~/github/projeto_PP_12/navier_stokes_paralelo_ingenua.c

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

#define NT 2000

// As funções auxiliares de alocação/liberação são as mesmas
double** allocate_grid(int nx, int ny) {
    double *data = (double*)malloc(nx * ny * sizeof(double));
    double **array = (double**)malloc(nx * sizeof(double*));
    for (int i = 0; i < nx; i++) {
        array[i] = &(data[i * ny]);
    }
    return array;
}

void free_grid(double** array) {
    free(array[0]);
    free(array);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Uso: %s <TAMANHO_DA_GRADE>\n", argv[0]);
        fprintf(stderr, "Exemplo: %s 512\n", argv[0]);
        return 1;
    }
    int NX = atoi(argv[1]);
    int NY = NX;

    double **u = allocate_grid(NX, NY);
    double **v = allocate_grid(NX, NY);
    double **u_new = allocate_grid(NX, NY);
    double **v_new = allocate_grid(NX, NY);

    // Inicialização (pode ser paralela, não impacta muito a análise)
    #pragma omp parallel for
    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/2.0, dy = j - NY/2.0;
            double dist = sqrt(dx*dx + dy*dy);
            if (dist < (NX / 25.0)) {
                u[i][j] += 2.0 * exp(-dist*dist/(NX/5.0));
                v[i][j] += 1.5 * exp(-dist*dist/(NX/5.0));
            }
        }
    }

```

```c
    double start_time = omp_get_wtime();

    for (int step = 0; step < NT; step++) {

        // --- INÍCIO DA ABORDAGEM INGÊNUA ---

        // GARGALO 1: Cria e destrói um time de threads apenas para este laço.
        #pragma omp parallel for
        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] + (0.001 * 0.01) * (d2u_dx2 + d2u_dy2);
                v_new[i][j] = v[i][j] + (0.001 * 0.01) * (d2v_dx2 + d2v_dy2);
            }
        }

        // GARGALO 2: Cria e destrói OUTRO time de threads apenas para as seções.
        #pragma omp parallel sections
        {
            #pragma omp section
            { // Contorno Horizontal
                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];
                }
            }
            #pragma omp section
            { // Contorno Vertical
                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];
                }
            }
        }

        // A troca de ponteiros ocorre serialmente, pela thread mestre, após os joins.
        double **temp_u = u;
        double **temp_v = v;
        u = u_new;
        v = v_new;
        u_new = temp_u;
        v_new = temp_v;

```

```
102             // --- FIM DA ABORDAGEM INGÊNUA ---
103         }
104
105         double end_time = omp_get_wtime();
106         printf("%.6f\n", end_time - start_time);
107
108         free_grid(u); free_grid(v); free_grid(u_new); free_grid(v_new);
109
110         return 0;
111     }
112
```