difusão_Nao_bloqueante_test.c

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

// Parâmetros da Simulação (Os mesmos para todas as versões)
#define GLOBAL_N 1000000
#define STEPS 10000
#define ALPHA 0.1

#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) {
        if (rank == 0) 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;

    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;


    MPI_Request requests[4];
    MPI_Status status;


    int inner_overlap_start = 2;
    int inner_overlap_end = local_size - 3;


    if (rank == 0) {
        for(int i = 1; i < local_data_size/2; i++) {
             u[i] = 10.0;
        }
    }

    MPI_Barrier(MPI_COMM_WORLD);
    double start_time = MPI_Wtime();

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



        // Envio/Recebimento na Direita
        MPI_Isend(&u[local_size - 2], 1, MPI_DOUBLE, right, TAG_RIGHT_TO_LEFT, MPI_COMM_WORLD, &requests[0]);
        MPI_Irecv(&u[local_size - 1], 1, MPI_DOUBLE, right, TAG_LEFT_TO_RIGHT, MPI_COMM_WORLD, &requests[1]);

        // Envio/Recebimento na Esquerda
        MPI_Isend(&u[1], 1, MPI_DOUBLE, left, TAG_LEFT_TO_RIGHT, MPI_COMM_WORLD, &requests[2]);
        MPI_Irecv(&u[0], 1, MPI_DOUBLE, left, TAG_RIGHT_TO_LEFT, MPI_COMM_WORLD, &requests[3]);

        // 2. Computação da Zona Interna (SOBREPOSIÇÃO)
        // O processador agora calcula os pontos internos que não dependem da comunicação.
        for (int i = inner_overlap_start; i <= inner_overlap_end; i++) {
            u_new[i] = u[i] + ALPHA * (u[i-1] - 2.0 * u[i] + u[i+1]);
        }

        int flag = 0;

        while (!flag) {

            int flag_recv_right = 0;
            int flag_recv_left = 0;

            MPI_Test(&requests[1], &flag_recv_right, &status);
            MPI_Test(&requests[3], &flag_recv_left, &status);

            flag = flag_recv_right && flag_recv_left;

        }

        u_new[1] = u[1] + ALPHA * (u[0] - 2.0 * u[1] + u[2]);
        u_new[local_size - 2] = u[local_size - 2] + ALPHA * (u[local_size - 3] - 2.0 * u[local_size - 2] + u[local_size - 1]);

        // Certifica-se que os envios também terminaram antes do próximo passo (Importante para o buffer)
        MPI_Wait(&requests[0], &status);
        MPI_Wait(&requests[2], &status);

        // 5. Trocar Ponteiros
        double *temp = u;
        u = u_new;
        u_new = temp;
    }

    double total_time = MPI_Wtime() - start_time;
    MPI_Barrier(MPI_COMM_WORLD);

    if (rank == 0) {
        printf("Versao 3 (Sobreposicao - Test): %.6f s\n", total_time);
    }

    free(u);
    free(u_new);
    MPI_Finalize();
    return 0;
}

```