Parallel and Distributed Computing
CSE4001

LU Decomposition using OpenMP

...

By Yashraj Agarwal(18BCI0183)

LU Decomposition

L U decomposition of a matrix is the factorization of a given square matrix into two triangular matrices, one upper triangular matrix and one lower triangular matrix, such that the product of these two matrices gives the original matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 0 \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

The Code

```
#include <bits/stdc++.h>
#include<omp.h>
using namespace std;

const int MAX = 100;

void luDecomposition(int mat[][MAX], int n)
{
        int lower[n][n], upper[n][n];
        memset(lower, 0, sizeof(lower));
        memset(upper, 0, sizeof(upper));

        int i,j,k;

        // Decomposing matrix into Upper and Lower
        // triangular matrix
        for ( i = 0; i < n; i++) {

                // Upper Triangular
                #pragma omp parallel for shared(mat,n,i) private(k,j) schedule(static,64)
                for ( k = i; k < n; k++) {

                        // Summation of L(i, j) * U(j, k)
                        int sum = 0;
```

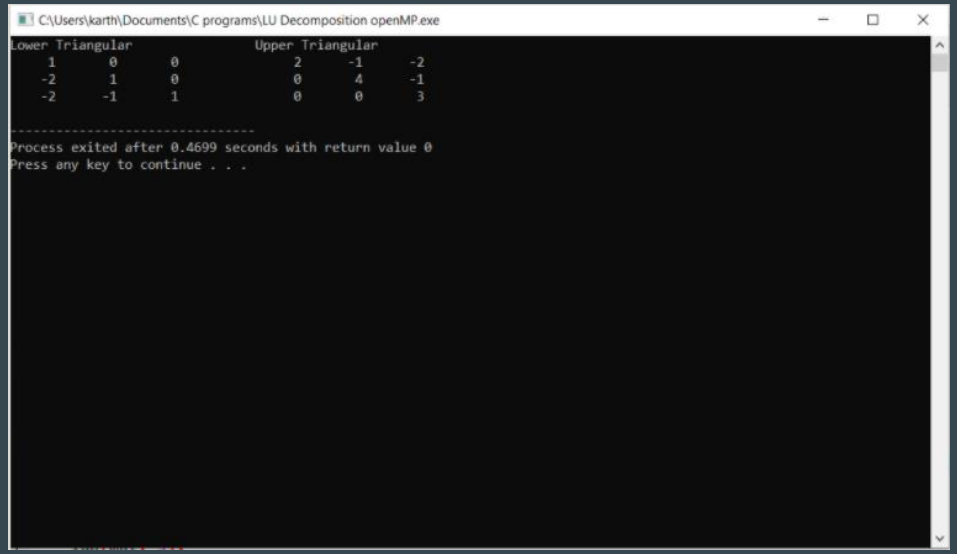
The Code

```
for ( j = 0; j < i; j++)
                sum += (lower[i][j] * upper[j][k]);

                // Evaluating U(i, k)
                upper[i][k] = mat[i][k] - sum;
}
// Lower Triangular
                        for ( k = i; k < n; k++) {
                                if (i == k)
                                        lower[i][i] = 1; // Diagonal as 1
                                else {


                                        // Summation of L(k, j) * U(j, i)
                                        int sum = 0;
                                        for ( j = 0; j < i; j++)
                                                sum += (lower[k][j] * upper[j][i]);

                                        // Evaluating L(k, i)
                                        lower[k][i] = (mat[k][i] - sum) / upper[i][i];
                                }
                        }
                }
```

The Code

```
// setw is for displaying nicely
        cout << setw(6) << "           Lower Triangular"
                << setw(32) << "Upper Triangular" << endl;

        // Displaying the result :
        for (int i = 0; i < n; i++) {
                // Lower
                for (int j = 0; j < n; j++)
                        cout << setw(6) << lower[i][j] << "\t";
                cout << "\t";

                // Upper
                for (int j = 0; j < n; j++)
                        cout << setw(6) << upper[i][j] << "\t";
                cout << endl;
        }
}

// Driver code
int main()
{
        int mat[][MAX] = { { 2, -1, -2 },
                                                        { -4, 6, 3 },
                                                        { -4, -2, 8 } };

        luDecomposition(mat, 3);
        return 0;
}
```

Output of the Code



Hence we find the solutions of the linear equations using the LU Decomposition Method implemented using OpenMP.