

ASSIGNMENT COVER SHEET

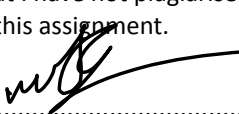
Student's name	(Family name) Myint Myat Thura	(Given names)	
ID number	31861067	E-mail	myin0010@student.monash.edu
Unit code & name	FIT3143 Parallel Computing	Unit code	FIT3143

Title of assignment	Assignment 2 - Distributed Memory Parallelism		
Lecturer/tutor	Dr. Shageenderan Sapai		
Is this an authorised group assignment? <input type="checkbox"/> Yes <input checked="" type="checkbox"/> 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? <input type="checkbox"/> Yes <input checked="" type="checkbox"/> No			
Tutorial/laboratory day & time	Thursday Laboratory 6 @ 8AM MYT		
Due date 16/OCT/2023	Date submitted 18 / OCT / 2023		

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

Has an extension been approved? Yes ☒ No ☐ If yes, please give the new submission date 18.../OCT/2023

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

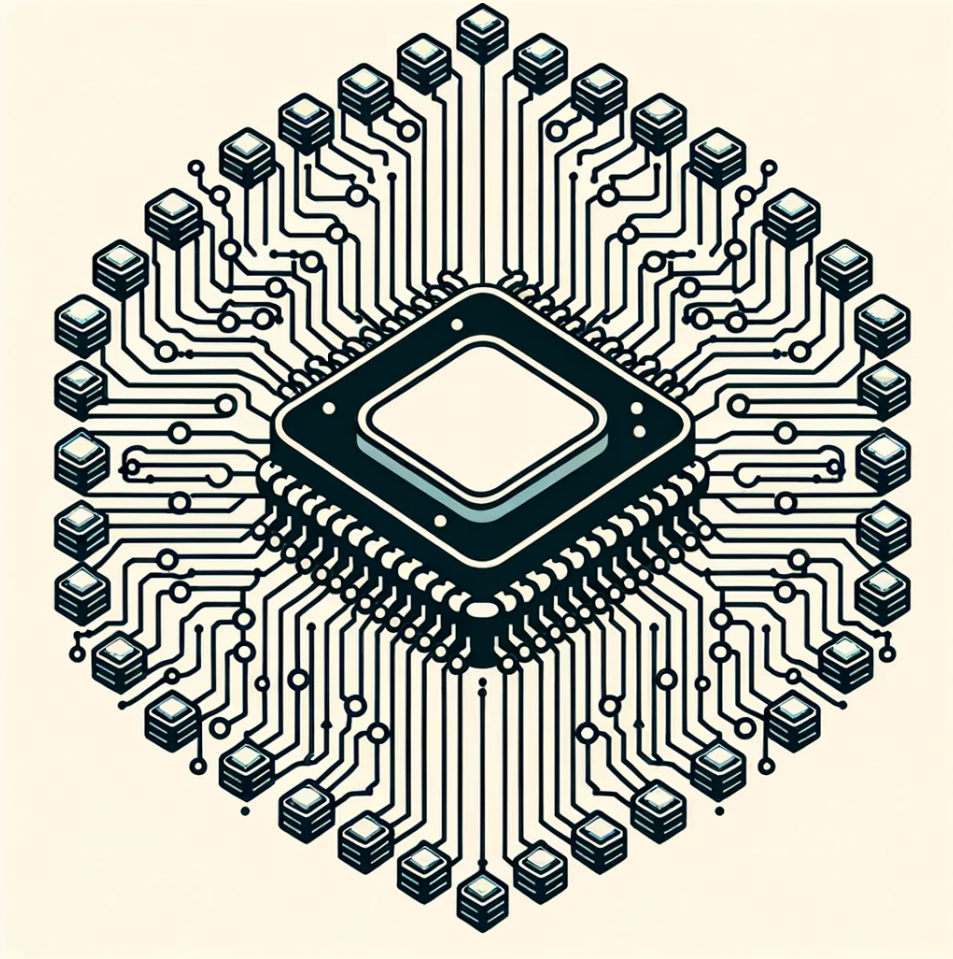
Student Statement:	<ul style="list-style-type: none"> I have read the University's Student Academic Integrity Policy and the University's Student Academic Integrity: Managing Plagiarism and Collusion Procedures. I understand the consequences of engaging in plagiarism and collusion as described in Assessment 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: <ul style="list-style-type: none"> provide to another member of faculty; and/or submit it to a plagiarism checking service; and/or 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 18 / OCT / 2023	
Privacy Statement	<p>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</p>

Assignment 2

Distributed Memory Parallelism

— Word count - 1622

By Myint Myat Thura (31861067)

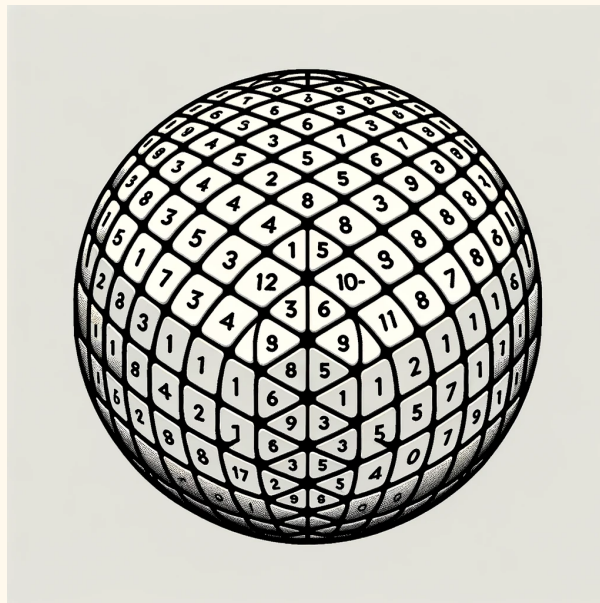


INTRODUCTION

This assignment aimed to assess our grasp of distributed memory parallelism through the simulation of an Electric Vehicle Charging Grid, managed by a distributed Wireless Sensor Network. We leveraged openMPI to accomplish this, employing Master-Slave interactions for communication and utilizing MPI Virtual Topologies to emulate the charging grid.

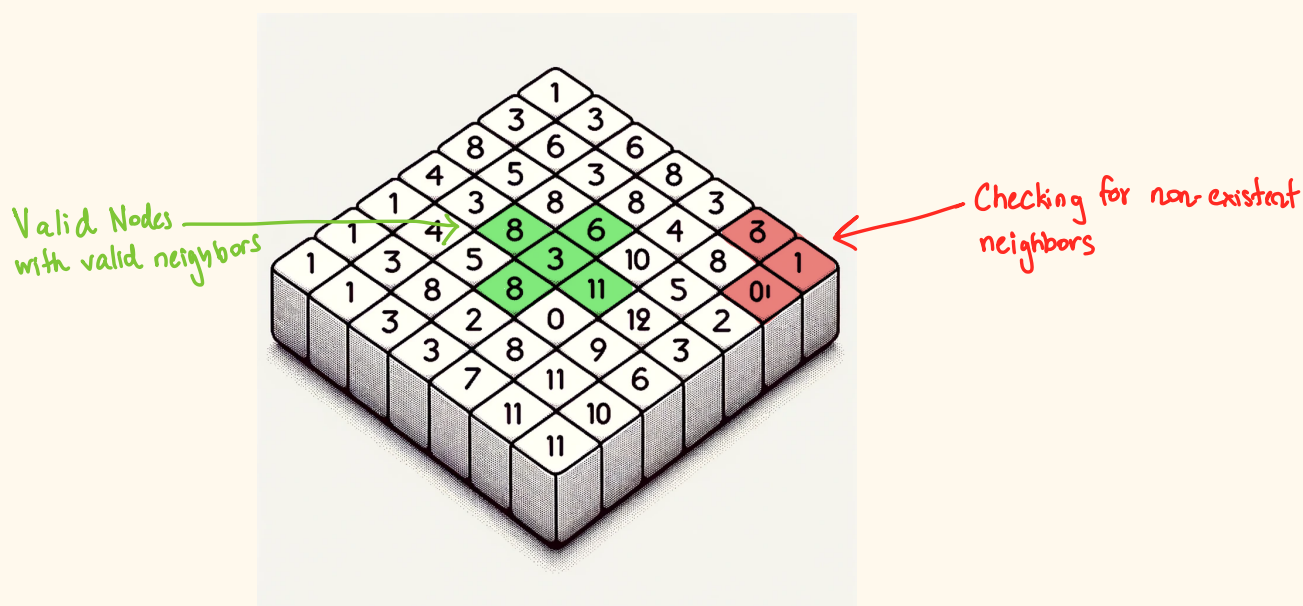
METHODOLOGY & OPTIMIZATION (Sphere Topological Architecture)

Given the intricate nature of this assignment, opportunities for optimization were admittedly limited. However, two significant enhancements emerged. One of the foremost is the conceptual representation of the grid architecture as if it were a sphere.



While the above illustration primarily serves a stylistic purpose, this spherically conceived "grid" is instrumental in bolstering the program's speed. A myriad of factors contribute to this, but paramount among them is the principle of "Symmetry."

In our setup, nodes communicate with their adjacent neighbors. Consider a conventional 2D grid as a reference point.



The central node in the grid would conventionally interact with its immediate neighbors, such as two nodes designated as '8', one '6', and an '11'. But what about an outlier node, say the '1' in the top right corner? '1' could certainly implement boundaries and selectively communicate only with valid neighbors. However, why take that route?

In the realm of distributed memory parallelism, especially within the context of openMPI, the sequence of message exchanges is vital. It's not just about dispatching and receiving messages, but ensuring they maintain a predictable order. Often, messages will pause to ensure sequence integrity and adhere to dependency chains.

It's evident that an outlier node, such as the one mentioned earlier, would complete its tasks significantly faster than a central node. If a swifter node stalls waiting for a more languid peer, our computational efficiency takes a hit. The overhead induced by complex boundary algorithms would overshadow the marginal data transmission overhead.

Total running time of a single Slave process when using Sphere topology (includes send, receive, computations, writing).	Total running time of a single Slave process when using 2D Grid topology (includes send, receive, computations, writing).
0.006841 seconds (worst case)	0.022189 seconds (worst case)

The 2D grid's computational time is about three times longer than the spherical model. This lag stems from MPI message waiting and algorithmic demands. By altering the architecture, we can potentially triple performance. By standardizing data processes, from transmission to computation, we achieve consistent operation, enhancing the program's efficiency.

METHODOLOGY & OPTIMIZATION (Sending minimal messages)

Minimizing the number of messages sent was just the beginning. Upon delving deeper, I discerned a consistent pattern governing the entire system's behavior. Essentially, there are two primary scenarios for data exchange:

1. When Slaves communicate with their neighbors, both requesting and sending data.
2. When a Slave process forwards data to the Master.

The insight here is the potential for Slaves to operate autonomously. By ensuring the Master only responds and never initiates communication, we can drastically reduce message traffic. Consider a setup with 1 Master and 2000 Slaves:

If both constantly communicated, the exchanged messages would total 4000. By preventing Master-initiated messages, this drops to 2000. While this minimizes message count by half, the Master remains key for processing, complex calculations and documenting received messages.

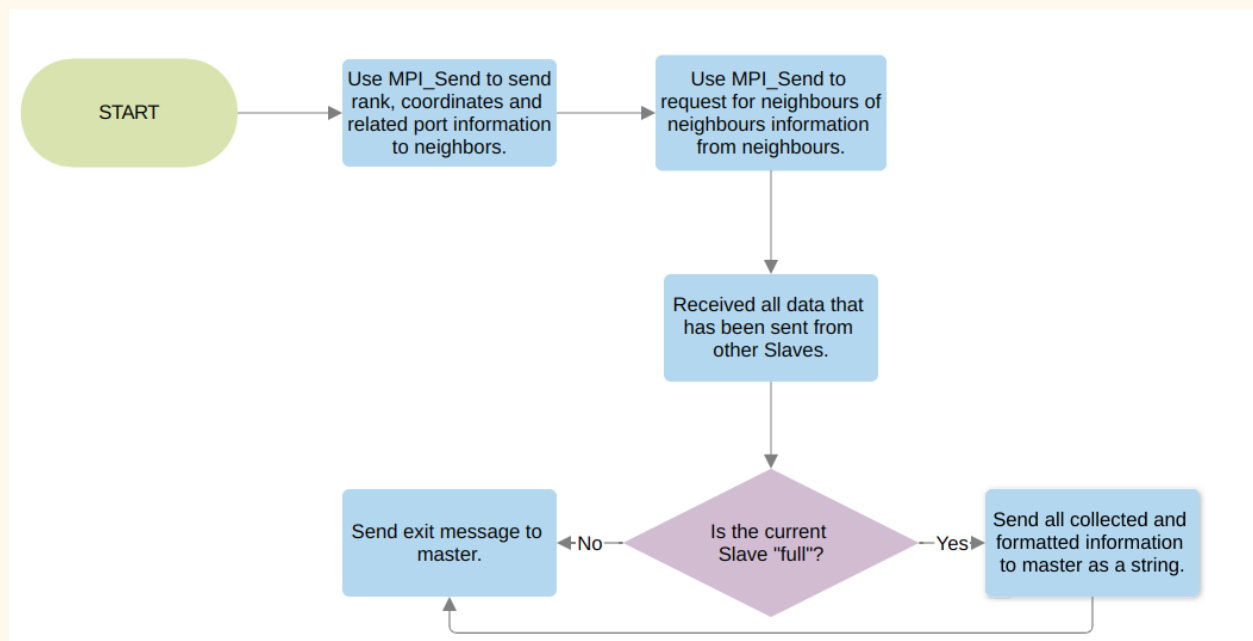
METHODOLOGY (Communication Between Neighbours)

Slaves necessitate several pieces of information from one another, specifically:

1. Rank Numbers
2. Port Availability
3. Coordinates
4. Neighbors of Neighbors

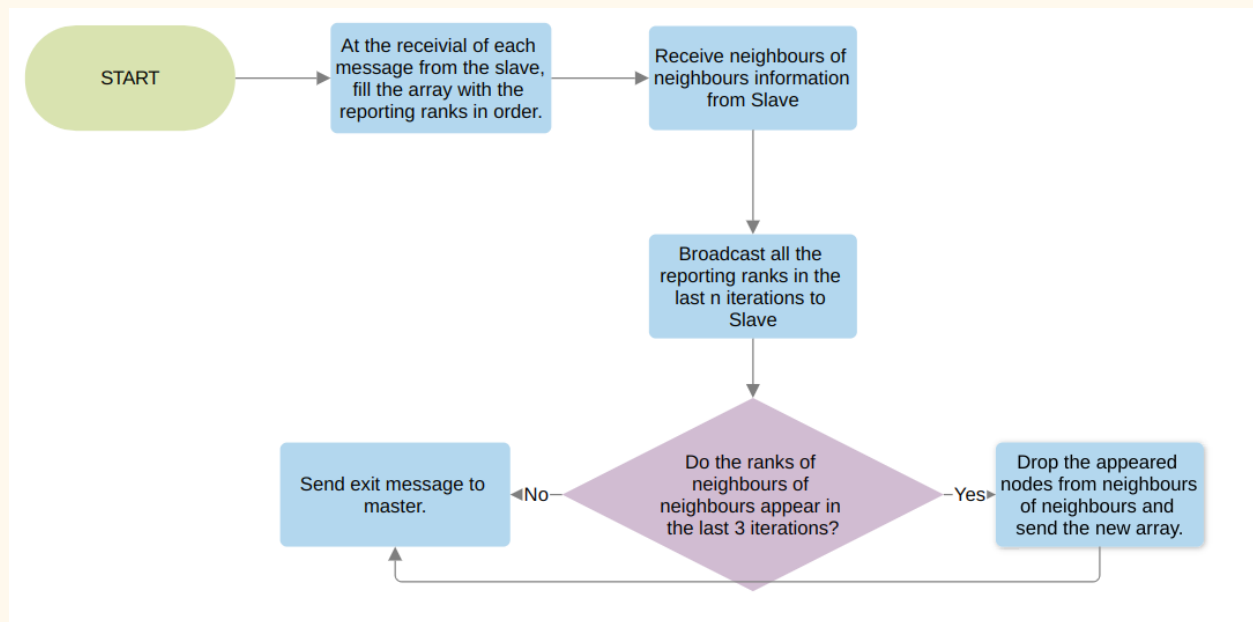
While aggregating requests using structs might simplify communication, it's crucial to proceed with caution. Sending complex messages in a single operation can lead to issues like buffer overflows, deadlocks, and synchronization errors. Bundling too much data can mask details and introduce unpredictability, which is unfavorable in synchronization-critical

programs. Thus, I've carefully divided my send and receive operations into more manageable segments, as shown in the following flowchart.



Another crucial condition ensuring optimal message transmission is to confirm that the Slave's ports are fully occupied. Only when the ports are full will messages be dispatched to the Master. The Master's role, in this scenario, is streamlined to merely receiving these messages. As a result of this efficient approach, the communication is limited to, at most, two messages to the Master: one for data and the other to signify exit.

METHODOLOGY (Calculations to recommend valid neighbors)



When the master process receives the Neighbours of Neighbours (NON) data from a slave process, it initiates a comparison with previously recorded ranks. The goal is to verify if the NON has been present in the past 'n' iterations. To accomplish this check, we use a specific method to “slice” the array:

The starting index for slicing is determined by the formula: Index to Start Slicing = $L - (n * w)$. Here, 'L' represents the length of the array, 'n' is the number of iterations we want to look back on, and 'w' is the total number of processes.

RESULTS TABULATION

Result Tabulation: Key Focus Areas

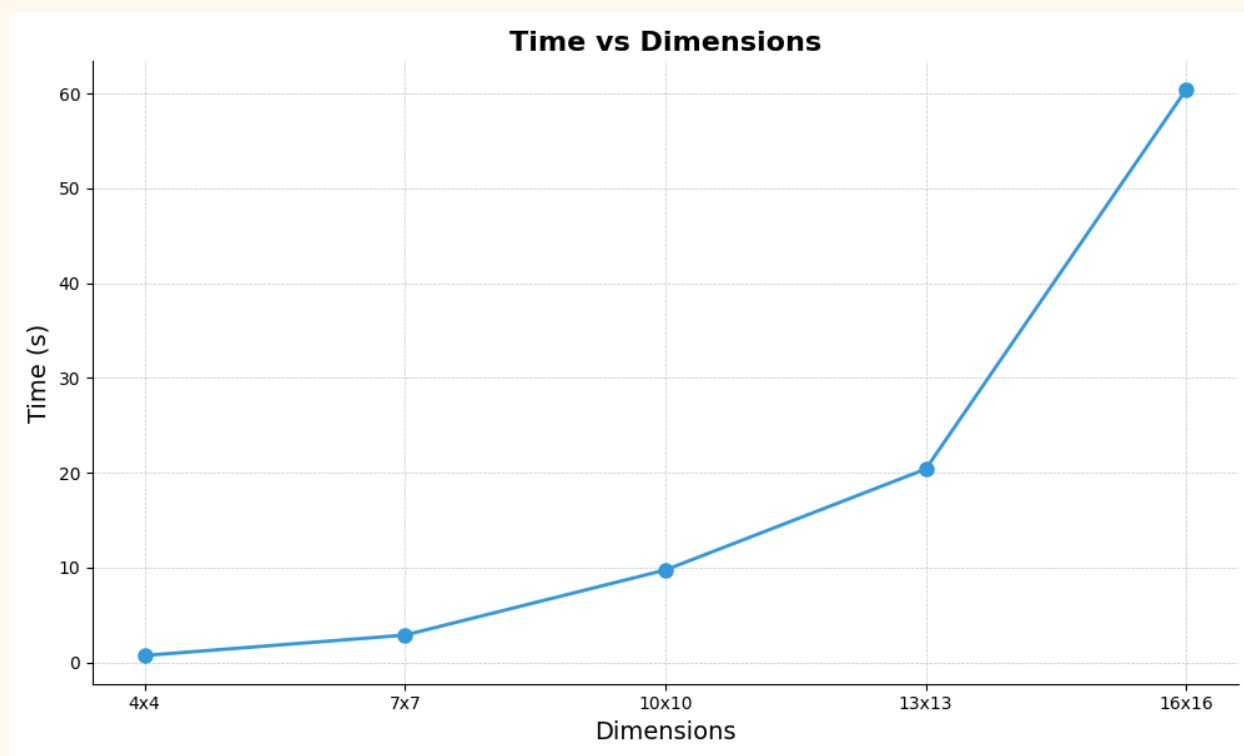
1. **Different Grid Sizes:** This aims to assess the influence of varying the number of processors on the cumulative running time.
2. **Different Number of Charging Ports:** This exploration intends to evaluate the individual runtime of each process, excluding the overarching timeline. Such an examination offers insight into the efficiency of algorithms deployed for sending, receiving, and other related operations.

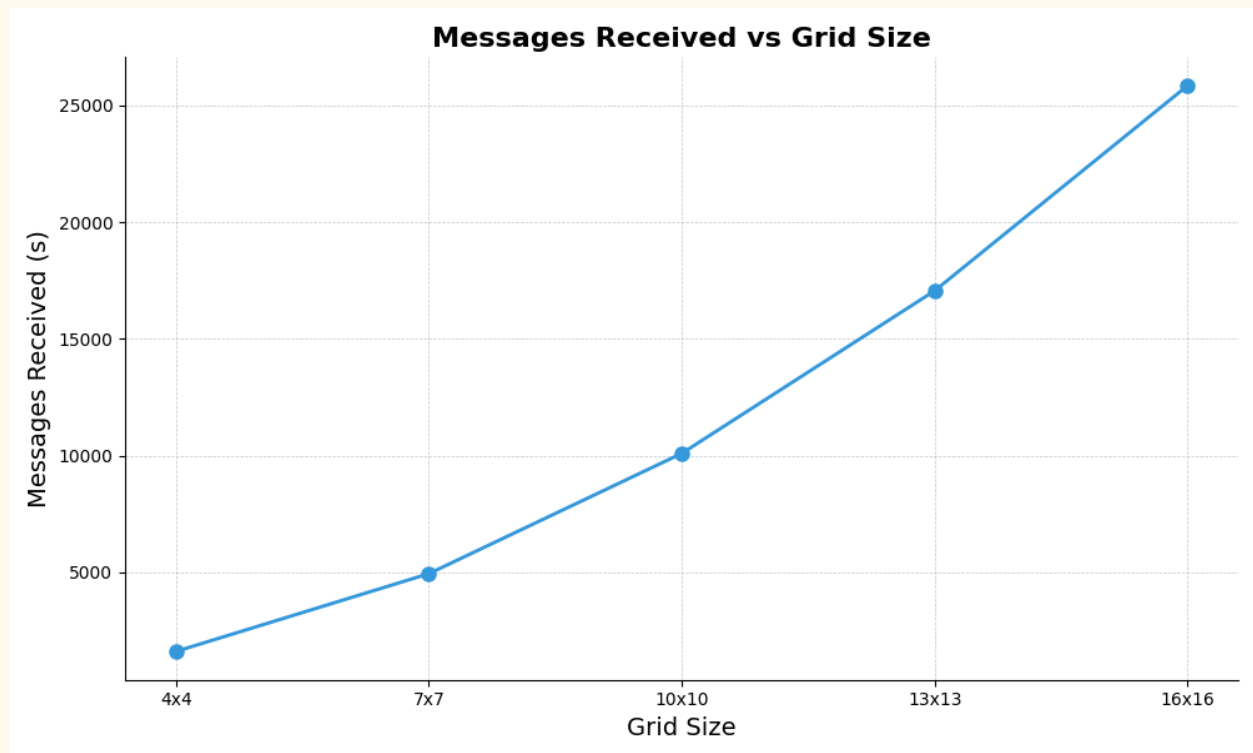
Assumptions for Simulations:

1. **Iteration Count:** Across Simulations 1, 2, and 3, the iteration count remains constant at 100.
2. **Grid Size:** For Simulations 2 and 3, the grid size is held steady at [10 x 10].
3. **Charging Ports Array:** Throughout Simulations 1, 2, and 3, the charging ports array is consistently set to 20.
4. **Full Consideration Percentage:** In Simulations 1 and 2, the threshold percentage for a port to be deemed "full" is maintained at 20%.

(Different Grid Sizes)

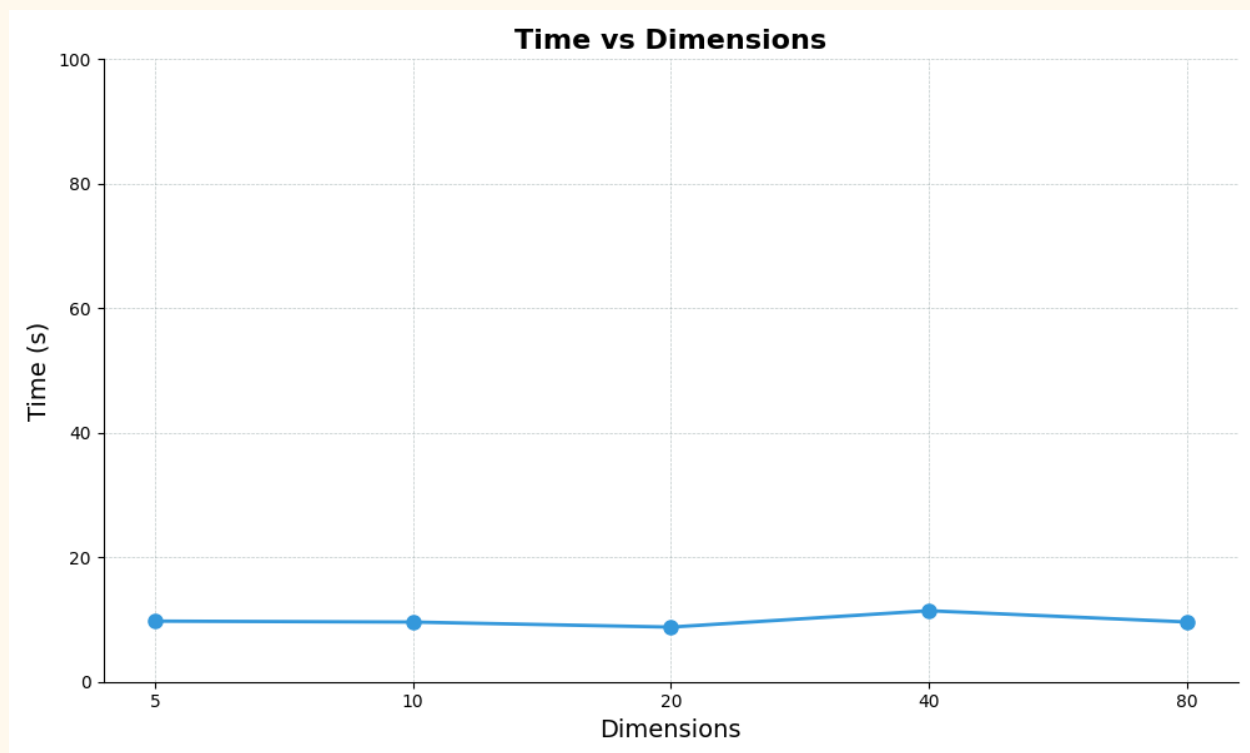
Grid Size	[4 x 4]	[7 x 7]	[10 x 10]	[13 x 13]	[16 x 16]
Running Time (s)	0.725 seconds	2.875 seconds	9.745 seconds	20.395 seconds	60.427 seconds
Messages Received (avg)	1616	4949	10100	17069	25856
Runs	3	3	3	3	3

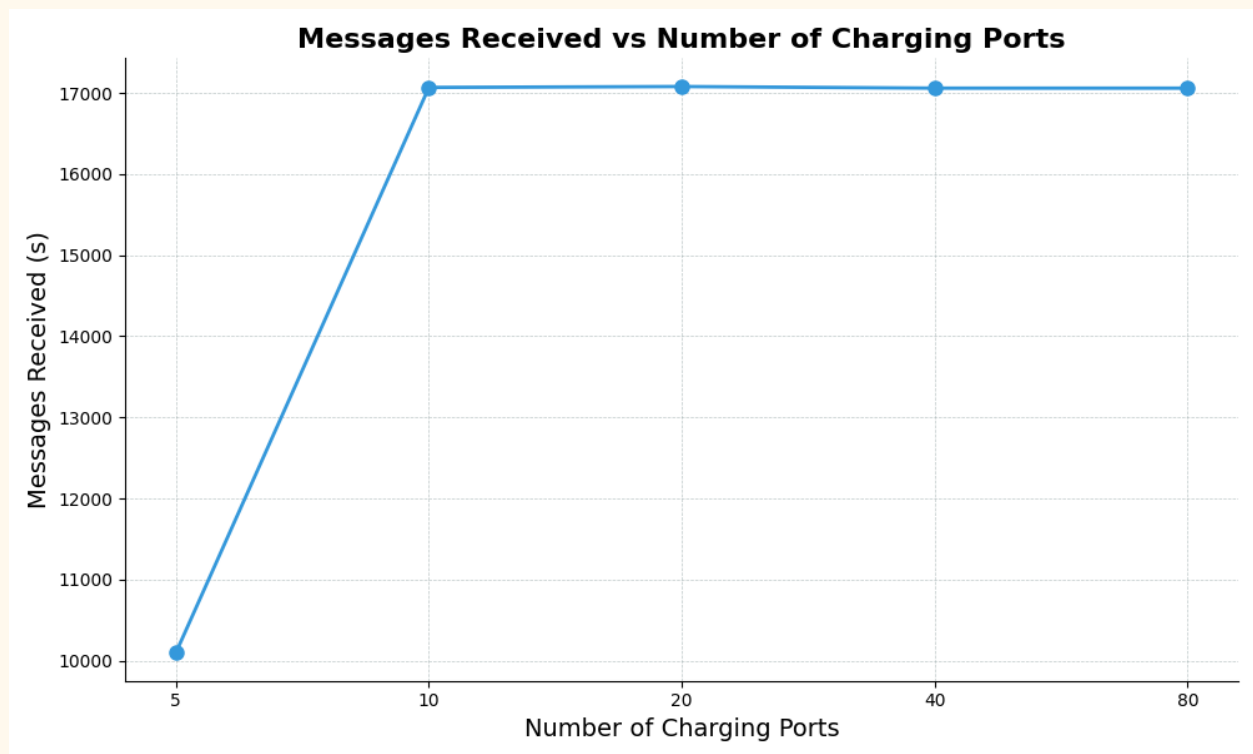




(Different Number of Charging Ports)

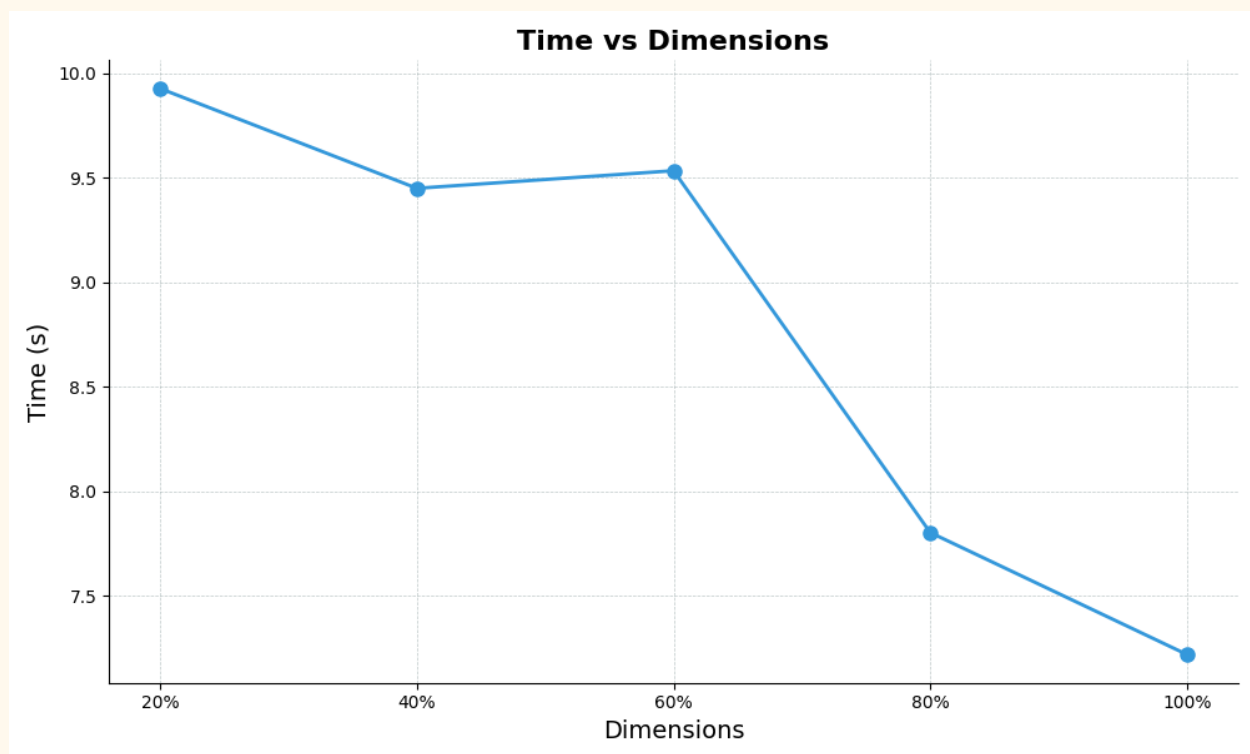
Number of Ports	5	10	20	40	80
Running Time (s)	9.745 seconds	9.597 seconds	8.78667 seconds	11.416 seconds	9.610 seconds
Messages (Received)	10100	17069	17080	17060	17060
Runs	2	2	2	2	2

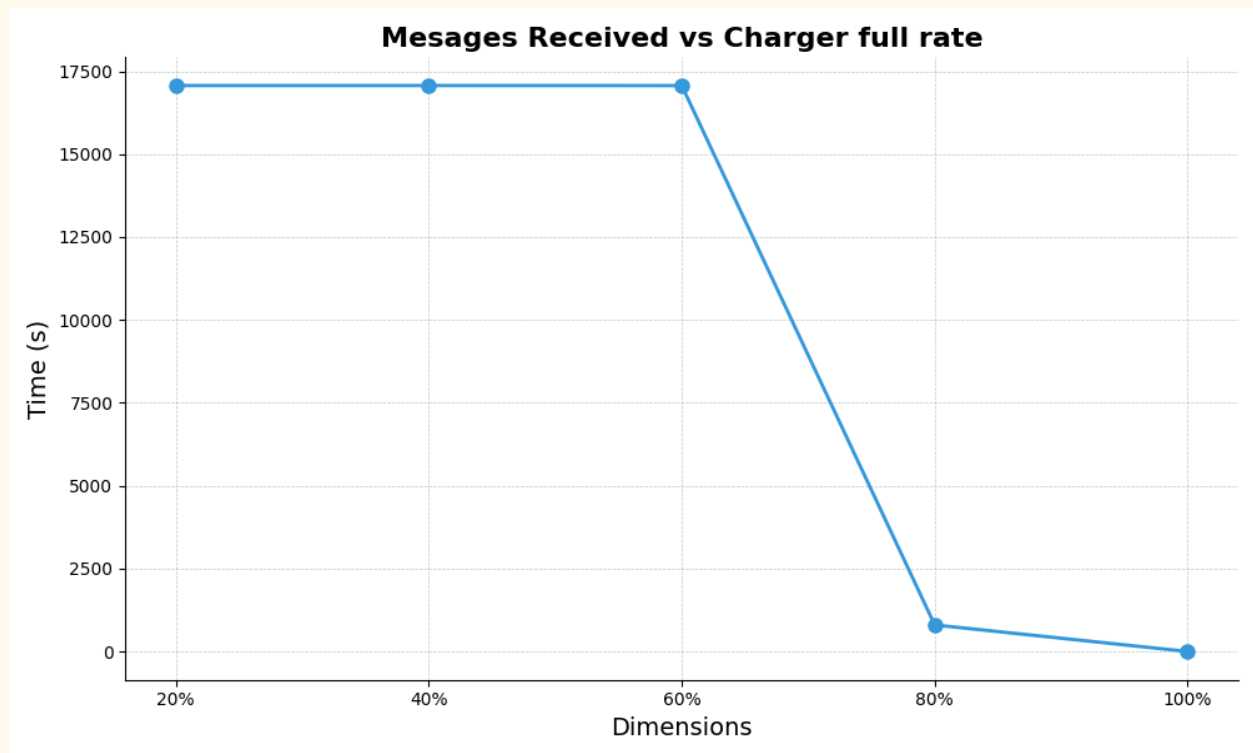




(Different Definition of Full Charging Ports)

Percentage of total ports to be considered full.	20% (At least 4 ports must be taken to be considered full)	40% (At least 8 ports must be taken to be considered full)	60% (At least 12 ports must be taken to be considered full)	80% (At least 16 ports must be taken to be considered full)	100% (All ports must be taken to be considered full)
Running Time (s)	9.927 seconds	9.450 seconds	9.534 seconds	7.804 seconds	7.221 seconds
Messages Received (avg)	17069	17069	17067	800	0
Runs	2	2	2	2	2





(SAMPLE LOG FILE OUTPUT)

```

*****
*****

Grid Dims [10 , 10]

-Number of Adjacent Nodes: 0
-Availability to be considered full: 40%

Reporting Node:

-Rank 1 , Coordinates (0,1), Port Size 20, Free Ports 10

Neighbours:

~~~~~Recommendations~~~~~

For node 1, we recommend:

-Node 81, Coordinates (8,1)
-Node 1, Coordinates (0,1)
-Node 90, Coordinates (9,0)
-Node 92, Coordinates (9,2)
-Node 21, Coordinates (2,1)
-Node 10, Coordinates (1,0)
-Node 12, Coordinates (1,2)
-Node 9, Coordinates (0,9)
-Node 3, Coordinates (0,3)

These Neighbours [81,90,92,21,12] have not sent reports in the last 3 iterations.
These neighbours [1,10,9,3] are most likely full.

Communication Time with Adjacent Nodes: [ 0.004492s ] (Not including Sleep)

*****
*****

```

The log file includes all essential information such as communication time, port numbers, etc.

HYPOTHESIS

I hypothesize that the size of the grids will directly influence both the time taken and the number of messages exchanged. More processors naturally allow for increased message exchanges. While the number of charging stations might not greatly impact the time, it should affect the message count due to increased communication avenues. Additionally, the percentage of free charging ports should influence the messages received. A higher threshold for a port to be considered full reduces the potential for message transmission, possibly driving the message count down to zero.

ANALYSIS AND DISCUSSION

From the figures presented, our primary focus was on gauging the impact of message reception and the time taken, based on three comprehensive simulations. Comparing the actual results to my hypothesis, there are some interesting results.

The grid size emerged as the predominant factor influencing both the time taken and the number of messages dispatched. I postulate that since the number of slave processes escalates in tandem with the grid size, there's an augmentation in processes available for message exchanges with the master. This directly results in a substantial surge in the volume of messages transmitted. Additionally, concerning time, my hypothesis is anchored in the notion that time scales linearly with the quantity of messages transmitted. It's a fundamental understanding that each message requires a certain time for transmission.

It's discernible that the volume of messages scales efficiently with an increment in the threshold delineating a port's status as "full". A conspicuous anomaly is the instances where the number of messages exchanged plummeted to negligible levels. I attribute this to the inherent probabilistic assignment mechanism: each port in the charger array is assigned a status of 0 (empty) or 1 (full) with an equal likelihood of 50%. Consequently, when the conditionality anticipates a homogenous status array (either entirely full or empty), we are poised on the precipice of a binary probability spectrum – leading to extreme messaging outcomes.

Contrastingly, the count of charging ports displayed a nominal impact on both the runtime and the volume of messages. This observation dovetails with the second point. I surmise that, given the consistent 50% probability assignment for port status, the volume remains unaffected. Even with an increased capacity like 80 charging spaces, there remains a consistent possibility of encountering a half-empty or half-full scenario, rendering the message variance negligible. The runtime, being largely invariant, reflects the efficiency of our employed algorithm, which primarily leverages for-loops. Such algorithmic efficiency was attainable courtesy of an adept topological framework, namely the "Sphere" architecture. This provided the latitude to refine our computational algorithms since the underlying MPI operations were already optimized.

CONCLUSION

In light of the hypothesis and subsequent detailed analysis, it's evident that grid size plays a pivotal role in influencing both time and message exchanges within the Electric Vehicle Charging Grid system. The architecture and probability dynamics involved further emphasize the

criticality of design decisions. Our optimized "Sphere" topology, combined with efficient algorithms, not only validated portions of our initial hypothesis but also shed light on nuanced behaviors, especially concerning the proportion of occupied charging ports and their impact on message transmission. This investigation underscores the intricate interplay of factors in distributed memory parallelism and the paramount importance of thoughtful system design.

REFERENCES

- Sabbaghi, R., & Doroud, H. (2013, January 18). *sphere-based topology for networks-on-chip* - researchgate. Sphere Based Topology. https://www.researchgate.net/publication/264440264_Sphere-based_topology_for_networks-on-chip
- MPI, open. (2019, March 17). 8.2.5. *Exch2: Extended cubed sphere topology*. 8.2.5. exch2: Extended Cubed Sphere Topology - MITgcm checkpoint68s-11-g1ac7194 documentation. https://mitgcm.readthedocs.io/en/latest/phys_pkgs/exch2.html
- Sabbaghi, R., & Doroud, H. (2013, January 15). *sphere-based topology for networks-on-chip* - researchgate. A novel idea for NoCs. <https://ieeexplore.ieee.org/document/6141375>

