CHOLESKY DECOMPOSITION OF A LARGE RANDOM MATRIX

The code was taken from:
https://www.geeksforgeeks.org/cholesky-decomposition-matrix-decomposition/

The Cholesky decomposition is the decomposition of a positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. The Cholesky decomposition of a Hermitian positive-definite matrix A is a decomposition of the form A = $[L][L]^T$, where L is a lower triangular matrix with diagonal entries, and L^T denotes the conjugate transpose of L.

<u>The code and its functionality:</u>

The code was modified to fit a matrix with random numbers.

- **Headers:** The code includes necessary C++ libraries for input/output, vector handling, random number generation, and timing measurements. We use vectors since this code deals with a large input size.
- **The Cholesky Decomposition Function:**
  - The choleskyDecomposition function takes three parameters, the original random matrix A, an empty matrix L that will store the lower triangular matrix, and the size N of the matrices.
  - It employs a nested loop structure to calculate the elements of the lower triangular matrix L.
  - The decomposition is performed iteratively for each row and column of the matrices, updating the elements of L.
    1. If j == i, it handles the diagonal elements of L differently by calculating the square root of the difference between the corresponding diagonal element of A and the sum of squares of previously calculated lower triangular elements.
    2. If j != i, it calculates the off-diagonal elements of L based on the Cholesky decomposition formula using the elements of both A and the already calculated lower triangular elements of L.
- **The main Function:** Defines the input matrix A (N x N) size. It initializes vector matrices A and L. Here, it generates random values for matrix A since the matrix consists of large input data. It also measures the execution time by recording the start and end times before and after the Cholesky Decomposition function. It also prints the time of execution.

In summary, the code generates a random matrix A, performs Cholesky decomposition, and measures the execution time.

The program's flow:

1. The program includes necessary libraries and declares the ‘choleskyDecomposition’ function.
2. The main function initializes necessary variables like the size of the matrix, generates a random matrix A (N x N), and initializes the L matrix as well. The current time is recorded as the start time and the choleskyDecomposition function is called.
3. The choleskyDecomposition function is called with matrices A and L as arguments, along with the size N. The Cholesky decomposition is performed and the result is stored in matrix L.
4. Then the main function measures the execution time and displays the time.

In summary, the code generates a random symmetric positive-definite matrix, performs Cholesky decomposition on that matrix, measures the time it takes to complete the decomposition, and outputs the running time to the console.

In the optimized code, OpenMP (Open Multi-Processing) is used to accelerate the Cholesky Decomposition. OpenMP is used for parallel processing of data, allowing the program to execute multiple threads concurrently, which can significantly improve the performance of the program.

OpenMP is used to accelerate Cholesky decomposition in the following ways:

1. **OpenMP Usage:** OpenMP processes multiple data elements in parallel. The code includes the ‘<omp.h>’ header, which provides access to OpenMP instructions.
2. **#pragma omp parallel for:** This statement is added before the outer loop in the choleskyDecomposition function. This pragma creates a parallel region, and the ‘for’ directive indicates that the following loop should be parallelized.
3. **omp_set_num_threads(14):** This is included in the main function to set the number of OpenMP threads to 14. This function call specifies the number of threads that OpenMP should use during parallel execution. To accelerate the program 14 threads are considered because threads determine the degree of parallelization.
4. **Parallelized Loop:** The outer loop of the choleskyDecomposition function is now parallelized. OpenMP automatically distributes the iterations of the loop across the specified number of threads.

Compilation steps and Flags:

I compiled the provided OpenMP-accelerated code for Cholesky decomposition, using a C++ compiler that supports OpenMP. Here are the compilation steps and flags needed for the code:
Compiler: I used a C++ compiler, such as GCC with VSCode.
Compile Command: Used the following compilation command:

g++ -fopenmp -o op optimized.cpp

Flags used:

- **'-fopenmp'**: This flag enables OpenMP support for parallelization. It is crucial for achieving the best performance.
- **'-o op'**: This flag specifies the output file name.
- **'optimized.cpp'**: Specifies the source code file to be compiled.

Run the compiled program: Did it using the following command,

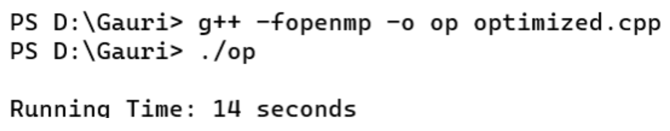
./op

This executes the Cholesky decomposition code using OpenMP.

Proof of achieved speedup:

The OpenMP-optimized code demonstrates a significant **average speedup of 4.76x** in Cholesky Decomposition compared to the non-optimized version. With OpenMP, the loops are parallelized and different threads can simultaneously compute separate elements of the elements of the output matrix. Thus execution time is reduced because the loops are parallelized and matrix calculations are executed with higher efficiency, which was way better than expected.

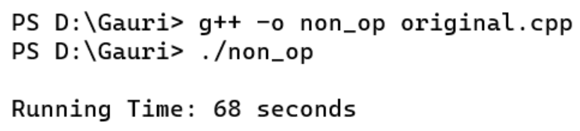
Execution time for the optimized Cholesky Decomposition code:



```
PS D:\Gauri> g++ -fopenmp -o op optimized.cpp
PS D:\Gauri> ./op

Running Time: 14 seconds
```

Execution time for the original Cholesky Decomposition code:



```
PS D:\Gauri> g++ -o non_op original.cpp
PS D:\Gauri> ./non_op

Running Time: 68 seconds
```