```
#include "generic_qcd_fields.h"
#include "layout_minsurface.h"
#include "options.h"
#include "random_vectors_cuda.h"
#include "cuda_vector_copy_v2.h"
#include "comm/comm_low.h"
#include "vector_util.h"
#include "vector_util_device.h"
#include "complex_vector_utils.h"
#include "dslash_multi_gpu.h"
#include "io_kentucky.h"
#include "timer.h"
#include "inverters_device.h"
#define CONTROL


using namespace qcd;


static struct {
  int dims[4];
  std::vector<double> kappas;
  std::string config, output;
} in;

void prepare_opts(options& opts)
{
  opts.add("nx", in.dims[0]);
  opts.add("ny", in.dims[1]);
  opts.add("nz", in.dims[2]);
  opts.add("nt", in.dims[3]);
  opts.add("config", in.config);
  opts.add("output", in.output);
}

// define the matrix multiplication you will be using
// for the inversion
struct my_mat_mult : matmult<device_wilson_field> {
 dslash_state_device<double>& ds;
 double kappa;

 my_mat_mult(dslash_state_device<double>& d, double k) : ds(d), kappa(k) {}

 void operator()(device_wilson_field& src, device_wilson_field& res)
 {
   ds.hopping(src, res, +1);
   res =  src/kappa - res;
 }

};

#define print0 if( get_node_rank() == 0) printf
int main(int argc, char** argv)
{
  init_machine(argc, argv);

  options opt;
  prepare_opts(opt);
  opt.read_options(argc,argv);

  //set layouy, read in the links, and copy them to the GPU
  layout_minsurface l(in.dims[0], in.dims[1], in.dims[2], in.dims[3]);
  su3_field lnkHost(&l);
  read_kentucky_lattice(in.config,lnkHost);
  device_gauge_field lnkDev(&l);
  copy_links_to_device(lnkHost, lnkDev);
:
  //define shifts, identify the most singular shift, and redefine shifts
  int iNoShifts = 4;
  double kappas[] = {0.14, 0.135, 0.13, 0.125, 0.120};
  double singularShift  = kappas[0];
  double pdShifts[iNoShifts];
  for(int i=0; i<iNoShifts;i++)  pdShifts[i] =1/kappas[i+1] - 1/singularShift;
```

```
  //generate a random source vector and copy it to the GPU
  device_random_field rnd(&l,1);
  device_wilson_field srcDev(&l);
  random_vector_uniform(srcDev, rnd);

  // allocate the solution vectors
  device_wilson_field* pvecSolutions[iNoShifts+1];
  for(int i=0; i<iNoShifts+1; i++) pvecSolutions[i] = new device_wilson_field(&l);

  // create instance of dslash and create the particular matrix multiplication being
 used
  dslash_state_device<double> ds(lnkDev);
  my_mat_mult myMatMult(ds, singularShift);

  timer t1;
  t1.start("BICGSTABM");
  int iter = qcd::bicgstabm_device(srcDev, iNoShifts, pdShifts, pvecSolutions, 1e-10
, 2000, myMatMult, rnd);
  t1.stop();


  print0("the number of iterations was %d\n",iter);
  fflush(stdout);

  //test the solutions
  device_wilson_field temp1Dev(&l);
  double result;

  // test the zero solution
  myMatMult(*pvecSolutions[0], temp1Dev);
  result = norm(temp1Dev - srcDev) / norm(srcDev);
  print0("vec0 residue is %20.15e\n", result);

  //test the rest
  for(int i=0; i<iNoShifts; i++)
  {
    myMatMult(*pvecSolutions[i+1], temp1Dev);
    result = norm(pdShifts[i] * *pvecSolutions[i+1] + temp1Dev - srcDev);
    result /= norm(srcDev);
    print0(" norm is %e\n",result);
  }
  for(int i=0; i<iNoShifts+1; i++) delete pvecSolutions[i];
  shutdown_machine();
  return 0;
}
```