# CSC453: Group project
# Parallel Bar Chart Generation

Tarfa Albanyan, Rawan Alotaibi, Norah Alguraishi, Sarah Alajlan and Nouf Alshaya

School of Computer Science, College of Computer and Information Sciences, KSU
Riyadh, Saudi Arabia

442201854@STUDENT.KSU.EDU.SA 442201374@STUDENT.KSU.EDU.SA 442201375@STUDENT.KSU.EDU.SA 442202314@STUDENT.KSU.EDU.SA

442200385@STUDENT.KSU.EDU.SA

**Abstract.** This code implements sequential, parallel using OpenMP, and parallel using MPI algorithms for finding the maximum value in an array and generating a bar chart based on the frequency of each element in the array. The sequential algorithm iterates through the array to find the maximum value and then counts the occurrence of each element. The OpenMP parallel algorithm divides the work between multiple threads to perform the same tasks concurrently, utilizing the parallel processing capabilities of modern processors. The MPI parallel algorithm distributes the workload among multiple processes, enabling parallel execution across different compute nodes. The performance of all three algorithms is evaluated in terms of time taken to execute and their scalability with varying array sizes, number of threads (for OpenMP), and number of processes (for MPI).

## 1 Introduction

This code endeavors to assess and compare the performance of sequential, OpenMP, and MPI algorithms designed for finding the maximum value in an array and constructing a bar chart based on element frequency. Section 2 illuminates the problem's essence, providing a comprehensive understanding of the program's objective. Following this, Section 3 delves into the sequential algorithm, offering both pseudo code and a performance analysis, emphasizing its linear execution pattern. Section 4 introduces the OpenMP parallel algorithm, elucidating its pseudo code and discussing performance results, particularly highlighting the advantages of parallelization. Section 5 extends the parallelization to distributed systems with the MPI algorithm, detailing the pseudo code and addressing complexities, scalability, and execution times. In Section 6, a performance graph compares the three codes, and a detailed discussion ensues to analyze the relative efficiency and scalability of each approach. The conclusion in Section 7 summarizes key findings, while Section 8 provides a list of references, in Section 9 provides a task distribution and Section 10 Houses an appendix for supplementary materials.

## 2 Problem definition

Given an array of integers, the problem is to find the maximum value in the array and generate a bar chart that represents the frequency of each element. The sequential algorithm iterates through the array to find the maximum value and then counts the occurrence of each element. The OpenMP parallel algorithm divides the work between multiple threads to perform the same tasks concurrently, exploiting the potential for parallelism in the problem.

## 3 Sequential Algorithm

### 3.1 Pseudo code

```
maxValueS → 1
FOR i ← 0 to size-1 DO
   IF numArray[i] > maxValueS THEN
      maxValueS → numArray[i]
   END IF
END FOR

Declare an array numCountArrayS of size (maxValueS + 1)

FOR i ← 1 to maxValueS DO
   numCountS → 0
   FOR j ← 0 to size-1 DO
      IF i = numArray[j] THEN
         numCountS → numCountS + 1
      END IF
   END FOR
   numCountArrayS[i] → numCountS
END FOR

PRINT "--- Bar chart ---\n"
FOR i ← 1 to maxValueS DO
   PRINT "\nData Point i: "
   FOR j ← 1 to numCountArrayS[i] DO
      PRINT "*"
   END FOR
END FOR
```

### 3.2 Performance

**Performance Analysis:**

The algorithm first finds the maximum value in the array (maxValueS).

Then, it initializes an array (numCountArrayS) to count the occurrences of each value from 1 to maxValueS in the original array.

Finally, it prints a bar chart based on the counts.

The time complexity of the algorithm can be analyzed in two parts:

**Finding the maximum value: O(N)**

**Counting occurrences and printing the bar chart**:

O(N * maxValueS) the overall time complexity of the algorithm is dominated by the second part since maxValueS can be significantly larger than N.

**Big O Notation:**

The time complexity of the provided code can be expressed as O(N* maxValueS), where N is the size of the array and maxValueS is the maximum value in the array. The space complexity is O(maxValueS) due to the numCountArrayS array.

**Performance Results:**

The average time for 10 runs is approximately 9.29 seconds.

The performance is measured with a dataset size of 2,000,000 elements. (figure 1)

## 4 OpenMP Algorithm

### 4.1 OpenMP pseudo code

```
maxValueS → 1

do in parallel
FOR i ← 0 to size-1 DO
   IF numArray[i] > maxValueS THEN
      maxValueS → numArray[i]
   END IF
END FOR

Declare an array numCountArrayS of size (maxValueS + 1)
FOR i ← 1 to maxValueS DO
   numCountS → 0
do in parallel
   FOR j ← 0 to size-1 DO
      IF i = numArray[j] THEN
```

```
          numCountS → numCountS + 1
        END IF
    END FOR
    numCountArrayS[i] → numCountS
END FOR

do in parallel
PRINT "--- Bar chart ---\n"
FOR i ← 1 to maxValueS DO
    PRINT "\nData Point i: "
    FOR j ← 1 to numCountArrayS[i] DO
        PRINT "*"
    END FOR
END FOR
```

## 4.2 OpenMP performance

For an example an array with 12 elements and 3 threads :

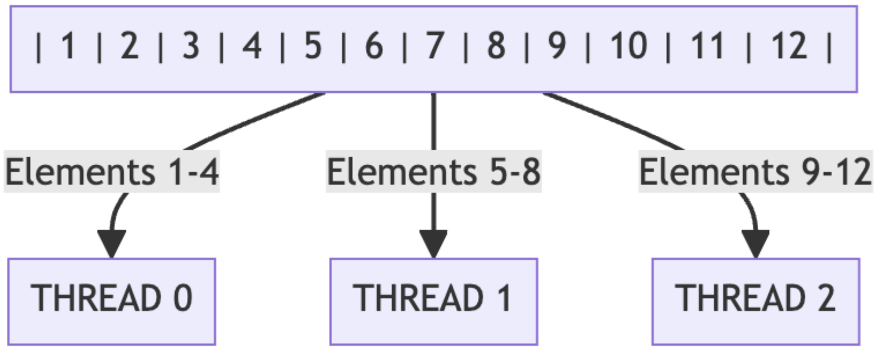


*figure 1: The division of work between the threads in OpenMP.*

**Performance Analysis (OpenMP):**

The OpenMP parallelized code follows a structured approach, starting with the parallelization of the process to find the maximum value (maxValue) in the array using the reduction(max : maxValue) clause. It then employs parallelization to count the occurrences of each value and generate a bar chart. The time complexity breakdown is as follows:

**Finding the Maximum Value (OpenMP):**
The parallelized process for finding the maximum value in the array has a time complexity of O(N/p), where N is the size of the array, and p is the number of threads.

**Counting Occurrences and Printing the Bar Chart (OpenMP):**
The parallelized counting and chart printing phase also has a time complexity of O(N/p) and O(1) respectively. The time after running our code on array size 2000 on 2 threads: 0.063542 (figure 6), 4 threads: 0.263320 (figure 7), 8 threads: 0.063542 (figure 7), 16 threads : 0.263320 (figure 8)and after running 10 times the average time for 8 threads: 0.035681 (figure 9).

**Big O Notation (OpenMP):**

The Big O notation for the OpenMP parallelized code can be expressed as O(N/p), where N is the size of the array, and p represents the number of threads utilized by OpenMP. The parallelization aims to distribute the workload among threads, leading to improved performance.

**Performance Results (OpenMP):**
The average time for 10 runs is approximately 3.074864 seconds.
The performance is measured with a dataset size of 2,000,000 elements. (figure 10)

## 5 MPI Algorithm

### 5.1 MPI pseudo code

```
START

Initialize MPI

Determine the rank (ID) of the current process and the total number of processes (numprocs)

IF rank = 0 THEN

    PRINT "Enter the size of the array: "

    READ size from user

END IF

Broadcast 'size' to all processes

Declare an array numArray of size 'size'
```

```
IF rank = 0 THEN
    Seed the random number generator
    FOR i ← 0 to size-1 DO
        numArray[i] → random number between 1 and size
    END FOR
END IF
Broadcast numArray to all processes
Start timer
Calculate elements per process and determine start and end indices for each process
elements_per_proc → size / numprocs
extra_elements → size % numprocs
start → rank * elements_per_proc + (rank < extra_elements ? rank : extra_elements)
end → start + elements_per_proc + (rank < extra_elements ? 1 : 0)
Find local maximum
localMax → 0
FOR i ← start to end-1 DO
    IF numArray[i] > localMax THEN
        localMax → numArray[i]
    END IF
END FOR
Reduce localMax to find global maximum (maxValue) at process 0
Broadcast maxValue to all processes
Declare an array numCountArray of size (maxValue + 1)
```

```
FOR i ← 0 to maxValue DO
   numCountArray[i] → 0
END FOR
Count occurrences in each process's portion of the array
FOR i ← start to end-1 DO
   numCountArray[numArray[i] - 1] → numCountArray[numArray[i] - 1] + 1
END FOR
Gather all numCountArray to process 0 into globalNumCounts array
IF rank = 0 THEN
   PRINT "--- Bar chart ---"
   FOR i ← 1 to maxValue DO
      PRINT "Data Point i: "
      FOR j ← 0 to numprocs-1 DO
         FOR k ← 1 to globalNumCounts[j * maxValue + i - 1] DO
            PRINT "*"
         END FOR
      END FOR
   END FOR
END IF
Stop timer
IF rank = 0 THEN
   PRINT execution time
END IF
```

```
Finalize MPI
END
```

## 5.2 MPI performance

The graph below shows How the work is divide between the thread

where E = size\numprocs the (size) is the total number of elements in the array, and (numprocs) is the total number of processes.

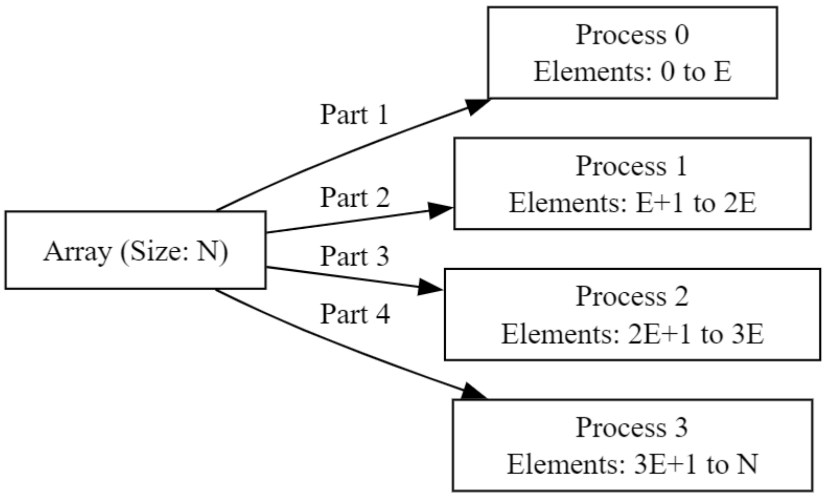


*figure 2: The division of work between the threads in MPI.*

The complexity of the algorithm (big O) with variable array size and variable number of processors O(N/p). The time after running our code on array size 2000 on 2 threads: 0.001784 (figure 11), 4 threads: 0.003691 (figure 12), 8 threads: 0.052934 (figure 13), 16 threads : 0.108509 (figure 14) and after running 10 times the average time for 8 threads: 0.0297869 you can refer to figure 15 to figure 24.

## 6 Result and discussion

The performance of our code with 2000 array size shows that the MPI outperforms the others, particularly with fewer threads,achieving the lowest runtime of 0.001784 seconds. Although MPI's runtime increases with more threads, it remains below the runtimes of sequential and OpenMP for lower thread count, showing superior efficiency. The sequential code, with a runtime of 0.011001 seconds, serves as a baseline. OpenMP's performance is slower than sequential with 2 and 8 threads and significantly deteriorates with 4 and 16 threads, possibly due to thread management overhead or inefficient parallelization. since MPI's runtime increases with more threads that suggests an increased communication overhead in higher thread counts. This highlights MPI's effectiveness in parallel processing, particularly in scenarios with lower thread counts.

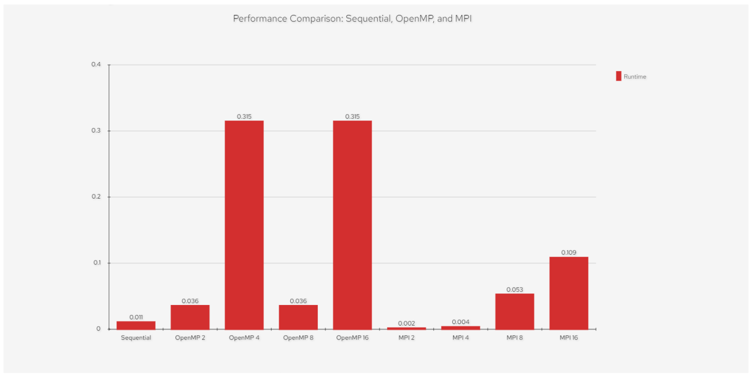


*figure 3:Preformance Comparison Sequential, OpenMP, and MPI.*

The performance of our code with 20000 array size shows that the sequential execution, with a runtime of 2.009415 seconds, is markedly slower, underscoring the advantages of parallel computing. OpenMP shows improved performance when increasing threads from 2 to 4, but its efficiency declines with 8 and 16 threads, indicating possible issues like thread overhead. On the other hand, MPI maintains a steady performance enhancement as threads increase, starting from a notably low runtime of 0.039211 seconds with 2 threads and gradually rising with higher thread counts, and still significantly outperforming the other methods. This pattern highlights MPI's effectiveness in handling parallel tasks, especially with more threads, and suggests potential inefficiencies in OpenMP's approach at higher thread levels.

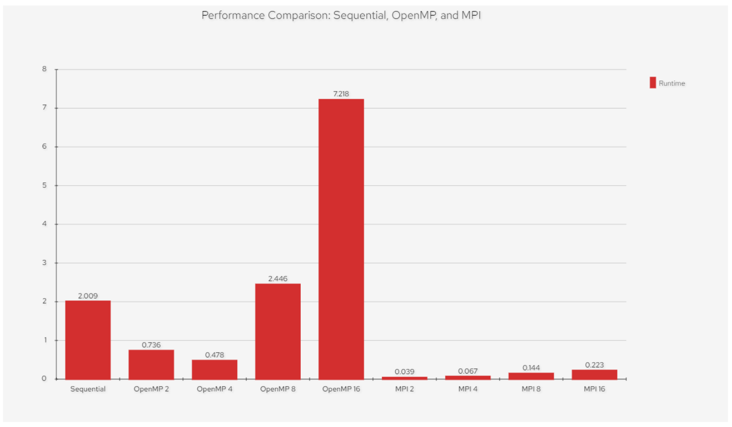


*figure 4:Preformance Comparison Sequential, OpenMP, and MPI.*

## 7 Conclusion and future work

In our project, we explored the efficiency of sequential, OpenMP, and MPI algorithms in the context of finding the maximum value in an array and generating parallel bar charts. Our findings reveal a clear superiority of MPI in terms of performance, particularly with fewer threads, showing its robustness in parallel processing. The sequential approach, while serving as a baseline, was significantly slower, underscoring the advantages of parallel computing. The OpenMP implementation, although beneficial in certain scenarios, had limitations, particularly at higher thread counts, possibly due to increased thread overhead.

For future work, it would be advantageous to delve deeper into optimizing the OpenMP implementation, especially at higher thread levels, to reduce overhead and improve efficiency. Additionally, exploring the scalability of MPI with larger datasets and more complex computational tasks could provide further insights into its potential and limitations.’

## 8- References

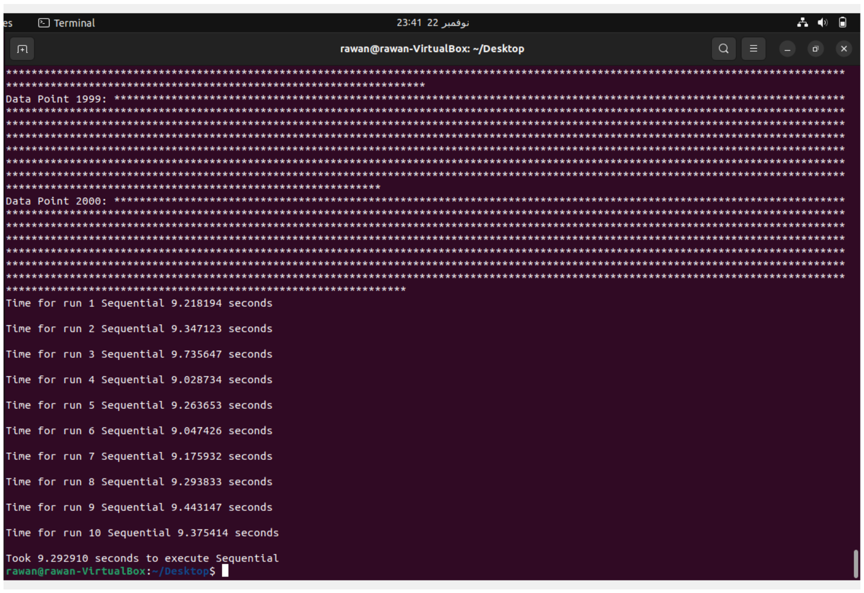
[1] "Lecture Notes in Computer Science LNCS | Springer | Springer — International Publisher," *Springer.com*, 2023. https://www.springer.com/gp/computer-science/lncs (accessed Nov. 25, 2023).

[2] "Introduction to Parallel Processing Algorithms and Architectures." Accessed: Nov. 25, 2023. [Online]. Available: https://doc.lagout.org/science/0_Computer%20Science/5_Parallel%20and%20Distributed/Introduction%20To%20Parallel%20Processing%20-%20Algorithms%20And%20Architectures.pdf

## 9 -Task Distribution

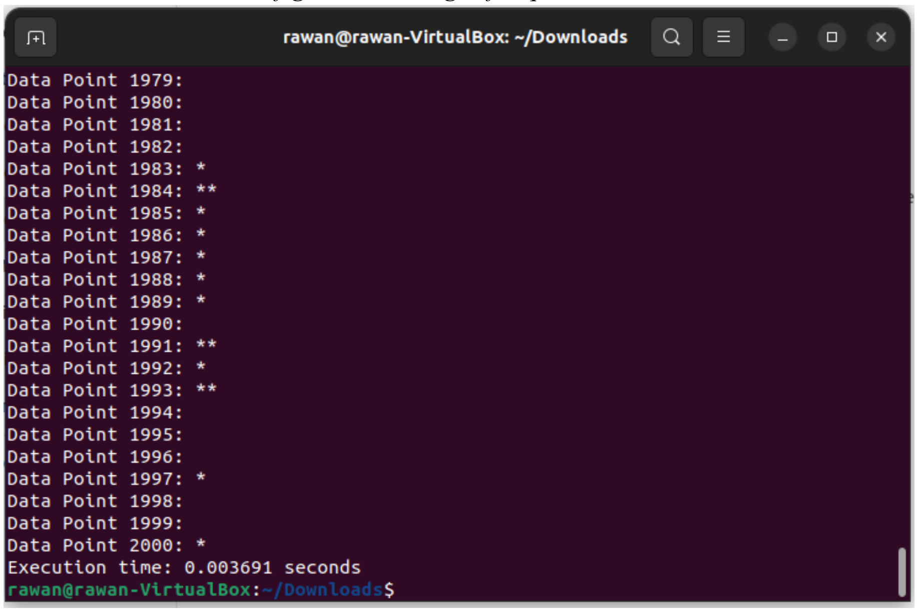
| Task | Students |
|---|---|
| Introduction and problem definition | Nouf Alshaya, Rawan Alotaib |
| Sequential | Tarfa Albanyan and Norah Alguraishi |
| OpenMP | Nouf Alshaya and Sarah Alajlan |
| MPI | Tarfa Albanyan and Rawan Alotaibi |
| Result and conclusion | Sarah Alajlan and Norah Alguraishi |
| Review | All |

## 10. Appendix A



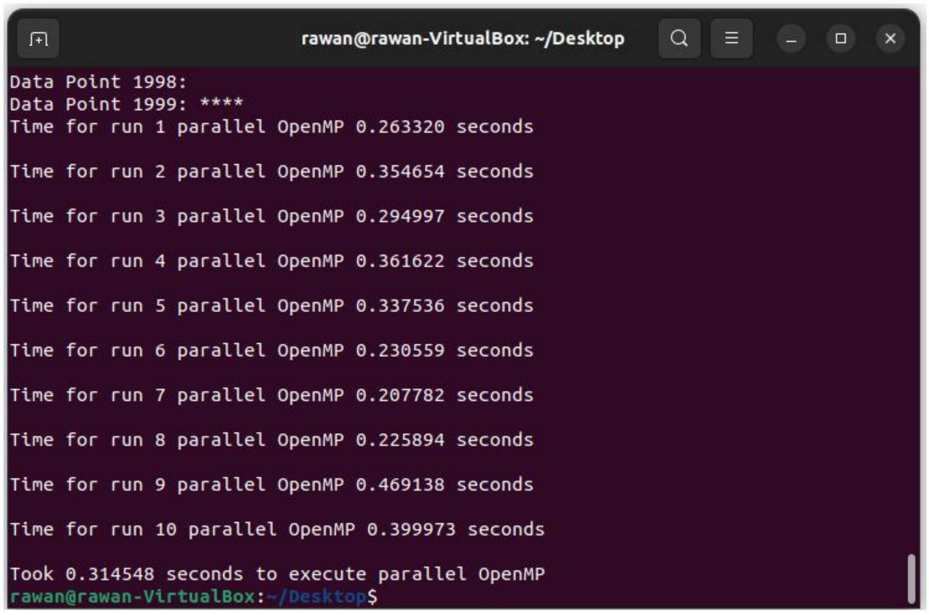
```
************************************************************************************************************************************
******************************************************************
Data Point 1999: *******************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
***********************************************************
Data Point 2000: *******************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
************************************************************************************************************************************
***************************************************************
Time for run 1 Sequential 9.218194 seconds

Time for run 2 Sequential 9.347123 seconds

Time for run 3 Sequential 9.735647 seconds

Time for run 4 Sequential 9.028734 seconds

Time for run 5 Sequential 9.263653 seconds

Time for run 6 Sequential 9.047426 seconds

Time for run 7 Sequential 9.175932 seconds

Time for run 8 Sequential 9.293833 seconds

Time for run 9 Sequential 9.443147 seconds

Time for run 10 Sequential 9.375414 seconds

Took 9.292910 seconds to execute Sequential
rawan@rawan-VirtualBox:~/Desktop$ 
```

*figure 5: Average of sequential time.*



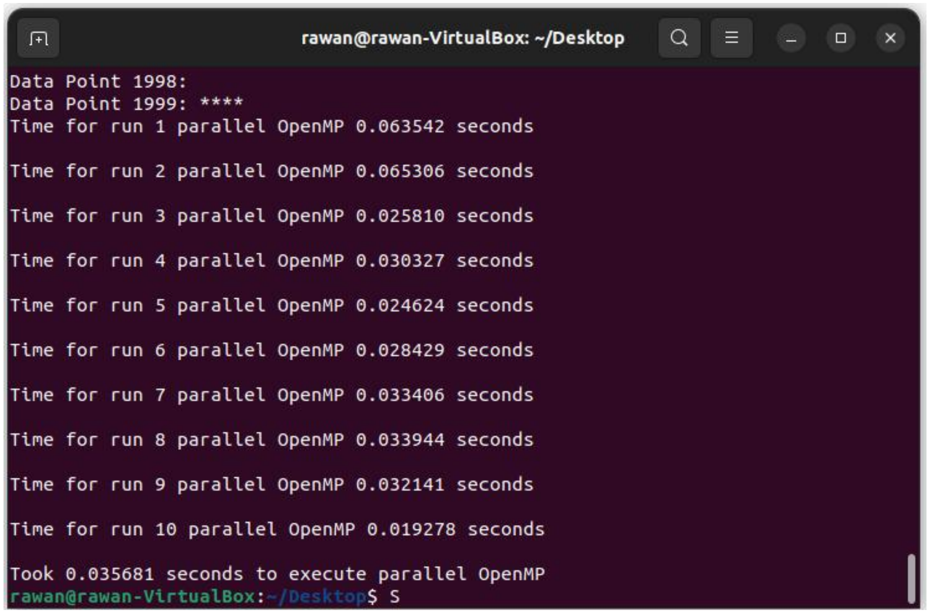
```
Data Point 1979:
Data Point 1980:
Data Point 1981:
Data Point 1982:
Data Point 1983: *
Data Point 1984: **
Data Point 1985: *
Data Point 1986: *
Data Point 1987: *
Data Point 1988: *
Data Point 1989: *
Data Point 1990:
Data Point 1991: **
Data Point 1992: *
Data Point 1993: **
Data Point 1994:
Data Point 1995:
Data Point 1996:
Data Point 1997: *
Data Point 1998:
Data Point 1999:
Data Point 2000: *
Execution time: 0.003691 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 6: Time for OpenMP with 2 processors*



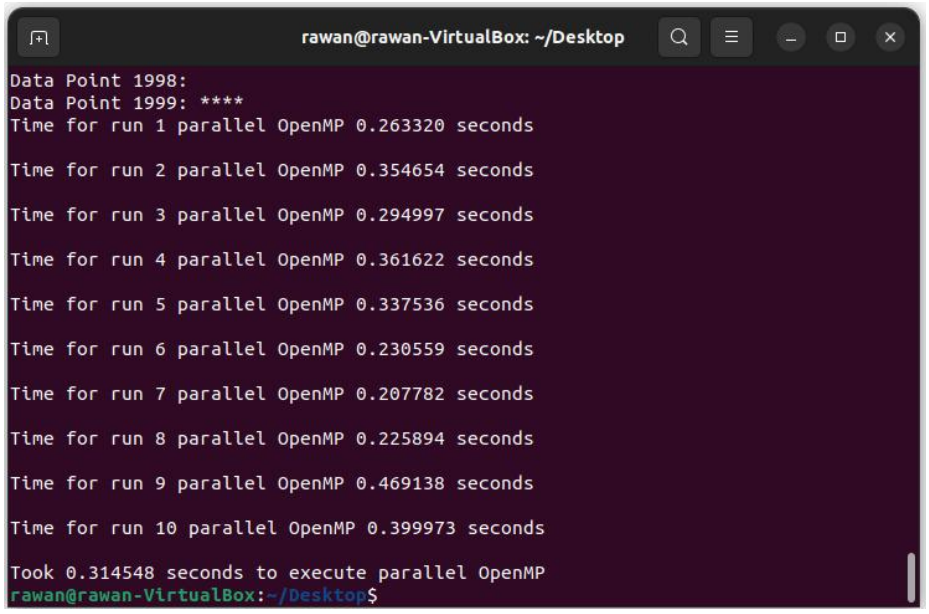
```
Data Point 1998:
Data Point 1999: ****
Time for run 1 parallel OpenMP 0.263320 seconds

Time for run 2 parallel OpenMP 0.354654 seconds

Time for run 3 parallel OpenMP 0.294997 seconds

Time for run 4 parallel OpenMP 0.361622 seconds

Time for run 5 parallel OpenMP 0.337536 seconds

Time for run 6 parallel OpenMP 0.230559 seconds

Time for run 7 parallel OpenMP 0.207782 seconds

Time for run 8 parallel OpenMP 0.225894 seconds

Time for run 9 parallel OpenMP 0.469138 seconds

Time for run 10 parallel OpenMP 0.399973 seconds

Took 0.314548 seconds to execute parallel OpenMP
rawan@rawan-VirtualBox:~/Desktop$
```

*figure 7:Time for OpenMP with 4 processors*



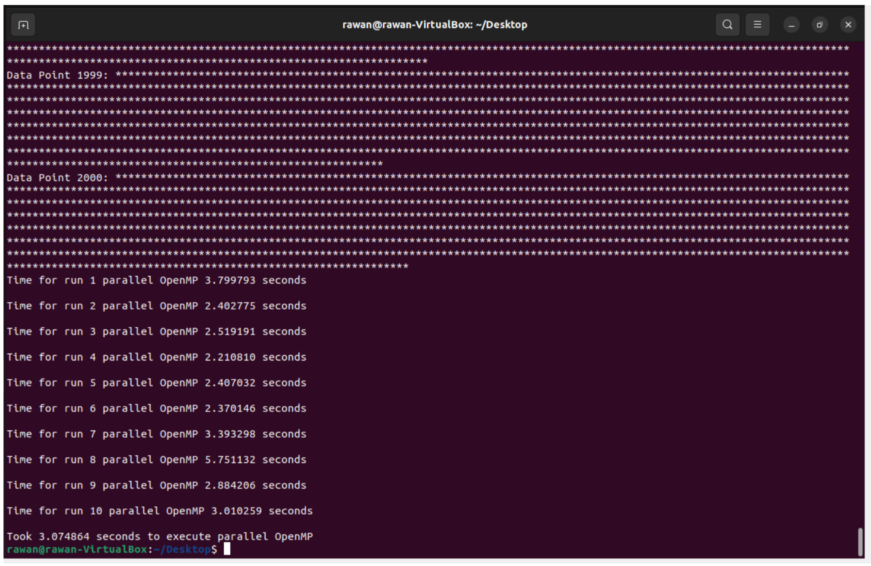
```
Data Point 1998:
Data Point 1999: ****
Time for run 1 parallel OpenMP 0.063542 seconds

Time for run 2 parallel OpenMP 0.065306 seconds

Time for run 3 parallel OpenMP 0.025810 seconds

Time for run 4 parallel OpenMP 0.030327 seconds

Time for run 5 parallel OpenMP 0.024624 seconds

Time for run 6 parallel OpenMP 0.028429 seconds

Time for run 7 parallel OpenMP 0.033406 seconds

Time for run 8 parallel OpenMP 0.033944 seconds

Time for run 9 parallel OpenMP 0.032141 seconds

Time for run 10 parallel OpenMP 0.019278 seconds

Took 0.035681 seconds to execute parallel OpenMP
rawan@rawan-VirtualBox:~/Desktop$ S
```

*figure 8: Time for OpenMP with 8 processors*



```
Data Point 1998:
Data Point 1999: ****
Time for run 1 parallel OpenMP 0.263320 seconds

Time for run 2 parallel OpenMP 0.354654 seconds

Time for run 3 parallel OpenMP 0.294997 seconds

Time for run 4 parallel OpenMP 0.361622 seconds

Time for run 5 parallel OpenMP 0.337536 seconds

Time for run 6 parallel OpenMP 0.230559 seconds

Time for run 7 parallel OpenMP 0.207782 seconds

Time for run 8 parallel OpenMP 0.225894 seconds

Time for run 9 parallel OpenMP 0.469138 seconds

Time for run 10 parallel OpenMP 0.399973 seconds

Took 0.314548 seconds to execute parallel OpenMP
rawan@rawan-VirtualBox:~/Desktop$
```

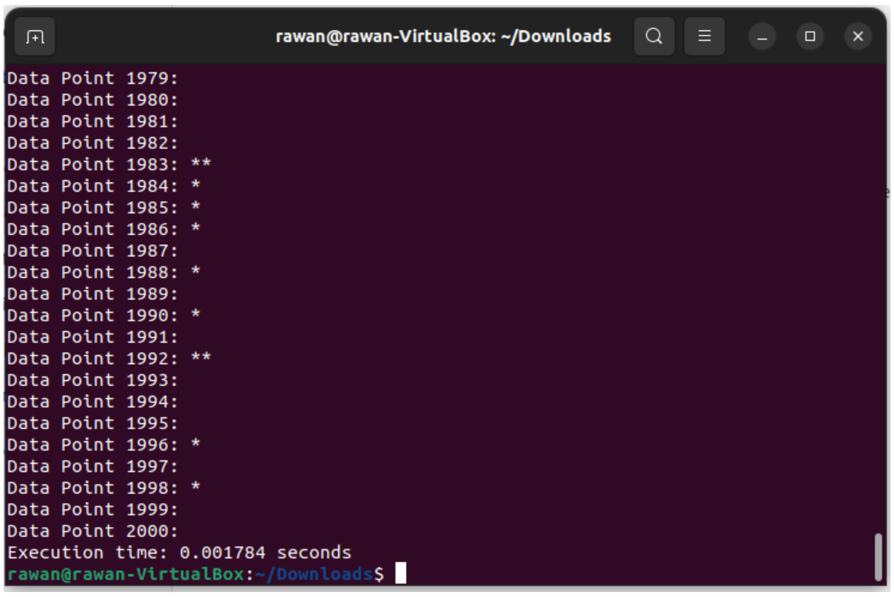
figure 9:Time for OpenMP with 16 processors



```
***********************************************************************************************************************************
******************************************************************
Data Point 1999: ******************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************
Data Point 2000: ******************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***********************************************************************************************************************************
***************************************************************
Time for run 1 parallel OpenMP 3.799793 seconds

Time for run 2 parallel OpenMP 2.402775 seconds

Time for run 3 parallel OpenMP 2.519191 seconds

Time for run 4 parallel OpenMP 2.210810 seconds

Time for run 5 parallel OpenMP 2.407032 seconds

Time for run 6 parallel OpenMP 2.370146 seconds

Time for run 7 parallel OpenMP 3.393298 seconds

Time for run 8 parallel OpenMP 5.751132 seconds

Time for run 9 parallel OpenMP 2.884206 seconds

Time for run 10 parallel OpenMP 3.010259 seconds

Took 3.074864 seconds to execute parallel OpenMP
rawan@rawan-VirtualBox:~/Desktop$
```

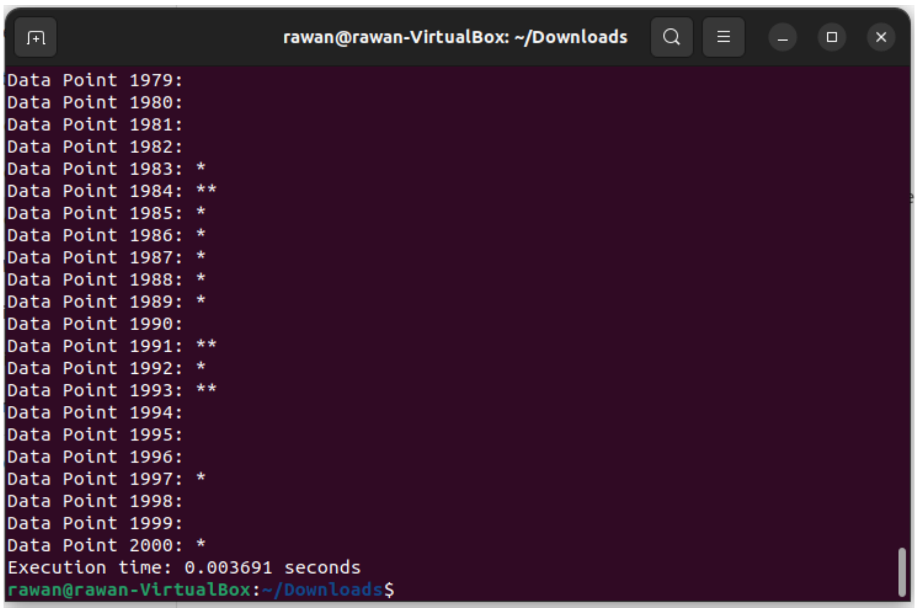
figure 10: average OpenMP parallel time.

1



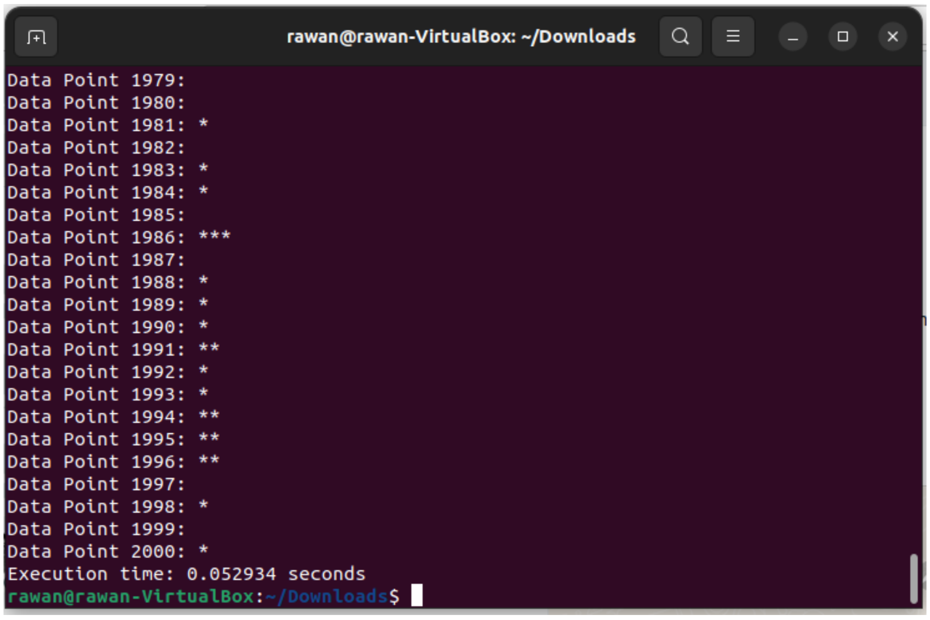
```
Data Point 1979:
Data Point 1980:
Data Point 1981:
Data Point 1982:
Data Point 1983: **
Data Point 1984: *
Data Point 1985: *
Data Point 1986: *
Data Point 1987:
Data Point 1988: *
Data Point 1989:
Data Point 1990: *
Data Point 1991:
Data Point 1992: **
Data Point 1993:
Data Point 1994:
Data Point 1995:
Data Point 1996: *
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000:
Execution time: 0.001784 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 11 : Time for MPI with 2 processors*



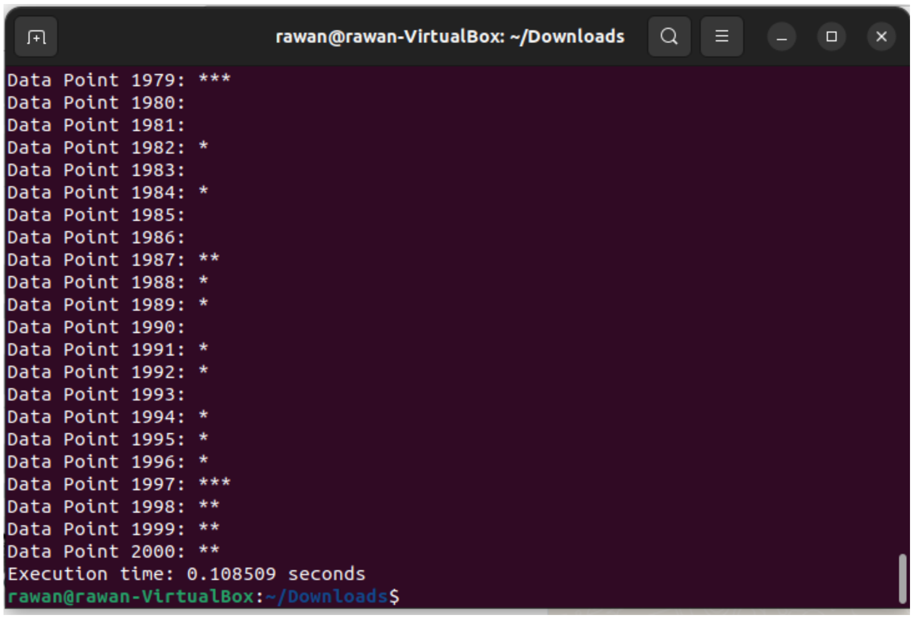
```
Data Point 1979:
Data Point 1980:
Data Point 1981:
Data Point 1982:
Data Point 1983: *
Data Point 1984: **
Data Point 1985: *
Data Point 1986: *
Data Point 1987: *
Data Point 1988: *
Data Point 1989: *
Data Point 1990:
Data Point 1991: **
Data Point 1992: *
Data Point 1993: **
Data Point 1994:
Data Point 1995:
Data Point 1996:
Data Point 1997: *
Data Point 1998:
Data Point 1999:
Data Point 2000: *
Execution time: 0.003691 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 12 : Time for MPI with 4 processors*



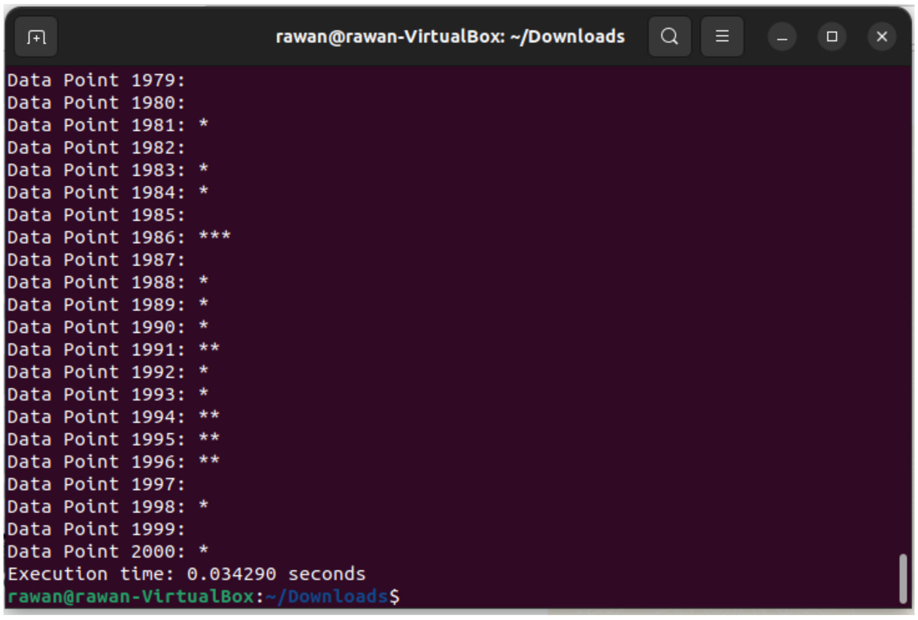
```
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.052934 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 13 : Time for MPI with 8 processors*



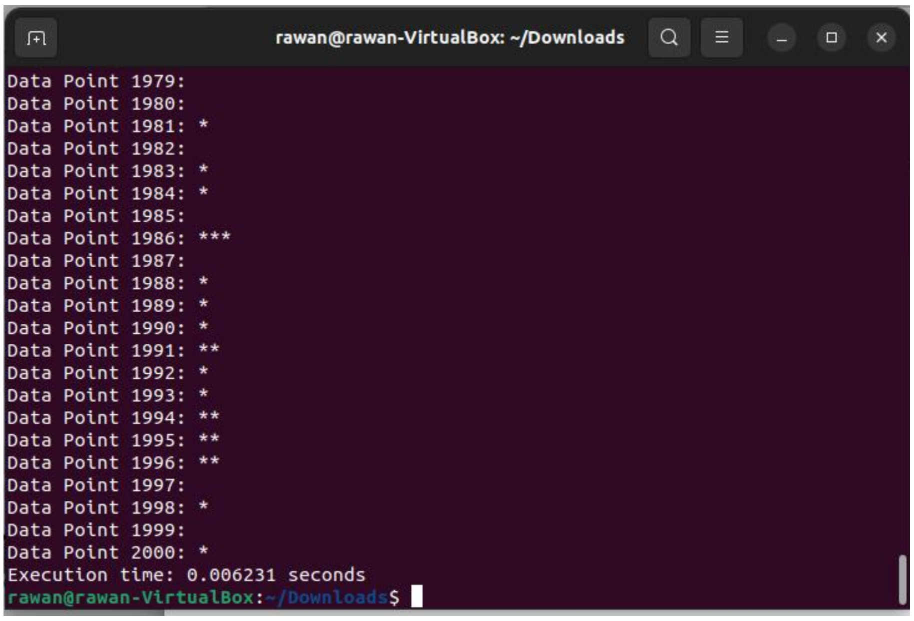
```
Data Point 1979: ***
Data Point 1980:
Data Point 1981:
Data Point 1982: *
Data Point 1983:
Data Point 1984: *
Data Point 1985:
Data Point 1986:
Data Point 1987: **
Data Point 1988: *
Data Point 1989: *
Data Point 1990:
Data Point 1991: *
Data Point 1992: *
Data Point 1993:
Data Point 1994: *
Data Point 1995: *
Data Point 1996: *
Data Point 1997: ***
Data Point 1998: **
Data Point 1999: **
Data Point 2000: **
Execution time: 0.108509 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 14 : Time for MPI with 16 processors*



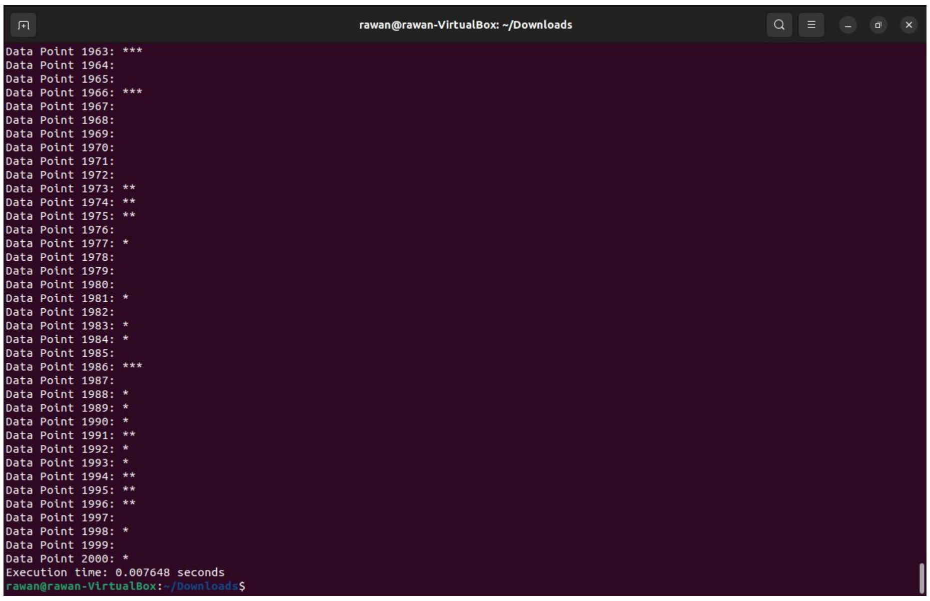
```
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.034290 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 15 : Run for MPI time 1.*



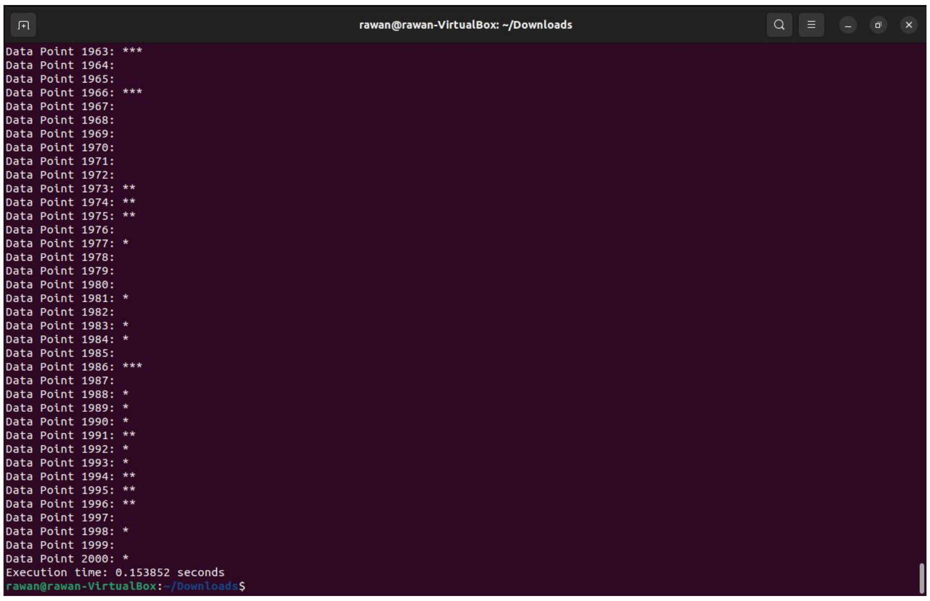
```
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.006231 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 16 : Run for MPI time 2.*



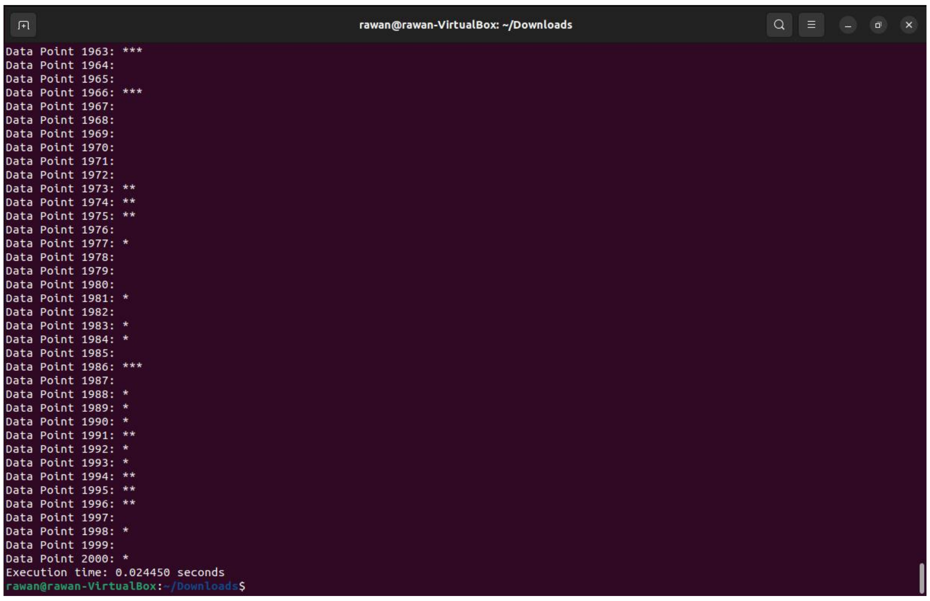
```
rawan@rawan-VirtualBox: ~/Downloads
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.007648 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 17 :Run for MPI time 3.*



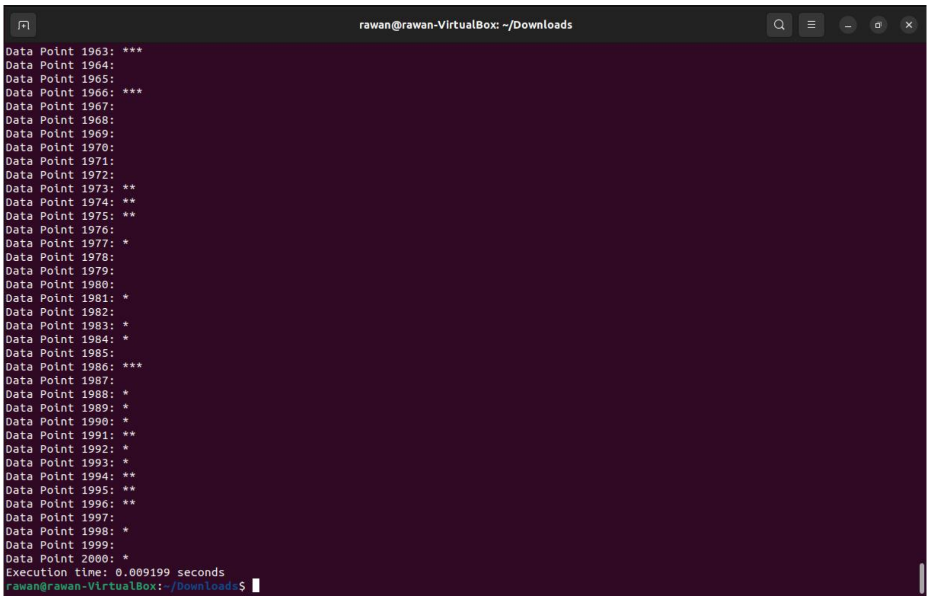
```
rawan@rawan-VirtualBox: ~/Downloads
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.153852 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 18 :Run for MPI time 4.*



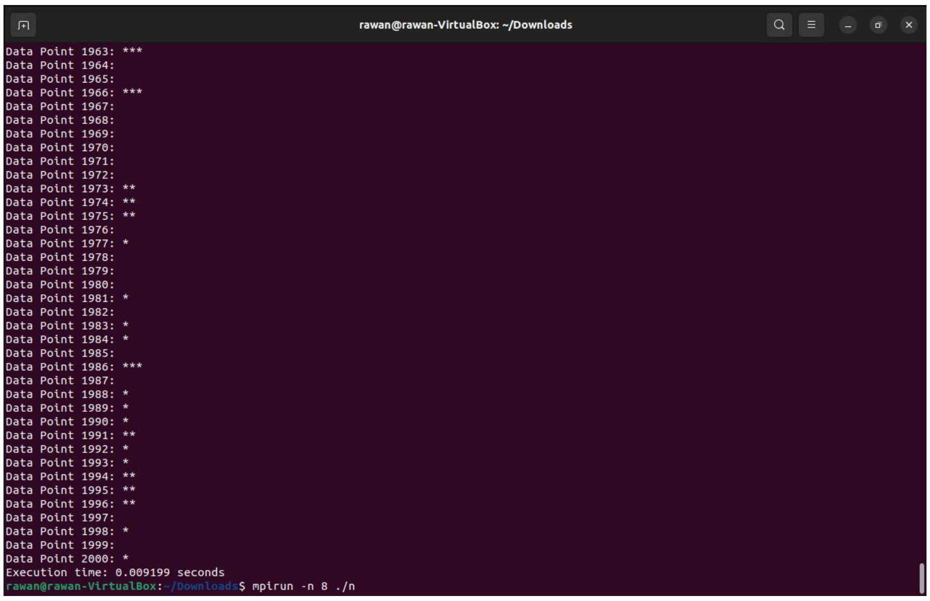
```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.024450 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 19 :Run for MPI time 5.*



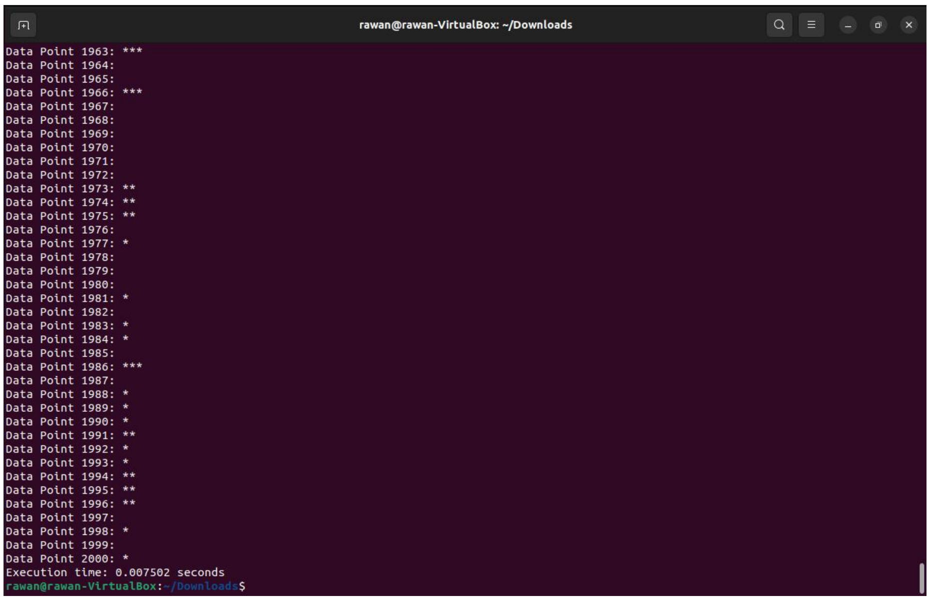
```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.009199 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 20 :Run for MPI time 6.*


rawan@rawan-VirtualBox: ~/Downloads

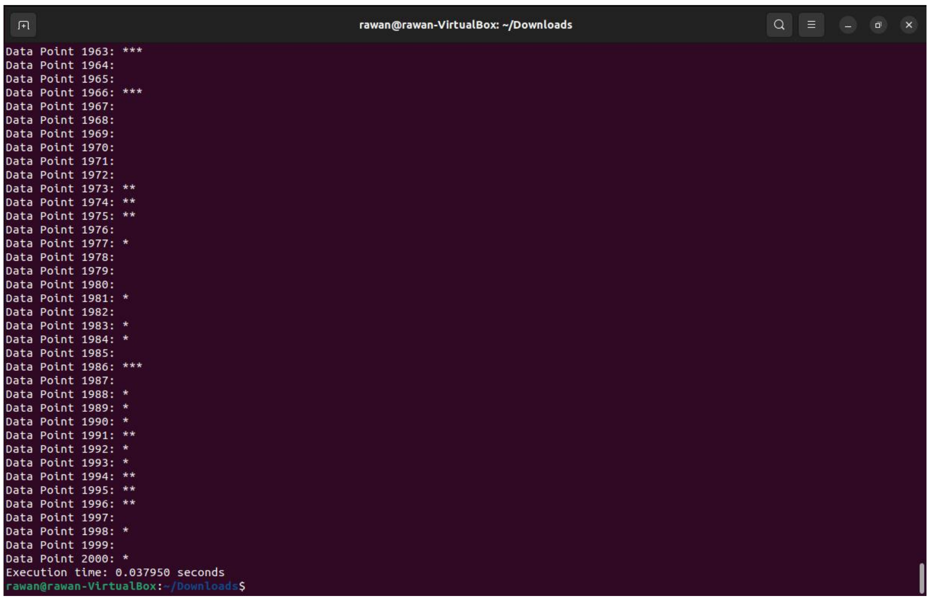
```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.009199 seconds
rawan@rawan-VirtualBox:~/Downloads$ mpirun -n 8 ./n
```

*figure 21  :Run for MPI time 7.*


rawan@rawan-VirtualBox: ~/Downloads

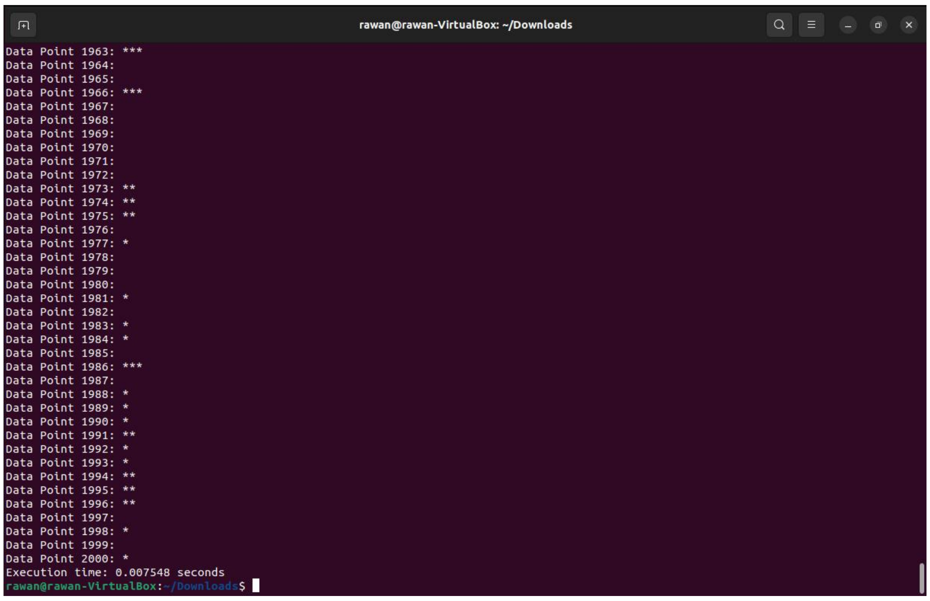
```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.007502 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 22 :Run for MPI time 8.*



```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.037950 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 23 :Run for MPI time 9.*



```
Data Point 1963: ***
Data Point 1964:
Data Point 1965:
Data Point 1966: ***
Data Point 1967:
Data Point 1968:
Data Point 1969:
Data Point 1970:
Data Point 1971:
Data Point 1972:
Data Point 1973: **
Data Point 1974: **
Data Point 1975: **
Data Point 1976:
Data Point 1977: *
Data Point 1978:
Data Point 1979:
Data Point 1980:
Data Point 1981: *
Data Point 1982:
Data Point 1983: *
Data Point 1984: *
Data Point 1985:
Data Point 1986: ***
Data Point 1987:
Data Point 1988: *
Data Point 1989: *
Data Point 1990: *
Data Point 1991: **
Data Point 1992: *
Data Point 1993: *
Data Point 1994: **
Data Point 1995: **
Data Point 1996: **
Data Point 1997:
Data Point 1998: *
Data Point 1999:
Data Point 2000: *
Execution time: 0.007548 seconds
rawan@rawan-VirtualBox:~/Downloads$
```

*figure 24: Run for MPI time 10.*