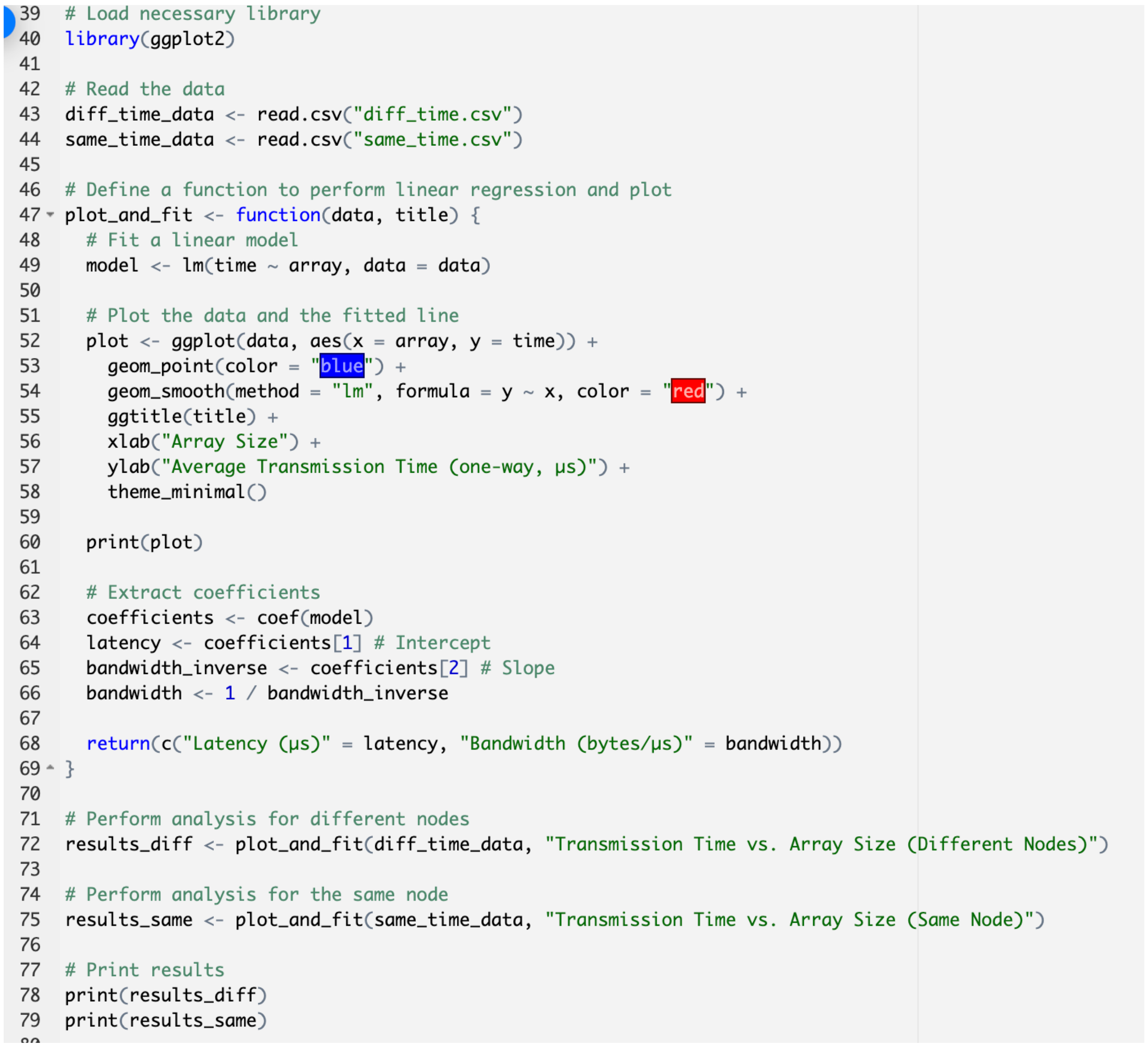
Project 5 CS4473 report

To Ly
The University of Oklahoma
Norman, OK
tohly@ou.edu

Problem 1:

Make plot, fitting and estimated values a linear regression function:

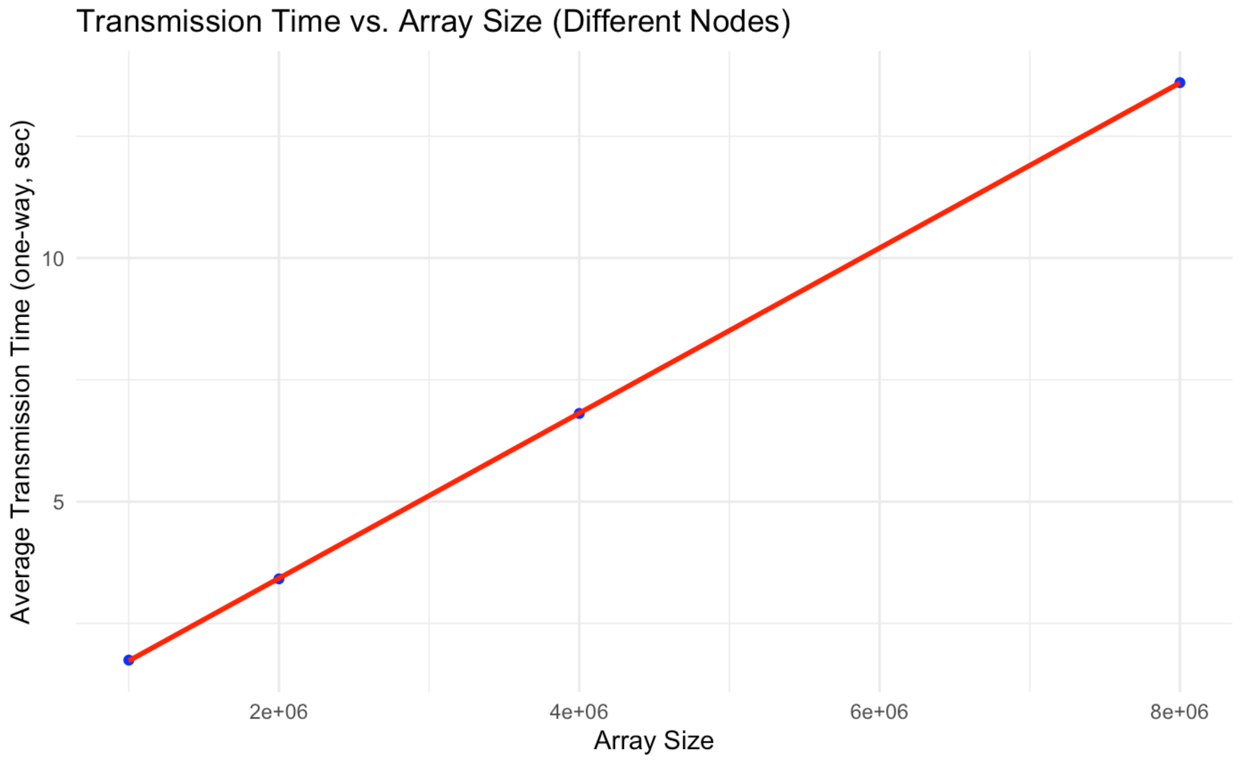
R-studio code:



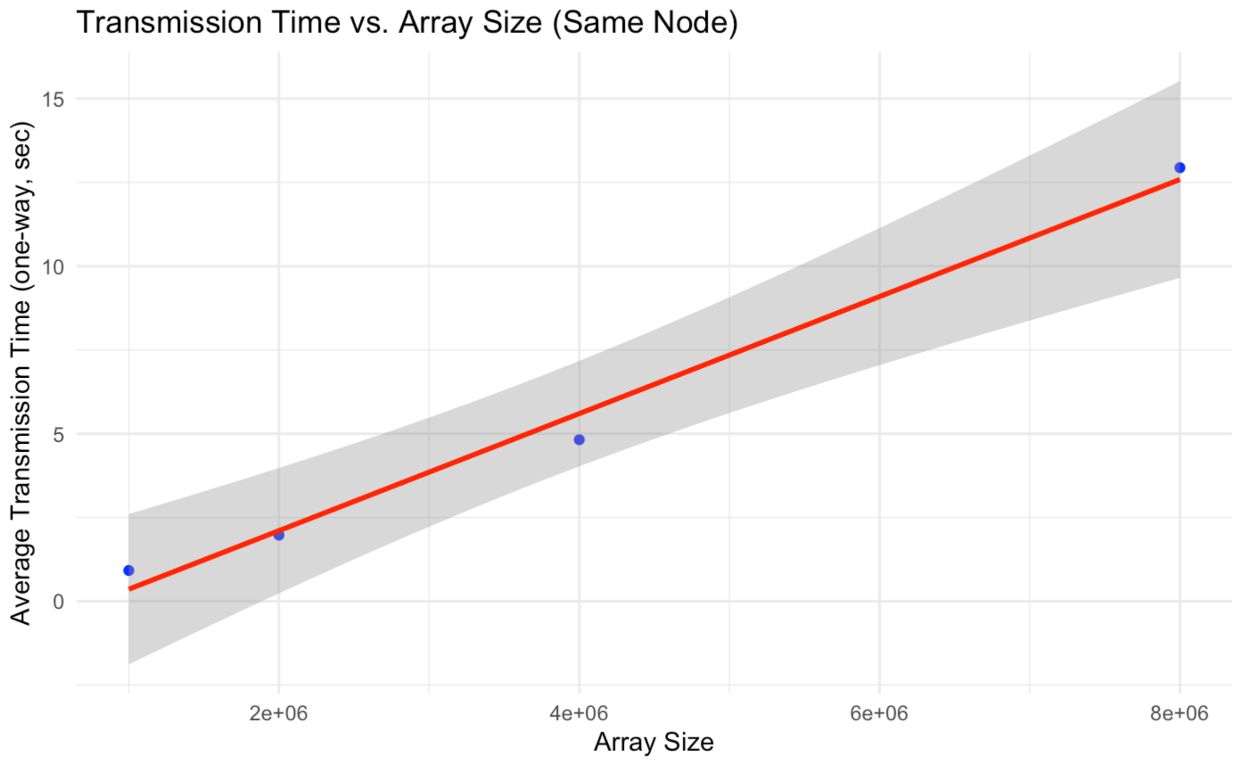
```r
# Load necessary library
library(ggplot2)

# Read the data
diff_time_data <- read.csv("diff_time.csv")
same_time_data <- read.csv("same_time.csv")

# Define a function to perform linear regression and plot
plot_and_fit <- function(data, title) {
  # Fit a linear model
  model <- lm(time ~ array, data = data)

  # Plot the data and the fitted line
  plot <- ggplot(data, aes(x = array, y = time)) +
    geom_point(color = "blue") +
    geom_smooth(method = "lm", formula = y ~ x, color = "red") +
    ggtitle(title) +
    xlab("Array Size") +
    ylab("Average Transmission Time (one-way, μs)") +
    theme_minimal()

  print(plot)

  # Extract coefficients
  coefficients <- coef(model)
  latency <- coefficients[1] # Intercept
  bandwidth_inverse <- coefficients[2] # Slope
  bandwidth <- 1 / bandwidth_inverse

  return(c("Latency (μs)" = latency, "Bandwidth (bytes/μs)" = bandwidth))
}

# Perform analysis for different nodes
results_diff <- plot_and_fit(diff_time_data, "Transmission Time vs. Array Size (Different Nodes)")

# Perform analysis for the same node
results_same <- plot_and_fit(same_time_data, "Transmission Time vs. Array Size (Same Node)")

# Print results
print(results_diff)
print(results_same)
```

Output:

Diff_node:



Same_node:



Estimate value:

```
# Print results
print(results_diff)
```

```
##   Latency (sec).(Intercept) Bandwidth (bytes/sec).array
##                3.626496e-02                 5.899792e+05
```

```
print(results_same)
```

```
##   Latency (sec).(Intercept) Bandwidth (bytes/sec).array
##                   -1.389076                572422.789461
```

For the different nodes (two processes in two different nodes):
- Latency: Approximately 0.03626496 seconds
- Bandwidth: Approximately 589,979 bytes per second

For the same node (two processes on the same node):
- Latency: Approximately -1.389076 seconds (which is not physically meaningful and should be considered as an artifact of the model fitting, effectively it can be treated as zero)
- Bandwidth: Approximately 572,422 bytes per second

In this case, the same node run faster than diff node.

Problem 2:

Array size	262144	524288	1048576
2 processes	0.000043775s	0.000046349s	0.000046137s
4 processes	0.000048155s	0.000049294s	0.000043162s
8 processes	0.000043579s	0.000032312s	0.000044462s

In this case I run this program in same node sbatch.
We can see the processes increasing but the time taken to complete tasks does not decrease as expected for a scalable MPI program. Ideally, more processes should mean faster execution, but communication overhead, load imbalance, and the inherent non-parallelizable parts of the program seem to limit this speedup. In short, the MPI program exhibits limited scalability based on the times recorded for the different numbers of processes.

Problem 3:

Array size	262144	524288	1048576
4 processes on the same node	0.020212s	0.041939s	0.088085s
4 processes on 4 different nodes	0.039590s	0.076377s	0.099020s

Scalability on the Same Node:
The execution time increases with the array size, which is expected. However, the rate of increase seems sublinear, suggesting that the program scales quite well when adding more data within the same node. This could be due to the efficient use of shared resources like memory and cache, and avoiding the network overhead associated with communication between different nodes.

Scalability on Different Nodes:
Execution times are higher across the board compared to the same node scenario, which indicates network communication overhead. This overhead is particularly impactful in distributed systems where data must be sent over the network. The times increase with the array size, and the rate of increase seems slightly higher than within the same node, suggesting that network latency and bandwidth are becoming significant factors in the program's overall scalability.

The observation that 2 processes are slower than 4 suggests that the MPI program might have a high setup overhead or benefits from optimized communication with more processes. This implies that the program may not simply be limited by raw communication costs but also by how effectively it utilizes available resources and parallelizes tasks. Therefore, while 2-process communication seems less efficient, adding more processes improves performance, indicating a potential for better scalability when the program can leverage more parallelism and perhaps shared resources within a node.