {

        #pragma omp parallel

        {

            SimulateChargingPorts(nPorts);

        }

        int i = nums;

        while (sending && i > 0) {

            Iprobe(Neighbors); // Listening

            if (received) {

                Recv(baseStation, Termination);

            }

            terminate();

            return;

        }

        Iprobe(Neighbors); // Listening

        if (received) {

            Recv(Neighbors, index);

        }

        if (sent && ports <= threshold) {

            sent = 0;

            for (each index, neighbor in neighbors) {

                Send(neighbor, index);

            }

        }

    }

    MPI\_Barrier(comm);

    while (sending && k > 0) {

        Iprobe(Neighbors); // Listening

        if (received) {

            Recv(Neighbors);

        }

        if (sent && ports <= threshold) {

            sent = 0;

            for (each msg, source in responsive message) {

                Send(source, msg);

            }

        }

    }

    MPI\_Barrier(comm);

    if (sent == 0) {

        WriteNodeLog();

        if (Number\_of\_neighbors - Not\_avail\_nodes <= 2) {

            Send(baseStation, AlertLog);

            Recv(baseStation, AvailNearestNode);

        }

        WriteNearestNode();

    }

    Time counter();

    sleep(1);

    MPI\_Barrier(world\_comm);

}

Communicator() {

    #pragma omp critical

    {

        while (count && global > 0) {

            Iprobe(EVStation);

            if (received) {

                Recv(EVStation, AlertLogs);

                StoreInformation();

                receive\_log[EVStation] = 1;

                global--;

            }

            count++;

        }

        #pragma omp barrier

        // To synchronize with the base station

        #pragma omp barrier

        for (EVStation) {

            if (received) {

                MPI\_Send(EVStation, NearestNode[EVStation]);

            }

        }

        #pragma omp barrier

    }

}

BaseStation() {

    #pragma omp parallel num\_threads(2)

    {

        if (threadID == 0) {

            MPI\_Barrier(world\_comm);

            while (is\_receiving) {};

            for (EVStation) {

                memset(nearest\_node[EVStation], -1);

            }

        }

        #pragma omp barrier

        #pragma omp critical

        {

            for (EVStation) {

                Analyze(NearestNode[EVStation], information[EVStation]);

            }

        }

        else if (threadID == 1) {

            while (iteration > 0) {

                Communicator(baseStation, information);

            }

            if (termination == 1) {

                for (EVStation) {

                    MPI\_Send(EVStation, terminate);

                }

            }

        }

    }

}

1. **Results Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of processors, n | Grid size, x \* y | Total number of iterations | Total Number of reported messages | Summary of events generated | Total Simulation Time (s) |
| 5 | 2x2 | 68 | 58 | Figure 1.1  Figure 1.2  Figure 1.3  Figure 1.4 | 2.7758 |
| 7 | 2x3 | 68 | 51 | Figure 2.1  Figure 2.2  Figure 2.3  Figure 2.4 | 3.845 |
| 10 | 3x3 | 68 | 86 | Figure 3.1  Figure 3.2  Figure 3.3  Figure 3.4 | 7.9453 |
| 9 | 2x4 | 68 | 70 | Figure 4.1  Figure 4.2  Figure 4.3  Figure 4.4 | 7.56 |
| 9 | 1x8 | 68 | 132 | Figure 5.1  Figure 5.2  Figure 5.3  Figure 5.4 | 8.0084 |

Table 1:

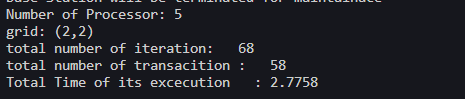


Figure 1.1

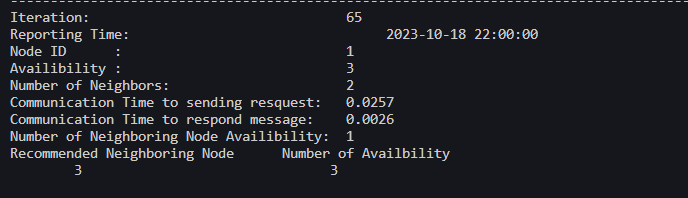


Figure 1.2

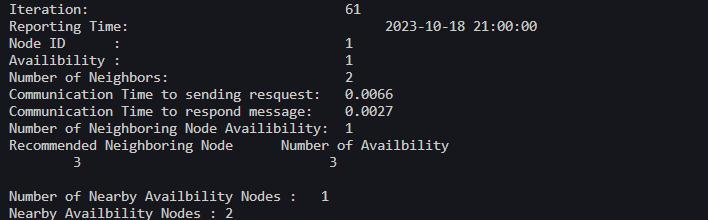


Figure 1.3

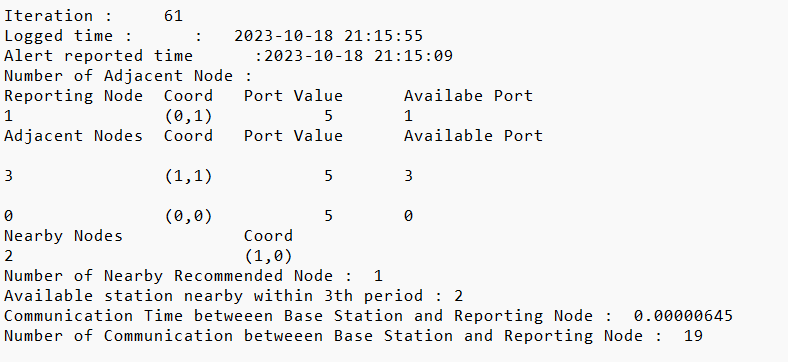


Figure 1.4

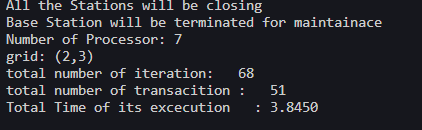


Figure 2.1

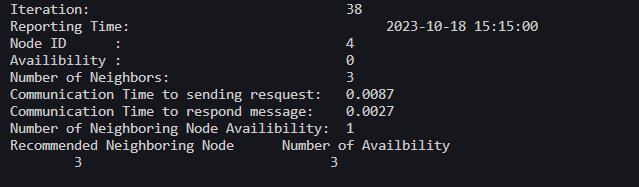


Figure 2.2

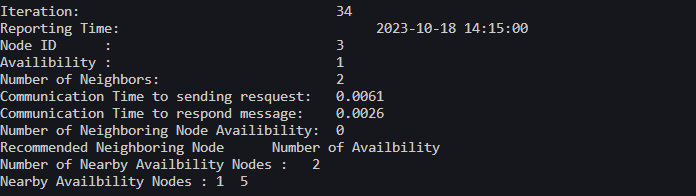


Figure 2.3

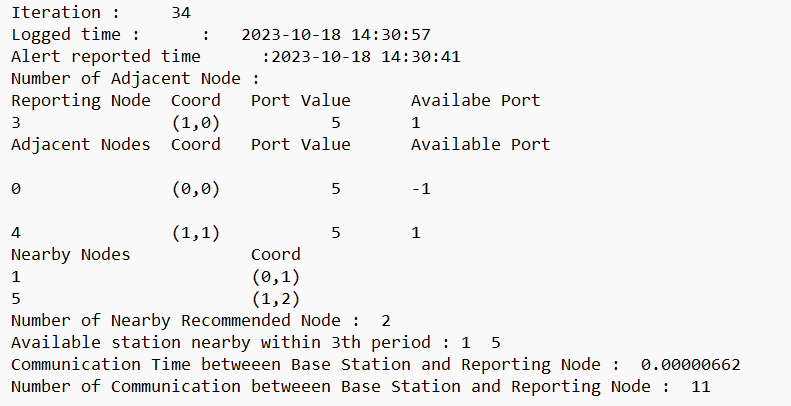


Figure 2.4

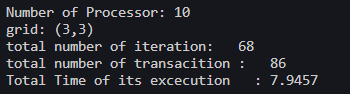


Figure 3.1

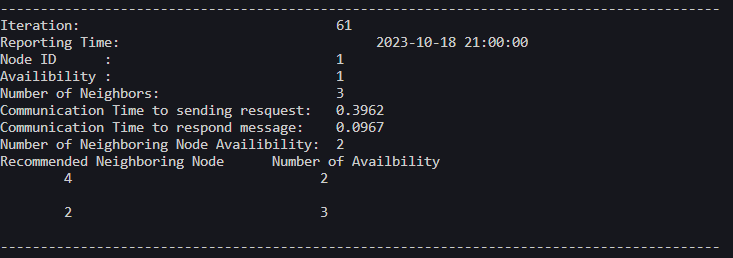


Figure 3.2

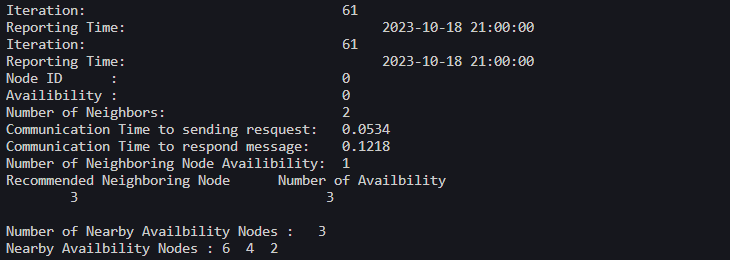


Figure 3.3

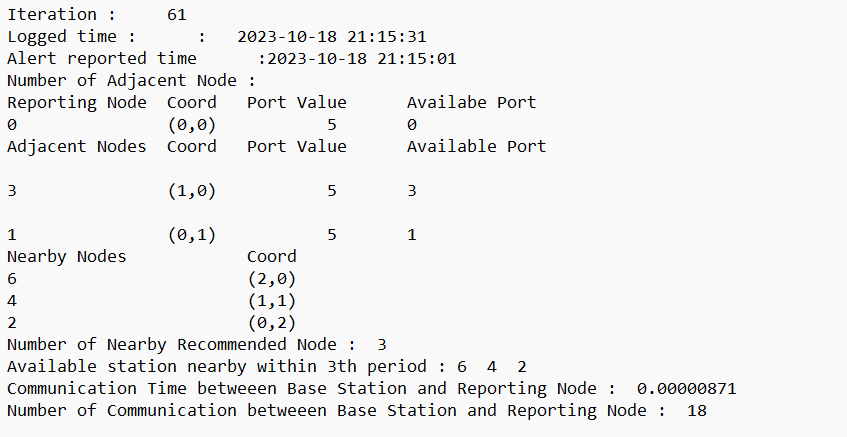


Figure 3.4

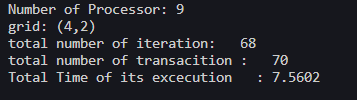


Figure 4.1

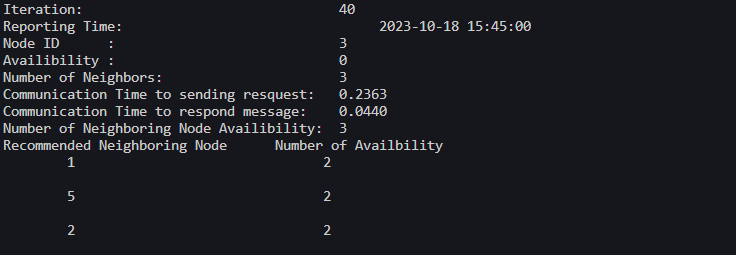


Figure 4.2

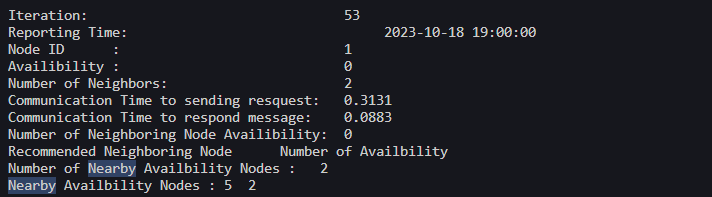


Figure 4.3

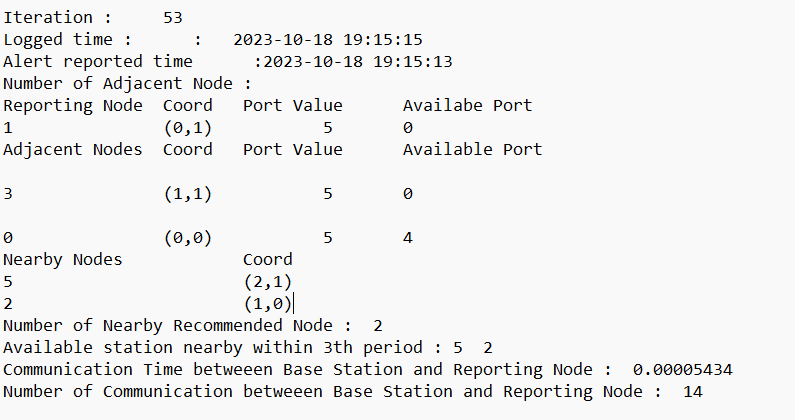


Figure 4.4

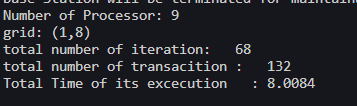


Figure 5.1

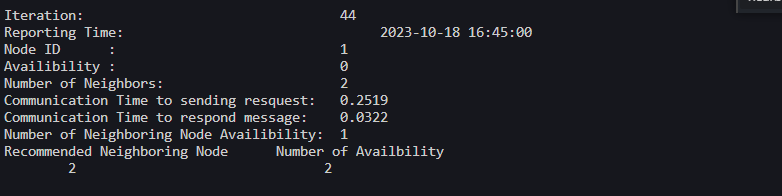


Figure 5.2

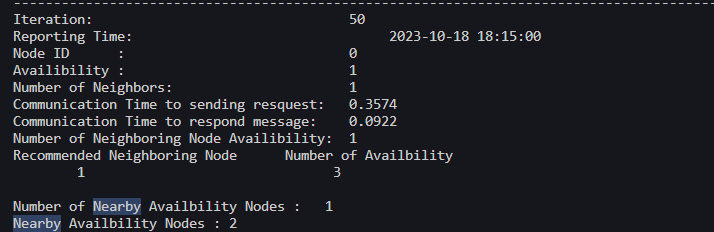


Figure 5.3

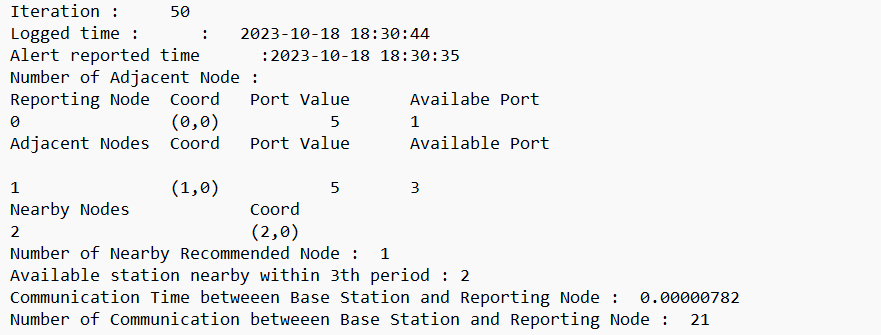


Figure 5.4

1. **Analysis & Discussion**

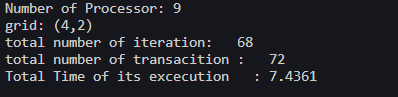
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of processors, n | Grid size, x \* y | Total number of iterations | Total Number of reported messages | Summary of events generated | Total Simulation Time (s) |
| 5 | 2x2 | 68 | 58 | Figure 1.1  Figure 1.2  Figure 1.3  Figure 1.4 | 2.7758 |
| 7 | 2x3 | 68 | 51 | Figure 2.1  Figure 2.2  Figure 2.3  Figure 2.4 | 3.845 |
| 10 | 3x3 | 68 | 86 | Figure 3.1  Figure 3.2  Figure 3.3  Figure 3.4 | 7.9453 |
| 9 | 2x4 | 68 | 70 | Figure 4.1  Figure 4.2  Figure 4.3  Figure 4.4 | 7.56 |
| 9 | 1x8 | 68 | 132 | Figure 5.1  Figure 5.2  Figure 5.3  Figure 5.4 | 8.0084 |

*From C. Tabulation Result Table 1:*

We can see that the difference grid dimension will has a huge impact to the total simulation time, which the first two result has not been used with oversubscribe, while the first two cases performs better, since there is no more oversubscribe. However, the times will be more precedently larger since the range of communication is much larger as number of processors is growing bigger, the time of the communication as well as processing the information will be much larger.

Although it is caused by overloading one or two more threads by oversubscribing the simulation time. Thus, it is hard to determine the real time for the time to be simulate, which it results in a big degradation of total run time for case 3 to case 5, it has a big difference when compared to case 1 and case 2, which the maximum cores to run was 8.

When comparing with the last three result, we can see the grid dimension will affect the simulation time. The case 4 and case 5, we can see that grid 1x8 has a larger execution time, since we can know that it has smaller surface of grid. Thus, each EV Station has less neighbour which it would be resulting sending more alert result to the Base Station which it results a larger communication time due to more transactions is sent during the execution. The reason is that each EV Station will be only has 1 or 2 neighbours which it will resulting more alerting message to be reported to the Base Station lead to larger extent of communication time.



When I run the program with grid size (4 x 2), it would get the approximately the same result as processor 9 which it would be leads to conclusion, the runtime of the programs will be also affect the grid size and the grid dimensions which it will be affecting the communication as a whole. The higher the number of transactions to the Base Station, the longer the simulation will be running since there is more message to be produced and to be sent.

**Conclusion:**

MPI Topology, (grid size and grid dimensions), will affect the communication time as it can causing less or more message to pass to the Master Node and then it would lead to more communication overhead. Thus, the execution times of the program will solely correlate to the grid size and grid dimension. The larger grid size will have more communication to deal with while the squared grid dimensions will have lesser communication to deal since each node will has more neighbour on the respective position of the grid cartesian

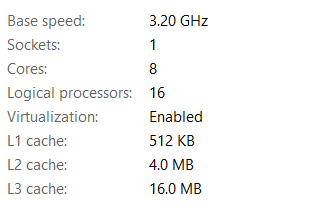
1. **Reference**

1. Aiso. (2023, July 7). Introduction to Circular Queue. https://www.geeksforgeeks.org/introduction-to-circular-queue/

2.Sergey L.Sergey L*Append to the end of a file in C*. Stack Overflow. https://stackoverflow.com/questions/19429138/append-to-the-end-of-a-file-in-c

3. GillesGilles  9. (1962, February 1). *Trouble understanding mpi\_type\_create\_struct*. Stack Overflow. https://stackoverflow.com/questions/33618937/trouble-understanding-mpi-type-create-struct /\*

*Appendix: Local PC*

**