

```

a2/FW/fw_sr_p.c

1  /*
2   * Recursive implementation of the Floyd-Warshall algorithm.
3   * command line arguments: N, B
4   * N = size of graph
5   * B = size of submatrix when recursion stops
6   * works only for N, B = 2^k
7   */
8
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <sys/time.h>
12 #include <omp.h>
13 #include "util.h"
14
15 inline int min(int a, int b);
16 void FW_SR (int **A, int arow, int acol,
17             int **B, int brow, int bcol,
18             int **C, int crow, int ccol,
19             int myN, int bsize);
20
21 int main(int argc, char **argv)
22 {
23     int **A;
24     int i,j,k;
25     struct timeval t1, t2;
26     double time;
27     int B=16;
28     int N=1024;
29
30     if (argc !=3){
31         fprintf(stdout, "Usage %s N B \n", argv[0]);
32         exit(0);
33     }
34
35     N=atoi(argv[1]);
36     B=atoi(argv[2]);
37
38     if ((N%B)!=0){
39         fprintf(stdout, "N must be multiple of B\n");
40         exit(0);
41     }
42
43     A = (int **) malloc(N*sizeof(int *));
44     for(i=0; i<N; i++) A[i] = (int *) malloc(N*sizeof(int));
45
46     graph_init_random(A, -1, N, 128*N);
47
48 //-----
49     gettimeofday(&t1,0);
50
51     #pragma omp parallel

```

```
52 #pragma omp single
53 {
54     FW_SR(A,0,0, A,0,0,A,0,0,N,B);
55 }
56
57 gettimeofday(&t2,0);
58
59     time=(double)((t2.tv_sec-t1.tv_sec)*1000000+t2.tv_usec-t1.tv_usec)/1000000;
60     printf("FW_SR,%d,%d,%4f\n", N, B, time);
61
62
63 // for(i=0; i<N; i++)
64 //     for(j=0; j<N; j++) fprintf(stdout,"%d\n", A[i][j]);
65
66
67     return 0;
68 }
69
70 inline int min(int a, int b)
71 {
72     if(a<=b) return a;
73     else return b;
74 }
75
76 void FW_SR (int **A, int arow, int acol,
77             int **B, int brow, int bcol,
78             int **C, int crow, int ccol,
79             int myN, int bsize)
80 {
81     int k,i,j;
82     /*we use different task paral depending on the blocks A,B,C use.
83      If they use same blocks therre may be future depedencies , else not*/
84
85     //Arrays check
86     if (A!=B || A!=C){
87         printf("Different arrays not supported yet\n");
88         exit (1);
89     }
90     //row check
91     int RAB= arow==brow;
92     int RAC= arow==crow;
93     int RBC= brow==crow;
94     //col check
95     int CAB= acol==bcol;
96     int CAC= acol==ccol;
97     int CBC= bcol==ccol;
98     //case check
99     int case_id;
100    if (RAB&&RAC&&CAB&&CAC) case_id=0; //A,B,C same block
101    else if (RAB&&CAB) case_id=1; //A,B same block
102    else if (RAC&&CAC) case_id=2; //A,C same block
103    else case_id=3; //A separate from B and C
104
105    /*
```

```

106     * The base case (when recursion stops) is not allowed to be edited!
107     * What you can do is try different block sizes.
108     */
109     if(myN<=bsize)
110         for(k=0; k<myN; k++)
111             for(i=0; i<myN; i++)
112                 for(j=0; j<myN; j++)
113                     A[arow+i][acol+j]=min(A[arow+i][acol+j], B[brow+i][bcol+k]+C[crow+k]
114 [ccol+j]);
115     else {
116
117         switch(case_id){
118             case 0: //A,B,C same block
119             {
120                 //call1
121                 FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
122
123                 #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
124                 shared(A,B,C)
125                 {
126                     //call2
127                     FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2,
128 bsize);
129                 }
130                 #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
131                 shared(A,B,C)
132                 {
133                     //call3
134                     FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
135 bsize);
136
137                     //call4
138                     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
139 myN/2, bsize);
140
141                     //call5
142                     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
143 ccol+myN/2, myN/2, bsize);
144
145                     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
146                     shared(A,B,C)
147                     {
148                         //call6
149                         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,

```

```

150         #pragma omp taskwait
151
152             //call8
153             FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
154         }
155         break;
156         case 1: //A,B same block
157         {
158             #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
159             shared(A,B,C)
160             {
161                 //call1
162                 FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
163                 //call2
164                 FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2,
165                 bsize);
166                 //call7
167                 FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
168                 myN/2, bsize);
169                 //call8
170                 FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2,
171                 bsize);
172                 //call9
173                 FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
174                 bsize);
175                 //call14
176                 FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
177                 myN/2, bsize);
178                 //call15
179                 FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
180                 ccol+myN/2, myN/2, bsize);
181                 //call16
182                 FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
183                 myN/2, bsize);
184             }
185             #pragma omp taskwait
186         }
187         break;
188         case 2: //A,C same block
189         {
190             #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
191             shared(A,B,C)
192             {
193                 //call1
194                 FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
195                 //call3
196                 FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
197                 bsize);
198                 //call16

```

```

193         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
194         myN/2, bsize);
195             //call18
196             FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2,
197             bsize);
198         }
199             //call12
200             FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2,
201             bsize);
202             //call14
203             FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
204             myN/2, bsize);
205             //call15
206             FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
207             ccol+myN/2, myN/2, bsize);
208             //call17
209             FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
210             myN/2, bsize);
211         }
212         #pragma omp taskwait
213     }
214     break;
215     case 3: //A separate from B and C
216     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
217     shared(A,B,C)
218     {
219         //call11
220         FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
221         //call18
222         FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
223     }
224     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
225     shared(A,B,C)
226     {
227         //call12
228         FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
229         //call17
230         FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
231         myN/2, bsize);
232     }
233     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
234     shared(A,B,C)
235     {
236         //call13
237         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
238         //call16
239         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
240         myN/2, bsize);
241     }

```

```
235     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
236     shared(A,B,C)
237     {
238         //call4
239         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
240         myN/2, bsize);
241         //call5
242         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
243         ccol+myN/2, myN/2, bsize);
244         }
245         #pragma omp taskwait
246
247     }
248
249 /*
250 call1
251     FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
252 call12
253     FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
254 call13
255     FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
256 call14
257     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2, myN/2,
258     bsize);
259 call15
260     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
261     myN/2, bsize);
262 call16
263     FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol, myN/2,
264     bsize);
265 call17
266     FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2, myN/2,
267     bsize);
268 call18
269     FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
*/
270
```