Matrix Multiplication using OpenMP

Mitul Verma
19210961
MCM Computing (Data Analytics)
Dublin City University
mitul.verma3@mail.dcu.ie

Plagiarism Statement

I certify that this assignment is my work, based on my study and research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment has not previously been submitted for assessment in any other unit. except some specific parts of the assignment which has been cited from all unit coordinators involved. I have not copied in part or whole or otherwise plagiarized the work of other students or persons**.**

Name: Mitul Verma (19210961)

Date: 19th April 2020

Race Condition:- In race condition every thread tries to execute the shared variable at the same time. This result in inconsistency of the output
Cache miss:- During an execution of code, the CPU first looks for the data in cache memory. If the data is found that is known as a cache hit. Otherwise cache miss.
Memory bound:- Refers to a situation where the time required to complete a computational problem directly related to the amount of memory required to hold data.

Abstract

OpenMP is an interface that is used to create multiprocessing, shared-memory application in C/C++, Fortran and on many other programming interfaces. This Document consist of a detailed explanation of how matrix multiplication can be optimized in C++ using OpenMP. The work describes the various methodology used to optimized Matrix multiplication and provided the best possible solution. The document consists of Introduction, Assumption, System Design, Code Efficiency, Results, Conclusion, and References.

Keywords—Open MP, Transpose, cache and Block.

I) INTRODUCTION

In this study, I investigated various methods that can be applied to reduce the execution time of matrix multiplication and created the most optimized code possible. The document talks about the approach to perform Matrix Multiplication followed by optimized approaches for matrix multiplication. The results generated at each step are used to compare different methods. Based on the result the best-optimized method is selected.

II) Assumptions

- The Matrix Multiplication is of order A*B.
- The Dimension of both Matrix is N*N.

III) Design Description

The Program consists of one main function followed by four user-defined functions. Each function shows different optimized ways of matrix multiplication in c/c++ using open MP. The Execution of the program starts with user inputs. The user is supposed to provide the dimension of the matrix and block size. Every next function is an optimized version of the one before it. Execution time of matrix multiplication for different dimensions are recorded and compared to provide evidence on the efficiency of each code and to select the best-optimized code.

Below section provide a brief introduction of all the function created.

A) The first method implemented show a Normal Matrix Multiplication.

```
for i =1 to n
for j=1 to n
{c[i][j] = 0
  for k =1 to n
   {c[i][j] += (matrix_a[i][k] * matrix_b[i][j])
```

B) The Second Method is optimization of the first one. The second method show implementation of parallel execution using OpenMP.

```
#Pragma omp for schedule (dynamic)
for i = 1 to n
for j = 1 to n
{c[i][j] = 0
 for k =1 to n
   {c[i][j] += (matrix_a[i][k] * matrix_b[i][j])
```

C) In third method, the transpose of Matrix B is calculated first. Then Multiplication of Matrix A*B is done using parallel execution.

Race Condition:- In race condition every thread tries to execute the shared variable at the same time. This result in inconsistency of the output
Cache miss:- During an execution of code, the CPU first looks for the data in cache memory. If the data is found that is known as a cache hit. Otherwise cache miss.
Memory bound:- Refers to a situation where the time required to complete a computational problem directly related to the amount of memory required to hold data.

```
#Pragma omp for schedule (dynamic)
for i = 1 to n
for j = 1 to n
    {c[i][j] = 0
for k = 1 to n
     {c[i][j] += matrix_a[i][j] * transpose_b[i][j]
```

D) This method implements both concept parallel program and blocks.

```
for jj = 1 to block_size
for kk =1 to block_size
for i = 1 to n
{for j = jj to (jj+block_size) || (j++)
   {num=0
      for k =kk to (kk+block_size) || (k++)
      {num += matrix_a[i][j] * matrix_b[k][j]
```

E) Some libraries are also imported to implement inbuilt functionalities. They are #include<ctime.h> to calculate the execution of each user-defined function. So, they can be compared. #include<omp.h> is used to implement OpenMP functionality i.e. parallel execution. #include<iostream> is used to read user input from standard input devices. #include<cstdlib> is used to generate random number. Matrix is initialized using the rand () function. The random number is generated in such a way that the range of numbers will be between 1 and 100. These numbers are generated and stored in the matrixes. We can also generate float numbers but for our program, we are using Integers. This have will not affect our final output. Using float numbers would have also provided similar results.

IV) Efficiency

The efficiency of the code has been improved by taking below factors into consideration.

I) Implementing parallel programming.

II) Increasing number of cache hits.

i). Implementing parallel programming.

OpenMP is used to implement parallel execution of a program. OpenMP uses Fork-Join architecture. In fork-join architecture, a single thread is split into several threads depending on the number of cores present. This is the default architecture. Users can also control the number of thread creation. #pragma omp parallel is the compiler directive. Anything which is written inside the parenthesis of the #pragma omp parallel is executed using parallel execution.

At the closing parenthesis of pragma, all the thread merged back to one and single thread begins execution of rest of the code. Using OpenMP, Matrix multiplication, which was executed by a single thread, is now executed by multiple threads. By default, an equal amount of workload is divided among threads by OpenMP, the user can also split the work among thread during thread scheduling, using this methodology, the work is divided among thread and the execution time decreases. OpenMP provides two methods for distribution of work among threads.

i. #pragma omp for schedule (static, chunk size)

ii. #pragma omp for schedule (dynamic, chunk size)

Race Condition:- In race condition every thread tries to execute the shared variable at the same time. This result in inconsistency of the output
Cache miss:- During an execution of code, the CPU first looks for the data in cache memory. If the data is found that is known as a cache hit. Otherwise cache miss.
Memory bound:- Refers to a situation where the time required to complete a computational problem directly related to the amount of memory required to hold data.

The first method will divide the work at compile time depending on the chunk size. In our code, we have used dynamic scheduling. This will allocate the work during runtime. The advantage of this methodology is that, if one thread gets free up early the CPU will schedule it instead of waiting for the other thread. The chunk size should not be too large nor too small.

While using parallel programming we need to consider the impact of race condition**. The race condition is avoided declaring the shared variable private. Once a thread is created, a corresponding stack memory is also created for that particular thread. A copy of variables, declared as private, stored in the stack of each thread. Now, these variables are independent of each other and act according to their corresponding thread. Thus, the change in one variable does not impact another shared variable.

ii). Reducing number of cache miss.

Cache is the fastest of all memory. If are able to store data in cache memory, then we can reduce the time taken by CPU to fetch data from the main memory. There are two important factors we need to consider before optimizing our code to make it cache compatible.

- Size of cache.
- Inline architecture of cache.

Size of cache.

The size of cache is very small as compared to the main memory. So, not all data can fit into the cache memory.

Inline architecture of cache.

The inline architecture of cache is built in such a way that it stores data as a single contiguous array of vectors (a row vector).

C/C++ are also designed in such a way that they store matrix as row of vectors in main memory. Since Matrix multiplication is a row-column major, Matrix transpose helps us to achieve cache inline architecture by using row-row major matrix multiplication. Now cache memory will pick one complete row store it in its memory. This optimization has been implemented in the function part- (C). Using this optimization, we have significantly reduced the number of cache miss**.

However, this is not the best approach. This optimization consists of two glitches. This approach is memory bound** and matrix transformation is itself a time-consuming process.

The other implementation, which overcomes this issue is the introduction of blocks. In blocks, the matrixes are divided into blocks of matrixes of b*b. where b is equivalent to the size of the cache. This approach will save the time required in matrix transpose and it is compute-bound instead of memory bound. In this approach i am able to increase the number of cache hits i.e. the time taken by CPU to fetch instruction and data is an expensive process. now the data fetching time is reduced because data is present in cache, therefore we have less cache miss, which result in a better output. Using this approach, the execution time of matrix multiplication reduced significantly and it also better than all other methods. Selecting b is itself a challenge. Here b is taken as 100. As at 100, we are getting the most optimized results.

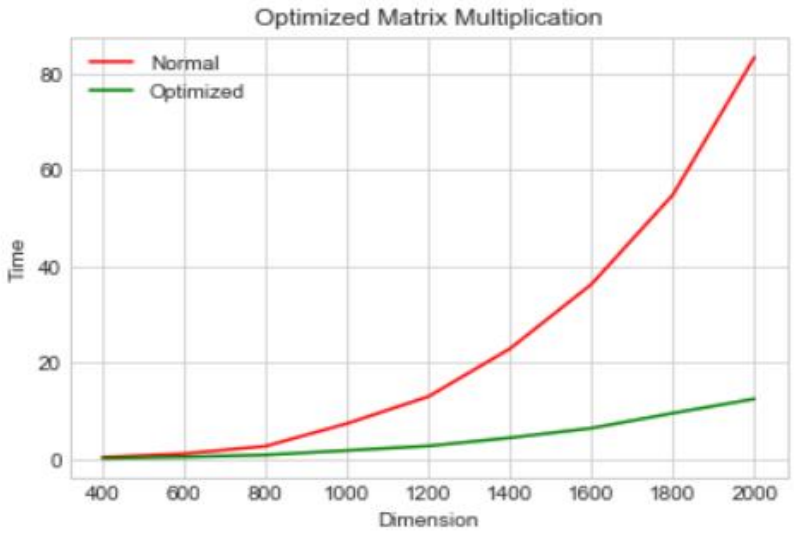
V) Results

The below table shows how the execution time is decreased as we keep on optimizing are algorithm. The best algorithm in our case is block with parallel programming. The time measured is in seconds.

Dimensions	Normal Matrix Multiplication	Matrix Multiplication using parallel programming	Matrix Multiplication using transpose and parallel programming	Matrix multiplication using parallel programming and blocks
400*400	0.312	0.14	0.116	0.094
600*600	1.062	0.64	0.412	0.385
800*800	2.615	1.502	0.906	0.797
1000*1000	7.33	3.706	1.756	1.724
1200*1200	12.95	6.214	3.05	2.66
1400*1400	22.903	11.208	4.821	4.321
1600*1600	36.306	16.456	7.193	6.302
1800*1800	54.93	24.93	10.265	9.426
2000*2000	83.53	34.045	14.356	12.456

Race Condition:- In race condition every thread tries to execute the shared variable at the same time. This result in inconsistency of the output
Cache miss:- During an execution of code, the CPU first looks for the data in cache memory. If the data is found that is known as a cache hit. Otherwise cache miss.
Memory bound:- Refers to a situation where the time required to complete a computational problem directly related to the amount of memory required to hold data.

VI) Graphical Representation of Results



VII) Conclusion

Using OpenMP and block multiplication we can reduce the execution time significantly. The number of threads used here are 4. If we increase the number of threads the efficiency will also increase. Better results are observed were CPU with more processor are used.

VIII) References

i). http://www.bowdoin.edu/~ltoma/teaching/cs3225-GIS/fall17/Lectures/openmp.html

ii). https://www.youtube.com/watch?v=a8R784VtXBg&list=PLJ5C_6qdAvBFMAko9JTyDJDIt1W48Sxmg

iii). https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

iv). https://www.geeksforgeeks.org/cache-memory-in-computer-organization/

v). https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf

vi). https://www.javatpoint.com/matrix-multiplication-in-cpp

Race Condition:- In race condition every thread tries to execute the shared variable at the same time. This result in inconsistency of the output
Cache miss:- During an execution of code, the CPU first looks for the data in cache memory. If the data is found that is known as a cache hit. Otherwise cache miss.
Memory bound:- Refers to a situation where the time required to complete a computational problem directly related to the amount of memory required to hold data.