paralelo/navier_stokes_simul_paralela.c

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

#define NX 512
#define NY 512
#define NT 10000
#define DT 0.001
#define NU 0.01

int main() {
    // Alocar memória
    double **u = malloc(NX * sizeof(double*));
    double **v = malloc(NX * sizeof(double*));
    double **un = malloc(NX * sizeof(double*));
    double **vn = malloc(NX * sizeof(double*));

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

    #pragma omp parallel for
    for (int i = 0; i < NX; i++) {
        for (int j = 0; j < NY; j++) {
            double dx = i - NX/2, dy = j - NY/2;
            double dist_sq = dx*dx + dy*dy;

            u[i][j] = 1.0;
            v[i][j] = 0.0;

            if (dist_sq < 400) {
                double perturbation = exp(-dist_sq/100.0);
                u[i][j] += 2.0 * perturbation;
                v[i][j] += 1.5 * perturbation;
            }
        }
    }

    double start = omp_get_wtime();

    // Loop de tempo PRINCIPAL
    for (int t = 0; t < NT; t++) {
        // 1. Atualização dos valores (um laço paralelo)
        #pragma omp parallel for
        for (int i = 1; i < NX-1; i++) {
            for (int j = 1; j < NY-1; j++) {
                un[i][j] = u[i][j] + DT*NU*(u[i+1][j] + u[i-1][j] + u[i][j+1] + u[i][j-1] - 4*u[i][j]);
                vn[i][j] = v[i][j] + DT*NU*(v[i+1][j] + v[i-1][j] + v[i][j+1] + v[i][j-1] - 4*v[i][j]);
            }
        }
        // 2. Aplicar condições de contorno (dois laços paralelos)
        #pragma omp parallel for
        for (int i = 0; i < NX; i++) {
            un[i][0] = un[i][NY-2];
            un[i][NY-1] = un[i][1];
            vn[i][0] = vn[i][NY-2];
            vn[i][NY-1] = vn[i][1];
        }

        #pragma omp parallel for
        for (int j = 0; j < NY; j++) {
            un[0][j] = un[NX-2][j];
            un[NX-1][j] = un[1][j];
            vn[0][j] = vn[NX-2][j];
            vn[NX-1][j] = vn[1][j];
        }

        // Swap pointers (feito pelo thread mestre, serialmente)
        double **ut = u, **vt = v;
        u = un; v = vn;
        un = ut; vn = vt;
    }

    double end = omp_get_wtime();
    printf("%.6f\n", end - start);

    // Cleanup
    for (int i = 0; i < NX; i++) {
        free(u[i]); free(v[i]); free(un[i]); free(vn[i]);
    }
    free(u); free(v); free(un); free(vn);

    return 0;
}
```