

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4 #include <time.h>
5 #include <math.h>
6
7 #include "includes/libpoisson.h"
8
9 // #define N 258
10
11 int main(int argc, char *argv[])
12 // int main(void)
13 {
14     int N;
15     if(argc != 2)
16     {
17         errorMsg("Erro na execucao!\nEx. ./[nome_executavel] [tamanho_da_malha+2]\n");
18         return (1);
19     }
20     else
21     {
22         N = atoi(argv[1]);
23     }
24
25     /*
26     * Fluxos atuais e antigos em cada um dos lados da célula espacial
27     * up (Upper), dn (Down), lf (Left), rh (Right)
28     */
29     nodeSides *q, *q_old;
30     // nodeSides **mq, **mq_old;
31
32     q = criaVetorNode(N, N);
33     // mq = criaMatrizNode(N, N, q);
34     q_old = criaVetorNode(N, N);
35     // mq_old = criaMatrizNode(N, N, q_old);
36
37     // mq = montaMatrizNode(N, N);
38     // mq_old = montaMatrizNode(N, N);
39
40     /*
41     * Multiplicadores de Lagrange em cada um dos lados da célula espacial
42     */
43     nodeSides *l, *l_old;
44     // nodeSides **ml, **ml_old;
45
46     l = criaVetorNode(N, N);
47     // ml = criaMatrizNode(N, N, l);
48     l_old = criaVetorNode(N, N);
49     // ml_old = criaMatrizNode(N, N, l_old);
50
51     // ml = montaMatrizNode(N, N);
52     // ml_old = montaMatrizNode(N, N);
53
54     /*
55     * Betas da condição de Robin em cada um dos lados da célula espacial
56     */
57
58     nodeSides *beta;
59     // nodeSides **mbeta;
60
61     beta = criaVetorNode(N, N);
62     // mbeta = criaMatrizNode(N, N, beta);
63
64     // mbeta = montaMatrizNode(N, N);
65
66     /*
67     * Pressões atuais e antigas em cada uma das células
68     */
69
70     double *p, *p_old;
```

```

71 //double **mp, **mp_old;
72
73 p = criaVetor(N, N);
74 //mp = criaMatriz(N, N, p);
75 p_old = criaVetor(N, N);
76 //mp_old = criaMatriz(N, N, p_old);
77
78 //mp = montaMatriz(N, N);
79 //mp_old = montaMatriz(N, N);
80
81 /*
82  * Parametros materiais da grade
83  */
84
85 nodeMaterial *pMat;
86 //nodeMaterial **mpMat;
87
88 pMat = criaVetorMaterial(N, N);
89 //mpMat = criaMatrizMaterial(N, N, pMat);
90
91 //mpMat = montaMatrizMaterial(N, N);
92
93 double lsize = 25600.00; /* dimensão da regioa */
94
95 /* Variáveis para auxiliar na contagem do tempo*/
96 double start, stop;
97 double startTime, elapsedTime;
98 double clockZero = 0.0;
99 clock_t ticks1, ticks2;
100 /* Variáveis auxiliares na implementação */
101 int i, j, k, n;
102 double h, aux,
103         c = 1., /* Valor para o calculo de beta */
104         Media, erro, /* Media das pressoes e erro na norma */
105         Keff; /* Valor medio da permeabilidade */
106 //double **mp_aux; /* Ponteiros para troca */
107 double *p_aux;
108 //nodeSides **mq_aux;
109 nodeSides *q_aux;
110
111 start = omp_get_wtime();
112 startTime = walltime( &clockZero );
113 ticks1 = clock();
114
115 /*
116  * Inicialização das variaveis
117  */
118
119
120
121 for(i=1; i < N-1; i++)
122     for(j=1; j < N-1; j++)
123     {
124         pMat[i*N+j].f = 0.0;
125         //mpMat[i][j].f = 0.0;
126         l[i*N+j].dn = l[i*N+j].lf = l[i*N+j].rh = l[i*N+j].up = 0.0;
127         //ml[i][j].dn = ml[i][j].lf = ml[i][j].rh = ml[i][j].up = 0.0;
128         q[i*N+j].dn = q[i*N+j].lf = q[i*N+j].rh = q[i*N+j].up = 0.0;
129         //mq[i][j].dn = mq[i][j].lf = mq[i][j].rh = mq[i][j].up = 0.0;
130         p[i*N+j]=0.0;
131         //mp[i][j]=0.0;
132     }
133
134 /*
135  * Cálculo de algumas variáveis auxiliares para o valor de permeabilidade
136  * en blocos verticais
137  */
138 n = N-2;
139 h = lsize/n;
140 k = n/2;

```

```

141
142     for(i=1; i<=k; i++)
143         for(j=1; j<N-1; j++)
144             {
145                 pMat[i*N+j].perm = 1.0e-10;
146                 //mpMat[i][j].perm = 1.0e-10;
147                 pMat[(i+k)*N+j].perm = 1.0e-11;
148                 //mpMat[i+k][j].perm = 1.0e-11;
149             }
150
151     /*
152     * Calcula os Beta da Condicao de Robin
153     */
154     k = N+1; //[1][1]
155     Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
156     //Keff = (2*mpMat[1][1].perm*mpMat[1][2].perm)/(mpMat[1][1].perm + mpMat[1][2].perm);
157     beta[k].up = c*h/Keff;
158     //mbeta[1][1].up = c*h/Keff;
159     Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
160     //Keff = (2*mpMat[1][1].perm*mpMat[2][1].perm)/(mpMat[1][1].perm + mpMat[2][1].perm);
161     beta[k].rh = c*h/Keff;
162     //mbeta[1][1].rh = c*h/Keff;
163
164     k = N+n; //[1][n]
165     Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
166     //Keff = (2*mpMat[1][n].perm*mpMat[1][n-1].perm)/(mpMat[1][n].perm + mpMat[1]
167     [n-1].perm);
168     beta[k].dn = c*h/Keff;
169     //mbeta[1][n].dn = c*h/Keff;
170     Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
171     //Keff = (2*mpMat[1][n].perm*mpMat[2][n].perm)/(mpMat[1][n].perm + mpMat[2][n].perm);
172     beta[k].rh = c*h/Keff;
173     //mbeta[1][n].rh = c*h/Keff;
174
175     k = n*N+1; //[n][1]
176     Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
177     //Keff = (2*mpMat[n][1].perm*mpMat[n][2].perm)/(mpMat[n][1].perm + mpMat[n][2].perm);
178     beta[k].up = c*h/Keff;
179     //mbeta[n][1].up = c*h/Keff;
180     Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
181     //Keff = (2*mpMat[n][1].perm*mpMat[n-1][1].perm)/(mpMat[n][1].perm + mpMat[n-1]
182     [1].perm);
183     beta[k].lf = c*h/Keff;
184     //mbeta[n][1].lf = c*h/Keff;
185
186     k = n*N+n; //[n][n]
187     Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
188     //Keff = (2*mpMat[n][n].perm*mpMat[n][n-1].perm)/(mpMat[n][n].perm + mpMat[n]
189     [n-1].perm);
190     beta[k].dn = c*h/Keff;
191     //mbeta[n][n].dn = c*h/Keff;
192     Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
193     //Keff = (2*mpMat[n][n].perm*mpMat[n-1][n].perm)/(mpMat[n][n].perm + mpMat[n-1]
194     [n].perm);
195     beta[k].lf = c*h/Keff;
196     //mbeta[n][n].lf = c*h/Keff;
197
198     for (i=2; i<n; i++)
199     {
200         k = i*N+1; //[i][1]
201         Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
202         //Keff = (2*mpMat[i][1].perm*mpMat[i][2].perm)/(mpMat[i][1].perm + mpMat[i]
203         [2].perm);
204         beta[k].up= c*h/Keff;
205         //mbeta[i][1].up= c*h/Keff;
206         Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
207         //Keff = (2*mpMat[i][1].perm*mpMat[i-1][1].perm)/(mpMat[i][1].perm + mpMat[i-1]
208         [1].perm);
209         beta[k].lf= c*h/Keff;
210         //mbeta[i][1].lf= c*h/Keff;

```

```

205     Keff = (2*pmat[k].perm*pmat[k+N].perm)/(pmat[k].perm + pmat[k+N].perm);
206     //Keff = (2*mpMat[i][1].perm*mpMat[i+1][1].perm)/(mpMat[i][1].perm + mpMat[i+1]
    [1].perm);
207     beta[k].rh= c*h/Keff;
208     //mbeta[i][1].rh= c*h/Keff;
209
210     k = i*N+n; //[i][n]
211     Keff = (2*pmat[k].perm*pmat[k-1].perm)/(pmat[k].perm + pmat[k-1].perm);
212     //Keff = (2*mpMat[i][n].perm*mpMat[i][n-1].perm)/(mpMat[i][n].perm + mpMat[i]
    [n-1].perm);
213     beta[k].dn= c*h/Keff;
214     //mbeta[i][n].dn= c*h/Keff;
215     Keff = (2*pmat[k].perm*pmat[k-N].perm)/(pmat[k].perm + pmat[k-N].perm);
216     //Keff = (2*mpMat[i][n].perm*mpMat[i-1][n].perm)/(mpMat[i][n].perm + mpMat[i-1]
    [n].perm);
217     beta[k].lf= c*h/Keff;
218     //mbeta[i][n].lf= c*h/Keff;
219     Keff = (2*pmat[k].perm*pmat[k+N].perm)/(pmat[k].perm + pmat[k+N].perm);
220     //Keff = (2*mpMat[i][n].perm*mpMat[i+1][n].perm)/(mpMat[i][n].perm + mpMat[i+1]
    [n].perm);
221     beta[k].rh= c*h/Keff;
222     //mbeta[i][n].rh= c*h/Keff;
223
224     k = N+i; //[1][i]
225     Keff = (2*pmat[k].perm*pmat[k+1].perm)/(pmat[k].perm + pmat[k+1].perm);
226     //Keff = (2*mpMat[1][i].perm*mpMat[1][i+1].perm)/(mpMat[1][i].perm + mpMat[1]
    [i+1].perm);
227     beta[k].up= c*h/Keff;
228     //mbeta[1][i].up= c*h/Keff;
229     Keff = (2*pmat[k].perm*pmat[k-1].perm)/(pmat[k].perm + pmat[k-1].perm);
230     //Keff = (2*mpMat[1][i].perm*mpMat[1][i-1].perm)/(mpMat[1][i].perm + mpMat[1]
    [i-1].perm);
231     beta[k].dn= c*h/Keff;
232     //mbeta[1][i].dn= c*h/Keff;
233     Keff = (2*pmat[k].perm*pmat[k+N].perm)/(pmat[k].perm + pmat[k+N].perm);
234     //Keff = (2*mpMat[1][i].perm*mpMat[2][i].perm)/(mpMat[1][i].perm + mpMat[2]
    [i].perm);
235     beta[k].rh= c*h/Keff;
236     //mbeta[1][i].rh= c*h/Keff;
237
238     k = n*N+i; //[n][i]
239     Keff = (2*pmat[k].perm*pmat[k+1].perm)/(pmat[k].perm + pmat[k+1].perm);
240     //Keff = (2*mpMat[n][i].perm*mpMat[n][i+1].perm)/(mpMat[n][i].perm + mpMat[n]
    [i+1].perm);
241     beta[k].up= c*h/Keff;
242     //mbeta[n][i].up= c*h/Keff;
243     Keff = (2*pmat[k].perm*pmat[k-1].perm)/(pmat[k].perm + pmat[k-1].perm);
244     //Keff = (2*mpMat[n][i].perm*mpMat[n][i-1].perm)/(mpMat[n][i].perm + mpMat[n]
    [i-1].perm);
245     beta[k].dn= c*h/Keff;
246     //mbeta[n][i].dn= c*h/Keff;
247     Keff = (2*pmat[k].perm*pmat[k-N].perm)/(pmat[k].perm + pmat[k-N].perm);
248     //Keff = (2*mpMat[n][i].perm*mpMat[n-1][i].perm)/(mpMat[n][i].perm + mpMat[n-1]
    [i].perm);
249     beta[k].lf= c*h/Keff;
250     //mbeta[n][i].lf= c*h/Keff;
251
252     for(j=2; j<n; j++)
253     {
254         k = i*N+j; //[i][j]
255         Keff = (2*pmat[k].perm*pmat[k+1].perm)/(pmat[k].perm + pmat[k+1].perm);
256         //Keff = (2*mpMat[i][j].perm*mpMat[i][j+1].perm)/(mpMat[i][j].perm + mpMat[i]
    [j+1].perm);
257         beta[k].up= c*h/Keff;
258         //mbeta[i][j].up= c*h/Keff;
259         Keff = (2*pmat[k].perm*pmat[k-1].perm)/(pmat[k].perm + pmat[k-1].perm);
260         //Keff = (2*mpMat[i][j].perm*mpMat[i][j-1].perm)/(mpMat[i][j].perm + mpMat[i]
    [j-1].perm);
261         beta[k].dn= c*h/Keff;
262         //mbeta[i][j].dn= c*h/Keff;

```

```

263         Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
264         //Keff = (2*mpMat[i][j].perm*mpMat[i+1][j].perm)/(mpMat[i][j].perm +
265         mpMat[i+1][j].perm);
266         beta[k].rh= c*h/Keff;
267         //mbeta[i][j].rh= c*h/Keff;
268         Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
269         //Keff = (2*mpMat[i][j].perm*mpMat[i-1][j].perm)/(mpMat[i][j].perm +
270         mpMat[i-1][j].perm);
271         beta[k].lf= c*h/Keff;
272         //mbeta[i][j].lf= c*h/Keff;
273     }
274 }
275
276 /*
277  * Inicializando valores da fonte
278  */
279 pMat[N+1].f=1.0e-7;
280 //mpMat[1][1].f=1.0e-7;
281 pMat[n*N+n].f=-1.0e-7;
282 //mpMat[n][n].f=-1.0e-7;
283
284 /*
285  * calculo de parâmetros que nao dependem das iterações
286  */
287 aux = 1/h;
288
289 for (i=1; i<=n; i++)
290     for (j=1; j<=n; j++)
291     {
292         pMat[i*N+j].shi=2*pMat[i*N+j].perm*aux;
293         //mpMat[i][j].shi=2*mpMat[i][j].perm*aux;
294         pMat[i*N+j].f*=h;
295         //mpMat[i][j].f*=h;
296     }
297
298 /*
299  * Ciclo até convergência do problema
300  */
301 k = 0; // quantidade de iterações
302
303 do
304 {
305     /*
306     mp_aux = mp;
307     mp = mp_old;
308     mp_old = mp_aux;
309
310     mq_aux = mq;
311     mq = mq_old;
312     mq_old = mq_aux;
313
314     ml_aux = ml;
315     ml = ml_old;
316     ml_old = ml_aux;
317     */
318
319     p_aux = p;
320     p = p_old;
321     p_old = p_aux;
322
323     q_aux = q;
324     q = q_old;
325     q_old = q_aux;
326
327     l_aux = l;
328     l = l_old;
329     l_old = l_aux;
330
331     k++;

```

```

331 //printf("Iteração %d \n",k);
332
333 /*Cálculo da pressão e dos fluxos em cada elemento */
334
335 /*Canto inferior esquerdo [1][1]*/
336 //canto_d_l(1, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
337 canto_d_lArray(1, 1, N, pMat, beta, q, q_old, l_old, p);
338
339 /*Canto superior esquerdo [1][N-2]*/
340 //canto_u_l(1, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
341 canto_u_lArray(1, n, N, pMat, beta, q, q_old, l_old, p);
342
343 /*Canto inferior direito [N-2][1]*/
344 //canto_d_r(n, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
345 canto_d_rArray(n, 1, N, pMat, beta, q, q_old, l_old, p);
346
347 /*Canto superior direito [N-2][N-2]*/
348 //canto_u_r(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
349 canto_u_rArray(n, n, N, pMat, beta, q, q_old, l_old, p);
350
351 /*Fronteira U [2...N-3][N-2]*/
352 //fronteira_u(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
353 fronteira_uArray(N, n, pMat, beta, q, q_old, l_old, p);
354
355 /*Fronteira D [2...N-3][1]*/
356 //fronteira_d(n, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
357 fronteira_dArray(N, 1, pMat, beta, q, q_old, l_old, p);
358
359 /*Fronteira R [N-2][2...N-3]*/
360 //fronteira_r(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
361 fronteira_rArray(n, N, pMat, beta, q, q_old, l_old, p);
362
363 /*Fronteira L [1][2...N-3]*/
364 //fronteira_l(1, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
365 fronteira_lArray(1, N, pMat, beta, q, q_old, l_old, p);
366
367 /*Elementos internos [2..N-3][2..N-3]*/
368 //internos(n, mpMat, mbeta, mq, mq_old, ml_old, mp);
369 internosArray(N, pMat, beta, q, q_old, l_old, p);
370
371 /* Atualização dos multiplicadores de lagrange e calculando a média da pressão*/
372 //Media = lagrangeUpdate(n, mbeta, mq, mq_old, ml, ml_old, mp);
373 Media = lagrangeUpdateArray(N, beta, q, q_old, l, l_old, p);
374
375 /* Impondo a média zero na distribuição de pressões
376  * e cálculo de verificação de convergência
377  */
378 //erro = mediaZero(n, Media, ml, mp, mp_old);
379 erro = mediaZeroArray(N, Media, l, p, p_old);
380
381 }
382 while(erro > 1e-5);
383
384
385 //free(mpMat);
386 free(pMat);
387 //free(mp_old);
388 free(p_old);
389 //free(mp);
390 free(p);
391 //free(mbeta);
392 free(beta);
393 //free(ml_old);
394 free(l_old);
395 //free(ml);
396 free(l);
397 //free(mq_old);
398 free(q_old);
399 //free(mq);
400 free(q);

```

```
401
402     /*
403     for(i=1; i < N-1; i++){
404         free(mpMat[i]);
405         free(mp_old[i]);
406         free(mp[i]);
407         free(mbeta[i]);
408         free(ml_old[i]);
409         free(ml[i]);
410         free(mq_old[i]);
411         free(mq[i]);
412     }
413     free(mpMat);
414     free(mp_old);
415     free(mp);
416     free(mbeta);
417     free(ml_old);
418     free(ml);
419     free(mq_old);
420     free(mq);
421     */
422
423     ticks2 = clock();
424     elapsedTime = walltime( &startTime );
425     stop = omp_get_wtime();
426
427     /*Dados finais mostrados na tela*/
428
429     printf("\n----- MALHA %d x %d",n,n);
430     printf("\n----- PRESSAO ----- ITERACAO %d ----- ERRO %6.4E", k, erro);
431     printf("\n----- CPU TIME %ld, %ld, %g, %g, %.4g", ticks2, ticks1,
432         (ticks1+1.-1.)/CLOCKS_PER_SEC, (ticks2+1.-1.)/CLOCKS_PER_SEC,
433         (ticks2+1.-1.)/CLOCKS_PER_SEC - (ticks1+1.-1.)/CLOCKS_PER_SEC);
434     printf("\n----- TOTAL TIME %.6f\n\n",elapsedTime);
435     printf("\n----- TOTAL TIME %.6f\n\n",stop - start);
436
437     return 0; /* fim da funcao main */
438
439     printf("Oi\n");
440     return 0;
441
442
443 }
444
```