Nome: Vinicius Oliveira Piccazzio

RA: 10419471

Multiplicação de matrizes em C++

1-)

Ordem de linha

```
Matriz A:
3 0 2
1 9 2
2 3 5

Matriz B:
7 8 1
3 0 3
5 5 1

Matriz C (Resultado):
31 34 5
44 18 30
48 41 16
```

```
Matriz A:
2 4 2
2 9 5
8 6 1

Matriz B:
5 4 6
7 4 2
1 9 4

Matriz C (Resultado):
40 42 28
78 89 50
83 65 64
```

Ordem de coluna

```
Matriz A:
7       2       7
0       8       0
8       0       7

Matriz B:
3       8       2
5       9       9
8       0       6

Matriz C (A * B):
87      74      74
40      72      72
80      64      58
```

```
Matriz A:
3       4       8
3       2       4
8       0       1

Matriz B:
4       9       2
8       7       3
0       5       3

Matriz C (A * B):
44      95      42
28      61      24
32      77      19
```

2-)

```
Matriz A :
83 86 77 15 93 35 86 92 49 21
77 34 90 26 24 57 14 68 5 58
51 64 90 63 91 72 37 37 59 28
85 97 0 97 8 29 49 13 79 34
51 80 20 97 0 88 61 96 74 1
46 39 84 82 50 72 43 86 68 19
61 95 40 68 40 97 41 75 41 53
49 0 35 38 97 40 34 46 86 30
77 28 96 1 30 88 3 10 80 22
56 37 91 16 85 24 81 49 87 90
...
Matriz B :
39 57 84 88 18 70 40 74 47 89
70 14 30 13 1 76 0 14 76 15
49 53 11 34 37 69 20 99 38 99
56 28 87 51 70 22 22 55 81 55
34 26 71 86 47 84 41 3 42 53
39 94 43 72 57 66 29 27 80 98
49 80 47 59 61 55 0 57 45 96
67 68 79 23 85 31 87 34 41 89
74 47 10 0 58 26 32 2 54 2
88 27 39 20 32 32 38 76 11 46
...
```

```
Tempo de execução sem otimização: 1.85646 segundos.
Matriz C (resultado da multiplicação):
1271617 1248132 1227663 1196748 1220011 1246892 1170789 1229905 1295198 1334982
1279901 1306906 1327764 1256047 1289961 1269144 1246013 1308410 1326751 1380512
1261397 1277765 1293980 1260687 1307874 1218761 1213754 1263774 1316612 1358394
1266088 1291663 1237722 1210117 1240449 1240216 1203825 1224303 1305642 1334461
1301468 1248154 1237727 1172233 1240946 1201590 1171549 1210668 1281682 1313427
1266675 1269751 1253317 1217440 1256806 1214442 1198363 1209255 1294630 1285923
1272446 1277695 1295448 1199532 1302944 1250278 1216061 1215367 1339756 1361024
1285922 1282453 1267791 1234445 1291682 1220852 1206671 1223428 1294751 1327511
1264289 1274535 1259383 1192359 1292441 1222236 1180190 1213311 1292333 1313958
1293165 1310350 1273692 1221407 1261956 1206671 1208108 1256717 1317309 1322170
...
Tempo de execução com otimização máxima: 1.98341 segundos.
Matriz C (resultado da multiplicação com otimização):
2543234 2496264 2455326 2393496 2440022 2493784 2341578 2459810 2590396 2669964
2559802 2613812 2655528 2512094 2579922 2538288 2492026 2616820 2653502 2761024
2522794 2555530 2587960 2521374 2615748 2437522 2427508 2527548 2633224 2716788
2532176 2583326 2475444 2420234 2480898 2480432 2407650 2448606 2611284 2668922
2602936 2496308 2475454 2344466 2481892 2403180 2343098 2421336 2563364 2626854
2533350 2539502 2506634 2434880 2513612 2428884 2396726 2418510 2589260 2571846
2544892 2555390 2590896 2399064 2605888 2500556 2432122 2430734 2679512 2722048
2571844 2564906 2535582 2468890 2583364 2441704 2413342 2446856 2589502 2655022
2528578 2549070 2518766 2384718 2584882 2444472 2360380 2426622 2584666 2627916
2586330 2620700 2547384 2442814 2523912 2413342 2416216 2513434 2634618 2644340
...
```

3-) Tive muita dificuldade de executar o comando VALGRIND para realizar a análise do código.Pois tive problemas na instalação de outros arquivos tanto para a compilação do código, quanto para rodar uma máquina virtual Linux, dessa forma, colocarei o código a seguir:

```
#include <iostream>

#include <chrono>

#include <vector>

using namespace std;
```

```
void matrix_multiply_blocking(const vector<vector<int>>& A, const vector<vector<int>>& B,
vector<vector<int>>& C, int block_size) {
    int n = A.size();
    for (int i = 0; i < n; i += block_size) {
        for (int j = 0; j < n; j += block_size) {
            for (int k = 0; k < n; k += block_size) {
                for (int ii = i; ii < min(i + block_size, n); ++ii) {
                    for (int jj = j; jj < min(j + block_size, n); ++jj) {
                        for (int kk = k; kk < min(k + block_size, n); ++kk) {
                            C[ii][jj] += A[ii][kk] * B[kk][jj];
                        }
                    }
                }
            }
        }
    }
}
void generate_matrix(vector<vector<int>>& matrix, int n) {
    matrix.resize(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            matrix[i][j] = rand() % 100;
        }
    }
}
void print_matrix(const vector<vector<int>>& matrix, int n) {
    int limit = min(n, 10);
    for (int i = 0; i < limit; ++i) {
        for (int j = 0; j < limit; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
```

```
    if (n > 10) {
        cout << "..." << endl;
    }
}
int main() {
    int n = 512;
    int block_size = 32;
    vector<vector<int>> A, B, C;
    generate_matrix(A, n);
    generate_matrix(B, n);
    C.resize(n, vector<int>(n, 0));
    cout << "Matriz A :" << endl;
    print_matrix(A, n);
    cout << "Matriz B :" << endl;
    print_matrix(B, n);
    auto start = chrono::high_resolution_clock::now();
    matrix_multiply_blocking(A, B, C, block_size);
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> duration = end - start;
    cout << "Tempo de execução sem otimização: " << duration.count() << " segundos." << endl;
    cout << "Matriz C (resultado da multiplicação):" << endl;
    print_matrix(C, n);
    start = chrono::high_resolution_clock::now();
    matrix_multiply_blocking(A, B, C, block_size);
    end = chrono::high_resolution_clock::now();
    duration = end - start;
    cout << "Tempo de execução com otimização máxima: " << duration.count() << " segundos." << endl;
    cout << "Matriz C (resultado da multiplicação com otimização):" << endl;
    print_matrix(C, n);
    return 0;
}
```