

## mxv\_teste\_grafico\_v1.c

```
1  #include <omp.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  // Função de comparação para usar com qsort em um array de doubles.
7  int compare_doubles(const void *a, const void *b) {
8      double da = *(const double *)a;
9      double db = *(const double *)b;
10     if (da > db) return 1;
11     if (da < db) return -1;
12     return 0;
13 }
14
15 // Aloca dinamicamente uma matriz 2D.
16
17 double **create_matrix(int rows, int cols) {
18     // Aloca um array de ponteiros (um para cada linha)
19     double **matrix = (double **)malloc(rows * sizeof(double *));
20     if (matrix == NULL) return NULL;
21
22     // Para cada ponteiro de linha, aloca a memória para as colunas
23     for (int i = 0; i < rows; i++) {
24         matrix[i] = (double *)malloc(cols * sizeof(double));
25         if (matrix[i] == NULL) {
26             // Se falhar no meio, libera o que já foi alocado
27             for(int k = 0; k < i; k++) free(matrix[k]);
28             free(matrix);
29             return NULL;
30         }
31     }
32     return matrix;
33 }
34
35 // Libera a memória de uma matriz alocada dinamicamente.
36
37 void free_matrix(int rows, double **matrix) {
38     if (matrix == NULL) return;
39     // Primeiro, libera a memória de cada linha
40     for (int i = 0; i < rows; i++) {
41         free(matrix[i]);
42     }
43     // Finalmente, libera o array de ponteiros
44     free(matrix);
45 }
46
47 // Preenche uma matriz e um vetor com valores aleatórios.
48
49 void fill_random_data(int rows, int cols, double **matrix, double *vector) {
50     // Usa o tempo atual como semente para o gerador de números aleatórios
51     srand(time(NULL));
```

```
52     for (int i = 0; i < rows; i++) {
53         for (int j = 0; j < cols; j++) {
54             matrix[i][j] = (double)(rand() % 10); // Números aleatórios de 0 a 9
55         }
56     }
57     for (int i = 0; i < cols; i++) {
58         vector[i] = (double)(rand() % 10);
59     }
60 }
61
62 // Executa a multiplicação de matriz por vetor (y = A * x).
63
64 void multiply_matrix_vector(int rows, int cols, double **A, double *x, double *y) {
65     for (int i = 0; i < rows; i++) {
66         y[i] = 0.0; // Garante que o valor inicial seja zero
67         for (int j = 0; j < cols; j++) {
68             y[i] += A[i][j] * x[j];
69         }
70     }
71 }
72
73 void multiply_matrix_vector_cols_outer(int rows, int cols, double **A, double *x, double
*y) {
74     for (int i = 0; i < rows; i++) {
75         y[i] = 0.0;
76     }
77
78     // 2. O laço externo agora itera sobre as COLUNAS (j) da matriz A.
79     for (int j = 0; j < cols; j++) {
80         // O laço interno itera sobre as LINHAS (i).
81         for (int i = 0; i < rows; i++) {
82             y[i] += A[i][j] * x[j];
83         }
84     }
85 }
86
87 int main() {
88     int n_inicial = 32;
89     int n_final = 16384;
90     int n_passo = 2;
91
92     double tempo_mediana_linhas, tempo_mediana_colunas;
93     double fator_lentidao;
94
95     // Cabeçalho do CSV
96     printf("Tamanho_N,Tempo_Mediana_Linhas_s,Tempo_Mediana_Colunas_s,Fator_Lentidao\n");
97
98     for (int N = n_inicial; N <= n_final; N *= n_passo) {
99
100         int M = N;
101
102         double **A = create_matrix(M, N);
103         double *x = (double *)malloc(N * sizeof(double));
104         double *y = (double *)malloc(M * sizeof(double));
```

```
105
106     if (A == NULL || x == NULL || y == NULL) {
107         fprintf(stderr, "Falha ao alocar para N=%d\n", N);
108         continue;
109     }
110
111     fill_random_data(M, N, A, x);
112
113     int repeticoes = 1001;
114     if (N >= 512) repeticoes = 51;
115     if (N >= 1024) repeticoes = 11;
116     if (N >= 2048) repeticoes = 5;
117
118     //Alocar arrays para armazenar os tempos de cada repetição ---
119     double *tempos_linhas = (double *)malloc(repeticoes * sizeof(double));
120     double *tempos_colunas = (double *)malloc(repeticoes * sizeof(double));
121     if (tempos_linhas == NULL || tempos_colunas == NULL) {
122         fprintf(stderr, "Falha ao alocar arrays de tempo para N=%d\n", N);
123         free_matrix(M, A); free(x); free(y);
124         continue;
125     }
126
127     // Teste 1: Acesso por Linhas (Coletando tempos individuais)
128     for(int r = 0; r < repeticoes; r++) {
129         double start_time = omp_get_wtime();
130         multiply_matrix_vector(M, N, A, x, y);
131         double end_time = omp_get_wtime();
132         tempos_linhas[r] = end_time - start_time;
133     }
134
135     // Teste 2: Acesso por Colunas (Coletando tempos individuais)
136     for(int r = 0; r < repeticoes; r++) {
137         double start_time = omp_get_wtime();
138         multiply_matrix_vector_cols_outer(M, N, A, x, y);
139         double end_time = omp_get_wtime();
140         tempos_colunas[r] = end_time - start_time;
141     }
142
143     // Ordena os tempos do acesso por linhas
144     qsort(tempos_linhas, repeticoes, sizeof(double), compare_doubles);
145
146     // Ordena os tempos do acesso por colunas
147     qsort(tempos_colunas, repeticoes, sizeof(double), compare_doubles);
148
149     // Calcula a mediana (pegando o elemento do meio do array ordenado)
150     tempo_mediana_linhas = tempos_linhas[repeticoes / 2];
151     tempo_mediana_colunas = tempos_colunas[repeticoes / 2];
152
153     // Calcula o fator de lentidão
154     if (tempo_mediana_linhas > 0) {
155         fator_lentidao = tempo_mediana_colunas / tempo_mediana_linhas;
156     } else {
157         fator_lentidao = 1.0;
158     }
```

```
159
160     // Imprime a linha de dados CSV
161     printf("%d,%.12f,%.12f,%.3f\n", N, tempo_mediana_linhas, tempo_mediana_colunas,
fator_lentidao);
162
163     // --- MODIFICAÇÃO 3: Liberar a memória dos arrays de tempo ---
164     free_matrix(M, A);
165     free(x);
166     free(y);
167     free(tempo_linhas);
168     free(tempo_colunas);
169 }
170
171 return 0;
172 }
```