serial/navier_stokes_simul_serial_tempo.c

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

#define NX 512
#define NY 512
#define NT 2000
#define DT 0.001
#define NU 0.01

int main() {
    // Alocar arrays 2D
    double **u = (double**)malloc(NX * sizeof(double*));
    double **v = (double**)malloc(NX * sizeof(double*));
    double **u_new = (double**)malloc(NX * sizeof(double*));
    double **v_new = (double**)malloc(NX * sizeof(double*));

    for (int i = 0; i < NX; i++) {
        u[i] = (double*)malloc(NY * sizeof(double));
        v[i] = (double*)malloc(NY * sizeof(double));
        u_new[i] = (double*)malloc(NY * sizeof(double));
        v_new[i] = (double*)malloc(NY * sizeof(double));
    }

    // Inicialização
    for (int i = 0; i < NX; i++) {
        for (int j = 0; j < NY; j++) {
            u[i][j] = 1.0;
            v[i][j] = 0.0;
            double dx = i - NX/2, dy = j - NY/2;
            double dist = sqrt(dx*dx + dy*dy);
            if (dist < 20.0) {
                u[i][j] += 2.0 * exp(-dist*dist/100.0);
                v[i][j] += 1.5 * exp(-dist*dist/100.0);
            }
        }
    }

    double start_time = omp_get_wtime();

    // Simulação principal
    for (int step = 0; step < NT; step++) {
        for (int i = 1; i < NX-1; i++) {
            for (int j = 1; j < NY-1; j++) {
                double d2u_dx2 = (u[i+1][j] - 2.0*u[i][j] + u[i-1][j]);
                double d2u_dy2 = (u[i][j+1] - 2.0*u[i][j] + u[i][j-1]);
                double d2v_dx2 = (v[i+1][j] - 2.0*v[i][j] + v[i-1][j]);
                double d2v_dy2 = (v[i][j+1] - 2.0*v[i][j] + v[i][j-1]);

                u_new[i][j] = u[i][j] + DT * NU * (d2u_dx2 + d2u_dy2);
                v_new[i][j] = v[i][j] + DT * NU * (d2v_dx2 + d2v_dy2);
            }
        }

        // Condições de contorno periódicas
        for (int i = 0; i < NX; i++) {
            u_new[i][0] = u_new[i][NY-2];
            u_new[i][NY-1] = u_new[i][1];
            v_new[i][0] = v_new[i][NY-2];
            v_new[i][NY-1] = v_new[i][1];
        }
        for (int j = 0; j < NY; j++) {
            u_new[0][j] = u_new[NX-2][j];
            u_new[NX-1][j] = u_new[1][j];
            v_new[0][j] = v_new[NX-2][j];
            v_new[NX-1][j] = v_new[1][j];
        }

        // Trocar arrays
        double **temp_u = u;
        double **temp_v = v;
        u = u_new;
        v = v_new;
        u_new = temp_u;
        v_new = temp_v;
    }

    double end_time = omp_get_wtime();

    // Saída apenas do tempo
    printf("%.6f\n", end_time - start_time);

    // Liberar memória
    for (int i = 0; i < NX; i++) {
        free(u[i]);
        free(v[i]);
        free(u_new[i]);
        free(v_new[i]);
    }
    free(u);
    free(v);
    free(u_new);
    free(v_new);

    return 0;
}
```