

Matrix Multiplication:

Native and Optimized Code using OpenMP:

The below code is an implementation of matrix multiplication using C language. The program first initializes a random number generator using the current time as the seed value. It then allocates memory for three matrices, A, B, and C, each of size $N \times N$, where N is set to 1300.

The program then initializes matrices A and B with random numbers using a for loop that iterates over all the elements of A and B matrices. The random numbers are generated using the `rand()` function and dividing it by `RAND_MAX` to generate values between 0 and 1.

Next, the program performs matrix multiplication of matrices A and B and stores the result in matrix C using three nested for-loops. The outer two loops iterate over the rows and columns of matrices A and B, respectively, while the inner loop calculates the dot product of corresponding row of matrix A and column of matrix B and accumulates the result in a variable named `sum`. Finally, the result is stored in the corresponding element of matrix C.

After performing the matrix multiplication, the program calculates the execution time of the non-optimized matrix multiplication code by measuring the difference between start and end times using the `clock()` function and converting it to seconds using

CLOCKS_PER_SEC constant. It then prints the execution time on the console.

Next, the program optimizes the matrix multiplication using OpenMP parallelization. The same matrix multiplication code is used, but the outermost loop is parallelized using the "#pragma omp parallel for" directive. This directive instructs the compiler to automatically parallelize the loop iterations among multiple threads. The program then measures the execution time of the optimized code and prints it on the console.

Finally, the program frees the allocated memory for matrices A, B, and C, and returns 0.

OpenMP.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1300

int main()
{
    // Initialize random number generator
    srand(time(NULL));

    // Allocate memory for matrices A, B, and C
    double *A = (double *)malloc(N * N *
sizeof(double));
```

```

    double *B = (double *)malloc(N * N *
sizeof(double));
    double *C = (double *)malloc(N * N *
sizeof(double));

    // Initialize matrices A and B with random numbers
    for (int i = 0; i < N * N; i++)
    {
        A[i] = (double)rand() / RAND_MAX;
        B[i] = (double)rand() / RAND_MAX;
    }

    // Multiply matrices A and B and store the result
in matrix C
    printf("Multiplying(Not-Optimized)...\n");
    clock_t start_time = clock();
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            double sum = 0;
            for (int k = 0; k < N; k++)
            {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
    clock_t end_time = clock();
    printf("Done\n");

    // Print execution time
    double execution_time = ((double)(end_time -
start_time)) / CLOCKS_PER_SEC;

```

```

    printf("Execution time(Not-Optimized): %f
seconds\n", execution_time);

    // Multiply matrices A and B and store the result
in matrix C
    printf("Multiplying using
OpenMP(Optimized)...\n");
    start_time = clock();
#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            double sum = 0;
            for (int k = 0; k < N; k++)
            {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
    end_time = clock();
    printf("Done\n");

    // Print execution time
    execution_time = ((double)(end_time - start_time))
/ CLOCKS_PER_SEC;
    printf("Execution time(Optimized with OpenMP): %f
seconds\n", execution_time);

    // Free memory
    free(A);
    free(B);
    free(C);

```

```
    return 0;
}
```

Output:

```
> g++ -fopenmp native.c
```

```
> a.exe
```

```
Multiplying(Not-Optimized)...
```

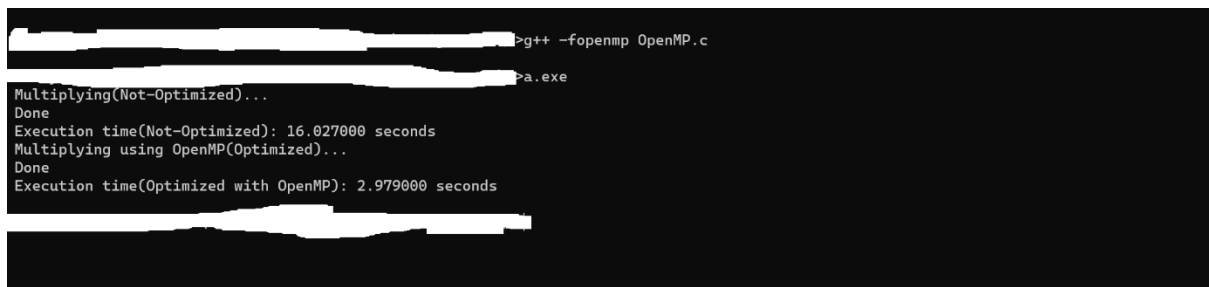
```
Done
```

```
Execution time(Not-Optimized): 16.027000 seconds
```

```
Multiplying using OpenMP(Optimized)...
```

```
Done
```

```
Execution time(Optimized with OpenMP): 2.979000 seconds
```



```
> g++ -fopenmp OpenMP.c
> a.exe
Multiplying(Not-Optimized)...
Done
Execution time(Not-Optimized): 16.027000 seconds
Multiplying using OpenMP(Optimized)...
Done
Execution time(Optimized with OpenMP): 2.979000 seconds
```

In summary, the below code implements matrix multiplication using C language and measures the execution time of both non-optimized and optimized (using OpenMP) codes. The OpenMP optimization parallelizes the outermost loop to utilize multiple threads and reduce the execution time. The code can be further optimized by using optimized matrix multiplication algorithms or using more advanced parallelization techniques like MPI or CUDA.

References:

- <https://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf>