CSC453: Group project
An Experimental Study of a Parallel Vectors Multiplication Algorithm

Lama Alrehaily, Enas Alzahrani, Aljwhra Almakhdoub, Reema Alqamshaa, Maha Almunawer and Raseel Alhaqbani

School of Computer Science, College of Computer and Information Sciences, KSU

Riyadh, Saudi Arabia
441200821@STUDENT.KSU.EDU.SA
441201082@STUDENT.KSU.EDU.SA
441201225@STUDENT.KSU.EDU.SA
441201227@STUDENT.KSU.EDU.SA
441201279@STUDENT.KSU.EDU.SA
441201421@STUDENT.KSU.EDU.SA

Abstract. Multiplying vectors sequentially takes a long time, especially when they are large, but reducing the calculation time by splitting the vector and then performing parallel calculations then combining the results is more efficient than sequential calculations and would have a significant impact on the time and performance. In this project, our goal is to create sequential and parallel vector multiplication algorithms, multiply the vectors several times with various sizes, and then compare the average time taken by each algorithm in each size. The algorithms we designed are the Sequential algorithm, OpenMP algorithm, and MPI algorithm. The MPI algorithm was the most efficient and took the least amount of time, followed by OpenMP, and then the Sequential algorithm which was the worst; it took the most amount of time. We conclude that splitting the vectors and performing parallel methods are more efficient and produce better results in terms of time and efficiency than sequential methods.

1 Introduction

An arrangement of numbers in rows and columns is called a matrix. A matrix's numbers might be either data or mathematical formulae. And a vector is a row or column of numbers. They are both utilized as a quick technique to approximate more difficult calculations. They were first developed to provide a description for systems of linear equations. Computers may multiply matrices or vectors to solve linear equation systems in applications like image processing and to give quick but good approximations of more complex calculations [1]. Using parallelism in a matrix or vector multiplication will make the code more efficient, make the calculations much faster, and ease the usage of very large data. Our goal in this project is to use OpenMP and MPI to parallelize the vector multiplication algorithm and compare between the parallel programs' speed.

In our report we state the definition of our problem, and how we solved it in three algorithms: Sequential, OpenMP, and OpenMPI. As well as the performance and time of each algorithm, and finally, the results, discussion, and conclusion.

2 Problem definition

The binary operation known as matrix multiplication creates a matrix from two matrices. The first matrix's columns must have the same number of rows as the second matrix's rows in order for matrices to be multiplied. The first matrix's number of rows and the second matrix's number of columns are combined to form the final matrix, or the matrix product [2].

$$\begin{bmatrix} 3 & 4 & 6 \\ 9 & 7 & 11 \\ 2 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 4 & 2 \\ 8 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 56 & 42 \\ 114 & 83 \\ 90 & 64 \end{bmatrix}$$

Fig.1. Matrix Multiplication Example

Vectors can be multiplied in two different ways. The dot product of two vectors and the cross product of two vectors are the two methods for multiplying vectors based on the fact that a vector contains both magnitude and direction. Given that the resultant value is a scalar quantity, the dot product of two vectors is also referred to as the scalar product. Due to the fact that the cross product produces a vector that is perpendicular to the original two vectors, it is known as the vector product [3].
Scalar product is the way of multiplication we will be using in our project however they will be one dimensional either a row or a column of a certain length. And the multiplication of two vectors will create a vector of the same length, and then the result of each multiplication is added up to conclude the result [4].

$$\begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} \times [6 \quad 2 \quad 9] = [30 \quad + \quad 4 \quad + \quad 36] = 70$$

Fig.2. Vector Multiplication Example

3 Sequential Algorithm

3.1 Pseudo code

```
Total := 0
    for i = 0 to size of vector - 1 do:
     Mx := V1x*V2x;
     Total = Total + Mx;
    end for
```

3.2 Performance

The complexity of the algorithm is O(n) on one processor with any variable vector size. The following table shows the average time for different n values:

n	Time in 10 runs	Average time
n = 1000	1= 10s 2= 11s 3= 22s 4= 14s 5= 10s 6= 17s 7= 10s 8= 24s 9= 9s 10= 48s	17.5 Seconds
n = 1500	1= 22s 2= 20s 3= 21s 4= 20s 5= 25s 6= 13s 7= 11s 8= 37s 9= 18s 10= 22s	20.9 Seconds
n = 3000	1= 28s 2= 107s 3= 35s 4= 31s 5= 74s 6= 29s 7= 30s 8= 46s 9= 71s 10= 47s	49.8 Seconds

Table.1. Sequential Algorithm Runs

4 OpenMP Algorithm

4.1 OpenMP pseudo code

```
Total := 0
  Processor x, 0 <= x < t
    for some number of threads t do:
     Mx := V1x*V2x;
     Total = Total + Mx;
    end for
```

OpenMP performance

We divided the threads by using (#pragma omp parallel for) also we have used the reduction clause to calculate the total. The complexity of our code is O(n/m) where n is the vector size and m is the number of threads, so we conclude that the complexity of our code belongs to O(n) class.
We ran our code using a Linux virtual machine that has 2 processors.

Num-ber of threads	Time in 10 runs											Average
2	n=1000	0.000283	0.000262	0.000149	0.000260	0.000231	0.000444	0.000409	0.000339	0.000520	0.000197	0.000309
	n=1500	0.000188	0.000247	0.000183	0.000276	0.000494	0.000316	0.000274	0.000216	0.000351	0.000329	0.000287
	n=3000	0.000258	0.000344	0.000300	0.0.000342	0.000331	0.000229	0.000304	0.000368	0.000444	0.000336	0.000324
4	n=1000	0.007922	0.000579	0.001111	0.000426	0.000670	0.000505	0.000820	0.000575	0.001000	0.000751	0.001436
	n=1500	0.000573	0.003597	0.000805	0.001102	0.002438	0.000880	0.001130	0.000727	0.000240	0.000660	0.001215
	n=3000	0.000433	0.000690	0.000714	0.000630	0.000600	0.002773	0.000556	0.000535	0.000482	0.000466	0.000788
8	n=1000	0.001144	0.001906	0.001178	0.007492	0.001193	0.000630	0.000946	0.001357	0.000913	0.001213	0.001797
	n=1500	0.001128	0.001157	0.001140	0.000976	0.001141	0.000794	0.001292	0.001131	0.000841	0.001117	0.001072
	n=3000	0.000940	0.001234	0.001128	0.001689	0.001194	0.0015255	0.001042	0.001239	0.001282	0.001306	0.001258
16	n=1000	0.003070	0.001628	0.001321	0.001347	0.001736	0.001935	0.001576	0.001656	0.001488	0.001729	0.001749
	n=1500	0.001581	0.001730	0.001709	0.001221	0.002551	0.001729	0.001758	0.001637	0.001710	0.001331	0.001696
	n=3000	0.001864	0.001750	0.009354	0.001155	0.001260	0.002204	0.001993	0.001881	0.002386	0.001387	0.002523

Table.2. OpenMP Algorithm Runs

5 MPI Algorithm

5.1 MPI pseudo code

```
rank_total := 0
for j = 0 to size Processor x, 0<=x<p for some number of processes p do:
  rank_total := rank_total + rank_vector1j * rank_vector2j;
end for
```

5.2 MPI performance

We divide the size of the vector into roughly equal sizes. First, we check if the sizes of the vectors are divisible by the number of processes if it is then the chunks are equally divided and sent using MPI_Send/Recv operations. Otherwise, if it is not divisible then the last process will get the remaining size of the vectors and will have to do more operations. The complexity of the algorithm in terms of sequential is O(n). Where is when it is divided into processes the complexity becomes O(n/p).
We ran our code using a Linux virtual machine that has 2 processors.

Number of processes	Time in 10 runs											Average
2	n=1000	0.00000152	0.00000149	0.00000144	0.00000153	0.00000146	0.00000146	0.00000141	0.00000136	0.00000145	0.00000152	0.000001464
	n=1500	0.00000217	0.00000199	0.00000206	0.00000403	0.00000219	0.00000355	0.00000353	0.00000399	0.00000204	0.00000207	0.000002762
	n=3000	0.00000682	0.00000725	0.00000695	0.00000741	0.00000712	0.00000850	0.00000724	0.00000698	0.00000837	0.00000688	0.000007352
4	n=1000	0.00000100	0.00000109	0.00000079	0.00000121	0.00000116	0.00000090	0.00000082	0.00000085	0.00000087	0.00000092	0.000000961
	n=1500	0.00000129	0.00000145	0.00000136	0.00000125	0.00000128	0.00000127	0.00000123	0.00000122	0.00000125	0.00000108	0.000001268
	n=3000	0.00000229	0.00000241	0.00000222	0.00000207	0.00000247	0.00000232	0.00000219	0.00000218	0.00000213	0.00000221	0.000002249
8	n=1000	0.00000053	0.00000056	0.00000057	0.00000060	0.00000044	0.00000057	0.00000052	0.00000056	0.00000082	0.00000074	0.000000591
	n=1500	0.00000097	0.00000059	0.00000064	0.00000071	0.00000096	0.00000072	0.00000080	0.00000079	0.00000076	0.00000075	0.000000769
	n=3000	0.00000212	0.00000127	0.00000126	0.00000123	0.00000138	0.00000114	0.00000129	0.00000121	0.00000127	0.00000122	0.000001339
16	n=1000	0.00000026	0.00000044	0.00000057	0.00000031	0.00000040	0.00000035	0.00000041	0.00000042	0.00000046	0.00000053	0.000000415
	n=1500	0.00000068	0.00000082	0.00000050	0.00000089	0.00000045	0.00000037	0.00000060	0.00000058	0.00000057	0.00000060	0.000000606
	n=3000	0.00000077	0.00000071	0.00000100	0.00000068	0.00000071	0.00000064	0.00000072	0.00000071	0.00000067	0.00000066	0.000000727

Table.3. MPI Algorithm Runs

6 Result and discussion

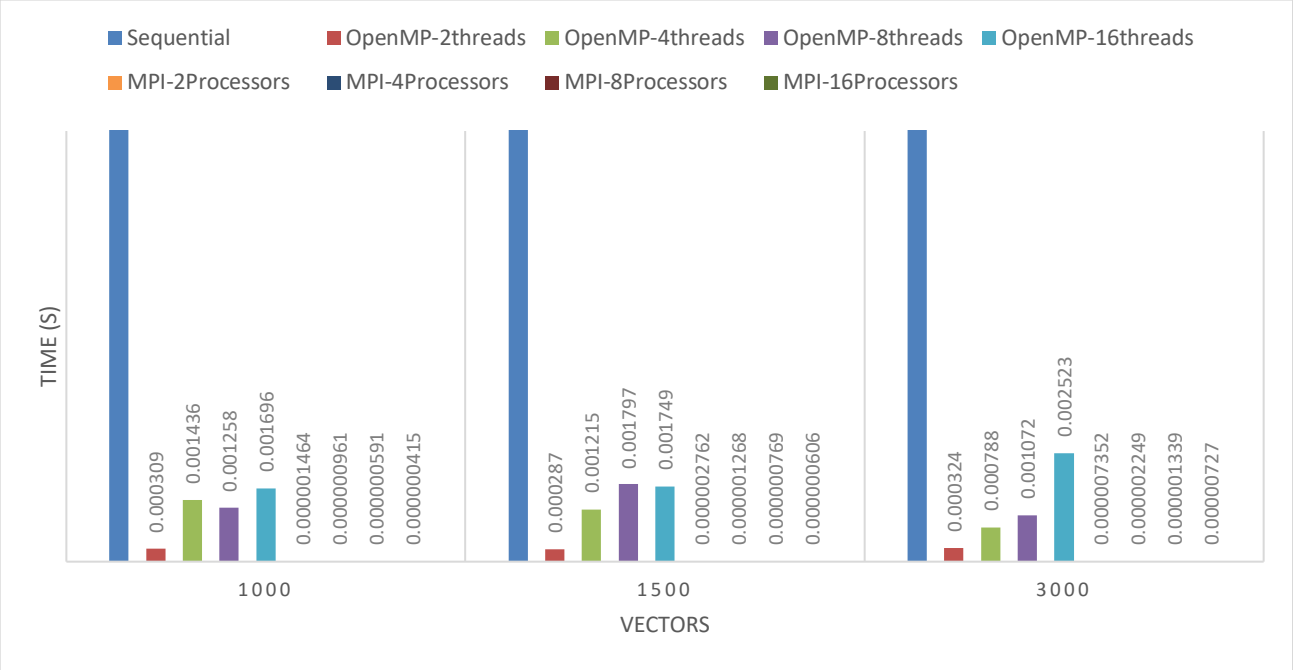


Fig.3. The algorithms time results comparison

In the graph, we showed different sizes of vectors (1000, 1500, 3000) representing different algorithms which are Sequential, OpenMP, and MPI. The parallel algorithms have more particular characteristics. OpenMP has threads that will increase the performance once the number of threads is increased, and MPI has processes that will also increase the performance once the number of processes is increased.

Sequential is a regular line-by-line algorithm when we start running the code. OpenMP allows shared-memory parallel computing, while MPI uses Message Passing Interface (MPI) for distributed-memory systems. Moreover, OpenMP allows the code to run on a single multi-core whereas the MPI will run it on multiple systems connected by a network. That leads to the MPI having higher performance due to its ability to provide API calls such as MPI_Send and MPI_Recv to allow communication between computation nodes. Unlike OpenMP, here each computational unit has to send its results to a master, and it manually compiles and aggregates the final result. Whereas OpenMP is relatively easy to implement as it is viewed in its code only. However, it has drawbacks, such as the problem of memory limitations for memory-intensive calculations. While MPI usually solves that problem well, which involves large memory as an advantage of shared memory that can be utilized within it.

As we analyzed the three algorithms and discussed their advantages and disadvantages, we can agree that the parallelized algorithms (OpenMP, MPI) are better than the Sequential algorithm but each of which has its own applicability and uses.

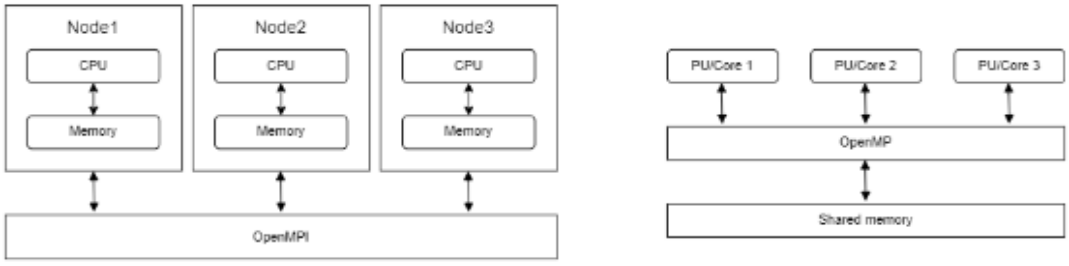


Fig.4. Architectural difference between MPI and OpenMP [5]

7 Conclusion and future work

The goal of this project was to use OpenMP and MPI to parallelize, the Sequential vector multiplication algorithm, and the aim was that by using these parallelism methods, we would make the algorithm much more efficient by having faster execution. We have thankfully accomplished that goal. The OpenMP algorithm is much faster in execution than the Sequential algorithm with the same time complexity O(n), and the MPI algorithm is faster in execution and has a better time complexity O(n/p) than the OpenMP algorithm. We hope that the concept of parallelism will not stop here, and we would consider it to solve many more algorithms such as the numerous types of Sorting, Matrices, Linked Lists, Shortest Path problems and much more!

References

[1] A. Sword, "What is a matrix?," 2016 2 2016. [Online]. Available: https://techmonitor.ai/what-is/what-is-a-matrix-4964527. [Accessed 29 10 2022].

[2] Wikipedia, "Matrix multiplication," 25 10 2022. [Online]. Available: https://en.wikipedia.org/wiki/Matrix_multiplication. [Accessed 29 10 2022].

[3] CUEMATH, "Multiplication of Vectors," [Online]. Available: https://www.cuemath.com/algebra/multiplication-of-vectors/. [Accessed 29 10 2022].

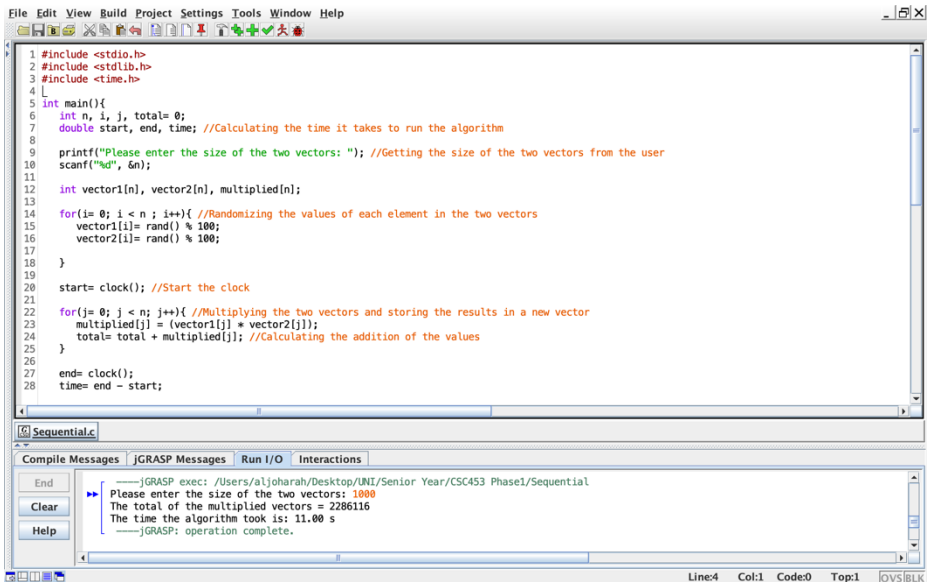
[4] CSE 331, "Matrix Vector Multiplication," [Online]. Available: https://cse.buffalo.edu/~erdem/cse331/support/matrix-vect/. [Accessed 29 10 2022].

[5] A. Abdur Rehman, "What is the difference between OpenMP and OpenMPI?," educative, [Online]. Available: https://www.educative.io/answers/what-is-the-difference-between-openmp-and-openmpi. [Accessed 05 11 2022].

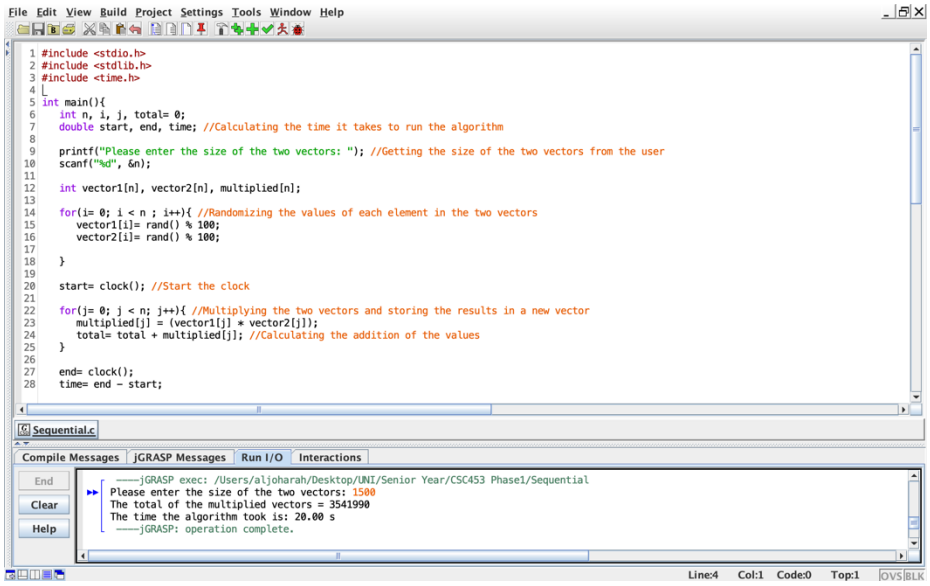
Appendix A

The Sequential code results:

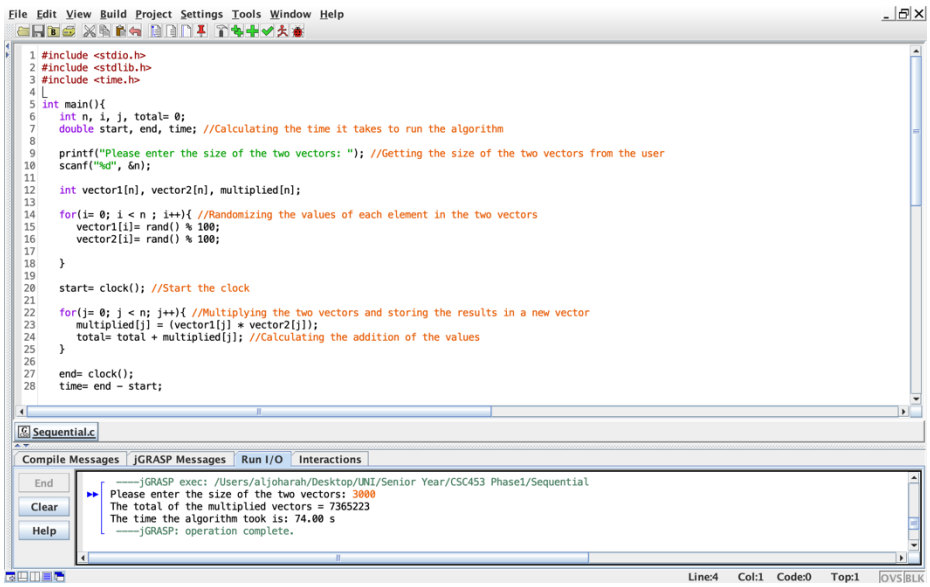
Vector size = 1000



Vector Size = 1500



Vector Size = 3000



The OpenMP code results:

Vector size = 1000, 1500, 3000, Number of Threads = 2

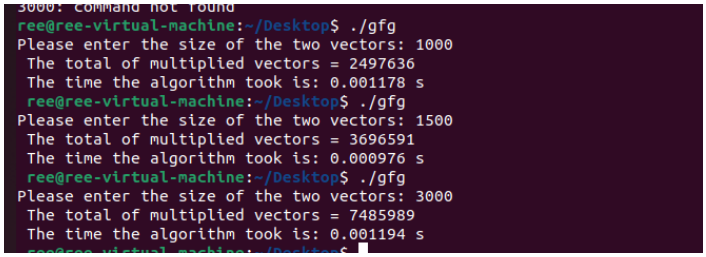
```
ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 1000
 The total of multiplied vectors = 2497636
 The time the algorithm took is: 0.000283 s
 ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 1500
 The total of multiplied vectors = 3696591
 The time the algorithm took is: 0.000247 s
 ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 3000
 The total of multiplied vectors = 7485989
 The time the algorithm took is: 0.000229 s
```

Vector size = 1000, 1500, 3000, Number of Threads = 4

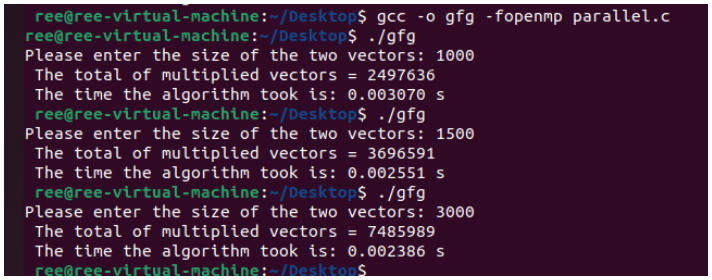
```
ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 1000
 The total of multiplied vectors = 2497636
 The time the algorithm took is: 0.007922 s
 ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 1500
 The total of multiplied vectors = 3696591
 The time the algorithm took is: 0.003597 s
 ree@ree-virtual-machine:~/Desktop$ ./gfg
Please enter the size of the two vectors: 3000
 The total of multiplied vectors = 7485989
 The time the algorithm took is: 0.000630 s
```

10

Vector size = 1000, 1500, 3000, Number of threads = 8

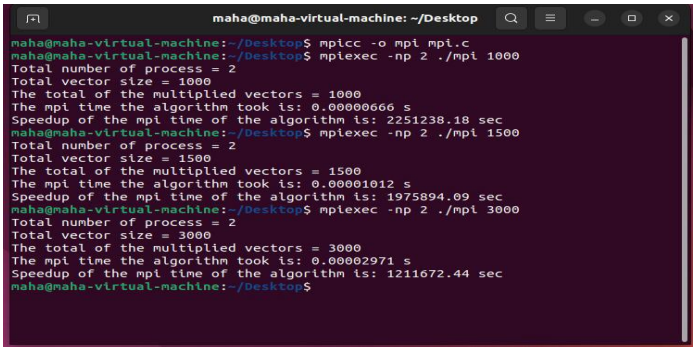


Vector size = 1000, 1500, 3000, Number of Threads = 16

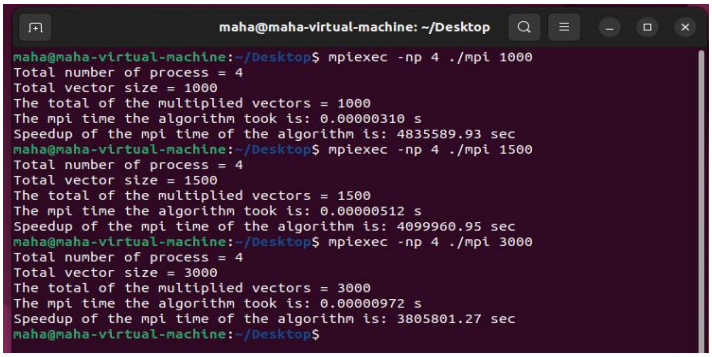


The MPI code results:

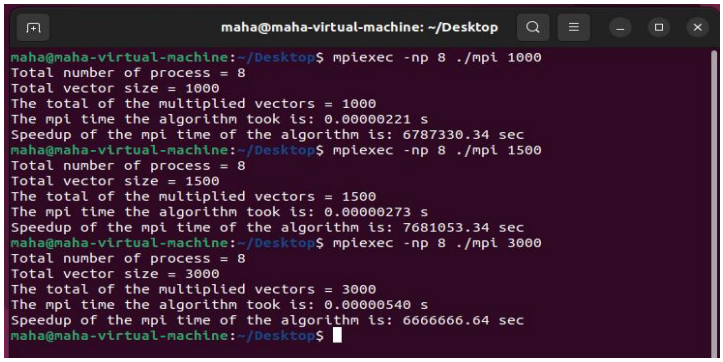
Vector size = 1000, 1500, 3000, Number of processes = 2



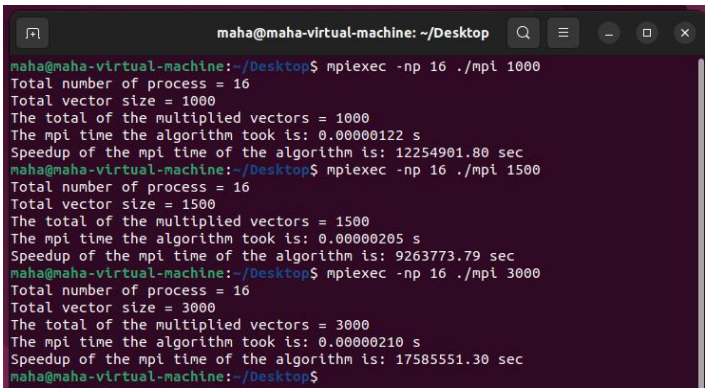
Vector size = 1000, 1500, 3000, Number of processes = 4



Vector size = 1000, 1500, 3000, Number of processes = 8



12
Vector size = 1000, 1500, 3000, Number of processes = 16



Appendix B

Task	Student
Introduction and Problem Definition	Aljwhra Almakhdoub
Sequential Part	Aljwhra Almakhdoub
OpenMP Part	Lama Alrehaily and Reema Alqamshaa
MPI	Enas Alzahrani, Maha Almunawer, and Raseel Alhaqbani
Result and Conclusion	All
Review	All