# Indian Institute of Information Technology, Vadodara

Parallel Programming(CS403)

Prof. Reshmi Mitra

**Name** - Dilip Puri
**ID** - 201351014
**Collaborators** - Hemant Kumar(201352026)
Govind Meena(201352010)

**Project 1**

**Submission Date** - September 12, 2016 **Deadline** - Sep12, 11.59 PM

## 1 Introduction

Suppose we have the system of equations

$$AX=B$$

Even most of the mathematical model follows these kind of systems. The motivation for an LU decomposition is based on the observation that systems of equations involving triangular coefficient matrices are easier to deal with. Indeed, the whole point of Gaussian Elimination is to replace the coefficient matrix with one that is triangular. The LU decomposition is another approach designed to exploit triangular systems. We suppose that we can write

$$A = LU$$

where L is a lower triangular matrix and U is an upper triangular matrix. Our aim is to find L and U and once we have done so we have found an LU decomposition of A. Let A be a square matrix. If there is a lower triangular matrix L with all diagonal entries equal to 1 and an upper matrix U such that A=LU, then we say that A has an LU-decomposition. It can be helpful in calculating various types of operation on matrices. Here L matrix has the upper triangular values as 0 and U has lower triangular values as 0 and diagonal values same as that of the original matrix.

## 2 Algorithm

To decompose matrix A into L and U we are using Doolittle's method which is described as follows

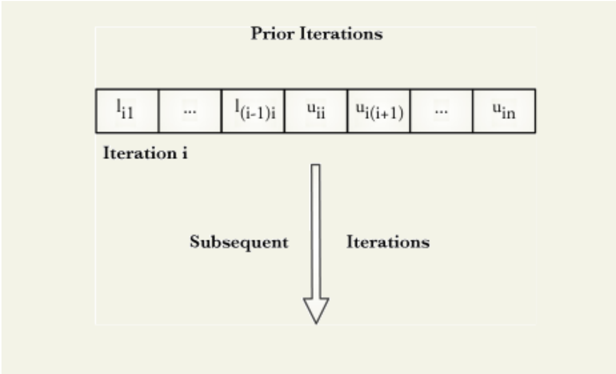


Figure 1: Computational Sequence of Doolittle's Method

**for** $i = 1, \cdots, n$
    **for** $j = 1, \cdots, i-1$
        $\alpha = a_{ij}$
        **for** $p = 1, \cdots, j-1$
            $\alpha = \alpha - a_{ip}a_{pj}$
        $a_{ij} = \dfrac{\alpha}{a_{jj}}$
    **for** $j = i, \cdots, n$
        $\alpha = a_{ij}$
        **for** $p = 1, \cdots, i-1$
            $\alpha = \alpha - a_{ip}a_{pj}$
        $a_{ij} = \alpha$

Figure 2: Doolittle's LU Decompostion Algorithm

# 3 Serial Code

Listing 1: Code

```c
#include<stdio.h>

int main(void){

    int i,j,k,n;
    printf("Enter the order of square matrix: ");
    scanf("%d",&n);

    float A[n][n],L[n][n], U[n][n];//initializing matrices

    printf("Enter matrix element:\n");

    for(i=0; i<n; i++) //matrix input from console
    {
        for(j=0; j<n; j++)
        {
            printf("Enter A[%d][%d] element: ", i,j);
            scanf("%f",&A[i][j]);
        }
    }
    //DooLittle's Algo.
    for(j=0; j<n; j++) //over row element
    {
        for(i=0; i<n; i++)//over column element
```

```
        {
            if(i<=j)     //check for diagonal and lower elements
            {
                U[i][j]=A[i][j];
                for(k=0; k<=i-1; k++)
                    U[i][j] = U[i][j] - L[i][k]*U[k][j];
                if(i==j)
                    L[i][j]=1;
                else
                    L[i][j]=0;
            }
            else         //check for upper elements
            {
                L[i][j]=A[i][j];
                for(k=0; k<=j-1; k++)
                    L[i][j]= L[i][j] - L[i][k]*U[k][j];
                L[i][j] = L[i][j] / U[j][j];
                U[i][j]=0;
            }
        }
    }

    printf("[L]: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            printf("%9.3f",L[i][j]);
        printf("\n");
    }
    printf("\n\n[U]: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            printf("%9.3f",U[i][j]);
        printf("\n");
    }

return 0;
}
```

# 4 Analysis using Valgrind

| Size | Time |
|---|---|
| 2x2 | 0m4.699s |
| 3x3 | 0m9.880s |
| 4x4 | 0m17.671s |

| Event Type | Incl. | Self | Short | Formula |
|---|---|---|---|---|
| Instruction Fetch | 0.15 | 0.15 | Ir | |
| L1 Instr. Fetch Miss | 0.61 | 0.61 | I1mr | |
| LL Instr. Fetch Miss | 0.62 | 0.62 | ILmr | |
| Data Read Access | 0.09 | 0.09 | Dr | |
| L1 Data Read Miss | 0.00 | 0.00 | D1mr | |
| LL Data Read Miss | 0.00 | 0.00 | DLmr | |
| Data Write Access | 0.16 | 0.16 | Dw | |
| L1 Data Write Miss | 0.00 | 0.00 | D1mw | |
| LL Data Write Miss | 0.00 | 0.00 | DLmw | |
| L1 Miss Sum | 0.24 | 0.24 | L1m = | I1mr + D1mr + D1mw |
| Last-level Miss Sum | 0.26 | 0.26 | LLm = | ILmr + DLmr + DLmw |
| Cycle Estimation | 0.22 | 0.22 | CEst = | Ir + 10 L1m + 100 LLm |

Figure 3: code analysis with valgrind and kchachegrind

You can find cachegrind here `https://raw.githubusercontent.com/dilippuri/parallel-programming-la` `master/projects/project_1/cachegrind.out.5590`