difusao_bloqueante.c

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Parâmetros da Simulação
#define GLOBAL_N 100000 // Tamanho total da barra
#define STEPS 5000       // Número de passos de tempo
#define ALPHA 0.1       // Coeficiente de difusão (precisa ser < 0.5 para estabilidade)


#define TAG_LEFT_TO_RIGHT 0

#define TAG_RIGHT_TO_LEFT 1

void compute_inner(double* u_new, double* u, int size) {

    for (int i = 1; i < size - 1; i++) {

        u_new[i] = u[i] + ALPHA * (u[i-1] - 2.0 * u[i] + u[i+1]);
    }
}

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size < 2) {
        fprintf(stderr, "Este programa requer pelo menos 2 processos.\n");
        MPI_Finalize();
        return 1;
    }


    int local_data_size = GLOBAL_N / size;

    int local_size = local_data_size + 2;

    // Alocação dos arrays: u (atual) e u_new (próxima iteração)
    double* u = (double*)calloc(local_size, sizeof(double));
    double* u_new = (double*)calloc(local_size, sizeof(double));


    int left = (rank > 0) ? rank - 1 : MPI_PROC_NULL;
    int right = (rank < size - 1) ? rank + 1 : MPI_PROC_NULL;

```

```c

    if (rank == 0) {
        // Inicializa uma seção com um valor alto para simular calor
        for(int i = 1; i < local_data_size/2; i++) {
             u[i] = 10.0;
        }
    }

    // --- Loop Principal ---
    double start_time = MPI_Wtime();

    for (int t = 0; t < STEPS; t++) {

        if (right != MPI_PROC_NULL) {
            MPI_Send(&u[local_size - 2], 1, MPI_DOUBLE, right, TAG_RIGHT_TO_LEFT, MPI_COMM_WORLD);
        }


        if (right != MPI_PROC_NULL) {
            MPI_Recv(&u[local_size - 1], 1, MPI_DOUBLE, right, TAG_LEFT_TO_RIGHT, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }


        if (left != MPI_PROC_NULL) {
            MPI_Send(&u[1], 1, MPI_DOUBLE, left, TAG_LEFT_TO_RIGHT, MPI_COMM_WORLD);
        }


        if (left != MPI_PROC_NULL) {
            MPI_Recv(&u[0], 1, MPI_DOUBLE, left, TAG_RIGHT_TO_LEFT, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }

        // --- FIM DA COMUNICAÇÃO BLOQUEANTE ---


        compute_inner(u_new, u, local_size);

        // 3. Trocar Ponteiros para o próximo passo de tempo
        double *temp = u;
        u = u_new;
        u_new = temp;
    }

    double total_time = MPI_Wtime() - start_time;

    if (rank == 0) {
        printf("Versao 1 (Bloqueante - Send/Recv) | N=%d, STEPS=%d: %.6f s\n", GLOBAL_N, STEPS, total_time);
    }
```

```
 98
 99      // Limpeza
100      free(u);
101      free(u_new);
102      MPI_Finalize();
103      return 0;
104  }
105
```