C++ Programming Work

Parallel CPU Matrix Multiplication Program Code Description

May 16, 2021

Jeremy J. Williams

1. Pseudocode

```
-   Begin CPU Matrix Multiplication Program
-   Enter Matrix Dimension (minimum 4):
        o   Validate if matrix dimension DimStr is a number or string.
                ▪   If number, then continue. Otherwise, return to the CHECK_DIM position.
        o   If dimension value $DimVal < 4$, then return to the START_DIM position.
        o   If dimension value $DimVal \geq 4$, then continue and store matrix dimension value.
-   Set Matrix A, B and C Dimensions
-   Enter Random Maximum Real Value (minimum 10):
        o   Validate if random maximum real RealVal is a number or string.
                ▪   If number, then continue. Otherwise, return to the CHECK_REAL_VAL position.
        o   If real value $RealVal < 10$, then return to the START_REAL_VAL position.
        o   If real value $RealVal \geq 10$, then continue and store random maximum real values.
-   Enter Random Maximum Imaginary Value (minimum 10):
        o   Validate if random maximum imaginary ImagVal is a number or string.
                ▪   If number, then continue. Otherwise, return to the CHECK_IMAG_VAL position.
        o   If random imaginary $ImagVal < 10$, then return to the START_IMAG_VAL position.
        o   If random real $ImagVal \geq 10$, then continue and store the random real values.
-   Set Matrix A and B with Random Maximum Complex (Real and Imaginary) Values
-   Set Matric C with Zero Values for Results Matrix
-   Display Matrix A and B with Complex Values.
        o   If matrix dimension value $DimVal \leq 30$, then display Matrix A and B.
        o   If matrix dimension value $DimVal > 30$, then do not display Matrix A and B.
-   Send Matrix A and B to non-parallel matrix multiplication function for serial processing.
-   Display non-parallel multiplication CPU elapsed time.
-   Send Matrix A and B to parallel matrix multiplication function for parallel processing.
-   Display parallel multiplication CPU elapsed time.
-   Display Results Matric C with Complex Values.
        o   If results matrix dimension value $DimVal \leq 30$, then display Matrix C.
        o   If results matrix dimension value $DimVal > 30$, then do not display Matrix C.
-   Display parallel processing processes difference.
-   Finish CPU matrix multiplication processes and calculations.
-   End Program
```

2. Source Code

2.1 Global Declarations

The following code are global declarations of headers, variables, and functions definitions.

```
#include <stdio.h>
#include <time.h>
#include <complex>      // std::complex, std::real, std::imag
#include <iostream>    // std::cout
#include <string>
#include <sstream>

typedef std::complex<double> TYPE;

//function identifier
void set_dimension(TYPE** &Matrix, size_t Dimension);
void set_random_value(TYPE** &Matrix, size_t Dimension, TYPE MaxRndVal_JW);
void display_matrix(TYPE** Matrix, size_t Dimension);
double matrix_multiplication(TYPE** MatrixA, TYPE** MatrixB, TYPE** &MatrixC, size_t Dimension);
double matrix_parallel_multiplication(TYPE** MatrixA, TYPE** MatrixB, TYPE** &MatrixC, size_t Dimension);

using namespace std;
```

2.2 Validate Entry Values

The following code are used to validate the dimension, real and imaginary values if entry values are a number or string value.

```
std::string DimVal;

//Validate Dimension Value if number or string
bool check_DIM_number(std::string DimVal) {
   for (int i = 0; i < DimVal.length(); i++)
   if (isdigit(DimVal[i]) == false)
      return false;
      return true;
}

std::string RealVal;

//Validate Real Value if number or string
bool check_REAL_number(std::string RealVal) {
   for (int i = 0; i < RealVal.length(); i++)
   if (isdigit(RealVal[i]) == false)
      return false;
      return true;
}

std::string ImagVal;

//Validate Imaginary Value if number or string
bool check_IMAG_number(std::string ImagVal) {
   for (int i = 0; i < ImagVal.length(); i++)
   if (isdigit(ImagVal[i]) == false)
      return false;
      return true;
}
```

2.3 Main

The following code is the main function for the entry point of the program.

This function starts the program and takes the dimension, real and imaginary values from the user.

This function waits for the user to press the Enter Key for each value entered, stores the values in memory, calculates the processes, displays the CPU processing times and calculates the difference of the CPU processing time of non-parallel (serial) and parallel matrix multiplication implementations.

```
int main()
{
        TYPE** MatrixA = NULL;
        TYPE** MatrixB = NULL;
        TYPE** MatrixC = NULL;
        size_t Dimension = 0;
        double _real = 0;
        double _imag = 0;
        TYPE MaxRndVal_JW = 0;
        double Elapsed = 0;
        double ParallelElapsed = 0;

    cout << "Parallel CPU Matrix Multiply Program" << "\n";
    cout << "\n";
        cout << "Hello and Thank You For Your Time" << "\n";
    cout << "\n";
    cout << "To Begin: Enter Your Matrix Dimension and Complex Values" << "\n";

CHECK_DIM:
    cout << "\n";
    cout << "Enter Matrix Dimension (minimum 4): ";
    cin >> DimVal;
    if (check_DIM_number(DimVal)){
      goto START_DIM;
    }
    else{
      cout << DimVal << " is a string. Please enter number (minimum 4)."<<endl;
      goto CHECK_DIM;
    }

START_DIM:
        Dimension = std::stoi(DimVal);
    if (Dimension < 4) {
                 cout << "Invalid Matrix Dimension. Please re-enter minimum 4." << endl;
                 goto CHECK_DIM;
        }
                 // set the dimensions
        set_dimension(MatrixA, Dimension);
        set_dimension(MatrixB, Dimension);
        set_dimension(MatrixC, Dimension);

CHECK_REAL_VAL:
    cout << "\n";
    cout << "Enter Random Maximum Real Value (minimum 10): ";
    cin >> RealVal;
    if (check_DIM_number(RealVal)){
      goto START_REAL_VAL;
    }
    else{
      cout << RealVal << " is a string. Please enter number (minimum 10)."<<endl;
      goto CHECK_REAL_VAL;
    }

START_REAL_VAL:
    _real = std::stoi(RealVal);
        MaxRndVal_JW.real(_real);
        if (real(MaxRndVal_JW) < 10) {
                 cout << "Invalid random maximum real value. Please re-enter minimum 10." << endl;
                 goto CHECK_REAL_VAL;
        }

CHECK_IMAG_VAL:
    cout << "\n";
    cout << "Enter Random Maximum Imaginary Value (minimum 10): ";
    cin >> ImagVal;
    if (check_DIM_number(ImagVal)){
      goto START_IMAG_VAL;
    }
    else{
      cout << ImagVal << " is a string. Please enter number (minimum 10)."<<endl;
      goto CHECK_IMAG_VAL;
    }

START_IMAG_VAL:
    _imag = std::stoi(ImagVal);
     MaxRndVal_JW.imag(_imag);
     if (imag(MaxRndVal_JW) < 10) {
                 cout << "Invalid random maximum imaginary value. Please re-enter minimum 10." << endl;
```

```
                goto CHECK_IMAG_VAL;
       }

        // set random value in the matrix
        set_random_value(MatrixA, Dimension, MaxRndVal_JW);
        set_random_value(MatrixB, Dimension, MaxRndVal_JW);
        set_random_value(MatrixC, Dimension, 0); // Set Result Matrix to zero

        //display matrix
        cout << "\n";
        cout << "Display Matrix A" << endl;
        cout << "\n";
        display_matrix(MatrixA, Dimension);
        cout << "\n";
        cout << "Display Matrix B" << endl;
        cout << "\n";
        display_matrix(MatrixB, Dimension);

        // matrix non-parallel multiplication
        cout << "\n";
        cout << "Start non-parallel multiplication..." << endl;
        Elapsed = matrix_multiplication(MatrixA, MatrixB, MatrixC, Dimension);

        // Display non-parallel multiplication elapsed time
        cout << "\n";
        cout << "Non-parallel multiplication elapsed time: " << Elapsed << "ms" << endl;

        // matrix parallel multiplication
        cout << "\n";
        cout << "Start parallel multiplication..." << endl;
        ParallelElapsed = matrix_parallel_multiplication(MatrixA, MatrixB, MatrixC, Dimension);

        // Display parallel multiplication elapsed time
        cout << "\n";
        cout << "Parallel multiplication elapsed time: " << ParallelElapsed << "ms" << endl;

        //display result matrix
        cout << "\n";
        cout << "Display Result Matrix C" << endl;
        cout << "\n";
        display_matrix(MatrixC, Dimension);

        // difference of two processes
        cout << "\n";
        cout << "Difference of the two (2) processes: " << Elapsed - ParallelElapsed << "ms" << endl;
        cout << "\n";
        cout << "Difference of the two (2) processes: " << (Elapsed - ParallelElapsed)/1000 << "s" << endl;
        cout << "\n";
        system("PAUSE");
        cout << "\n";
        cout << "Parallel Processing Application Completed Successfully!" << "\n";
        cout << "\n";
        cout << "Good Bye and Thank You For Your Time" << "\n";
        cout << "\n";
        cout << "Exiting Parallel CPU Matrix Multiply Program..." << "\n";

    return 0;
}
```

2.4 Serial Matrix Multiplication

The following code uses a function to calculate and process the serial/non-parallel multiplication of matrices Matrix A and Matrix B.

This function enters the serial/non-parallel process results in Matrix C.

```
double matrix_multiplication(TYPE** MatrixA, TYPE** MatrixB, TYPE** &MatrixC, size_t Dimension) {

        double start = clock();

        for (size_t i = 0; i < Dimension; i++) {
                for (size_t j = 0; j < Dimension; j++) {
                        for (size_t k = 0; k < Dimension; k++) {
                                MatrixC[i][j] += MatrixA[i][k] * MatrixB[k][j];
                        }
                }
        }

        double end = clock();
        return end - start;
}
```

2.5 Parallel Matrix Multiplication

OpenMP Application Program Interface (API) is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications.

An OpenMP executable directive applies to the succeeding structured block or an OpenMP construct. Each directive starts with "**#pragma omp**". The remainder of the directive follows the conventions of the C and C++ standards for compiler directives. A structured-block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

The following code uses a function to calculate, uses OMP directives and process the parallel multiplication of matrices Matrix A and Matrix B.

For the parallel process begin, OMP directive "**#pragma omp parallel**" was inserted to define a parallel region, which is executed with many threads in parallel.

The clause "**shared(MatrixA,MatrixB,MatrixC)**" was inserted to specify that one or all three (3) variables "MatrixA", "MatrixB" and "MatrixC" should be shared among all threads in the parallel region.

The clause "**private(i,j,k)**" was inserted to specify that each thread should have its own instance of "i", "j" and "k" in the parallel region.

OMP directive "**#pragma omp for**" for was inserted to cause the work done in a "**for loop**" inside the parallel region to be divided among threads.

The clause "**schedule(static)**" was inserted to ensure that the same bounds exists for all the loops. The **static** schedule clause is was selected to have each thread consistently refer to the same set of elements of an array in a series of loops, even if some threads are assigned relatively less work in some of the loops.

OMP directives were used for the parallel matrix multiplication for the fundamental construct to start the parallel execution of the function; which enters the parallel process results in Matrix C.

```
//#pragma omp parallel for
#pragma omp parallel shared(MatrixA,MatrixB,MatrixC) private(i,j,k)
        {
#pragma omp for schedule(static)
                for (size_t i = 0; i < Dimension; i++) {
                    #pragma omp for schedule(static)
                        for (size_t j = 0; j < Dimension; j++) {
                            #pragma omp for schedule(static)
                                for (size_t k = 0; k < Dimension; k++) {
                                    #pragma omp atomic
                                        MatrixC[i][j] += MatrixA[i][k] * MatrixB[k][j];
                                }
                        }
                }
        }
        double end = clock();
        return end - start;
}
```

2.6 Set Matrix Dimension Memory Allocation

The following code uses a function to setup the memory allocation using the Matrix Dimension entered by the user.

```
void set_dimension(TYPE** &Matrix, size_t Dimension) {

        Matrix = (TYPE**)malloc(Dimension * sizeof(TYPE*));

        for (size_t i = 0; i < Dimension; i++) {
                Matrix[i] = (TYPE*)malloc(Dimension * sizeof(TYPE));
        }
}
```

2.7 Set Matrix Random Values

The following code uses a function to load random values in required matrices.

```
void set_random_value(TYPE** &Matrix, size_t Dimension, TYPE MaxRndVal_JW) {

        for (size_t i = 0; i < Dimension; i++) {
                for (size_t j = 0; j < Dimension; j++) {
                        if (real(MaxRndVal_JW) < 1 || imag(MaxRndVal_JW) < 1) {
                                Matrix[i][j].real(0);
                                Matrix[i][j].imag(0);
                        }
                        else
                        {
                                double _real, _imag;
                                _real = real(MaxRndVal_JW);
                                _real = rand() % (size_t)_real;
                                _imag = imag(MaxRndVal_JW);
                                _imag = rand() & (size_t)_imag;
                                Matrix[i][j].real(_real);
                                Matrix[i][j].imag(_imag);
                        }

                }
        }
}
```

2.8 Display Matrix

The following code uses a function to display the Matrix if the Dimension variable ≤ 30.

```
void display_matrix(TYPE** Matrix,size_t Dimension) {

        if (Dimension > 30) {
                cout << "Display is not available. Matrix Dimension < 30." << endl;
                return;
        }
        for (size_t i = 0; i < Dimension; i++) {
                for (size_t j = 0; j < Dimension; j++) {
                        cout << Matrix[i][j] << " ";
                }
                cout << endl;
        }
}
```

3. Execution of Source Code

See execution screenshots using the following sample user entry matrix execution values, Dimension Value (DimVal), Real Value (RealVal) and Imaginary Value (ImagVal):

- DimVal = **4**,
  RealVal = **10** and
  ImagVal = **10i**

- Dimension = **50**,
  RealVal = **100** and
  ImagVal = **100i**

- Dimension = **250**,
  RealVal = **500** and
  ImagVal = **500i**

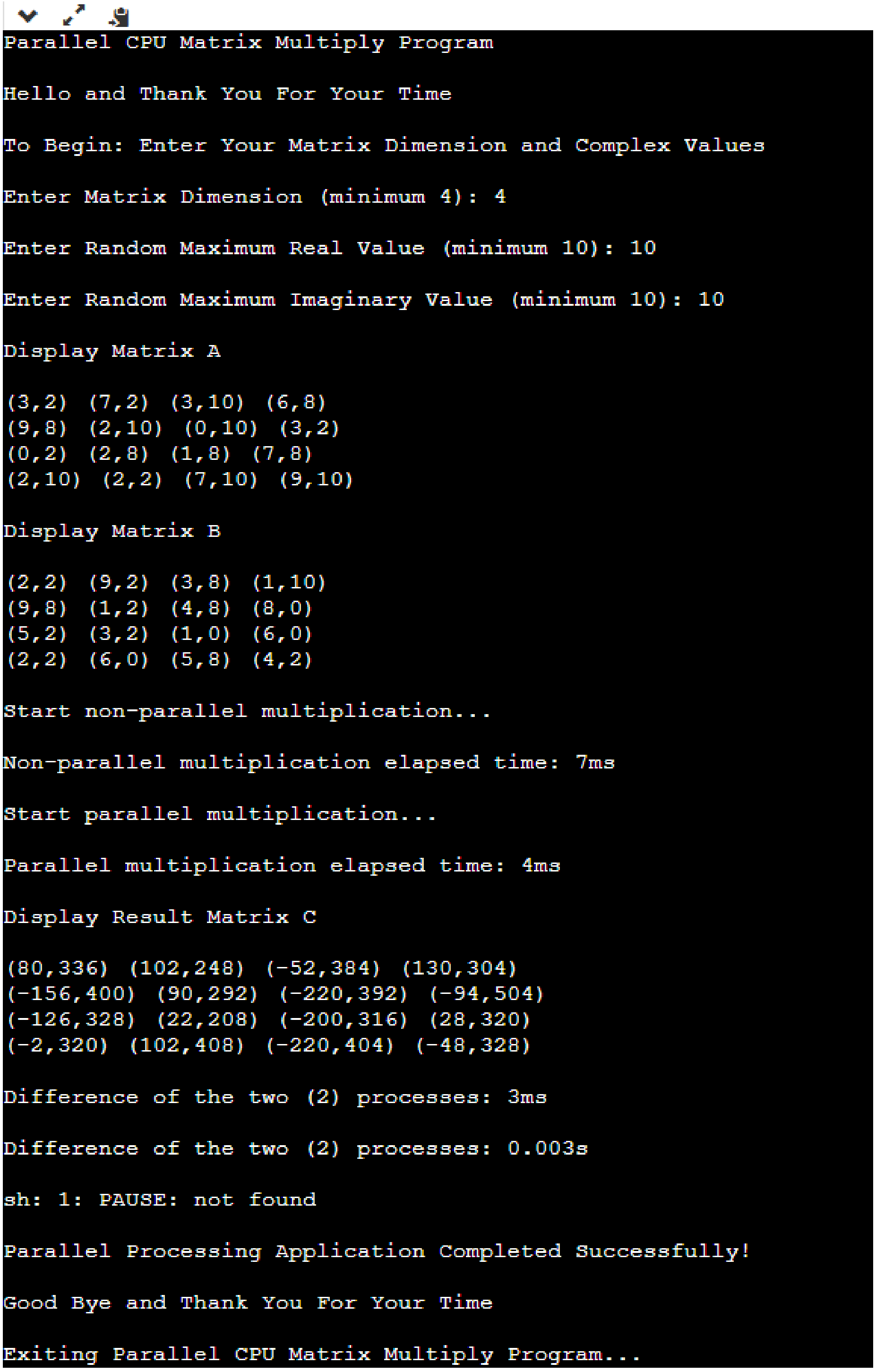
- Dimension = **500**,
  RealVal = **1000** and
  ImagVal = **1000i**

- Dimension = **750**,
  RealVal = **1500** and
  ImagVal = **1500i**

- Dimension = **1000**,
  RealVal = **2000** and
  ImagVal = **2000i**

3.1 Execution #1

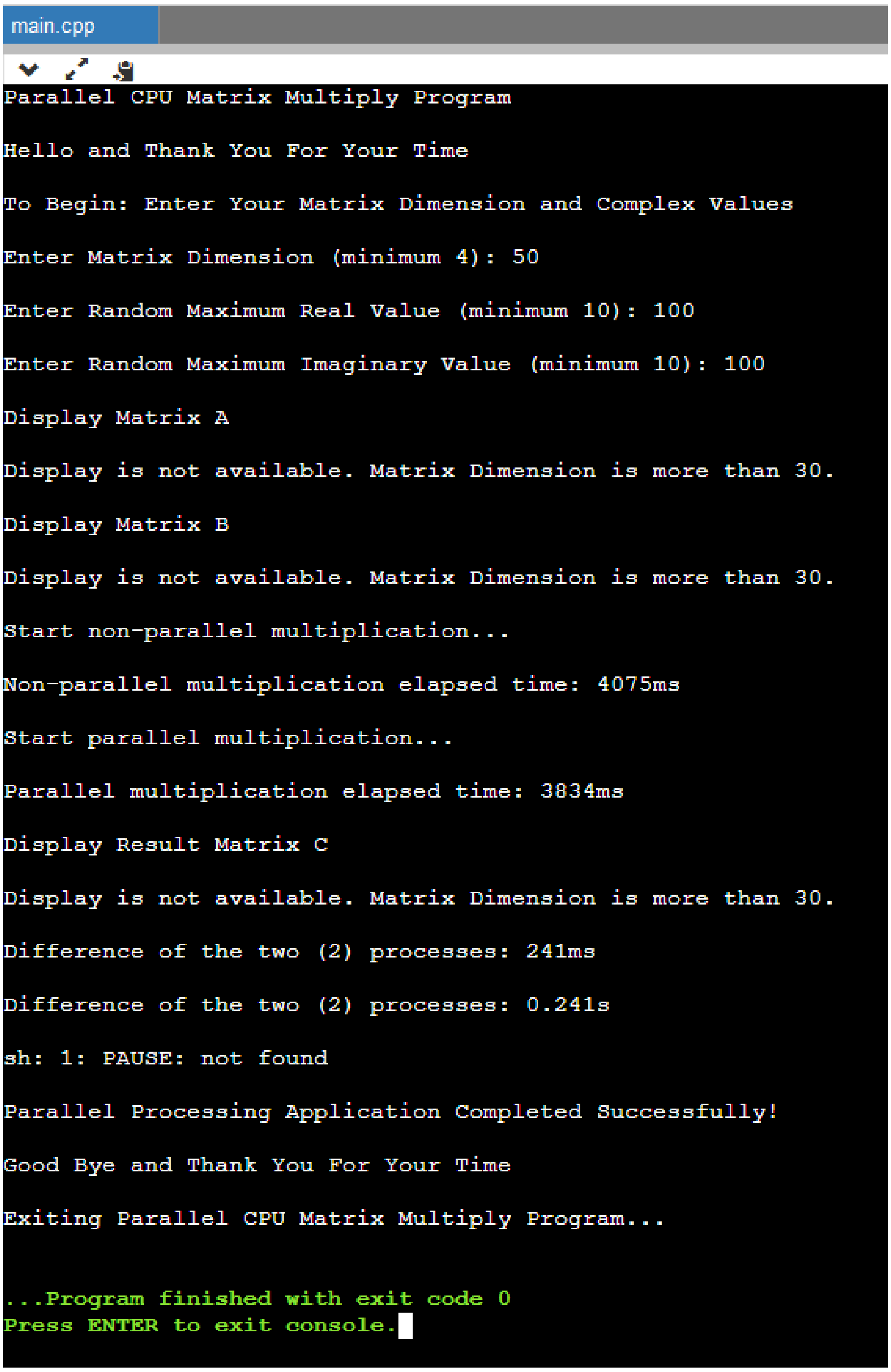
- DimVal = **4**, RealVal = **10** and ImagVal = **10i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 4

Enter Random Maximum Real Value (minimum 10): 10

Enter Random Maximum Imaginary Value (minimum 10): 10

Display Matrix A

(3,2) (7,2) (3,10) (6,8)
(9,8) (2,10) (0,10) (3,2)
(0,2) (2,8) (1,8) (7,8)
(2,10) (2,2) (7,10) (9,10)

Display Matrix B

(2,2) (9,2) (3,8) (1,10)
(9,8) (1,2) (4,8) (8,0)
(5,2) (3,2) (1,0) (6,0)
(2,2) (6,0) (5,8) (4,2)

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 7ms

Start parallel multiplication...

Parallel multiplication elapsed time: 4ms

Display Result Matrix C

(80,336) (102,248) (-52,384) (130,304)
(-156,400) (90,292) (-220,392) (-94,504)
(-126,328) (22,208) (-200,316) (28,320)
(-2,320) (102,408) (-220,404) (-48,328)

Difference of the two (2) processes: 3ms

Difference of the two (2) processes: 0.003s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...
```

3.2 Execution #2

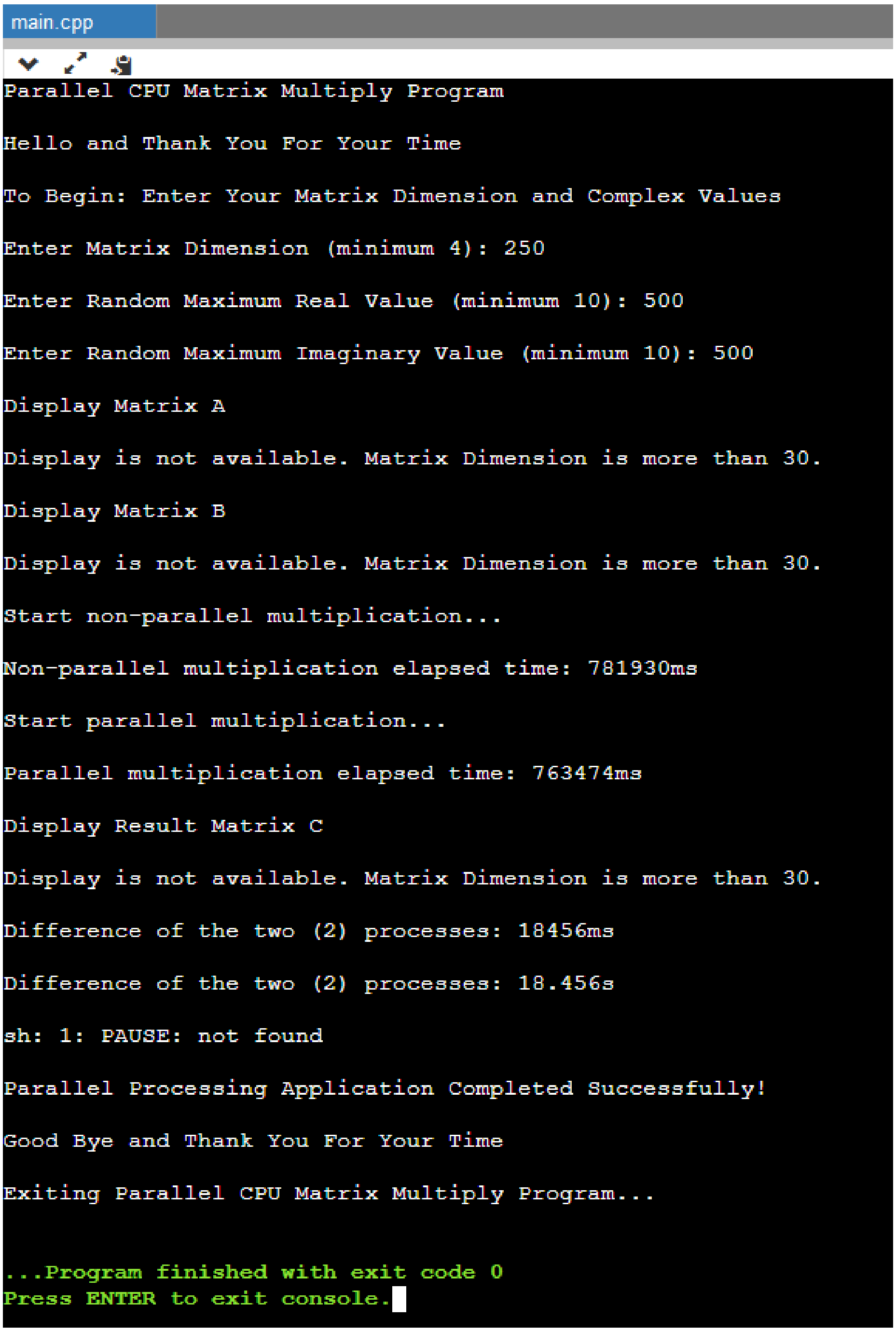
- DimVal = **50**, RealVal = **100** and ImagVal = **100i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 50

Enter Random Maximum Real Value (minimum 10): 100

Enter Random Maximum Imaginary Value (minimum 10): 100

Display Matrix A

Display is not available. Matrix Dimension is more than 30.

Display Matrix B

Display is not available. Matrix Dimension is more than 30.

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 4075ms

Start parallel multiplication...

Parallel multiplication elapsed time: 3834ms

Display Result Matrix C

Display is not available. Matrix Dimension is more than 30.

Difference of the two (2) processes: 241ms

Difference of the two (2) processes: 0.241s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...


...Program finished with exit code 0
Press ENTER to exit console.
```

3.3 Execution #3

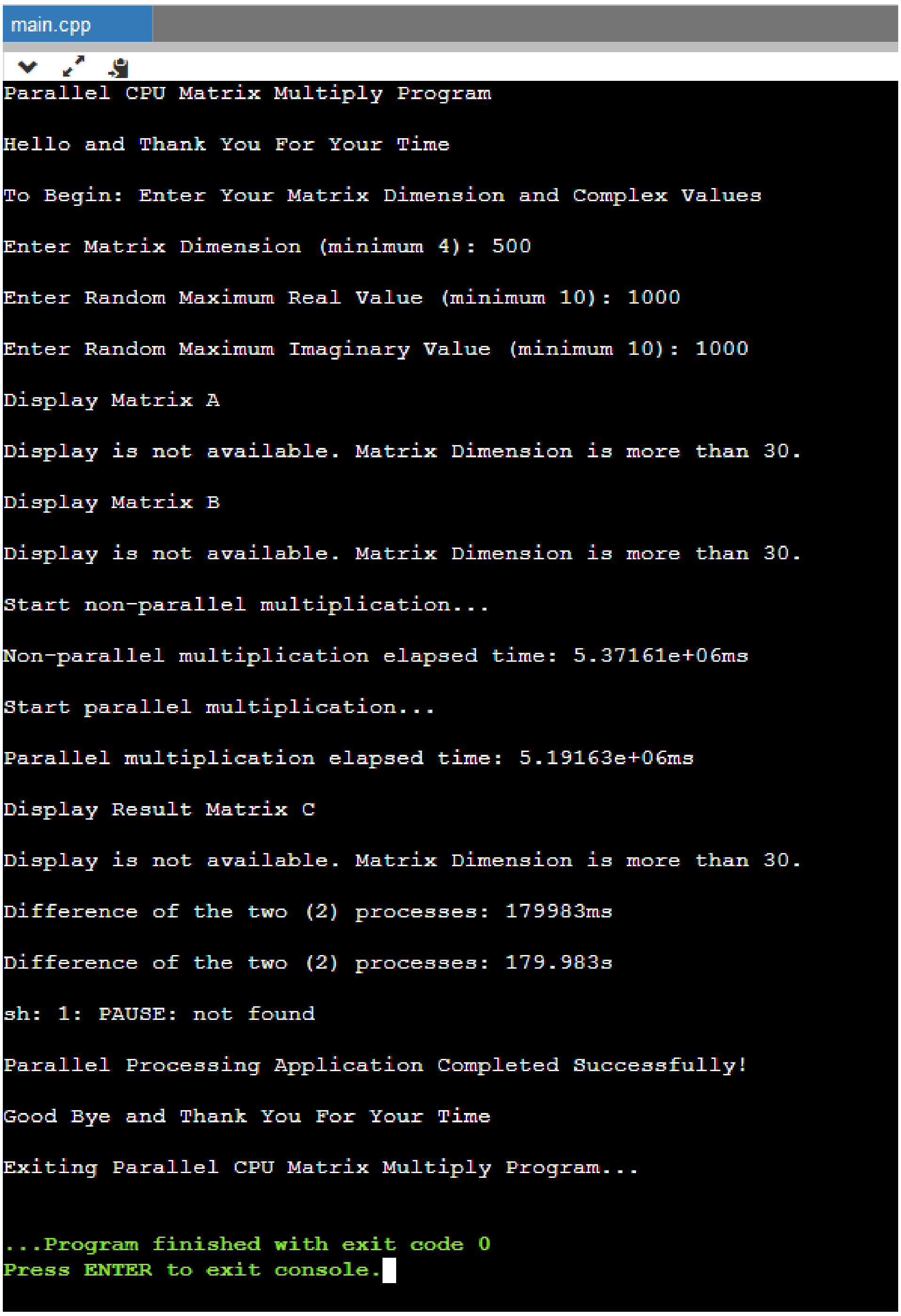
- DimVal = **250**, RealVal = **500** and ImagVal = **500i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 250

Enter Random Maximum Real Value (minimum 10): 500

Enter Random Maximum Imaginary Value (minimum 10): 500

Display Matrix A

Display is not available. Matrix Dimension is more than 30.

Display Matrix B

Display is not available. Matrix Dimension is more than 30.

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 781930ms

Start parallel multiplication...

Parallel multiplication elapsed time: 763474ms

Display Result Matrix C

Display is not available. Matrix Dimension is more than 30.

Difference of the two (2) processes: 18456ms

Difference of the two (2) processes: 18.456s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...


...Program finished with exit code 0
Press ENTER to exit console.
```

3.4 Execution #4

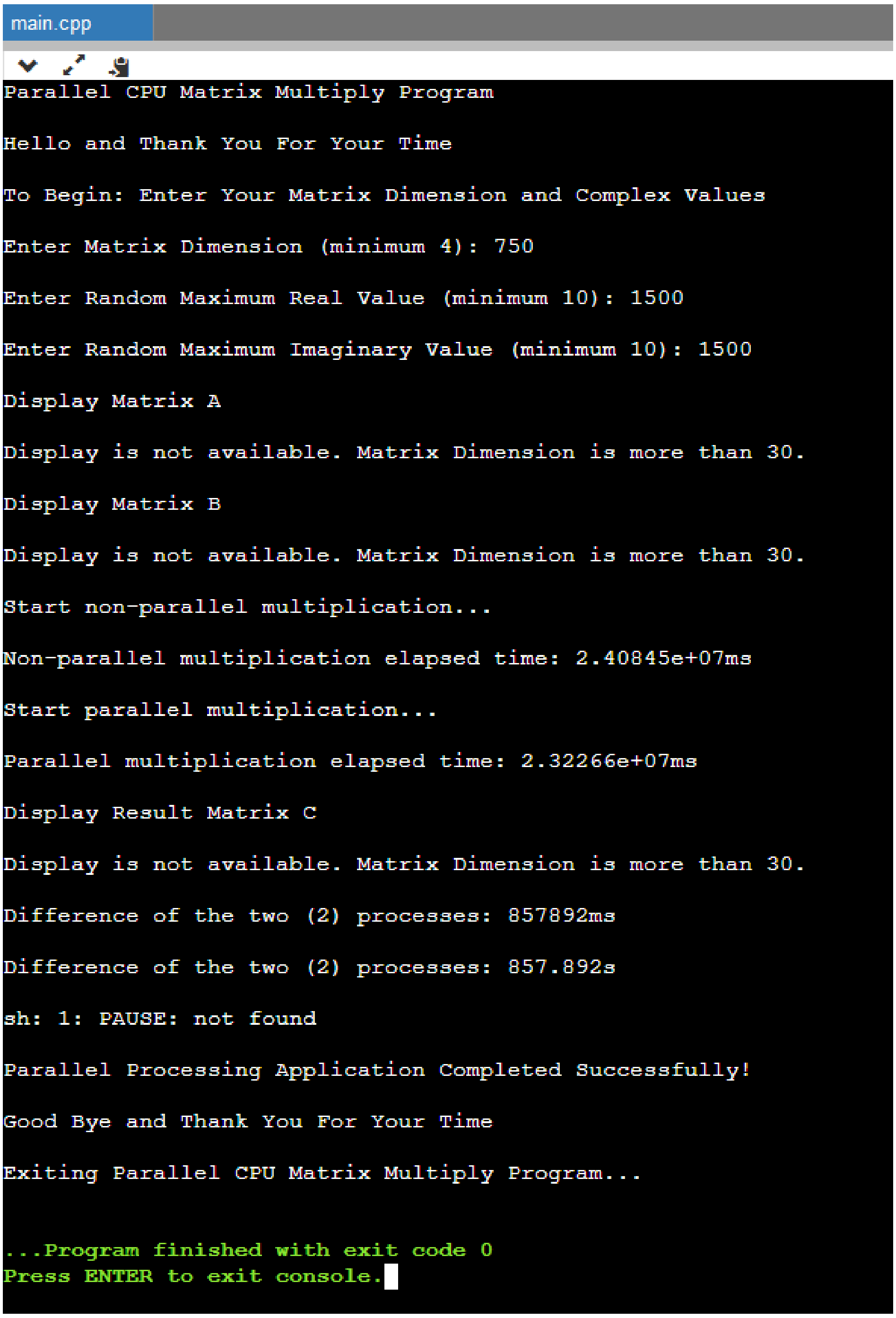
- DimVal = **500**, RealVal = **1000** and ImagVal = **1000i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 500

Enter Random Maximum Real Value (minimum 10): 1000

Enter Random Maximum Imaginary Value (minimum 10): 1000

Display Matrix A

Display is not available. Matrix Dimension is more than 30.

Display Matrix B

Display is not available. Matrix Dimension is more than 30.

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 5.37161e+06ms

Start parallel multiplication...

Parallel multiplication elapsed time: 5.19163e+06ms

Display Result Matrix C

Display is not available. Matrix Dimension is more than 30.

Difference of the two (2) processes: 179983ms

Difference of the two (2) processes: 179.983s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...


...Program finished with exit code 0
Press ENTER to exit console.
```

3.5 Execution #5

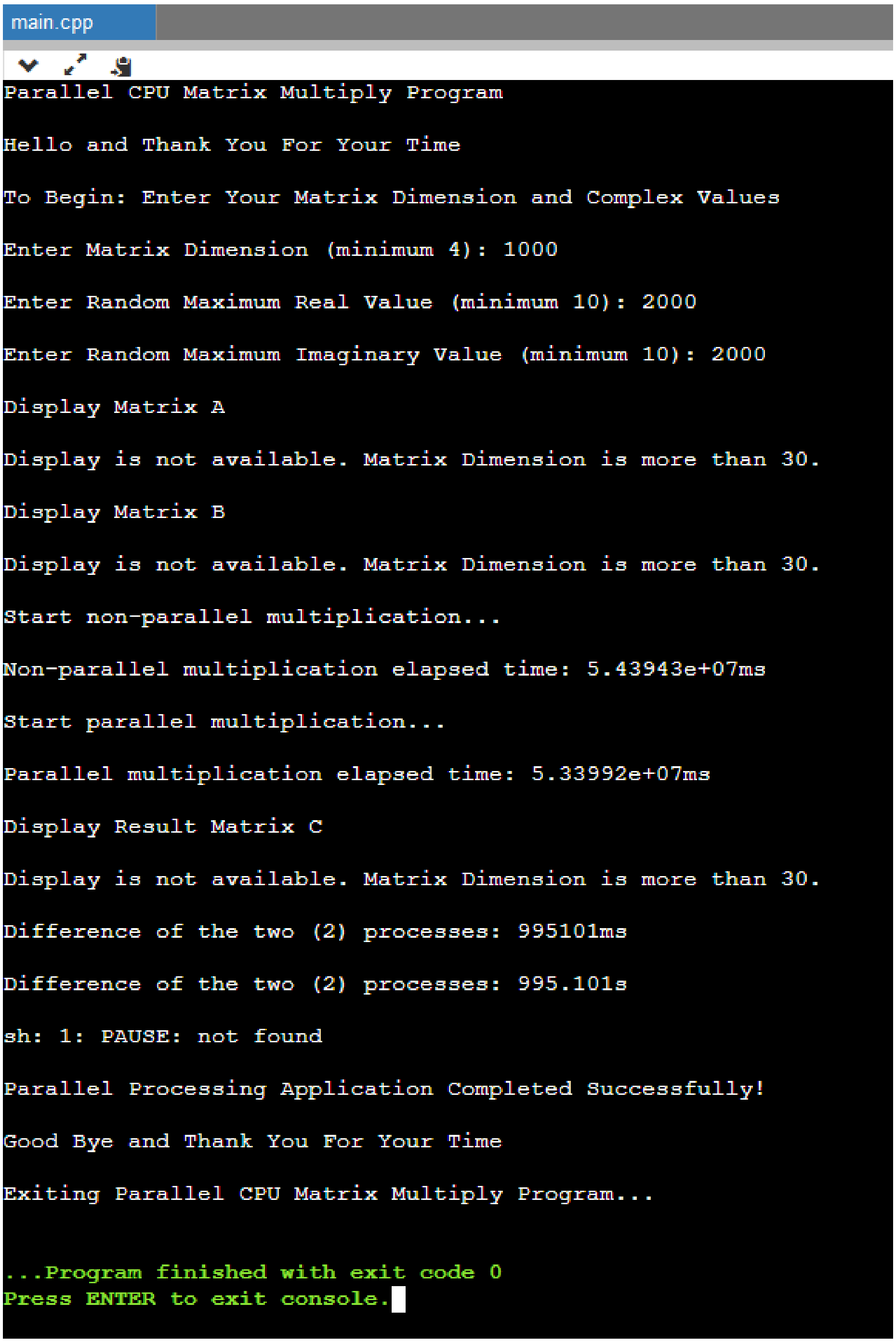
- DimVal = **750**, RealVal = **1500** and ImagVal = **1500i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 750

Enter Random Maximum Real Value (minimum 10): 1500

Enter Random Maximum Imaginary Value (minimum 10): 1500

Display Matrix A

Display is not available. Matrix Dimension is more than 30.

Display Matrix B

Display is not available. Matrix Dimension is more than 30.

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 2.40845e+07ms

Start parallel multiplication...

Parallel multiplication elapsed time: 2.32266e+07ms

Display Result Matrix C

Display is not available. Matrix Dimension is more than 30.

Difference of the two (2) processes: 857892ms

Difference of the two (2) processes: 857.892s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...


...Program finished with exit code 0
Press ENTER to exit console.
```

3.6 Execution #6

- DimVal = **1000**, RealVal = **2000** and ImagVal = **2000i**



```
Parallel CPU Matrix Multiply Program

Hello and Thank You For Your Time

To Begin: Enter Your Matrix Dimension and Complex Values

Enter Matrix Dimension (minimum 4): 1000

Enter Random Maximum Real Value (minimum 10): 2000

Enter Random Maximum Imaginary Value (minimum 10): 2000

Display Matrix A

Display is not available. Matrix Dimension is more than 30.

Display Matrix B

Display is not available. Matrix Dimension is more than 30.

Start non-parallel multiplication...

Non-parallel multiplication elapsed time: 5.43943e+07ms

Start parallel multiplication...

Parallel multiplication elapsed time: 5.33992e+07ms

Display Result Matrix C

Display is not available. Matrix Dimension is more than 30.

Difference of the two (2) processes: 995101ms

Difference of the two (2) processes: 995.101s

sh: 1: PAUSE: not found

Parallel Processing Application Completed Successfully!

Good Bye and Thank You For Your Time

Exiting Parallel CPU Matrix Multiply Program...


...Program finished with exit code 0
Press ENTER to exit console.
```

4. References

[1] **Wikipedia contributors**. "Matrix (mathematics)." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 12 May. 2021. Web. 16 May. 2021.

[2] **Wikipedia contributors**. "Matrix multiplication." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 5 May. 2021. Web. 16 May. 2021.

[3] **Wikipedia contributors**. "Complex number." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 6 May. 2021. Web. 16 May. 2021.

[4] **Cplusplus.com**, C++ Language, https://www.cplusplus.com/doc/tutorial, Web. 18 Apr. 2021.

[5] **GeeksforGeeks**, Multiplication of Matrix using threads, https://www.geeksforgeeks.org/multiplication-of-matrix-using-threads, Web. 18 Apr. 2021.

[6] **OnlineGDB.com**, Online_C++_compiler, https://www.onlinegdb.com/online_c++_compiler, Web. 16 May. 2021.

[7] **Github.com**, Research-Programming-Work, https://github.com/jeremyjwsc/Research-Programming-Work, Web. 16 May. 2021.