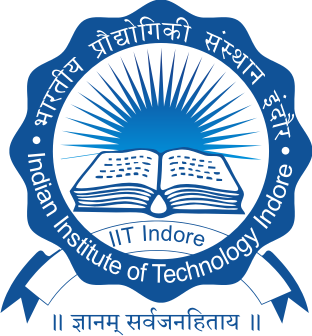
INDIAN INSTITUTE OF TECHNOLOGY INDORE



Department Of Computer Science and Engineering

Project Report

Parallel Implementation for Doolittle Algorithm using OpenMP for Multicore Machines

3rd Year (Autumn 2017)

Parallel Computing

Under the Guidance of

Dr. Surya Prakash

Submitted By

Priyanka Meena (150001023)

Kunal Sikri (150001016)

Table Of Contents:

* Acknowledgement…………………………
* Introduction…………………………………
* Abstract……………………………………...
* Algorithm Design
  + Sequential Algorithm Design……….
  + Parallel Algorithm Design…………..
  + Comparing Times…………………….
* Experimental Results…………………….
* Conclusion and Further Work…………..
* References…………………………………….

Acknowledgment

We would like to express our sincere gratitude to our supervisor Dr. Surya Prakash and teaching assistants Mr. G Iyyakutti Iyyapan and Mr. Sadaf Ali for providing their invaluable guidance, comments and suggestions throughout the course of the project. We would specially thank Dr. Surya

Prakash for constantly motivating us to work harder.

Also, we would like to thank the above-mentioned teaching assistants for their thorough guidance and efforts during the lab sessions in order to clear out our concepts which helped in carrying out our preparation of the project.

Introduction

With the rapid increase in use of multicore processors, parallelization of applications has become a priority task. In order to take advantage of the multi-core architecture of modern processors, the legacy serial code must be analyzed to discover the regions where the parallelization effort can be more rewarding.We present a parallel implementation of Doolittle Algorithm using OpenMP allowing the users to utilize the multiple cores present in the modern CPUs.

The Serial Doolittle Algorithm is analyzed for computing the solution of dense system of linear equations, and is parallelized in C using the OpenMP library which makes it highly efficient, cross-platform compatible and scalable.

Abstract

The computations of dense matrices are very important and mostly used scientific program kernels appears in various scientific applications. The LU factorization is a direct scheme that can solve big systems of linear equations which come from most important area of applications, such as simulation of circuits, structural analysis and power networks. Given a problem AX = B, we can apply several methods to frame a solution.The recent method to solve a linear equation AX = B is to find initially the A = LU factorization.

LU decomposition can be used to find inverse of a matrix, Determinant of a matrix as well as to solve system of linear equations.

The sequential Doolittle algorithm is a matrix decomposition which writes a matrix as the product of a upper triangular matrix and an lower triangular matrix.

The main challenge was to implement the algorithm parallelly such that a speedup is observed in case of higher number of unknowns.

Algorithm Design

Sequential Algorithm Design :-

The algorithm that was used to calculate Lower triangular matrix (L) and upper triangular matrix (U)from a given matrix A.

The matrix generation is using rand() function which generates random values for matrices A and Initialized the values of Upper triangular matrix and lower triangular matrix to Zero.

Since the rand() function generates fixed random values we are using srand() to pass seed as an argument where seed is initialized by using the function gettimeoftheday(). srand() is basically used to randomize rand() by using this seed.

*for(k=0;k<n;k++) {*

*lwr[k][k]=1;*

*for(j=k;j<n;j++){*

*ld sum=0;*

*for(int s=0;s<=k-1;s++){*

*sum+=lwr[k][s]\*upr[s][j];*

*}*

*upr[k][j]=a[k][j]-sum;*

*}*

*for(i=k+1;i<n;i++){*

*ld sum=0;*

*for(int s=0;s<=k-1;s++) {*

*sum+=lwr[i][s]\*upr[s][k];*

*}*

*lwr[i][k]=(a[i][k]-sum)/upr[k][k];*

*}*

*}*

Parallel Algorithm Design :-

To have speed up in the algorithm we need to implement it parallelly i.e. it was necessary to divide the tasks of computation among the CPU processors.The legacy sequential code was analyzed to determine the regions which are independent of each other and where the execution takes more time and was implemented as parallel code. Data parallelism is used to parallelize the algorithm.

In our analysis we found that data independence exists between calculating the row elements of any column in lower triangular matrix and any other row element in that column.Similarly, calculating the column elements of any row in an upper triangular matrix is independent of any other column element in that row. This independence in calculation of elements in upper and lower triangular matrix gave us the scope to use data parallelism. The row or column is split and the row or column elements are calculated simultaneously.This is done in order to reduce the execution time as compared to serial execution.

The implementation strategy mentioned here is based on an already existing sequential implementation of the algorithm written in C. The appropriate OpenMP directives are added to it and the necessary changes have been made. Doolittle algorithm was implemented that uses the block wise decomposition in parallel.The for loops are parallelized in a manner that blocks of matrices are decomposed by dividing the work among parallel threads.

*#pragma omp parallel shared(lwr,upr,a,nthreads) private(tid,i,k,j)*

*{*

*tid=omp\_get\_thread\_num();*

*if(tid==0)*

*nthreads=omp\_get\_num\_threads();*

*for(k=0;k<n;k++){*

*lwr[k][k]=1;*

*#pragma omp for schedule(static,chunk)*

*for(j=k;j<n;j++){*

*ld sum=0;*

*for(int s=0;s<=k-1;s++){*

*sum+=lwr[k][s]\*upr[s][j];*

*}*

*upr[k][j]=a[k][j]-sum;*

*}*

*#pragma omp for schedule(static,chunk)*

*for(i=k+1;i<n;i++){*

*ld sum=0;*

*for(int s=0;s<=k-1;s++){*

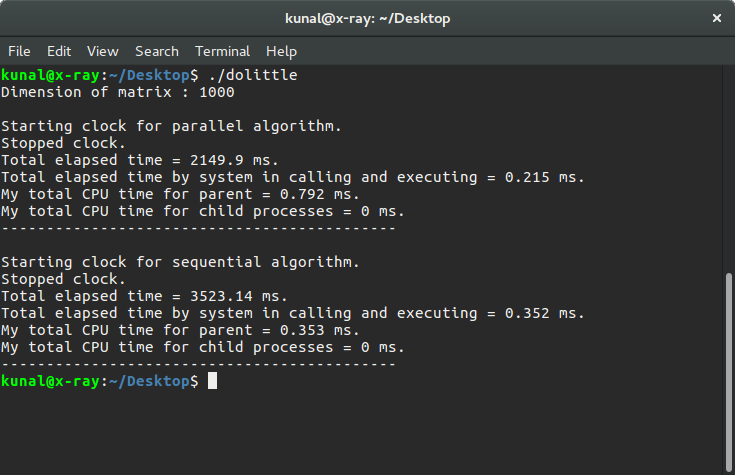
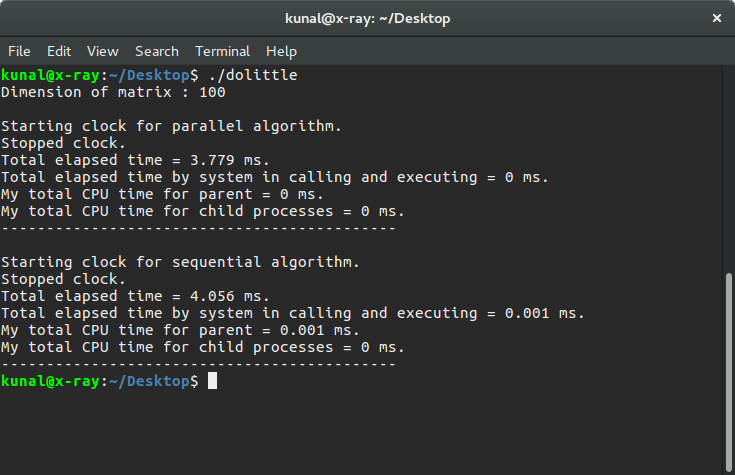
*sum+=lwr[i][s]\*upr[s][k];*

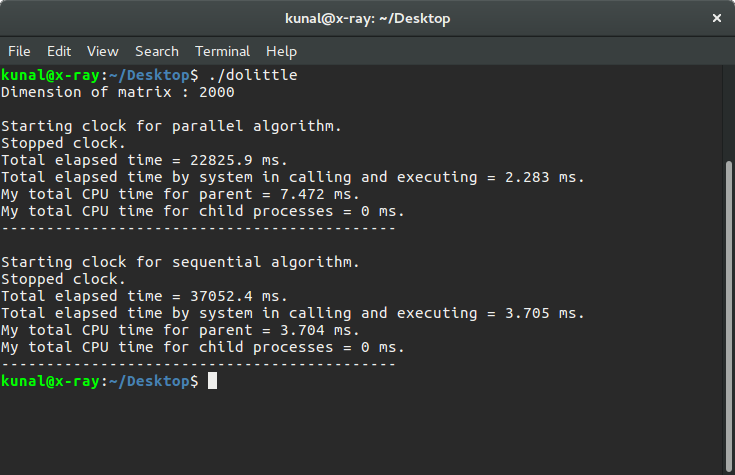
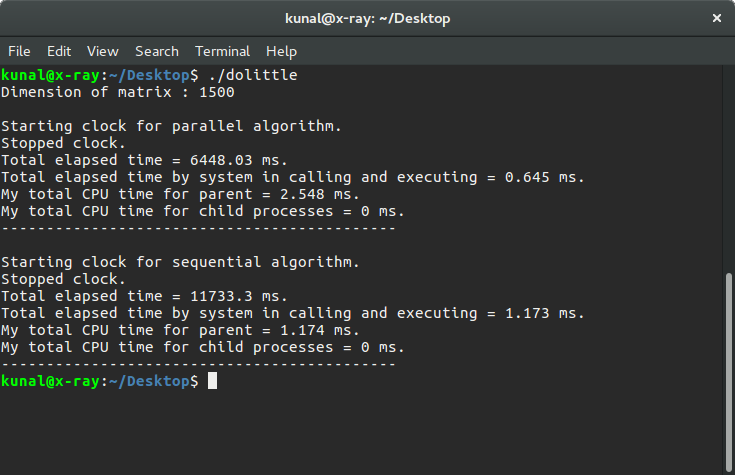
*}*

*lwr[i][k]=(a[i][k]-sum)/upr[k][k];}}}*

Comparing Times

To observe the speedup or the time difference between the sequential and parallel implementations, we used built-in structures in C which include timezone and timeval which were used along with gettimeoftheday(). All the mentioned functions and structures are included in the library times.h and sys/times.h. For calculating CPU time elapses, we used the structure tms which include CPU start and end time for calling function and also for child function. The tms structure include u\_time for calculating the user time and s\_time for system time which helps in observing to the greater precision.





Experimental Results:

We implemented both Serial Doolittle Algorithm and the Parallel Doolittle Algorithm in the Quad core machine with the following configuration:

Processor: 4x Intel(R) Core(™) i5-3210M CPU @2.50GHz

RAM: 4 GB, DDR3 1600 MHz

OS: Ubuntu 16.04.3 LTS

Compiler: GCC, OpenMP

In this work Doolittle algorithm was implemented that uses the block wise decomposition in parallel using OpenMp.The or loops are parallelized in a manner that blocks of matrices are decomposed by dividing the work among parallel threads.Algorithm was evaluated for input matrix of sizes 100, 1000,1500,2000.

The results on a multi-core processor show that the proposed Parallel Doolittle Algorithm achieves good performance (speedup) compared to the sequential algorithm.

Conclusion and Future Work

In this work we have presented parallel implementation of the Doolittle Algorithm using OpenMP allowing the users to utilize the multiple cores present in the modern CPUs. The experimental results on a multi-core processor show that the proposed Parallel Doolittle Algorithm achieves good performance (speedup) compared to the sequential algorithm. For large order matrices Parallel Doolittle Algorithm can be readily used. In future, to further reduce the execution time of the Doolittle Algorithm we can utilize GPGPU (General Purpose Computing on Graphical Processing Unit) [6] and parallelize using CUDA, OpenCL.

References

1. <http://www.geeksforgeeks.org/>
2. <http://ieeexplore.ieee.org/>
3. http://mathonline.wikidot.com/