```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include "libs/bspedupack.h"
#include "libs/bspfuncs.h"
#include "libs/vecio.h"
#include "libs/paullib.h"
#include "libs/debug.h"

#define EPS (10E-12)
#define KMAX (1500)

/*
 * This program takes as input:
 *  - a matrix distributed over n processors
 *  - vector distributions u and v (which processor owns which value)
 *  - values for vector v (reals)
 *
 *  and outputs a vector u such that
 *  A . u == v
 *
 *  by using the conjugate gradient method, generalised to parallel
 *  as detailed in the documentation.
 */

int P;

char vfilename[STRLEN], ufilename[STRLEN], matrixfile[STRLEN];

void bspcg(){

    int s, p, n, nz, i, iglob, nrows, ncols, nv, nu,
        *ia, *ja, *rowindex, *colindex, *vindex, *uindex;
    double *a, *v, *u, *r, time0, time1, time2;

    bsp_begin(P);

    p= bsp_nprocs(); /* p=P */
    s= bsp_pid();

    pid_t pid;

    /* get the process id */
    if ((pid = getpid()) < 0) {
        bsp_abort("unable to get pid\n");
    } else {
        HERE("The process id is %d\n", pid);
    }
    time0= bsp_time();

    // only proc 0 reads the files.
    if(s==0) {
        HERE("Start of BSP section.\n");
        char my_cwd[1024];
        getcwd(my_cwd, 1024);
        HERE("My working dir: PWD=%s\n", my_cwd);

        if(!file_exists(matrixfile)) {
            HERE("Matrix file doesn't exist. (%s)\n", matrixfile);
            bsp_abort("matrix doesn't exist\n");
        }
        if(!file_exists(vfilename)) {
            HERE("V-distrib file doesn't exist. (%s)\n", vfilename);
            bsp_abort("vector v doesn't exist\n");
        }
        if(!file_exists(ufilename)) {
            HERE("U-distrib file doesn't exist. (%s)\n", ufilename);
            bsp_abort("vector u doesn't exist\n");
        }
    }
    if (s==0){
        printf("CG solver\n");
```

```
 74          printf("   using %d processors\n",p);
 75      }
 76
 77      /* Input of sparse matrix */
 78      bspinput2triple(matrixfile, p,s,&n,&nz,&ia,&ja,&a);
 79      HERE("Done reading matrix file.\n");
 80
 81      /* Convert data structure to incremental compressed row storage */
 82      triple2icrs(n,nz,ia,ja,a,&nrows,&ncols,&rowindex,&colindex);
 83      HERE("Done converting to ICRS. nrows = %d, ncols = %d\n", nrows, ncols);
 84      vecfreei(ja);
 85
 86      int *owneru, *indu;
 87      /* Read vector distributions */
 88      bspinputvec(p,s,ufilename,&n,&nu,&uindex, &u, &owneru, &indu);
 89      HERE("Loaded distribution vec u (nu=%d).\n",nu);
 90      for(i=0; i<nu; i++){
 91          iglob= uindex[i];
 92          HERE("original input vec %d = %lf\n", iglob, u[i]);
 93      }
 94      for(i=0;i<n;i++) {
 95          HERE("u: global idx %d, and proc %d has it at spot %d\n", i,owneru[i
    ],indu[i]);
 96      }
 97
 98      int *ownerv, *indv;
 99      bspinputvec(p,s,vfilename,&n,&nv,&vindex, &v, &ownerv, &indv);
100      HERE("Loaded distribution vec v (nv=%d). set to zero.\n",nv);
101      zero(nv,v);
102      for(i=0;i<n;i++) {
103          HERE("v: global idx %d, and proc %d has it at spot %d\n", i,ownerv[i
    ],indv[i]);
104          if(ownerv[i] == s)
105              assert(i==vindex[indv[i]]); //sanity check.
106      }
107
108      //if(p!=1)
109      //    assert(nv!=nu); // we want interesting testcases.
110      HERE("Loaded a %d*%d matrix, this proc has %d nz.\n", n,n,nz);
111      if(s==0)
112          printf("Loaded a %d*%d matrix, proc 0 has %d nz.\n", n,n,nz);
113
114      if (s==0){
115          HERE("Initialization for matrix-vector multiplications\n");
116      }
117      bsp_sync();
118
119      // alloc metadata arrays
120      int *srcprocv, *srcindv, *destprocu, *destindu;
121
122      srcprocv  = vecalloci(ncols);
123      srcindv   = vecalloci(ncols);
124      destprocu = vecalloci(nrows);
125      destindu  = vecalloci(nrows);
126
127      // do the heavy lifting.
128      bsp_sync();
129      time1= bsp_time();
130
131      int k;
132
133      k = 0; // iteration number
134
135      // initialise mv data structures for doing u <- A.v
136      bspmv_init(p,s,n,nrows,ncols,nv,nu,rowindex,colindex,vindex,uindex,
137                 srcprocv,srcindv,destprocu,destindu);
138
139      r = vecallocd(nu);
140      // corresponds to:
141      // r := b - Ax,
142      // but our guess for x = 0;
143      // therefore the first time it corresponds to copying b into r
144      for(i=0; i< nu; i++) {
```

```
145         r[i] = u[i];
146     }
147
148     long double rho = bspip(p,s,nu,nu,r,uindex,r,owneru,indu);
149     long double alpha,gamma,rho_old,beta;
150     rho_old = 0; // just kills a warning.
151
152     double *pvec = vecallocd(nv);
153     double *w    = vecallocd(nu);
154
155     HERE("rho (r.r) turned out to be = %Lf\n", rho);
156     bsp_sync();
157     while ( k < KMAX &&
158             sqrt(rho) > EPS * bspip(p,s,nv,nv,v,vindex,v,ownerv,indv)) {
159         if(s==0)
160             printf("[Iteration %02d] rho  = %e\n", k+1, sqrt(rho));
161         if ( k == 0 ) {
162             // do p := r
163             copyvec(s,nu, nv,r,pvec, uindex, ownerv, indv);
164         } else {
165             beta = rho/rho_old;
166             // p:= r + beta*p
167             scalevec(nv, beta, pvec);
168             addvec(nv,pvec,vindex, nu, r, owneru, indu);
169         }
170         // w := Ap
171         bspmv(p,s,n,nz,nrows,ncols,a,ia,srcprocv,srcindv,
172               destprocu,destindu,nv,nu,pvec,w);
173
174         // gamma = p.w
175         gamma = bspip(p,s,nv,nu,pvec,vindex,w,owneru,indu);
176
177         alpha = rho/gamma;
178
179         // x := x + alpha*p
180         local_axpy(nv,alpha,pvec,v,
181                                  v);
182
183         // r := r - alpha*w
184         local_axpy(nu,-alpha,w,r,
185                                 r);
186
187         rho_old = rho;
188         // rho := ||rho||^2
189         rho = bspip(p,s,nu,nu,r,uindex,r,owneru,indu);
190
191         k++;
192
193     }
194
195     // end heavy lifting.
196
197     // postcondition:
198     // v s.t. A.v = u
199
200     bsp_sync();
201     time2= bsp_time();
202
203     if (s==0){
204         HERE("End of matrix-vector multiplications.\n");
205         printf("Initialization took only %.6lf seconds,\n", time1-time0);
206         printf("%d CG iterations took only %.6lf seconds (KMAX = %d).\n", k,
    (time2-time1), KMAX);
207         printf("The computed solution is:\n");
208     }
209
210     for(i=0; i<nv; i++){
211         iglob=vindex[i];
212         HERE("FINAL ANSWER *** proc=%d v[%d]=%lf \n",s,iglob,v[i]);
213     }
214
215     double* answer = vecallocd(n);
216     bsp_push_reg(answer,n*SZDBL);
```

```
217      int* nz_per_proc = vecalloci(P);
218      bsp_push_reg(nz_per_proc,P*SZINT);
219
220      bsp_sync();
221
222      for(i=0; i<nv; i++){
223          iglob=vindex[i];
224          bsp_put(0, &v[i], answer, iglob*SZDBL, SZDBL);
225      }
226      bsp_put(0, &nz, nz_per_proc, s*SZINT, SZINT);
227      bsp_sync();
228
229      if(s==0) {
230
231          int total_nz = 0;
232          for(i=0; i<p;i++)
233              total_nz += nz_per_proc[i];
234
235          printf("========= Solution =========\n");
236          printf("Final error = %e\n\n", sqrt(rho_old));
237          printf("csv_answer_head:\tP,N,nz,time,iters,success\n");
238          printf("csv_answer_data:\t%d,%d,%d,%lf,%d,%d\n",P,n,total_nz,(time2-
     time1),k,k<KMAX);
239
240  #ifdef DEBUG
241          for(i=0; i<n; i++) {
242              printf("solution[%d] = %lf\n", i, answer[i]);
243          }
244  #endif
245      }
246
247      bsp_pop_reg(answer);
248      bsp_pop_reg(nz_per_proc);
249
250      vecfreed(answer);   vecfreei(nz_per_proc);
251      vecfreed(w);        vecfreed(pvec);
252      vecfreed(r);
253
254      vecfreei(destindu); vecfreei(destprocu);
255      vecfreei(srcindv);  vecfreei(srcprocv);
256      vecfreed(u);        vecfreed(v);
257      vecfreei(uindex);   vecfreei(vindex);
258      vecfreei(rowindex); vecfreei(colindex);
259      vecfreei(ia);       vecfreed(a);
260      bsp_end();
261
262  } /* end bspcg */
263
264  int main(int argc, char **argv){
265
266      bsp_init(bspcg, argc, argv);
267      P = bsp_nprocs();
268
269      if(argc != 4){
270          fprintf(stderr, "Usage:\n");
271          fprintf(stderr, "\t%s [mtx-dist] [u-dist] [v-dist]\n\n", argv[0]);
272          exit(1);
273      }
274
275      strcpy(matrixfile, argv[1]);
276      strcpy(ufilename, argv[2]);
277      strcpy(vfilename, argv[3]);
278
279      bspcg();
280      exit(0);
281  }
```