```c
#include "bspfuncs.h"
#include "bspedupack.h"
#include "debug.h"

// This is my own version; since bspip from BSPedupack
// cannot handle v1 and v2 having arbitrary distributions.

/*
 * bspip computes the inner product of two vectors, arbitrarily distributed
 * over a number of processors.
 *
 * The parameters:
 *
 * - p: number of processors
 * - s: my processor id
 * - nv1 and nv2: the length of the vectors
 * - v1 and v2: the locally-stored components of v1 and v2
 * - v1index: the array which maps my local indexing of v1 to the global index
 * - procv2 and indv2: arrays mapping global vector indices to owner and offset on owner. (i.e. this tells us which processor owns a given nonzero)
 *
 * @return the inproduct of the two vectors
 */

double bspip(int p,int s,
        int nv1, int nv2,
        double* v1, int*v1index,
        double *v2, int *procv2, int *indv2)
{
    /* Compute inner product of vectors x and y of length n>=0 */
    // here v1 is the local vector and v2 is the remote vector.

    double *v2_locals;
    v2_locals = vecallocd(nv1);

    int i;
    bsp_push_reg(v2, nv2*SZDBL);

    bsp_sync();
    for(i=0; i<nv1; i++) {
        // get all the vector components from v2, from where ever they're
        // stored. the array procv2 tells us this, but it is indexed by the
        // component's global index.
        bsp_get(procv2[v1index[i]], v2, indv2[v1index[i]]*SZDBL, &v2_locals[i], SZDBL);
    }

    double myip=0.0;

    bsp_sync();
    for(i=0;i<nv1;i++) {
        myip += v1[i]*v2_locals[i];
    }

#ifdef __GNUC__
    size_t tagsz;
#else
    int tagsz;
#endif
    tagsz = SZINT;
    bsp_set_tagsize(&tagsz);
    bsp_sync();

    for(i=0;i<p;i++) {

        if(s==i) // don't send myself messages.
            continue;

        // tell all the other processors what my inproduct
        // contribution is
        bsp_send(i, &s, &myip, SZDBL);
    }

    bsp_sync();
    int nsums;
#ifdef __GNUC__
    size_t nbytes;
#else
    int nbytes;
#endif

    double alpha = myip; //since we won't be receiving this component

    bsp_qsize(&nsums, &nbytes);
    int status, tag;
    bsp_get_tag(&status, &tag);

    // get all the messages for this processor, which are
    // the inproduct contributions for the other processors.
    for(i=0;i<nsums; i++) {
        bsp_move(&myip, SZDBL);

        alpha += myip; // simply sum all the messages
        bsp_get_tag(&status, &tag);

    }

    bsp_pop_reg(v2);

    free(v2_locals);
    bsp_sync();

    return alpha;

} /* end bspip */

/*
 * Copy distributed vec v into u. Note that v and u may have different
 * distributions.
 *
 * - s: my processor id
 * - nv and nu: the length of the vectors
 * - v and u: the locally-stored components of v and u
 * - vindex: the array which maps my local indexing of v to the global index
 * - procu and indu: arrays mapping global vector indices to owner and offset on owner. (i.e. this tells us which processor owns a given nonzero)
 *
 * This function doesn't return anything, but places a copy of vector v into u, so
 * after the function terminates, u == v == \old{v}.
 */
void copyvec(int s,
        int nv, int nu,
        double* v, double* u,
        int* vindex,
        int* procu, int* indu)
{
    int i;

    bsp_push_reg(u, nu*SZDBL);
    bsp_sync();

    for(i=0;i<nv;i++) {

        // put my v into u somewhere remote;
        // we look up the correct processor and position on
        // that processor using the metadata arrays procu and indu
        bsp_put(procu[vindex[i]], &v[i], u, indu[vindex[i]]*SZDBL, SZDBL);
    }

    bsp_sync();
    bsp_pop_reg(u);
}

/*
```

```
 * Add some other distributed vector (r) to v (local)
 *
 * - nv and nr: the length of the vectors
 * - v and remote: the locally-stored components of v and remote vector
 * - vindex: the array which maps my local indexing of v to the global index
 * - procr and indr: arrays mapping global vector indices to owner and offset on owner. (i.e. this tells us which processor owns a given nonzero)
 *
 * Ensures that afterwards, v = \old{v} + remote, componentwise and on each processor
 */

void addvec(int nv, double *v, int*vindex, int nr, double *remote,
        int *procr, int *indr) {

    double *tmp = vecallocd(nv);
    bsp_push_reg(remote,nr*SZDBL);
    bsp_sync();

    int i;
    for(i=0;i<nv;i++) {
        // similar to how bspip above gets the proc that's relevant, and knows where
        // that processor stores the vector component we want.
        bsp_get(procr[vindex[i]], remote, indr[vindex[i]]*SZDBL, &tmp[i], SZDBL);
    }
    bsp_pop_reg(remote);
    bsp_sync();

    for(i=0;i<nv;i++) {
        v[i] += tmp[i];

    }

    free(tmp);
}
```