**PARALLEL AND DISTRIBUTED PROCESSING**

**Programming Assignment\_2**

**Problem Statement:**

Use OpenMP application program interface to parallelize a Jacobi iteration. Compare your solution with serial version of the algorithm and show how your code scales with increasing number of threads. Additionally, compare the answer of the parallel code to that from the serial code and, if they differ, explain why that might legitimately be the case.

**Solution Approach:**

Initially we can explain about Jacobian method and its usage

**Jacobi Method:** Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges. In [numerical linear algebra](https://en.wikipedia.org/wiki/Numerical_linear_algebra), it is an iterative algorithm for determining the solutions of a [diagonally dominant](https://en.wikipedia.org/wiki/Diagonally_dominant_matrix) [system of linear equations](https://en.wikipedia.org/wiki/System_of_linear_equations).

**Implementation of code:**

* Initially we will ask user to enter the input of matrix size whether it is 2\*2 matrix or 3\*3 matrix or any square matrix possible, then the matrix input elements are entered and displayed in output as a conformation for the user.
* We can control number of threads using omp\_set\_num\_threads(NUM\_OF\_THREADS);
* Allow the user to enter the right hand side matrix.
* Here we can ask the user to enter the running mode whether it can be serial or parallel, if it is serial,solve\_jacobi\_sequential(matrix,matrix\_size, right\_hand\_side);

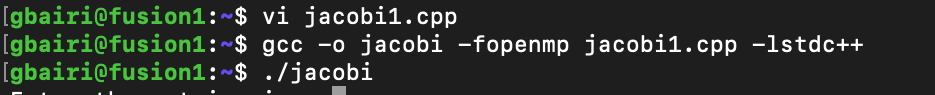
Will be called else if it is parallel case1 solve\_jacobi\_parallel(matrix, matrix\_size, right\_hand\_side), will be called after setting number of threads to defined value.

And also elapsed time is printed float(clock).

* Here first we check whether the matrix entered is dominant or not and if it is dominant then we will proceed further else we will ask user to enter the dominant matrix.
* In solve\_jacobi\_parallel we use “ #pragma omp parallel for schedule(dynamic, 1)” that takes care about all the details of the parallelization. It creates a team of threads and distributes the iterations between the threads, but in solve\_jacobi\_sequential we won’t use pragma function which makes it serial.

So now we will run the code in different ways whether serial / parallel and for different matrix and finally with different number of threads and check performance analysis.

**Compilation steps;**



**Analysis:**

* 3\*3 matrix ( 4 threads) Serial Elapsed time : 103.0000 ms
* 3\*3 matrix ( 4 threads) Parallel Elapsed time : 2176.000 ms
* 2\*2 matrix ( 4 threads) Serial Elapsed time : 194.0000 ms
* 2\*2 matrix ( 4 threads) Parallel Elapsed time : 3165.00000 ms
* 3\*3 matrix ( 6 threads) Serial Elapsed time : 106.0000 ms
* 3\*3 matrix ( 6 threads) Parallel Elapsed time : 3405.0000 ms
* 4\*4 large int matrix ( 6 threads) Parallel Elapsed time : 3507.0000 ms
* 4\*4 large int matrix ( 6 threads) Serial Elapsed time : 86.0000 ms

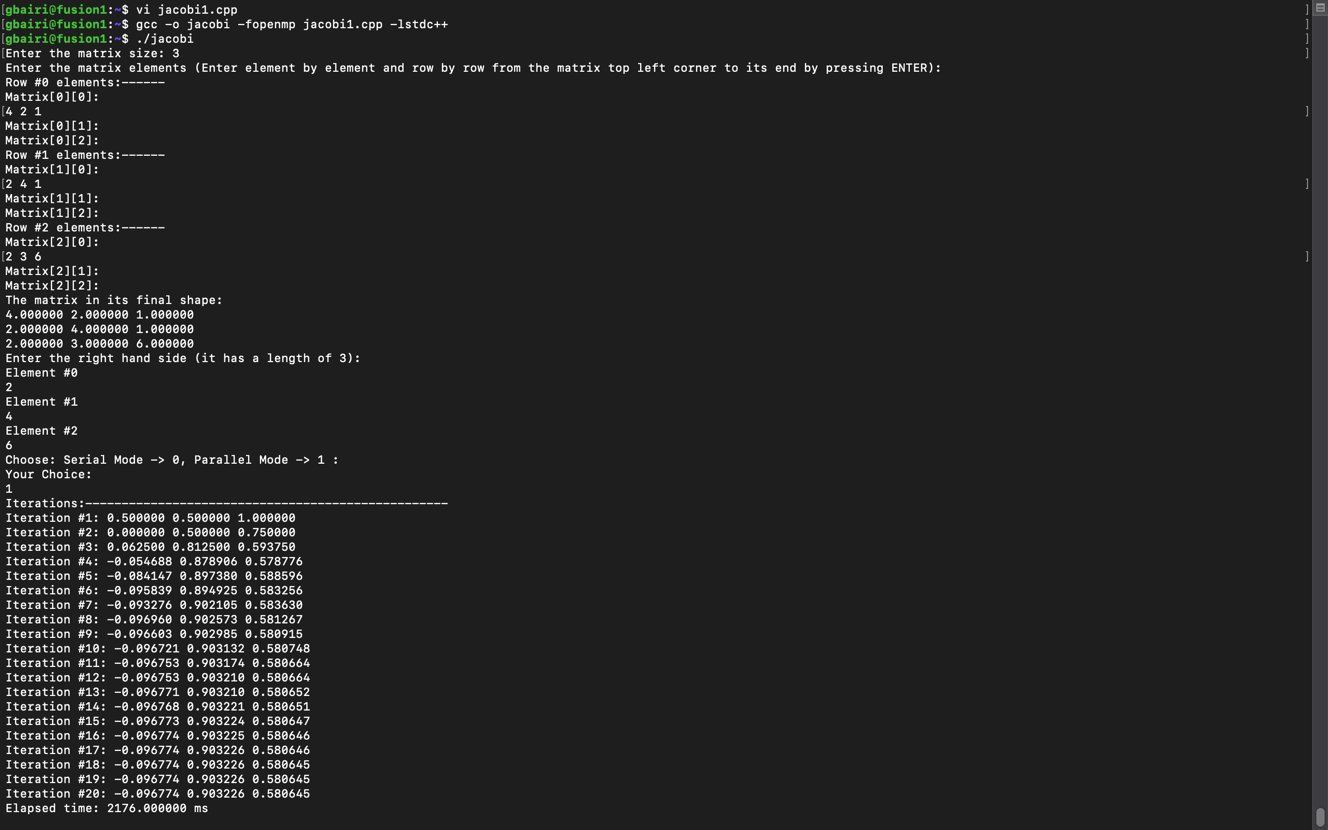
Code file: jacobi.cpp

>>>>>continuation in next page attached output analysis Scrennshot>>>>>>>>>

**PERFORMANCE RESULTS:**

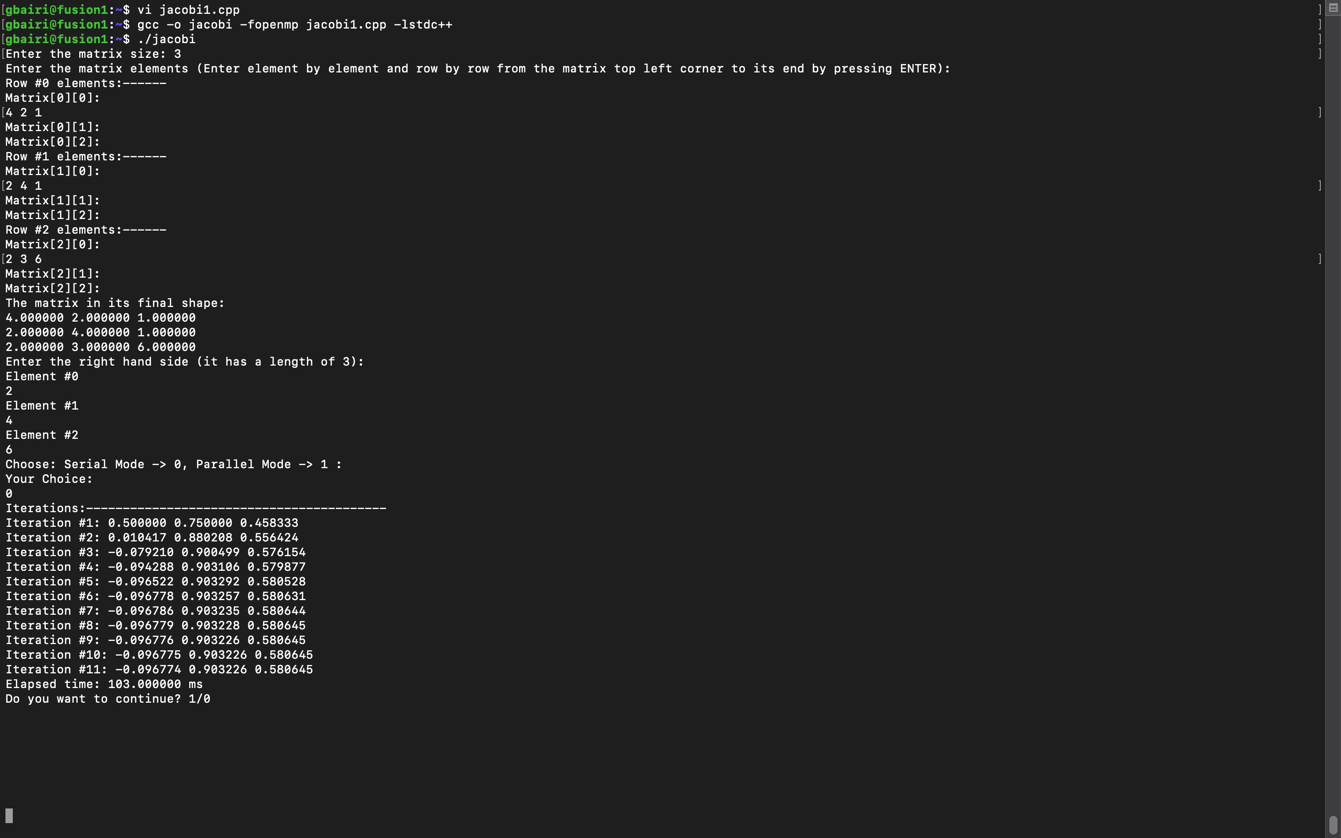
**Running Code in parallel with 3\*3 matrix ( 4 threads)**

**[MatrixElements: R1: 4 2 1, R2: 2 4 1, R3: 2 3 6, 2ndMatrixElements: R1: 2, R2: 4, R3: 6 ]**



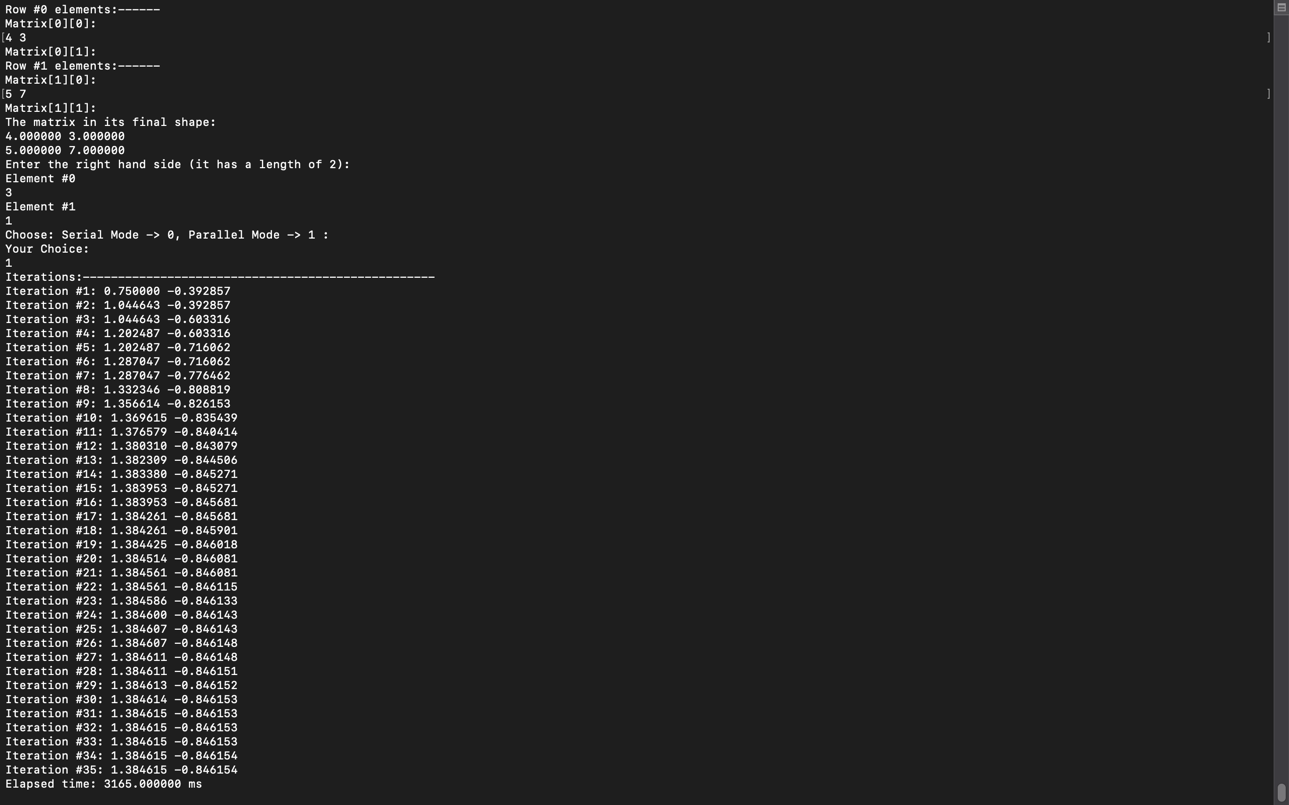
**Running Code in Serial with 3\*3 matrix ( 4 threads)**

**[MatrixElements: R1: 4 2 1, R2: 2 4 1, R3: 2 3 6, 2ndMatrixElements: R1: 2, R2: 4, R3: 6 ]**



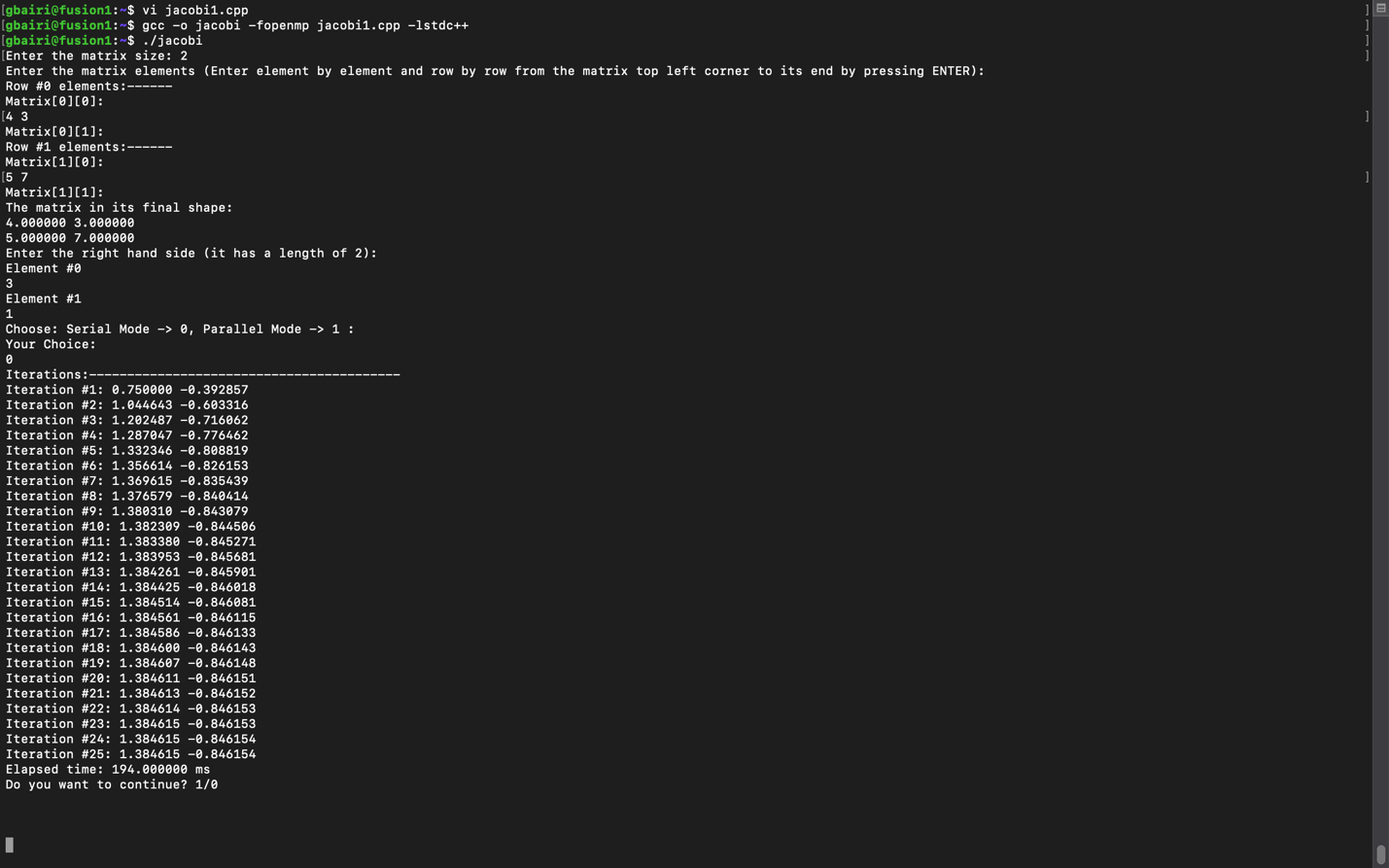
**Running Code in parallel with 2\*2 matrix ( 4 threads)**

**[MatrixElements: R1: 4 3, R2: 5 7, 2ndMatrixElements: R1: 3, R2: 1 ]**



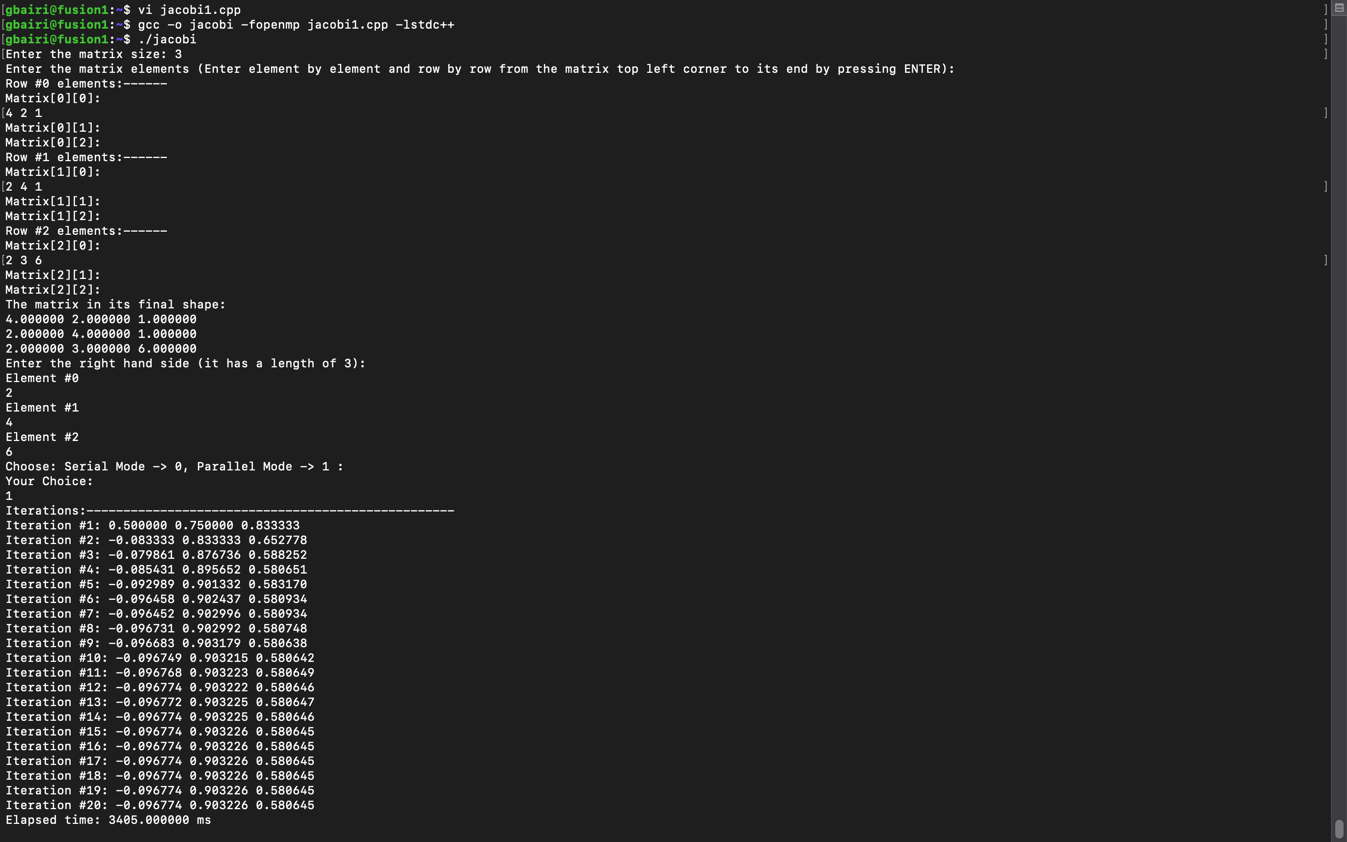
**Running Code in Serial with 2\*2 matrix ( 4 threads)**

**[MatrixElements: R1: 4 3, R2: 5 7, 2ndMatrixElements: R1: 3, R2: 1 ]**



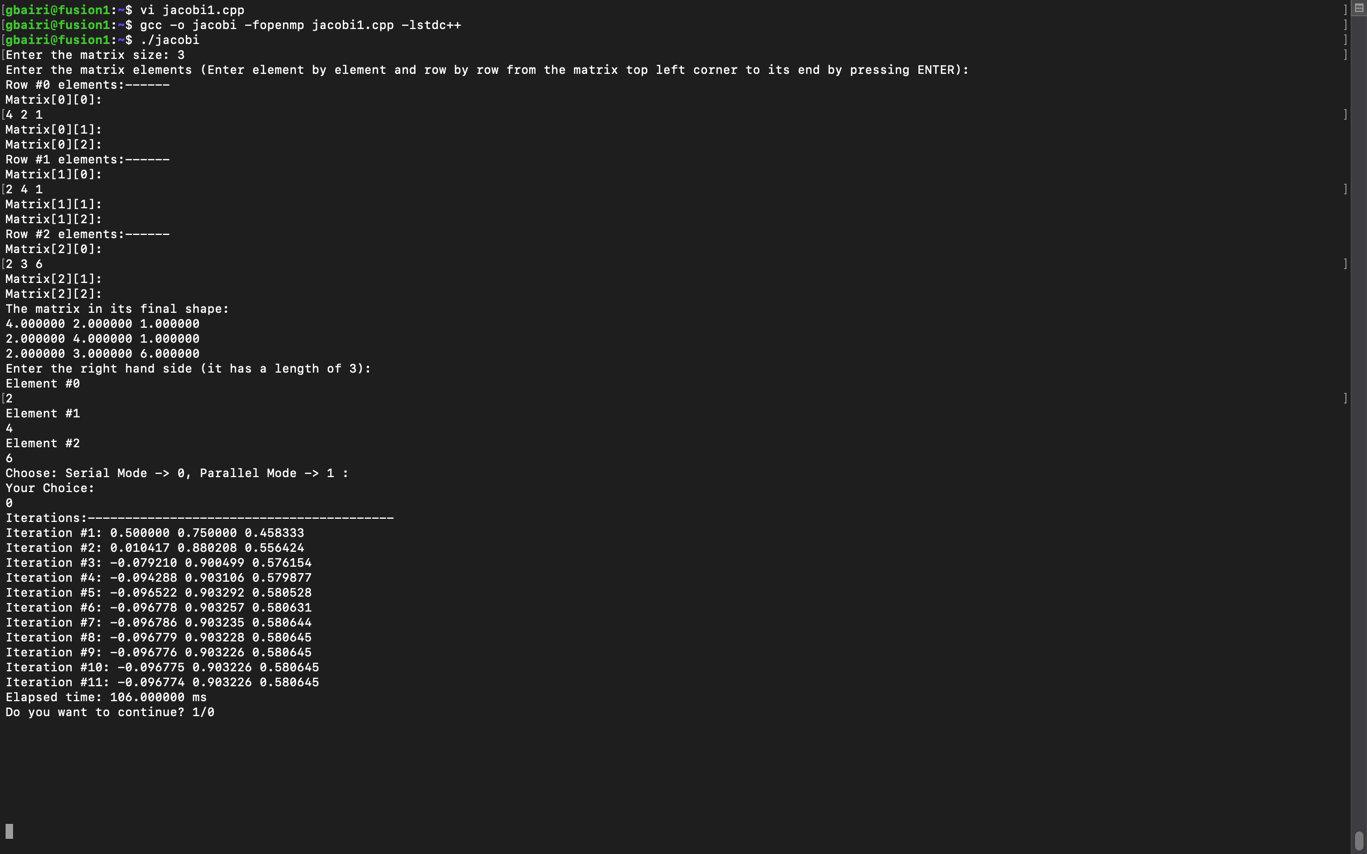
**Running Code in parallel with 3\*3 matrix ( 6 threads)**

**[MatrixElements: R1: 4 2 1, R2: 2 4 1, R3: 2 3 6, 2ndMatrixElements: R1: 2, R2: 4, R3: 6 ]**

****

**Running Code in Serial with 3\*3 matrix ( 6 threads)**

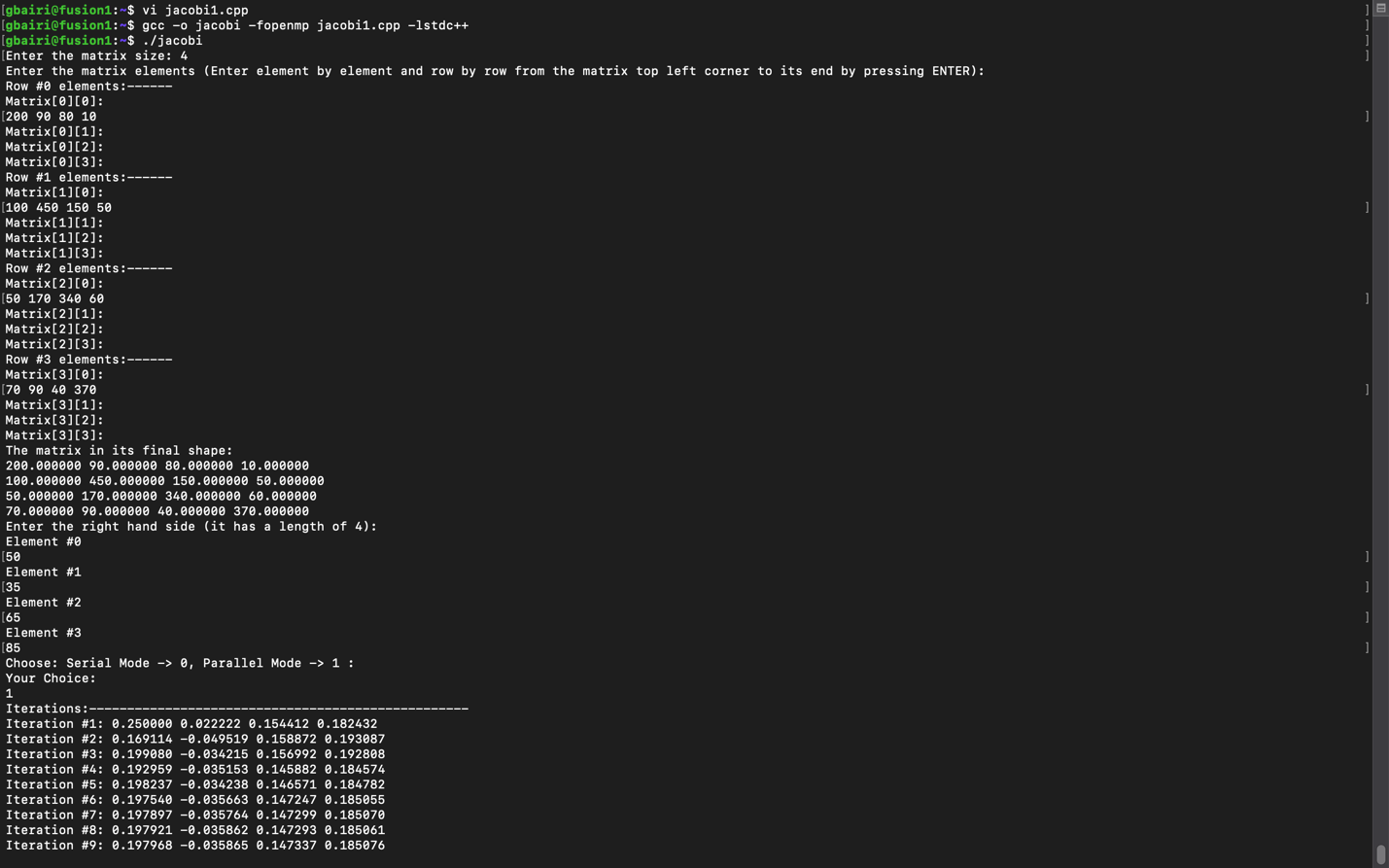
**[MatrixElements: R1: 4 2 1, R2: 2 4 1, R3: 2 3 6, 2ndMatrixElements: R1: 2, R2: 4, R3: 6 ]**

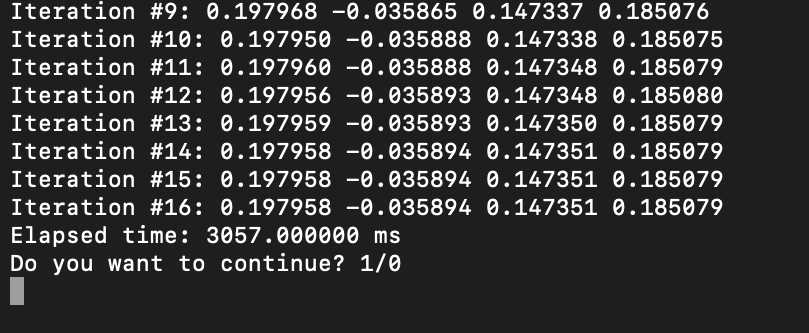


**Running Code in parallel with 4\*4 matrix ( 6 threads) [ large integers ]**

**[MatrixElements: R1: 200 90 80 10, R2: 100 450 150 50, R3: 50 170 340 60, R4: 70 90 40 370**

**2ndMatrixElements: R1: 50, R2: 35, R3: 65, R4:85]**





**Running Code in Serial with 4\*4 matrix ( 6 threads) [ large integers ]**

**[MatrixElements: R1:** 200 90 80 10**, R2:** 100 450 150 50**, R3:** 50 170 340 60**, R4:** 70 90 40 370

**2ndMatrixElements: R1:** 50**, R2:** 35**, R3:** 65**, R4:**85**]**

