Source Code

A serial C program to perform matrix multiplication on two 1024x1024matrices with data type double.

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

#define N 1000

double** allocateMatrix(int rows, int cols) {
double** mat = (double**)malloc(rows * sizeof(double*));
    for (int i = 0; i < rows; i++) {
        mat[i] = (double*)malloc(cols * sizeof(double));
    }
    return mat;
}

void freeMatrix(double** mat, int rows) {
for (int i = 0; i < rows; i++) {
        free(mat[i]);
    }
    free(mat);
}

void initializeMatrix(double** mat, int rows, int cols) {
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
            mat[i][j] = (double)rand() / RAND_MAX;
        }
    }
}

void multiplyMatrices(double** A, double** B, double** result) {
  for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++) {
   result[i][j] = 0;
   for (int k = 0; k < N; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main() {
    srand(time(NULL));

    double** A = allocateMatrix(N, N);
    double** B = allocateMatrix(N, N);
    double** result = allocateMatrix(N, N);

    initializeMatrix(A, N, N);
    initializeMatrix(B, N, N);

    clock_t start, end;
    start = clock();

    multiplyMatrices(A, B, result);

    end = clock();
    double time_spent = (double)(end - start) / CLOCKS_PER_SEC;

    printf("Time taken for matrix multiplication: %f seconds\n", time_spent);

    freeMatrix(A, N);
    freeMatrix(B, N);
    freeMatrix(result, N);
    return 0;
}
```

Using OpenMP ,Parallel C program to perform matrix multiplication on two 1024x1024matrices with data type double using two threads

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

#define N 1024

double** allocateMatrix(int rows, int cols) {
double** mat = (double**)malloc(rows * sizeof(double*));
for (int i = 0; i < rows; i++) {
        mat[i] = (double*)malloc(cols * sizeof(double));
    }
    return mat;
}

void freeMatrix(double** mat, int rows) {
for (int i = 0; i < rows; i++) {
        free(mat[i]);
    }
    free(mat);
}

void initializeMatrix(double** mat, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            mat[i][j] = (double)rand() / RAND_MAX;
        }
    }
}

void multiplyMatrices(double** A, double** B, double** result) {
    #pragma omp parallel for num_threads(2)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = 0;
            for (int k = 0; k < N; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

```
int main() {
    srand(time(NULL));
    double** A = allocateMatrix(N, N);
    double** B = allocateMatrix(N, N);
    double** result = allocateMatrix(N, N);

    initializeMatrix(A, N, N);
    initializeMatrix(B, N, N);

    double start, end;
    start = omp_get_wtime();

    multiplyMatrices(A, B, result);

    end = omp_get_wtime();
    double time_spent = end - start;
printf("Time taken for matrix multiplication with 2 threads: %f seconds\n", time_spent);

    freeMatrix(A, N);
    freeMatrix(B, N);
    freeMatrix(result, N);
 return 0;
}
```

Using OpenMP ,Parallel C program to perform matrix multiplication on two 1024x1024matrices with data type double using three threads .

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

#define N 1024

double** allocateMatrix(int rows, int cols) {
    double** mat = (double**)malloc(rows * sizeof(double*));
    for (int i = 0; i < rows; i++) {
        mat[i] = (double*)malloc(cols * sizeof(double));
    }
    return mat;
}

void freeMatrix(double** mat, int rows) {
    for (int i = 0; i < rows; i++) {
        free(mat[i]);
    }
    free(mat);
}

void initializeMatrix(double** mat, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            mat[i][j] = (double)rand() / RAND_MAX;
        }
    }
}

void multiplyMatrices(double** A, double** B, double** result) {
    #pragma omp parallel for num_threads(3)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double sum = 0.0;
            for (int k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            result[i][j] = sum;
        }
    }
}

int main() {
    srand(time(NULL));

    double** A = allocateMatrix(N, N);
    double** B = allocateMatrix(N, N);
    double** result = allocateMatrix(N, N);

    initializeMatrix(A, N, N);
    initializeMatrix(B, N, N);

    double start, end;
    start = omp_get_wtime();

    multiplyMatrices(A, B, result);

    end = omp_get_wtime();
    double time_spent = end - start;

    printf("Time taken for matrix multiplication with 3 threads: %f seconds\n", time_spent);

    freeMatrix(A, N);
    freeMatrix(B, N);
    freeMatrix(result, N);
return 0;
}
```

Using OpenMP ,Parallel C program to perform matrix multiplication on two 1024x1024matrices with data type double using four threads .

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

#define N 1024

double** allocateMatrix(int rows, int cols) {
double** mat = (double**)malloc(rows * sizeof(double*));
    for (int i = 0; i < rows; i++) {
        mat[i] = (double*)malloc(cols * sizeof(double));
    }
    return mat;
}

void freeMatrix(double** mat, int rows) {
for (int i = 0; i < rows; i++) {
        free(mat[i]);
    }
    free(mat);
}

void initializeMatrix(double** mat, int rows, int cols) {
for (int i = 0; i < rows; i++)
{
for (int j = 0; j < cols; j++)
{
mat[i][j] = (double)rand() / RAND_MAX;
        }
    }
}

void multiplyMatrices(double** A, double** B, double** result) {
 #pragma omp parallel for num_threads(4)
for (int i = 0; i < N; i++)
{
 for (int j = 0; j < N; j++) {
 double sum = 0.0;
for (int k = 0; k < N; k++) {
```

```
                sum += A[i][k] * B[k][j];
            }
            result[i][j] = sum;
        }
    }
}

int main() {
    srand(time(NULL));

    double** A = allocateMatrix(N, N);
    double** B = allocateMatrix(N, N);
    double** result = allocateMatrix(N, N);

    initializeMatrix(A, N, N);
    initializeMatrix(B, N, N);

    double start, end;
    start = omp_get_wtime();

    multiplyMatrices(A, B, result);

    end = omp_get_wtime();
    double time_spent = end - start;

    printf("Time taken for matrix multiplication with 4 threads: %f seconds\n", time_spent);

    freeMatrix(A, N);
    freeMatrix(B, N);
    freeMatrix(result, N);
return 0;
}
```

Report:

The performance evaluation report compares the execution times of a serial matrix multiplication program with the parallelize program using OpenMP. The matrix size used for the calculations is 1024 x 1024 . The goal for this report is to analyze the impact of parallelization on the execution time of the matrix multiplication task.

Test Environment:

Hardware:

Total RAM:7.6GB

Used RAM : 2.9GB

Free RAM:689MB

Shared RAM:672MB

Buffers/Cache:4.1GB

Available RAM:3.8GB

Compiler:

Compiler Name: GCC (GNU Compiler Collection)

Version: 11.4.0

Distribution:Ubuntu 11.4.0-1ubuntu1~22.04

Copyright: © 2021 Free Software Foundation, Inc.

License: This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Test Scenarios and Results:

Serial Program:

Execution Time: 23.22 seconds

The serial program executed the matrix multiplication task in 23.22 seconds.

Parallel Program with 2 Threads:

Execution Time: 14.345 seconds

The parallel program with 2 threads executed the matrix multiplication task in 14.345 seconds, which is significantly faster than the serial program.

Parallel Program with 3 Threads:

Execution Time: 10.23 seconds

The parallel program with 3 threads executed the matrix multiplication task in 10.23 seconds, showing further improvement in execution time compared to 2 threads.

Parallel Program with 4 Threads:

Execution Time: 8.875 seconds

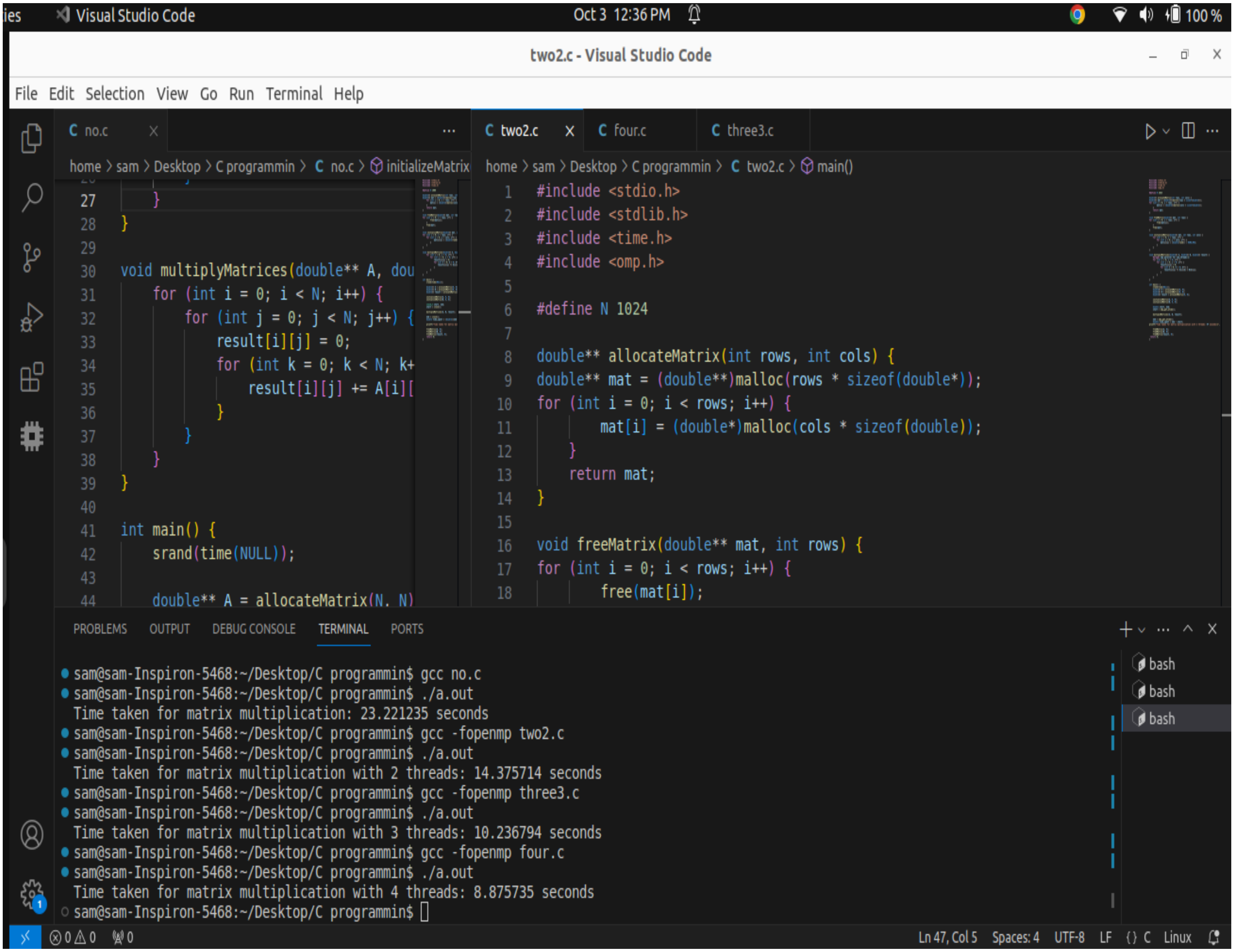
The parallel program with 4 threads executed the matrix multiplication task in 8.875 seconds, demonstrating the benefits of increased parallelism.

Conclusion

Here the conclusion is that the performance of the program is advanced using OpenMP.

Increase the number of threads reduced the computing time.

Screen-shot of Results:


ies
Visual Studio Code
Oct 3 12:36 PM
100 %
two2.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C no.c
C two2.c
C four.c
C three3.c
home › sam › Desktop › C programmin › C no.c › initializeMatrix
home › sam › Desktop › C programmin › C two2.c › main()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash
bash
bash
⊗ 0 ⚠ 0
0
Ln 47, Col 5
Spaces: 4
UTF-8
LF
{} C
Linux

```c
    }
}

void multiplyMatrices(double** A, dou
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = 0;
            for (int k = 0; k < N; k+
                result[i][j] += A[i][
            }
        }
    }
}

int main() {
    srand(time(NULL));

    double** A = allocateMatrix(N, N)
```

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

#define N 1024

double** allocateMatrix(int rows, int cols) {
double** mat = (double**)malloc(rows * sizeof(double*));
for (int i = 0; i < rows; i++) {
        mat[i] = (double*)malloc(cols * sizeof(double));
    }
    return mat;
}

void freeMatrix(double** mat, int rows) {
for (int i = 0; i < rows; i++) {
        free(mat[i]);
```

```
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ gcc no.c
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ ./a.out
  Time taken for matrix multiplication: 23.221235 seconds
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ gcc -fopenmp two2.c
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ ./a.out
  Time taken for matrix multiplication with 2 threads: 14.375714 seconds
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ gcc -fopenmp three3.c
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ ./a.out
  Time taken for matrix multiplication with 3 threads: 10.236794 seconds
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ gcc -fopenmp four.c
● sam@sam-Inspiron-5468:~/Desktop/C programmin$ ./a.out
  Time taken for matrix multiplication with 4 threads: 8.875735 seconds
○ sam@sam-Inspiron-5468:~/Desktop/C programmin$ 
```