memory_bound_v2.c

```c

////////////////////////////////////////////////////////////////////
// memory_bound_V2.c
//Programa Memory-Bound: Matriz Column-Major com Colunas Aleatórias
////////////////////////////////////////////////////////////////////

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

// Para a inicializaçãoda matriz
#define DIM 10000

#define IDX(i, j) ((long)(i) * DIM + (j))

// Função para embaralhar um vetor de índices (Fisher-Yates shuffle)
void shuffle(int *array, size_t n) {
    if (n > 1) {
        for (size_t i = 0; i < n - 1; i++) {
            size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
            int t = array[j];
            array[j] = array[i];
            array[i] = t;
        }
    }
}

int main() {
    // Alocação de memória para as matrizes
    double *a = (double*)malloc((long)DIM * DIM * sizeof(double));
    double *b = (double*)malloc((long)DIM * DIM * sizeof(double));
    double *c = (double*)malloc((long)DIM * DIM * sizeof(double));

    // Alocação do vetor para os índices das colunas
    int *col_indices = (int*)malloc(DIM * sizeof(int));

    if (a == NULL || b == NULL || c == NULL || col_indices == NULL) {
        fprintf(stderr, "Erro ao alocar memória.\n");
        return 1;
    }

    printf("Preparando os dados e embaralhando os índices das colunas...\n");

    // Inicializa as matrizes e o vetor de índices de colunas (0, 1, 2, ...)
    #pragma omp parallel for
    for (int i = 0; i < DIM; i++) {
        col_indices[i] = i;
        for (int j = 0; j < DIM; j++) {
            a[IDX(i, j)] = 1.0;
            b[IDX(i, j)] = 2.0;
```

```
        }
    }

    // Embaralha o vetor de índices de colunas
    srand(time(NULL));
    shuffle(col_indices, DIM);

    printf("Programa Memory-Bound: Matriz Column-Major com Colunas Aleatórias\n");
    printf("Dimensão da Matriz: %d x %d\n", DIM, DIM);
    printf("--------------------------------------------------------------------------------\n");

    for (int num_threads = 1; num_threads <= 16; num_threads *= 2) {
        omp_set_num_threads(num_threads);

        double start_time = omp_get_wtime();

        // O loop externo percorre o VETOR DE ÍNDICES DE COLUNAS embaralhado.
        #pragma omp parallel for
        for (int k = 0; k < DIM; k++) {
            int j = col_indices[k]; // Pega uma coluna aleatória

            // O loop interno ainda percorre as linhas, causando o acesso column-major.
            for (int i = 0; i < DIM; i++) {
                c[IDX(i, j)] = a[IDX(i, j)] + b[IDX(i, j)];
            }
        }

        double end_time = omp_get_wtime();
        printf("Threads: %2d | Tempo de Execução: %f segundos\n", num_threads, end_time -
start_time);
    }

    free(a);
    free(b);
    free(c);
    free(col_indices);

    return 0;
}
```