Technische Universität Berlin
Faculty IV
Dept. of Computer Engineering and Microelectronics
Embedded Systems Architecture Group

Advanced Computer Architecture Course
Instructor: Nadjib Mammeri
Summer Semester 2018

Report

4th Assignment
Multithreading

Schenck, Tianhao Alissa
Mor, Keyshav

July 17, 2018

1. Introduction

In this assignment we worked with a sequential code implementing the LU decomposition algorithm. The objective of this algorithm is to factor a matrix A as the product of a lower triangular matrix L and an upper triangular matrix U. The algorithm that was applied in this assignment resulted in a unit lower triangular matrix, i.e. the diagonal elements of L are ones, as shown in equation 1. The code of the sequential program is shown in listing 1. First, we parallelized this sequential LU Decomposition program with OpenMP directives. Then we modified the algorithm and parallelized it again using OpenMP to provide an optimized version for best performance.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \dots & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \dots & 0 \\ l_{31} & l_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix} \tag{1}$$

2. Parallized LU Decomposition

As a first step of Parallelization we have to identify these parts of the program that can run in parallel. The algorithm shown in listing 1 reads values in the i-th loop iteration that have been written in the (i-1)-th iteration, hence the iterations of i-loop can not run in parallel. The two j-loops iterate through the elements of one row or column of the matrix. The operations that are performed on the elements in one row or column are independent, thus the iterations of both j-loops can be parallelized, but the upper j-loop has to finish before the lower one can start executing and vice versa. Also the k-loops that are nested in the j-loops need to run in order because they accumulate a value in the variable sum.

With this knowledge in mind we insert the OpenMP directive #pragma omp parallel for private(j,k,sum) shared(a,size,i) above the j-loops to implement the multithreading. This directive opens a parallel contruct. The parallel construct creates a team of threads and the iterations of the j-loops will be divided among the threads. Unless otherwise specified the number of threads equals the number of cores. When a thread encounters the nested k-loop, it will executed it in order, because no new parallel construct is opened here. OpenMP is a parallel programming model for shared-memory multi-processor systems. In order to achieve correct program behavior we have to specify which variables are shared and which ones are private. The variables a, $size$ and i need to be shared, because all threads operate on the same a, $size$ and i. The algorithm would not work, if each thread had a local copy with a possibly different value. The variables j, k and sum need to be private in order for the iterations of the j-loop to run in parallel.

The parallelized program 'lud_par.cpp' is part of the deliverables.

3. Optimized LU Decomposition

4. Evaluation of Speedup

A. Sequential Program Code

```cpp
void compute_lud(float *a, int size) {

    int i,j,k;
    float sum;

    for (i=0; i<size; i++) {
        for (j=i; j<size; j++) { //computes the i-th row of U
            sum=a[i*size+j];
            for (k=0; k<i; k++) {
                sum -= a[i * size + k] * a[k * size + j];
            }
            a[i*size+j]=sum;
        }

        for (j=i+1;j<size; j++){ //computes the i-th column of L
            sum=a[j*size+i];
            for (k=0; k<i; k++) {
                sum -= a[j * size + k] * a[k * size + i];
            }
            a[j*size+i]=sum/a[i*size+i];
        }
    }
}
```

Listing 1: lud_seq.cpp