

navier_stokes_paralelo_close.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <omp.h>
5
6  // NT agora é uma constante interna.
7  #define NT 2000
8
9  // Funções auxiliares agora recebem as dimensões como parâmetros
10 double** allocate_grid(int nx, int ny) {
11     double *data = (double*)malloc(nx * ny * sizeof(double));
12     double **array = (double**)malloc(nx * sizeof(double*));
13     for (int i = 0; i < nx; i++) {
14         array[i] = &(data[i * ny]);
15     }
16     return array;
17 }
18
19 void free_grid(double** array) {
20     free(array[0]);
21     free(array);
22 }
23
24 int main(int argc, char *argv[]) {
25     // Agora esperamos o tamanho da grade (NX) como argumento
26     if (argc != 2) {
27         fprintf(stderr, "Uso: %s <TAMANHO_DA_GRADE>\n", argv[0]);
28         fprintf(stderr, "Exemplo: %s 512\n", argv[0]);
29         return 1;
30     }
31     int NX = atoi(argv[1]);
32     int NY = NX; // NY será sempre igual a NX
33
34     // Alocação de memória usa as variáveis NX e NY
35     double **u = allocate_grid(NX, NY);
36     double **v = allocate_grid(NX, NY);
37     double **u_new = allocate_grid(NX, NY);
38     double **v_new = allocate_grid(NX, NY);
39
40     // Inicialização usa as variáveis NX e NY
41     #pragma omp parallel for
42     for (int i = 0; i < NX; i++) {
43         for (int j = 0; j < NY; j++) {
44             u[i][j] = 1.0; v[i][j] = 0.0;
45             double dx = i - NX/2.0, dy = j - NY/2.0;
46             double dist = sqrt(dx*dx + dy*dy);
47             if (dist < (NX / 25.0)) { // Condição inicial relativa ao tamanho
da grade
48                 u[i][j] += 2.0 * exp(-dist*dist/(NX/5.0));
49                 v[i][j] += 1.5 * exp(-dist*dist/(NX/5.0));
```

```
50     }
51   }
52 }
53
54 double start_time = omp_get_wtime();
55
56 #pragma omp parallel proc_bind(close)
57 {
58     for (int step = 0; step < NT; step++) {
59
60         #pragma omp for collapse(2) schedule(static)
61         for (int i = 1; i < NX-1; i++) {
62             for (int j = 1; j < NY-1; j++) {
63                 double d2u_dx2 = (u[i+1][j] - 2.0*u[i][j] + u[i-1][j]);
64                 double d2u_dy2 = (u[i][j+1] - 2.0*u[i][j] + u[i][j-1]);
65                 double d2v_dx2 = (v[i+1][j] - 2.0*v[i][j] + v[i-1][j]);
66                 double d2v_dy2 = (v[i][j+1] - 2.0*v[i][j] + v[i][j-1]);
67
68                 u_new[i][j] = u[i][j] + (0.001 * 0.01) * (d2u_dx2 +
d2u_dy2); // DT e NU podem ser fixados
69                 v_new[i][j] = v[i][j] + (0.001 * 0.01) * (d2v_dx2 +
d2v_dy2);
70             }
71         }
72
73         #pragma omp for
74         for (int i = 0; i < NX; i++) {
75             u_new[i][0] = u_new[i][NY-2];
76             u_new[i][NY-1] = u_new[i][1];
77             v_new[i][0] = v_new[i][NY-2];
78             v_new[i][NY-1] = v_new[i][1];
79         }
80
81         #pragma omp for
82         for (int j = 0; j < NY; j++) {
83             u_new[0][j] = u_new[NX-2][j];
84             u_new[NX-1][j] = u_new[1][j];
85             v_new[0][j] = v_new[NX-2][j];
86             v_new[NX-1][j] = v_new[1][j];
87         }
88
89         #pragma omp single
90         {
91             double **temp_u = u;
92             double **temp_v = v;
93             u = u_new;
94             v = v_new;
95             u_new = temp_u;
96             v_new = temp_v;
97         }
98     }
99 }
100
```

```
101     double end_time = omp_get_wtime();
102     printf("%.6f\n", end_time - start_time);
103
104     free_grid(u); free_grid(v); free_grid(u_new); free_grid(v_new);
105
106     return 0;
107 }
108
```