

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         switch(case_id){
117             case 0: //A,B,C same block
118                 {
119                     //call1
120                     FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
121
122                     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
123                     shared(A,B,C)
124                     {
125                         //call2
126                         FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2,
127 bsize);
128                     }
129                     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
130                     shared(A,B,C)
131                     {
132                         //call3
133                         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
134 bsize);
135                     }
136                     #pragma omp taskwait
137
138                     //call4
139                     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
140 myN/2, bsize);
141
142                     //call5
143                     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
144 ccol+myN/2, myN/2, bsize);
145
146                     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
147                     shared(A,B,C)
148                     {
149                         //call6
150                         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
151 myN/2, bsize);
152                     }
153                     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
154                     shared(A,B,C)
155                     {
156                         //call7
157                         FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
158 myN/2, bsize);
159                     }

```

```

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             //call11
162             FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
163             //call12
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         }
173         #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
174         shared(A,B,C)
175         {
176             //call13
177             FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
178             bsize);
179             //call14
180             FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
181             myN/2, bsize);
182             //call15
183             FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
184             ccol+myN/2, myN/2, bsize);
185             //call6
186             FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
187             myN/2, bsize);
188         }
189
190         #pragma omp taskwait
191     }
192     break;
193     case 2: //A,C same block
194     {
195         #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
196         shared(A,B,C)
197         {
198             //call11
199             FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
200             //call13
201             FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2,
202             bsize);
203             //call6

```

```

193         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
myN/2, bsize);
194         //call8
195         FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2,
bsize);
196     }
197     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
shared(A,B,C)
198     {
199         //call2
200         FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2,
bsize);
201         //call4
202         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
myN/2, bsize);
203         //call5
204         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
ccol+myN/2, myN/2, bsize);
205         //call7
206         FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
myN/2, bsize);
207     }
208
209     #pragma omp taskwait
210 }
211 break;
212 case 3: //A separate from B and C
213     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
shared(A,B,C)
214     {
215         //call11
216         FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
217         //call8
218         FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
219     }
220
221     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
shared(A,B,C)
222     {
223         //call2
224         FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
225         //call7
226         FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
myN/2, bsize);
227     }
228     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
shared(A,B,C)
229     {
230         //call3
231         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
232         //call6
233         FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol,
myN/2, bsize);
234     }

```

```
235     #pragma omp task firstprivate(arow,acol,brow,bcol,crow,ccol,myN,bsize)
shared(A,B,C)
236     {
237         //call4
238         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2,
myN/2, bsize);
239         //call5
240         FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2,
ccol+myN/2, myN/2, bsize);
241     }
242     #pragma omp taskwait
243
244     break;
245 }
246 }
247 }
248
249 /*
250 call1
251     FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
252 call2
253     FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
254 call3
255     FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
256 call4
257     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2, myN/2,
bsize);
258 call5
259     FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol+myN/2,
myN/2, bsize);
260 call6
261     FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol, myN/2,
bsize);
262 call7
263     FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2, myN/2,
bsize);
264 call8
265     FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
266 */
267
268
269
```