**Exercise 0: Explain your system?**

|  |  |
| --- | --- |
| Processor | i7-5500U , 2.40GHz |
| Cores | 4 |
| Operating system | Windows 64 Bit |
| Ram | 8GB |
| Programming Language | Python 3.7.7 |

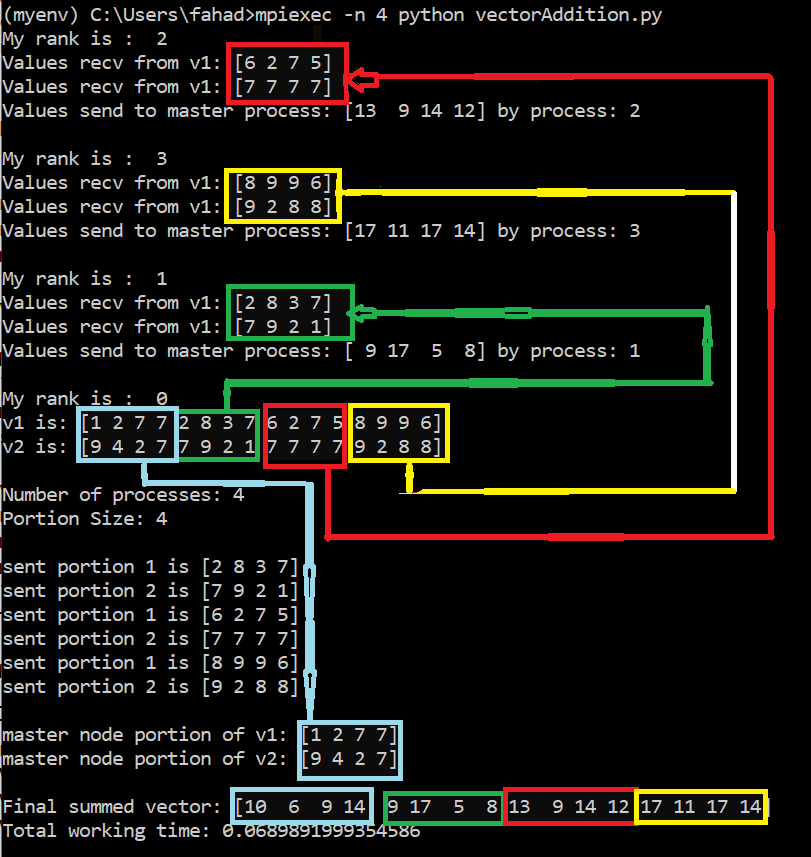
**Exercise 1: Basic Parallel Vector Operations with MPI?**

**Part a)**

Steps:

1. Consider 1st process as master node and other as worker node.
2. Master process will create two vectors v1, v2.
3. Then divide the vectors in equal portion and send that portion to other worker processes.
4. The worker receive chunk of vectors and perform addition on that vectors and return the result.
5. The master node performs addition on its chunk of vector.
6. The master node collects the result from other worker.
7. It appends and displays the results.

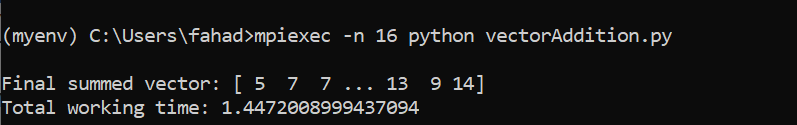
**Processes: 4, Vectors size: 16**

****

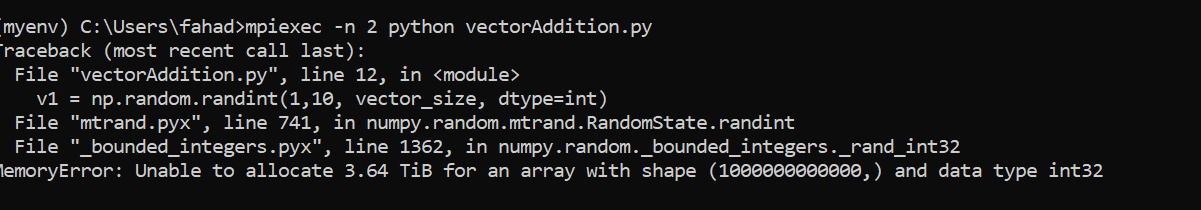
**Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| Size  P | 10^7 | 10^12 | 10^15 |
| 1 | 0.24 | Error | Error |
| 2 | 0.29 |  |  |
| 3 | 0.32 |  |  |
| 4 | 0.34 |  |  |
| 5 | 0.36 |  |  |
| 6 | 0.38 |  |  |
| 7 | 0.48 |  |  |
| 8 | 0.50 |  |  |
| 16 | 0.75 |  |  |

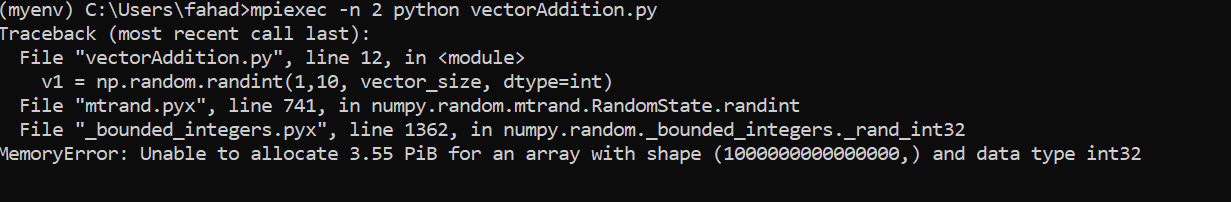
**Vector size**: 10^7



**Vector size**: 10^12



**Vector size**: 10^15



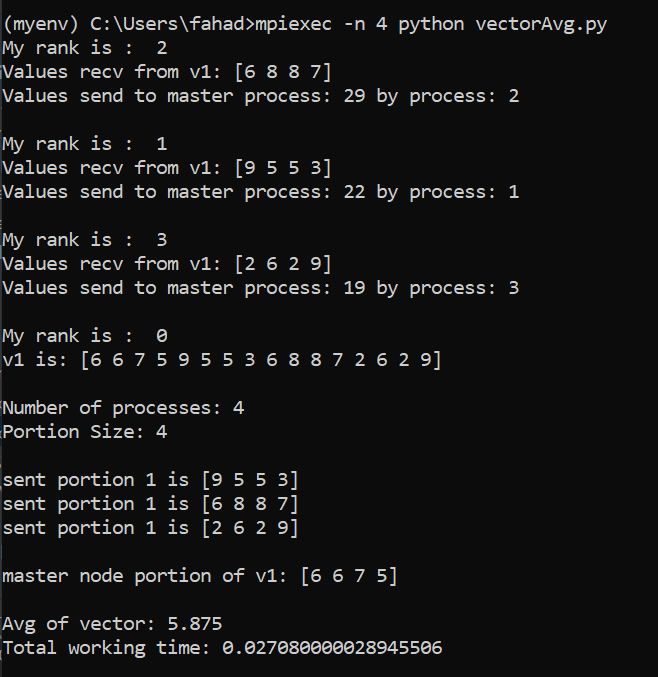
**Code in file VectorAddition.py and output with results are shown in file VectorAddition.html**

**Part b)**

Steps:

1. Consider 1st process as master node and other as worker node.
2. Master process will create vectors v1.
3. Then divide the vectors in equal portion and send that portion to other worker processes.
4. The worker receive chunk of vector and perform addition on that vector and return the result.
5. The master node performs addition on its chunk of vector.
6. The master node collects the result from other worker.
7. It adds the results and divides by size of vector and displays the average of values in vector.

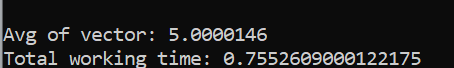
**Processes: 4, Vectors size: 16**

****

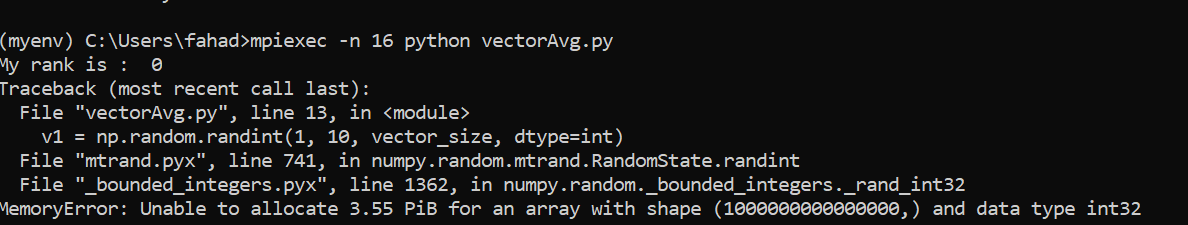
**Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| Size  P | 10^7 | 10^12 | 10^15 |
| 1 | 0.22 | Error | Error |
| 2 | 0.25 |  |  |
| 3 | 0.26 |  |  |
| 4 | 0.29 |  |  |
| 5 | 0.32 |  |  |
| 6 | 0.37 |  |  |
| 7 | 0.35 |  |  |
| 8 | 0.37 |  |  |
| 16 | 0.67 |  |  |

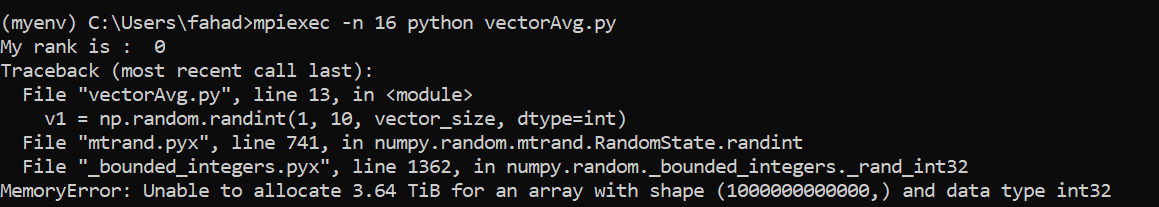
**Vector size**: 10^7



**Vector size**: 10^12



**Vector size**: 10^15



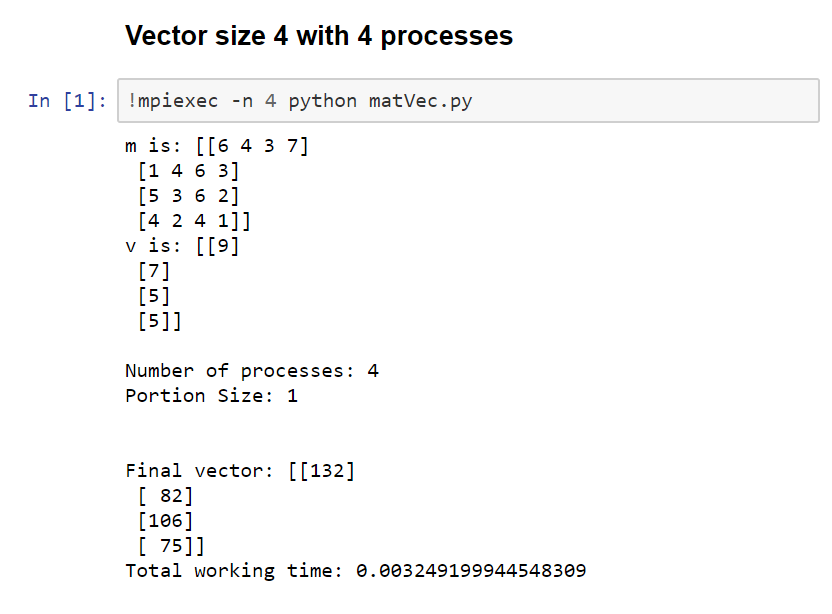
**Code in file VectorAvg.py and output with results are shown in file VectorAvg.html**

**Exercise 2: Parallel Matrix Vector multiplication using MPI?**

Steps:

1. Consider 1st process as master node and other as worker node.
2. Master process will create matrix m and vector v.
3. Then divide the matrix by dividing matrix size with total number of available process and send that portion of matrix and vector to other worker processes.
4. The workers receive chunk of matrix and vector and perform multiplication and return the results.
5. The master node performs multiplication on its chunk of matrix and vector.
6. The master node collects the results from other workers , append and display results.

**Processes: 4, Matrix size: (4,4), Vector size:(4,1)**

****

**Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| Size  P | 10^2 | 10^3 | 10^4 |
| 1 | 0.0003 | 0.02 | 2.3 |
| 2 | 0.001 | 0.03 | 2.8 |
| 3 | 0.005 | 0.03 | 3.13 |
| 4 | 0.002 | 0.04 | 3.2 |
| 5 | 0.123 | 0.07 | 3.23 |
| 6 | 0.055 | 0.139 | 3.56 |
| 7 | 0.22 | 0.095 | 3.39 |
| 8 | 0.066 | 0.121 | 3.18 |
| 16 | 0.206 | 0.34 | 3.64 |

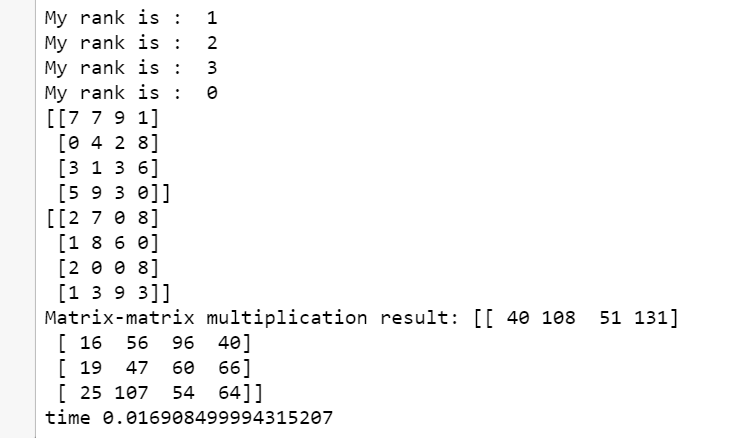
**Code in file MatVec.py and output with results are shown in file MatVecRes.html**

**Exercise 3: Parallel Matrix Operation using MPI?**

Steps:

1. Consider 1st process as master node and other as worker node.
2. Through collective communication broadcast matrix m1\_share, m2\_share and their mat size.
3. All the processes divide the matrix by dividing matrix size with total number of available process and multiply that chunk of m1\_share matrix to other m2\_share matrix and send result back to master node.
4. The master node performs its multiplication on its chunk of matrix m1\_share and m2\_share.
5. The master node collects the results from other workers, append, reshape and display results.

**Processes: 4, Matrix size: (4,4)**

****

**Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| Size  P | 10^2 | 10^3 | 10^4 |
| 1 | 0.002 | 1.57 | Taking too much time |
| 2 | 0.003 | 1.28 |  |
| 3 | 0.002 | 1.15 |  |
| 4 | 0.004 | 1.17 |  |
| 5 | 0.010 | 1.25 |  |
| 6 | 0.057 | 1.24 |  |
| 7 | 0.102 | 1.26 |  |
| 8 | 0.104 | 1.6 |  |
| 16 | 0.24 | 2.4 |  |

**Code in file MatrixMultiplication.py and output with results are shown in file MatrixMultiplication.py**

**Conclusion for all questions:**

* If the number of processes that we run through mpi4py is more than then available cores then OS has to run multiple processes on one core. In this case there is context switching between processes and it increases the time taken to do our operations. As you can see in question 3 time table when number of processes (5, 6, 7, 8, and 16) is more than the available cores (4), then mostly instead of decreasing time for matric multiplication, time take is increased. But in some case time is decreased due to efficient handling of processes by OS internally.
* Also, when we increase the # of process from 1 to 2 then time should be decreased because now we are doing multiprocessing. But results show that time taken when # of processes are 2 is more than when # of processes were 1. This may be because our OS is still running the 2 processes on single core due to which there is context switching involved which instead of increases the time, decreases it.