MONASH University

GROUP ASSIGNMENT COVER SHEET

Student ID Number	Surname	Given Names
29357071	Ooi	Yi Xun
29610672	Chan	Jing Yao
30666503	Chan	Guan Yu

* Please include the names of all other group members.

Unit name and code	FIT3143 Parallel Computing	
Title of assignment	Assignment 2	
Lecturer/tutor	Vishnu Monn Baskaran	
Tutorial day and time	Wednesday 4pm-6pm	**Campus** Malaysia
Is this an authorised group assignment? ✅ **Yes** ☐ **No**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ **Yes** ✅ **No**		
Due Date 18/10/2021		**Date submitted** 19/10/2021

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 The material used can be from any source (staff, students or the internet, published and unpublished works).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
- I 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 ..... *Date……*

* delete (iii) if not applicable

Signature:Yi Xun Ooi Date:18/10/2021
Signature:Jing Yao Chan Date:18/10/2021
Signature: Guan Yu Chan Date:18/10/2021

Privacy Statement
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 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, 2021
Assignment 2 - Report

Team Name (or Number): Lab 01 Team 09

Student email address	Student First Name	Student Last Name	Contribution %	Contribution details*
yooi0003@student.monash.edu	Yi Xun	Ooi	20%	Task 2 Code, Report Methodology & Results Tabulation
jcha0075@student.monash.edu	Jing Yao	Chan	20%	Task 1 & 3 Code, Results Tabulation
gcha0018@student.monash.edu	Guan Yu	Chan	60%	Task 1, 2 & 3 Code, Methodology, Results Tabulation, Analysis and Discussion, Full Documentation (e.g comments)

**Your contribution details include the report, code, or both.*

Note: Please refer to Assignment specifications, FAQ and marking rubric (two member or three member teams) for details to be included in the following sections of this report.

Include the word count here (for Sections A to C): 1895

Simulation of Tsunami Detection in a Distributed Wireless Sensor Network using MPI and POSIX

Team Name: Lab 01 Team 09
Team Member: Yi Xun Ooi (yooi0003@student.monash.edu)
Jing Yao Chan (jcha0075@student.monash.edu)
Guan Yu Chan (gcha0018@student.monash.edu)

MONASH University

A. Methodology

Grid Topology Implementation

In this assignment, a wireless sensor network is implemented using grid topology to simulate the sensor nodes and the base station.

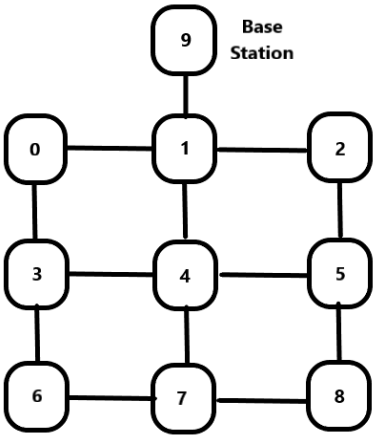


Figure 0: An example of 3X3 Grid Topology

As shown in the figure, each node will be a MPI process with the number of processes to be allocated by the user. For example, if the number of processes is 10, a 3X3 grid of sensor nodes will be formed by process 0 to process 8 while process 9 acts as the base station node.

Each sensor node can communicate with its adjacent nodes (top, bottom, left, right) and also the base station node. The base station node can communicate with every sensor node individually.

After the sensor network is implemented, each sensor node obtains its own coordinate using MPI_Cart_coords and calculates its number of adjacent nodes using MPI_Cart_shift.

Simulated components

Sensor Nodes

A sensor node first starts by preparing to receive requests from neighbour nodes, using a non-blocking operation. These requests are only received later on. To generate a simple moving average, an array of size 5 is created (height_arr), to store the height values. Since initially the array is empty, the node will populate the array by creating 5 random height values and inserting them into the array. This is so that the initial, first moving average generated can be large enough to be compared with the threshold. Note that the threshold is fixed to be 6000, the random height values range from 5700 to 6300.

While the sensor node has yet to receive a termination message, it starts and creates a new height value and updates height_arr. Then, the new moving average is calculated. The new moving

average is the current reading. Next, the current reading is compared with the threshold. If the threshold is exceeded, the sensor node will send a request to all its valid neighbour nodes using a synchronous send, MPI_Ssend. The reason why MPI_Ssend is chosen is because MPI_Ssend will wait until the recipient has properly received whatever that has been sent. Then the sensor node will wait for its neighbour nodes to send their readings back to it. This is done using a non-blocking operation.

On the other hand, if the threshold is not exceeded, the sensor node will try to check if there are any requests sent from its neighbour nodes. That is done using MPI_Testsome. If there is at least 1 request, the sensor node will send its reading to the neighbour node that has requested it via a synchronous send, similar to when the sensor node is requesting for its neighbour nodes' readings. To put it simply, while the sensor node sends a request and waits for a response (neighbour node's reading), it will also check if any neighbour node has sent to it.

Recall that the receive operations are non-blocking, so a MPI_Waitall must be called eventually. If the sensor node has sent requests to its neighbour nodes, it will wait and receive their readings. Then, it simply checks if there are at least 2 matches between its own reading and its neighbour nodes' readings. If there are at least 2 matches, then it will populate the structure for a report with the relevant values, then call MPI_Ssend to the base station node. A MPI_Iprobe is then called to always be ready to receive a termination message from the base station node. If the sensor node receives a termination message, it will then do what is called "resolving requests". It is likely that certain sensor nodes will always finish faster, which means that other sensor nodes will keep waiting for the readings of those sensor nodes that have already been finished. This will cause the program to run indefinitely. This is prevented by simply looping MPI_Testsome and checking for a constant amount of times, whether there are new requests from neighbour nodes or not. So, the sensor nodes that have finished faster will then "stay" and check for any more requests and "resolve" them.

Pseudocode for the sensor node:

```
Be prepared to receive requests from neighbour nodes
While sensor node does not have to terminate
        Obtain reading
        Check if reading has exceeded threshold or not
                If yes, request readings from neighbour nodes, and wait to receive their readings
                If no, check if there are requests sent from neighbour nodes
                        If yes, send own reading to neighbour nodes
                        If yes, also prepared to receive subsequent requests
                Wait for neighbour nodes to sent their readings back
                Check if there are at least 2 matches with neighbour nodes' readings
                        If yes, send a report to base station node
                Check if need to terminate or not
                        If yes, exit while loop
                        If no, sleep interval of 0.01 seconds
Check if there are any remaining requests left and resolve them
```

Satellite Altimeter

The satellite altimeter is simulated using a POSIX thread created by the base station node. It consists of an array and a function. The shared global array is an array of structures, which represents entries in the array, containing relevant information. The function, satellite_altimeter starts running, by creating an array with the size being the total number of processes subtracted by 1. Similarly to height_arr in sensor nodes, the shared global array called satellite is initially populated by entries.

It enters a while loop, where there is an interval of 5 seconds. For every 5 seconds, create a new entry and update the shared global array satellite with the newest entry. The method of updating is the first in, first out (FIFO) method. Everytime a new entry is created, the first entry in the array is replaced by the second entry and so on. The new entry will be stored in the last element of the array. Its termination signal is basically a variable called running, which is either true or false. If the base station wants the satellite altimeter to stop running, it will change the variable running to false, which will cause the while loop to stop executing.

There are three pieces of information in an entry. Height value, which always exceeds the threshold. The threshold is 6000, so the height value ranges between 6001 to 6500. Coordinates, which are randomly generated that are valid and the reporting time.

Pseudocode for the satellite altimeter:

Allocate space for the shared global array, size is row x column
Populate the array with entries, which are structures
Loop (row x column) times
    Create a new entry
    Insert into the array
While satellite altimeter does not have to stop running
    Sleep interval of 5 seconds
    Create a new entry
    Use FIFO method to insert entry into the array

Base Station

The base station node checks if an input iteration is given. If there is, that will be the max iteration. While the max iteration is yet to be reached, MPI_Iprobe to always be ready to receive a report from any node at every iteration. If there are reports sent to the base station node, the base station node will check if the tag is 1. If the tag is 1, it means that there is a report. A for loop is iterated row x column times. In each iteration, check if a report is sent by a node, with a rank of i. The report is received using MPI_Recv.

A report sent by a node contains reported time, communication time, its reading, its neighbour nodes' ranks and their readings, its coordinates and the number of matches between its readings and its neighbour nodes' readings.

The base station node starts by checking if the report coordinates are found in the shared global array called satellite. If the report coordinates are not found, all the information related to the

satellite altimeter will be outputted as “N/A”. If the report coordinates are found, all the information related to the satellite altimeter will be outputted according to the information of the entry in the shared global array called satellite. A variable called node_satellite_match is set to true, indicating that it is a true alert. The base station will then start to properly output the required information to the log file by calling out a number of fprintf’s. The report’s reading and entry reading is also compared and if their readings are not close enough to each other (tolerance range is 100), it indicates a false alert. Every other part in the for loop iteration is just to output the required information into the log file. After the for loop iterations are all done, the while loop iteration count is increased. Note that even when the iteration count has reached max iteration, it will still continue to run, for a constant number of iterations, particularly a constant called RESOLVE (has a value of 10). This is similar to the “resolving requests” part in the sensor node. It basically spends the last RESOLVE iterations to call MPI_Send to all the sensor nodes, telling them to terminate, by setting the termination message variable to true.

Pseudocode for the base station:

While iteration is lesser than (max iteration + RESOLVE)
    Check if any node has sent a report
    Check whether the report is a true or false alert
    If the report coordinates can be found in the shared global array, it could be a true alert
    Check if the report reading and entry reading is close enough to each other
    If yes, fprintf out information including entry information
    If no, fprintf out information not including entry information
    Once the iteration is equal to max iteration
        Send termination message to all sensor nodes
        Send termination signal to satellite altimeter
    Keep increasing iteration (till it is equal to max iteration + RESOLVE)
    Sleep interval of 1 second
fprintf out the summary information
Let POSIX thread for satellite altimeter to terminate properly

B. Results Tabulation

Simulation experiment setup #1

- Platform tested on:
    - Virtual machine - Oracle VM VirtualBox
- Specifications of the platform:
    - Logical processors: 4
    - System memory: 6144 MB
    - Operating system: Ubuntu (64-bit)
- Specification of test run:
    - Size of the grid: 3 x 3
    - Sea water column height threshold: 6000
    - Number of iterations at the base station: 10

Specifications	Values
Number of test runs	3
Grid dimensions	3 x 3
Sea water column height threshold	6000
Range in which random sea water column height float values are created	Min: 5700 Max: 6300
Average total communication time based on the number of test runs (in seconds)	27.374
Average total number of messages passed throughout the network based on the number of test runs	164
Average number of true alerts	5
Average number of false alerts	21

Table 1: Results of simulation experiment setup #1

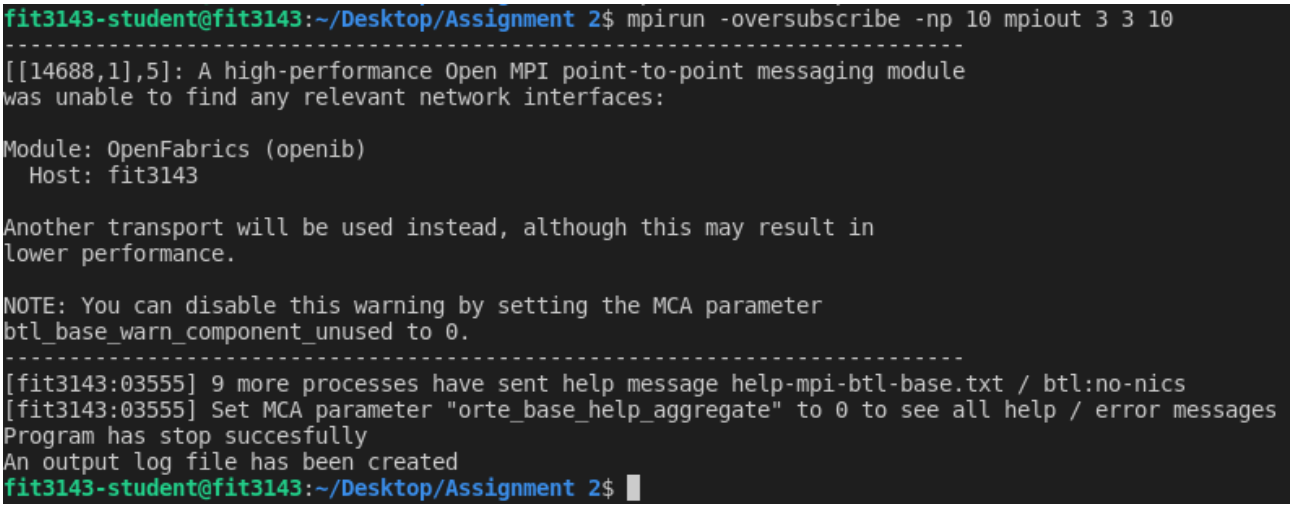
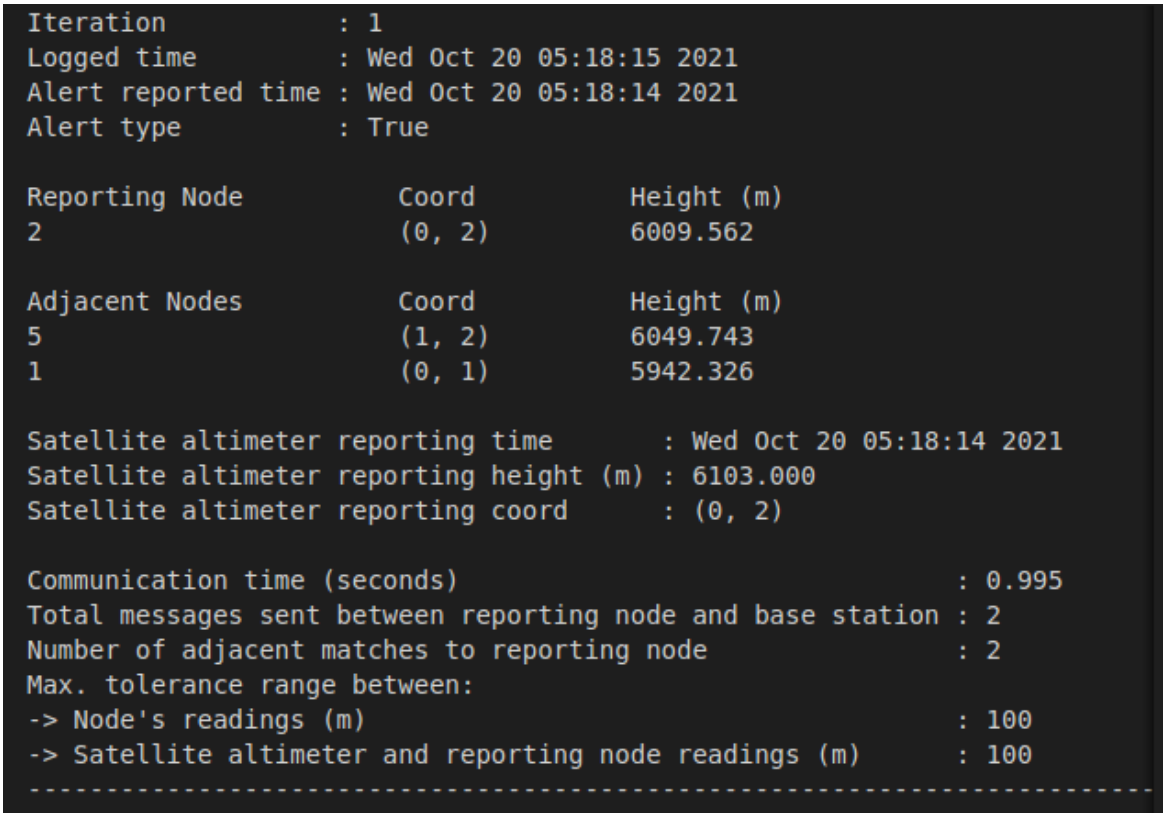
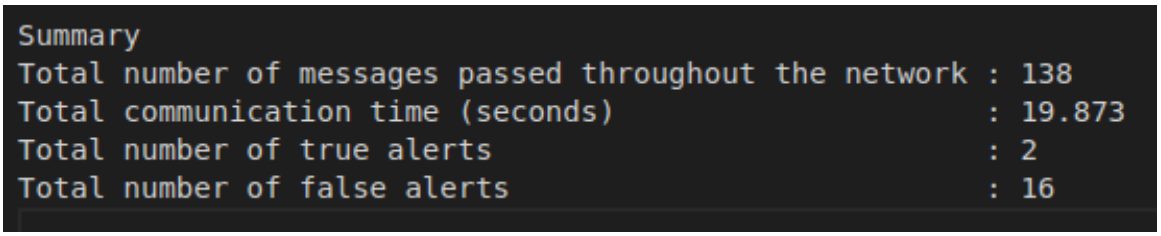


Figure 1.0: Execution of program without any errors



```
Iteration           : 1
Logged time         : Wed Oct 20 05:18:15 2021
Alert reported time : Wed Oct 20 05:18:14 2021
Alert type          : True

Reporting Node          Coord          Height (m)
2                       (0, 2)         6009.562

Adjacent Nodes          Coord          Height (m)
5                       (1, 2)         6049.743
1                       (0, 1)         5942.326

Satellite altimeter reporting time       : Wed Oct 20 05:18:14 2021
Satellite altimeter reporting height (m) : 6103.000
Satellite altimeter reporting coord      : (0, 2)

Communication time (seconds)                                  : 0.995
Total messages sent between reporting node and base station : 2
Number of adjacent matches to reporting node                  : 2
Max. tolerance range between:
-> Node's readings (m)                                        : 100
-> Satellite altimeter and reporting node readings (m)        : 100
--------------------------------------------------------------------------------
```

Figure 1.1: An example of an entry in the log file



```
Summary
Total number of messages passed throughout the network : 138
Total communication time (seconds)                     : 19.873
Total number of true alerts                            : 2
Total number of false alerts                           : 16
```

Figure 1.2: An example of the summary in the log file

Simulation experiment setup #2

- Platform tested on:
  - Virtual machine - Oracle VM VirtualBox
- Specifications of the platform:
  - Logical processors: 4
  - System memory: 6144 MB
  - Operating system: Ubuntu (64-bit)
- Specification of test run:
  - Size of the grid: 2 x 2
  - Sea water column height threshold: 6000
  - Number of iterations at the base station: 5

Specifications	Values
Number of test runs	3
Grid dimensions	2 x 2
Sea water column height threshold	6000
Range in which random sea water column height float values are created	Min: 5700 Max: 6300
Average total communication time based on the number of test runs (in seconds)	7.646
Average total number of messages passed throughout the network based on the number of test runs	68.67
Average number of true alerts	1
Average number of false alerts	6

Table 2: Results of simulation experiment setup #2

```
fit3143-student@fit3143:~/Desktop/Assignment 2$ mpirun -oversubscribe -np 5 mpiout 2 2 5
--------------------------------------------------------------------------
[[15210,1],1]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: fit3143

Another transport will be used instead, although this may result in
lower performance.

NOTE: You can disable this warning by setting the MCA parameter
btl_base_warn_component_unused to 0.
--------------------------------------------------------------------------
[fit3143:04073] 4 more processes have sent help message help-mpi-btl-base.txt / btl:no-nics
[fit3143:04073] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
Program has stop succesfully
An output log file has been created
fit3143-student@fit3143:~/Desktop/Assignment 2$
```

Figure 2.0: Execution of program without any errors

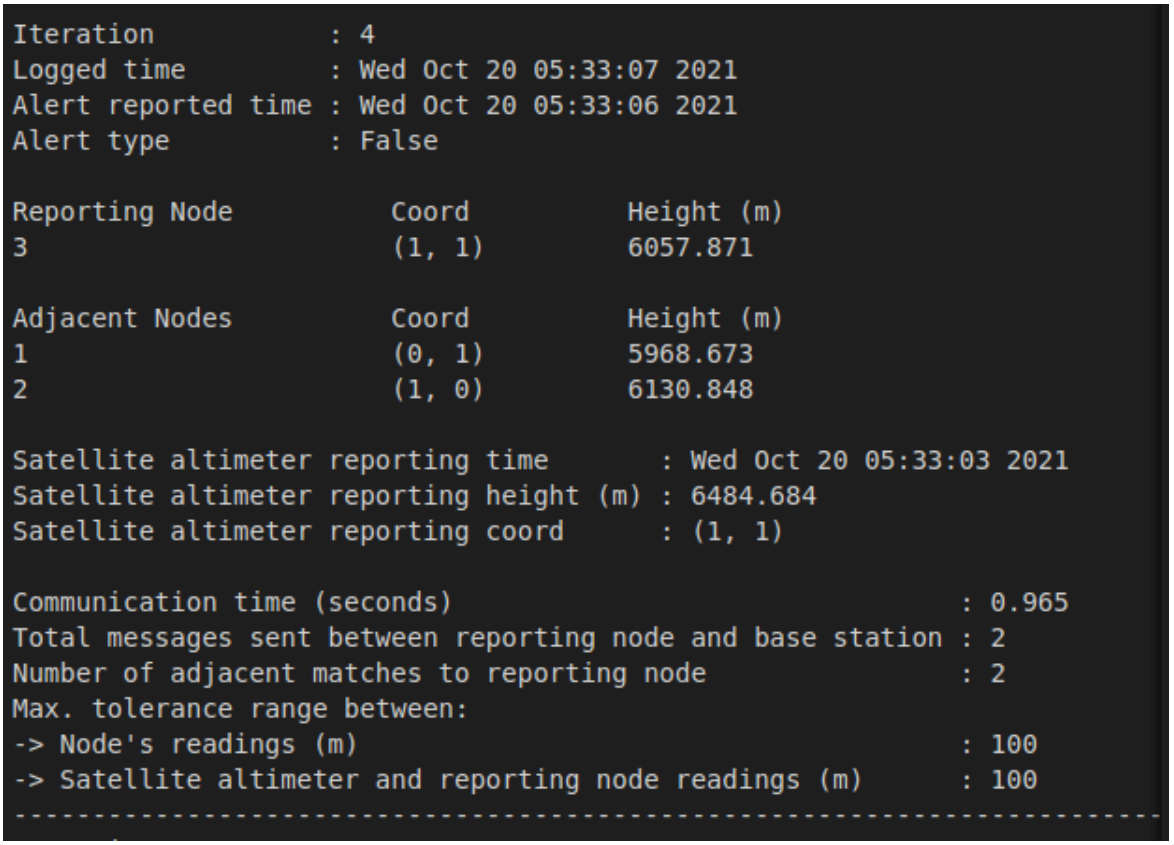


Figure 2.1: An example of an entry in the log file

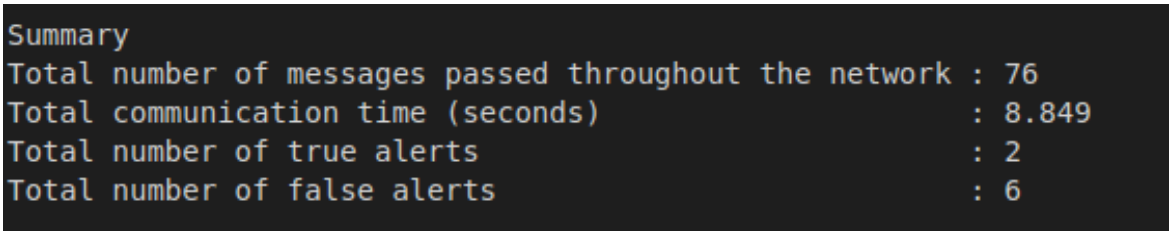


Figure 2.2: An example of the summary in the log file

Simulation experiment setup #3

- Platform tested on:
  - Virtual machine - Oracle VM VirtualBox
- Specifications of the platform:
  - Logical processors: 4
  - System memory: 6144 MB
  - Operating system: Ubuntu (64-bit)
- Specification of test run:
  - Size of the grid: 3 x 2
  - Sea water column height threshold: 6000
  - Number of iterations at the base station: 5

Specifications	Values
Number of test runs	3
Grid dimensions	3 x 2
Sea water column height threshold	6000
Range in which random sea water column height float values are created	Min: 5700 Max: 6300
Average total communication time based on the number of test runs (in seconds)	11.729
Average total number of messages passed throughout the network based on the number of test runs	72.67
Average number of true alerts	2
Average number of false alerts	9

Table 3: Results of simulation experiment setup #3

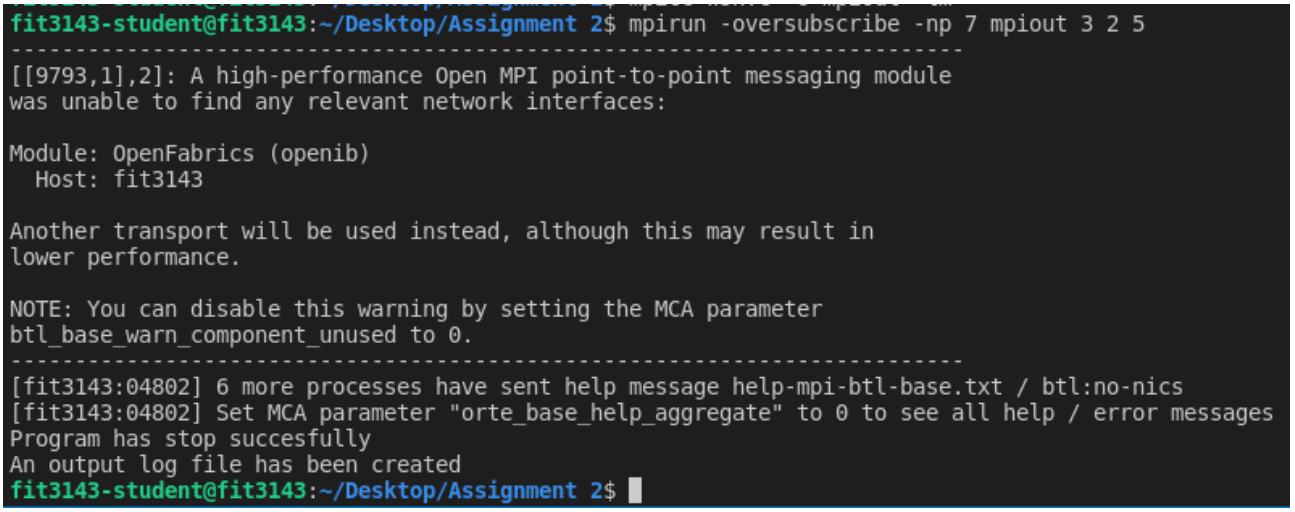


Figure 3.0: Execution of program without any errors

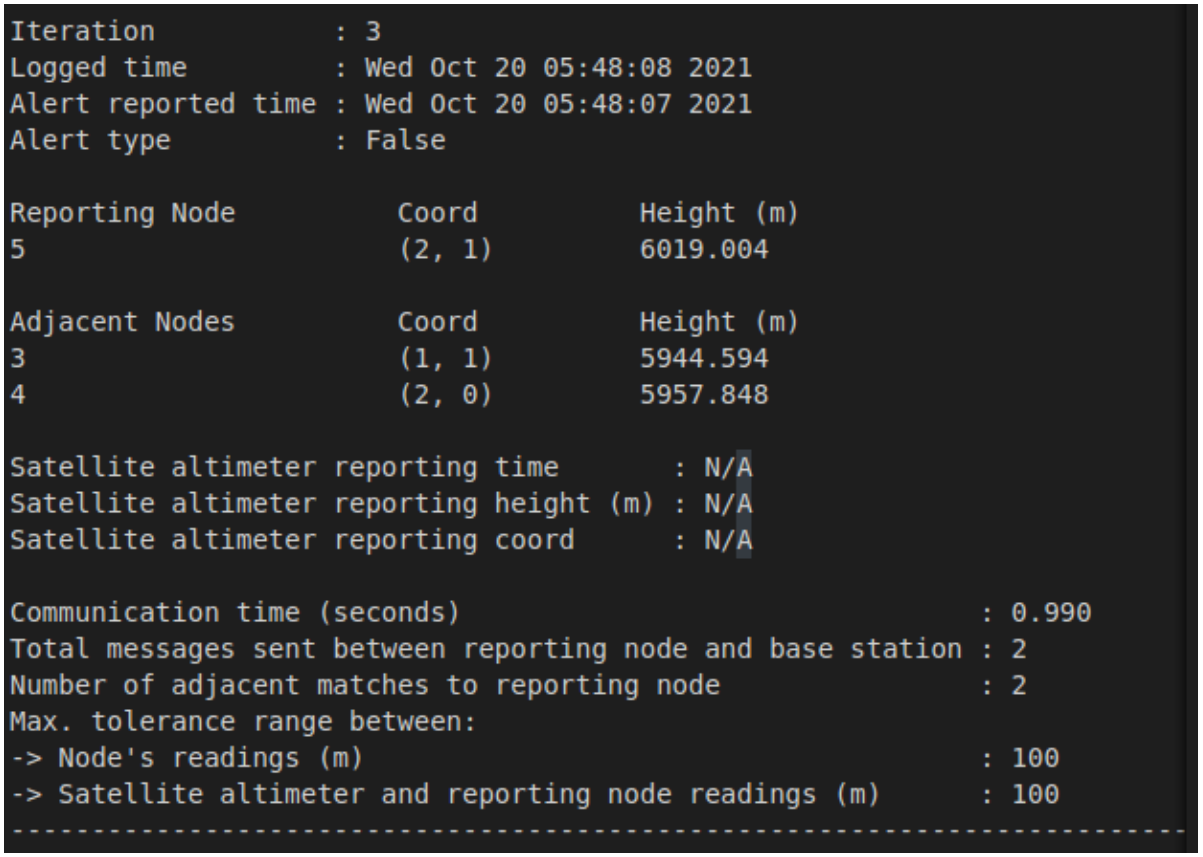


Figure 3.1: An example of an entry in the log file

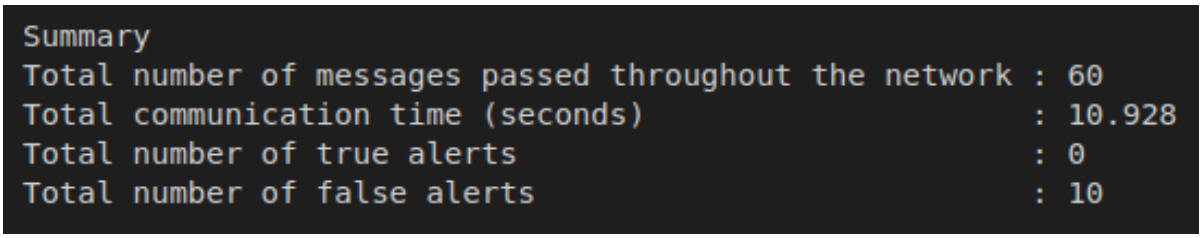


Figure 3.2: An example of the summary in the log file

Specifications	Values
CPU of machine that ran the virtual machine for program execution	Intel(R) Core(TM) i7-7700HQ
RAM of machine that ran the virtual machine for program execution	12 GB
Bit of machine that ran the virtual machine for program execution	64

Table 4: Specifications of machine that ran the virtual machine for program execution

Grid dimensions	Average total communication time based on the number of test runs (in seconds)	Average total number of messages passed throughout the network based on the number of test runs	Average number of true alerts	Average number of false alerts
2 x 2	7.646	68.67	1	6
3 x 2	11.729	72.67	2	9
3 x 3	27.374	164	5	21

Table 5: The grouping of a few result values of the 3 simulation experiment setups

C. Analysis and Discussion

From Table 4 shown above, we can see that the specifications of the machine that ran the virtual machine for program execution are not top notch. For instance, having a 12 GB RAM machine to run the virtual machine for program execution. 12 GB RAM might not be sufficient, in the sense that while running the virtual machine to execute the program, there might be other background processes as well as other software running such as Google Chrome, Discord or an antivirus software. These factors, although not significantly, could still impact the speed in which the program can be executed on the virtual machine.

After conducting several simulation experiments, we can group some tabulated results from each of the simulation experiments. From the table shown above (Table 5), we can see that the average total communication time and average total number of messages passed throughout the network based on the number of test runs, average number of true and false alerts increase, when the multiplication of the grid dimensions increase.

For example, 2 x 2 has an average total communication time of around 8 seconds, while 3 x 3 has an average total communication time of around 27 seconds. Another example, 3 x 2 has an average number of false alerts of 9, while 3 x 3 has an average number of false alerts of 21. From what we can observe, we can come up with a hypothesis.

The hypothesis is that for a constant number of test runs, the total communication time and total number of messages passed throughout the network increases for an increasing value of multiplication of the grid dimensions (the value is the result of row multiplied by column).

MONASH University

D. References

MPI documentation. RookieHPC. (2021). Retrieved 18 October, 2021.
https://www.rookiehpc.com/mpi/docs/index.php

Simple Moving Average (SMA). Investopedia. (2021). Retrieved 17 October, 2021.
https://www.investopedia.com/terms/s/sma.asp

C structures. Programiz. (2021). Retrieved 18 October, 2021.
https://www.programiz.com/c-programming/c-structures

Difference between MPI_Send() and MPI_Ssend(). Stack Overflow. (2013). Retrieved 18 October, 2021.
https://stackoverflow.com/questions/17582900/difference-between-mpi-send-and-mpi-ssend

MONASH University

Declaration:

I declare that this assignment report and the submitted code represent work within my team. I have not copied from any other teams' work or from any other source except where due acknowledgment is made explicitly in the report and code, nor has any part of this submission been written for me by another person outside of my team.

Signature of student 1: Yi Xun Ooi

Signature of student 2: Jing Yao Chan

Signature of student 3: Guan Yu Chan