Declaration on Plagiarism

This form must be filled in and completed by the student(s) submitting an assignment

Name:	Ajit Kumar
Student Number:	19210438
Programme:	Concurrent Programming
Module Code:	CA670
Assignment Title:	Efficient Large Matrix Multiplication in OpenMP
Submission Date:	20 April 2020
Module Coordinator:	Dr. David Sinclair

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found at
http://www.dcu.ie/info/regulations/plagiarism.shtml,
https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines

Name: Ajit Kumar
Date: 20th April 2020

Efficient Large Matrix Multiplication in OpenMP

Introduction

The program implements an efficient large matrix multiplication algorithm using ijk implementation method in OpenMP.OpenMP is an API which is used for writing multi-threaded application which works on fork join model.it contains a set of compiler directives and library routines for parallel application programs which greatly simplifies writing multi-threaded programs. An OpenMP framework often starts with a single control thread, called the master thread, which persists for program length. In this problem statement efficiency of traditional matrix multiplication, matrix multiplication using pragma parallel construct and an optimized matrix multiplication using pragma parallel construct has been analysed on the square matrix of different dimensions ranging from 200 to 2000 with a step size of 200.

Design

- This program is designed in a very simple way, it contains five different functions named generate_random_matrix,convert2d_to_1d,matrix_multiplication, parallel_matrix_multiplication, optimized_matrix_multiplication and execution of the program begins with int main() function which is the entry point of the program.

- Inbuilt malloc () function which comes under <stdlib.h> is used for dynamically assigning the memory block of a specified size at heap as we have to work on matrices of different dimensions in order to analyse the performance. In this an array of a pointer for different matrix a[i][j], matrix b[i][j] and resultant matrix mul[i][j] is used to dynamically assigned the memory using double pointer. After the program gets executed the space which is created to solve the problem is then free using free () function in order to avoid the "Segmentation fault".

- To get the time taken by all the three different methods to complete the process clock () function is used. Clock () is used at the beginning and at the end of the task, then the value is subtracted to get the difference and the same is divide by CLOCK_PER_SEC to get the processor time.

- Generate_random_matrix () the function takes the dimensions of the matrix row r, column c, matrix a and matrix b as an argument and used to generate random matrix of the respective dimension depends on user input. In this Rand () function with modulo 100 is used to generate random number between 0 to 99 which is system dependent.

- Matrix_multiplication () the function takes the dimensions of the matrix row r, column c, matrix a, matrix b and resultant matrix mul as an argument. This is the very first approach of solving the problem which is a traditional way of matrix multiplication. Here matrix a[i][j] and matrix b[i][j] are the input matrix whereas matrix mul[i][j] is resultant matrix. In this operation of each dimension at the resultant matrix takes in a sequential manner.

- parallel_matrix_multiplication () the function takes the dimensions of the matrix row r, column c, matrix a, matrix b and resultant matrix mul as an argument. In this loop, parallelization is implemented using the OpenMP library. OpenMP directive #pragma omp parallel for loop construct was used to parallelize the outermost for loop. It takes the immediate following for loop and split it up to its iteration between different threads.

- optimized_matrix_multiplication () the function takes the dimensions of the matrix row r, column c, matrix a, matrix b and resultant matrix mul as an argument.in this various optimization techniques are used so that different instruction can be executed efficiently. Firstly, inside this #pragma omp parallel for loop along with schedule, shared, private and number of threads keyword. Here schedule keyword used to give information to the compiler which says here is how the iteration of the loop are broken down and split between the threads. #pragma omp for schedule(static) is used assuming that each iteration has a relatively stable cost per iteration. #pragma omp parallel shared(mul) private(i,j,k,newi,newj,total) num_threads(40) is used which means that matrix mul is used as a shared resource to avoid race condition. Data scoping private clause is used to specify that the variable is local to each thread and 40 threads are used to do the multiplication process. Secondly, memory optimization is done as our matrices are stored in heap and accessing them after every iteration through heap will increase the cost, so the data from the heap is stored to the stack before multiplication. The 2D matrices are converted to 1D matrices using convert2d_to_1d () function and the same has been parallelized using #pragma omp parallel for. Thirdly localization of variable is done as in ijk algorithm it keeps on updating the same variable mul[i][j],so in order to improve efficiency a local variable total is used to store the sum and after the iteration of innermost loop is over, the same is updated to the mul[i][j].

Results

The following below mentioned table includes time taken in seconds at every approach for each dimension matrix.

Dimension of matrix	Traditional Approach	Matrix multiplication using parallel for loop	Optimized Matrix multiplication using parallel for loop
200	0.092	0.056	0.036
400	0.828	0.384	0.140
600	2.897	0.989	0.464
800	7.984	2.257	1.127
1000	17.619	4.989	2.189
1200	34.141	8.576	3.782
1400	58.278	14.145	5.996
1600	89.569	23.079	8.918
1800	133.643	32.799	12.846
2000	213.626	67.558	17.862

Conclusion

Optimized matrix multiplication using parallel for loop using OpenMP is suitable and efficient for large matrix multiplication. As seen in the above table that it 12 times faster than the traditional approach and 4 times faster than matrix multiplication using the parallel for loop approach as the dimension of the matrix increases.

References

[1] C. É. Drevet, M. Nazrul Islam, and É. Schost, “Optimization techniques for small matrix multiplication,” Theor. Comput. Sci., vol. 412, no. 22, pp. 2219–2236, 2011.

[2] N. Anchev, M. Gusev, S. Ristov, and B. Atanasovski, “Some optimization techniques of the matrix multiplication algorithm,” Proc. Int. Conf. Inf. Technol. Interfaces, ITI, pp. 71–76, 2013.

[3] A. Mathematics, S. Kartheeswaran, and G. Arts, “Analyzing the Matrix Multiplication Performance in Shared memory processor Under Multicore Architecture Using OpenMP,” vol. 119, no. 15, pp. 3249–3256, 2018.

[4] https://github.com/roshanmadhushanka/Parallel-Matrix-Multiply/blob/master/pmp.c

[5] Generate a Random Matrix
https://scriptverse.academy/tutorials/c-program-generate-random-matrix.html

[6] Dynamic Memory Allocation in C using malloc (), calloc(), free() and realloc()
https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

[7] How to dynamically allocate a 2D array in C
https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/

[8] How to measure time taken by a function in C
https://www.tutorialspoint.com/how-to-measure-time-taken-by-a-function-in-c