HOMEWORK 2

Matrix Multiplication Using OpenMP

Camila Paladines

Parallel Programming

Professor: Roger Alfonso Gómez Nieto

March 22, 2021

1 How the algorithm works

The most general definition of the algorithm (not including ways to display or create data) can be represented as follows:

- **Input:** a matrix A of dimensions $r_1 \times c_1$ and a matrix B of dimensions $r_2 \times c_2$, where $c_1 = r_2$.
- **Output:** a matrix $C = AB$ of dimensions $r_1 \times c_2$.

The steps for this algorithm are:

1. Create a matrix C of dimensions $r_1 \times c_2$.
2. For each input C_{ij} $(0 \leq i < r_1,\ 0 \leq j < c_2)$
   (a) Initialize C_{ij} to 0.
   (b) For each k $(0 \leq k < c_1)$
      - Add $A_{ik} * B_{kj}$ to C_{ij}

Although the matrix multiplication algorithm is prepared for whatever the dimension of the resulting matrix (respecting the algorithm restrictions), it was assumed that for ease of data collection, matrices A and B were of size $n \times n$.

2 Code

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 500
#define M 10
#define lli long long int

void buildRandomMatrix(int matrix[N][N], lli rows, lli cols){
    for (lli i = 0; i < rows; i++){
        for (lli j = 0; j < cols; j++)
            matrix[i][j] = rand() % M;
   }
}

double multiplyMatrices(int A[N][N], int B[N][N], int C[N][N], lli r1,
     lli c1, lli r2, lli c2){
    double t_start, t_wall_clock;
    omp_set_nested(1);
    t_start = omp_get_wtime();
    #pragma omp parallel for
    for (lli i = 0; i < r1; i++){
        #pragma omp parallel for
        for (lli j = 0; j < c2; j++){
            C[i][j] = 0;
            for (lli k = 0; k < c1; k++)
                C[i][j] += A[i][k] * B[k][j];
        }
    }
    t_wall_clock = omp_get_wtime() - t_start;
    return t_wall_clock;
}

int main(int argc, char *argv[]){
    int A[N][N], B[N][N], C[N][N];
    lli max=500;
    buildRandomMatrix(A, max, max);
    buildRandomMatrix(B, max, max);
    double t_wall_clock = multiplyMatrices(A, B, C, max, max, max, max);
    printf("Time (N = %lld): %f\n", max, t_wall_clock);
    return 0;
}
```

3 Time measure

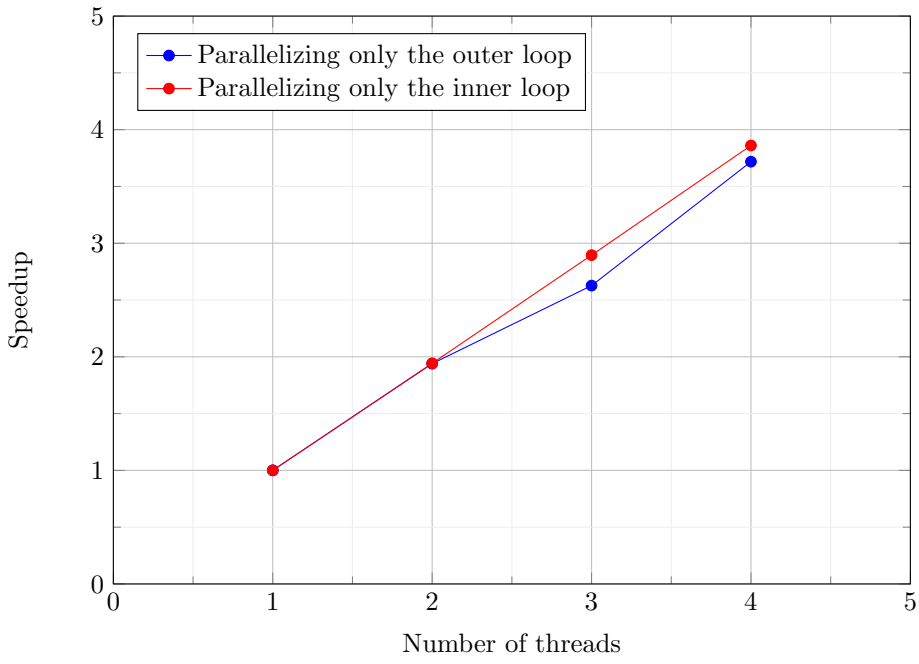
3.1 Experimental environment

The experimental environment where the tests were taken are as follows:

Configuration items	Item value
System version	Ubuntu 18.04.5
Compiler	GCC 7.5.0
Server model	Unknown
CPU	Intel Core i5-7200U 2.50GHz x4
GPU	Intel HD Graphics 620
Memory capacity	11,5 GiB
Memory version	Unknown
OpenMP version	OpenMP 4.5

3.2 Speedup by number of threads

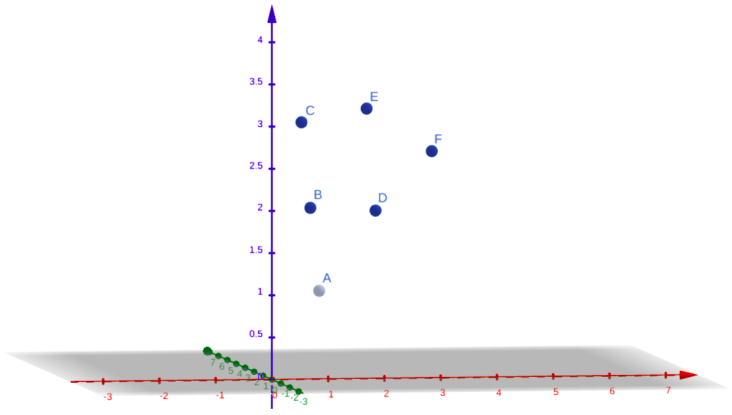
3.2.1 Assigning threads to only one loop



These results can also be seen in the following table:

Threads	Speedup outer loop	Speedup inner loop
1	1	1
2	1.940	1.943
3	2.627	2.895
4	3.718	3.860

3.2.2 With nested parallelism



These results can also be seen in the following table:

Point	Threads (outer loop)	Threads (inner loop)	Speedup
A	1	1	1
B	1	2	1.936
C	1	3	2.904
D	2	1	1.943
E	2	2	3.106
F	3	1	2.640

As can be seen in the table and the graphic, when the number of threads is balanced for both loops in 2 and 2, it obtains better speedup than in the other thread allocation options.