Parallel Computing for GIS Problem

ZHANG ZHEN

Supervisor: Kwai Wong, Cheng Liu,
Lonnie Crosby, Nicholas Nagle

August 21, 2016

Abstract

In this project, we perform dasymetric mapping in parallel. Dasymetric mapping, as a commonly used mapping method in geography and demography, can be very time-consuming and can occupy a lot of space if performed on large areas. If we can develop some easy parallel method for dasymetric mapping, then time and space can both be saved. We are using MPI to fulfill this task. At the first stage, we try to use a simple model of dasymetric mapping–Regression Weighting Method on a state level. This requires 3 input data sets: NLCD data set, boundary data of block groups in a state, and ACS summary table. The output will be a matrix containing population in 30m*30m grids. The result turns out to be good. In our next step we are going to repeat the same process on a larger scale. Also we will implement the Penalized Maximum Entropy Dasymetric mapping(P-MEDM) method in parallel.

1 Introduction

1.1 Dasymetric Mapping

Most of the time, population data that are publicly available don't have a satisfactory resolution. In reality, we can only know how many people are there in a block group(6.1.1). But we don't know where people live within that block group. Actually, in GIS applications and other areas, sometimes we have to know the population distribution in a high resolution. Dasymetric mapping can help us with that. Dasymetric mapping will use an ancillary-level data(most of the time land type data), which is at a higher resolution level, to help us determine the population distribution more accurately.
In this project, we will cover two main methods for dasymetric mapping. The first one is weighting regression method, which is easier to implement. The second one is Penalized Maximum Entropy Dasymetric Mapping, which is more complicated but more accurate.

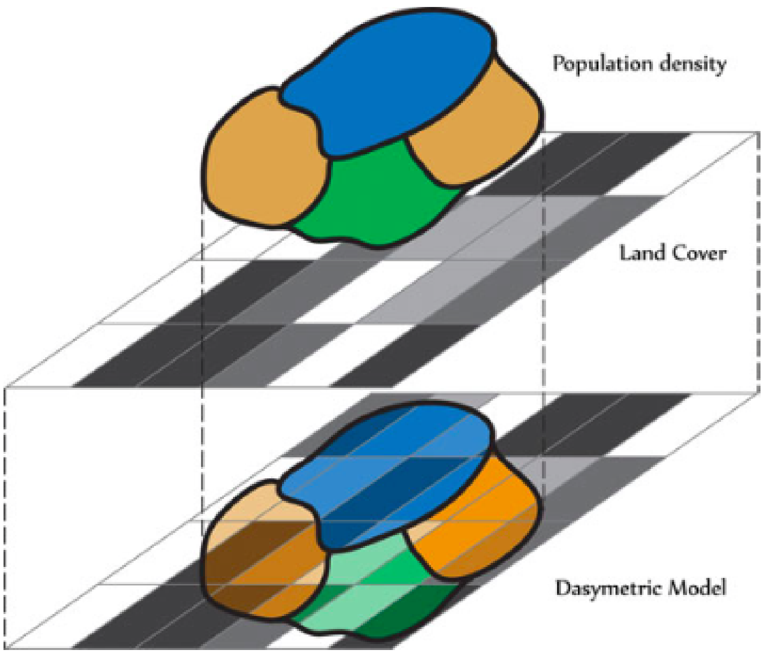

Figure 1: Dasymetric Mapping

1.2 Parallel Method

We would like to perform dasymetric mapping in parallel based on two main concerns. First, it will take a long time to do dasymetric mapping on the whole U.S. Later, if we want to perform dasymetric mapping on the whole world, time cost will be terrible. Second, space is also an issue. All the data (50GB for the whole states using my method, maybe need more space if using more complicated algorithms) can hardly be fit into a single memory. Thus we have to distribute data and work among different processes. We use MPI to fulfill this task.

2 Regression Weighting method

2.1 Steps

4 steps are needed in regression weighting method: data preparation, regression estimation, calculation, parallelize, as is shown in figure 2.

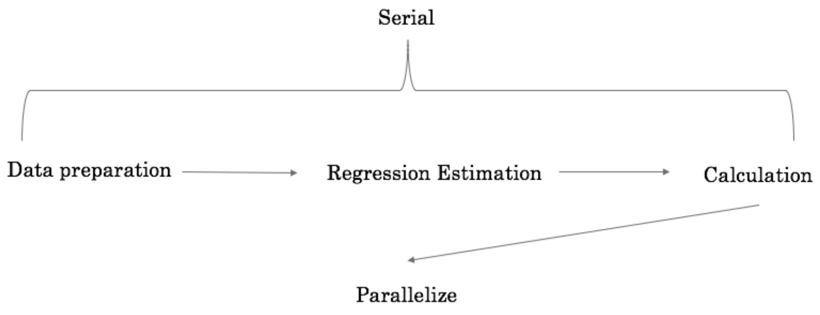

Figure 2: Steps needed for Regression Weighting Method

2.2 Data Preparation

In general, 3 data sets are needed for Regression Weighting method. The first one is National Land Cover data(NLCD), the second one is block group boundary file, the third one is American Community Survey(ACS) Summary table. Because the three datasets are on different enumeration unit and are in different format, we need to make some changes on them, to make them compatible with each other.

2.2.1 National Land Cover Data

National Land Cover Database 2011 (NLCD 2011) is the most recent national land cover product created by the Multi-Resolution Land Characteristics (MRLC) Consortium. NLCD 2011 provides the capability to assess national land cover changes and trends across the United States from 2001 to 2011. NLCD 2011 uses 16-class land cover classification scheme that has been applied consistently across the United States at a spatial resolution of 30 meters.
The file has a size of about 16GB, and it is a 104424×161190 matrix. Each cell contains a number represents the land type.
The data set can be downloaded from `http://www.mrlc.gov/nlcd2011.php`
After downloading the package, we can see that two files are larger, one is in .rrd, another is in .ige. These contains the main information of land type. But we will not use them directly. We will manipulate on .img file instead, because

there is a pointer in .img file that points to these two files. We can open the .img file using QGIS. (.img means the file is raster file, which is a matrix.) A map of US will appear like figure 3:

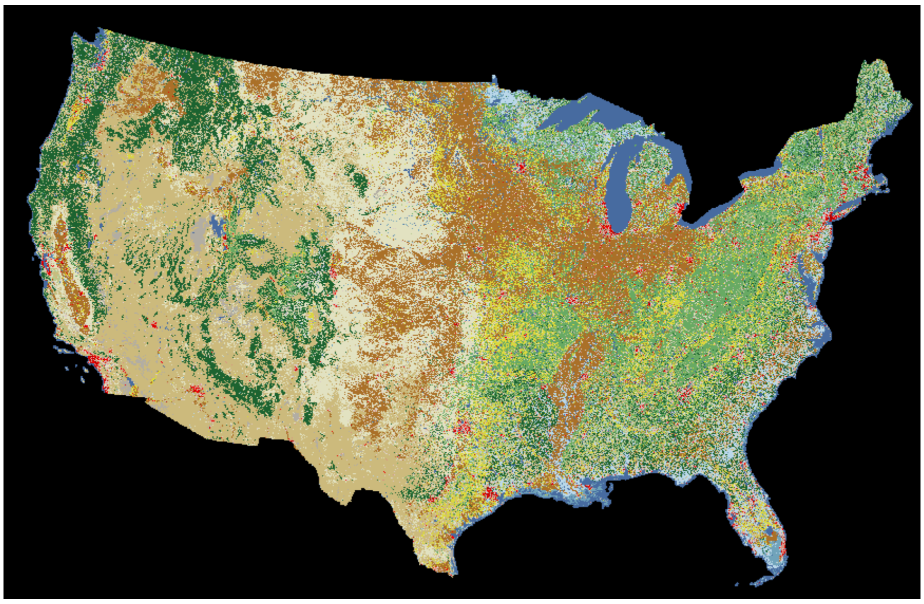

Figure 3: NLCD data under QGIS

However, we only need the land information of Tennessee at this stage. So we will cut Tennessee out from this data set. This requires a few steps:
1. Download the state-boundary file for the whole US from `https://www.census.gov/geo/maps-data/data/cbf/cbf_state.html`
2. Load state-boundary file and NLCD data into QGIS.
3. Highlight Tennessee in state-boundary file and save it as a separate image in QGIS.
4. Use the boundary of Tennessee(from 3) to clip NLCD data. This step is still done using QGIS.
Finally we get a picture like figure 4: This picture contains the land type data

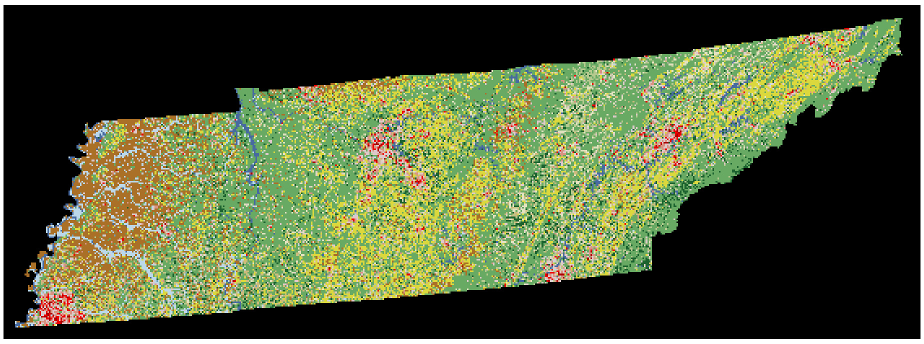

Figure 4: Land Type data of Tennessee

that we want. We can read them out using GDAL later.

2.2.2 Block Group Boundary File

Block group boundary file is in state level. It tells the shape of block groups in Tennessee. In QGIS, it looks like figure 5: However, this file is not a raster

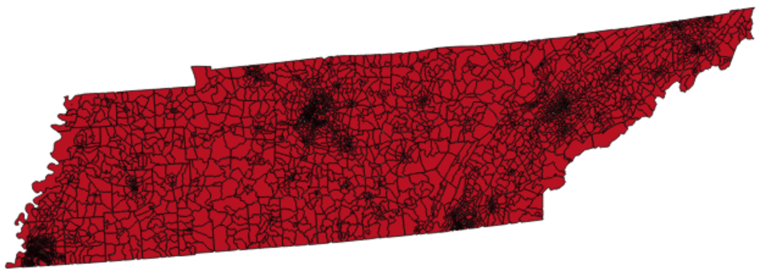
Figure 5: Block group boundary of Tennessee

file. We cannot manipulate it as a matrix. Thus, we have to rasterize it using QGIS. Steps are shown below:
1. Download the block group boundary file of Tennessee in 2015 from `https://www.census.gov/geo/maps-data/data/cbf/cbf\_blkgrp.html`.
2. Load this file into QGIS.
3. Add a new attribute called "GROUPID" into block group boundary file. We will use a unique number between 1 and 4125 to identify a block group(There are 4125 block groups in Tennessee). We want to rasterize the file based on this attribute. However, values in a raster file can only range from 0 to 255. Thus, we add an attribute named "QUOTIENT", which is equal to GROUPID/256, another attribute named "REMAINDER", which is equal to GROUPID%256. Then these two attributes can uniquely identify a block group, and their values are between 0 and 255. We will rasterize the shape file into 2 matrices according to QUOTIENT and REMAINDER separately. This can be done using QGIS easily.

2.2.3 American Community Survey Summary Table

American Community Survey(ACS) Summary Table, is a .csv file. It tells the total population, population by race, population by sex etc. in each block group. Because we only need some columns of this data set, we will use R to select some columns of it and reorder them according to the GROUPID we defined in section 2.2. Because we will do dasymetric mapping in C rather than in R, we still have to load our R matrix into C. This is a little bit complicated. We use a C program that calls R function to fulfill this task. It is implemented in serial.c/parallel.c as a function named read_bg() as well as an R function named ACS.r.
Details of what ACS.r does is omitted here. You can read it for more reference.

2.3 Regression Estimation

Now we have the land type data for Tennessee, however, it cannot be used directly to determine population distribution. However, it can be used to give a brief estimate of population density in each 30m*30m cell. We can use a linear regression model to model the relation between land type and density:

$$P_s = \alpha + \sum_{c=1}^{C} \beta_c A_{sc} + \epsilon_s$$

where P_s is the population of zone s; α is the intercept term; β_c is the coefficient for land cover c; $A_s c$ is the area of land cover c within zone s; C is the number of populated land cover types occurring within study region; and ϵ_s is a random error term. It can also be written as:

$$E(P_s|A_s) = \alpha + \beta A_s$$

where β and A_S are vectors. Negative population totals can also be avoided by using Poisson regression:

$$E(P_s|A_s) = \exp(\alpha + \beta A_s)$$

Here s(source zone) is a block group. The values in β are the estimated population density for cells with different land types.

2.4 Calculation

We will use the formula below to get our final result:

$$\widehat{Pop_t} = Pop_s \frac{w_t}{\sum_{t \in s} w_t}$$

Here s, the source region is block group. t, the target region is 30m*30m cell. That means, the population in a cell is equal to the population in a block group times the weight of that cell divided by the total weights of that block group. Because we have determined the estimated population density for cells with different land types, we can use them as the weights w_t here.
By this formula, we get our final result, we should have interpreted it as the population in each 30m*30m cell. However, because this number is always less than 1, we prefer to interpret it as the possibility for a cell to reside people. The result looks like figure 6 (Darker color mean higher possibility to reside people).

2.5 Parallelize

This project is quite open because actually there are many parts that can be parallelized. I have done the parallelization on 2 aspects. First, because we would like to do dasymetric mapping on different groups of people, we can use different processes to analyze on different groups of people, as is shown in figure 7 and figure 8. What's more, figure 8 shows the the result of parallel dasymetric

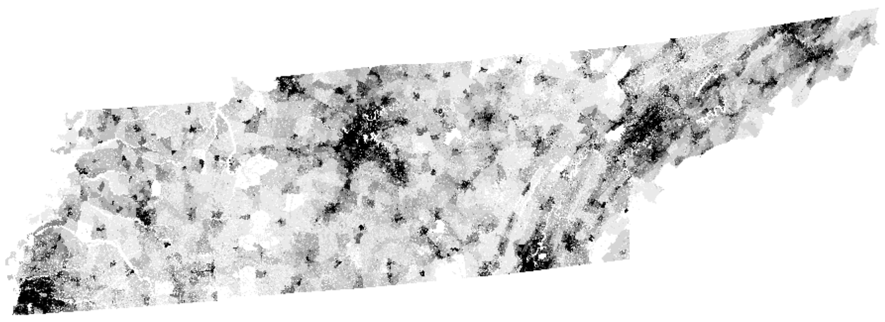

Figure 6: Population Density Map under Dasymetric Mapping

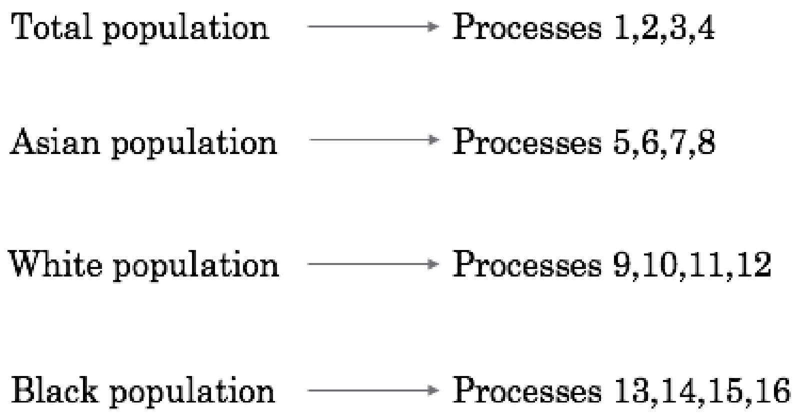

Figure 7: Work distribution(1)

mapping. Each picture shows the population distribution of a certain group of people(Total, White, Asian, Black). After we have distributed work into small groups, we can further distribute within these small groups. We can cut the input data by rows and assign them separately to different processes. Then every process can do their own calculation with little communication, as is shown in figure 9. The original time cost to run Regression Weighting Method on Tennessee analyzing 4 groups of people will be about half a minute, after parallelization, time cost is reduced to only 2 seconds. Although this seems unnecessary, if we apply the method on the whole U.S., analyzing more features, more importantly, if we apply P-MEDM method in the next section, this step will become very useful.

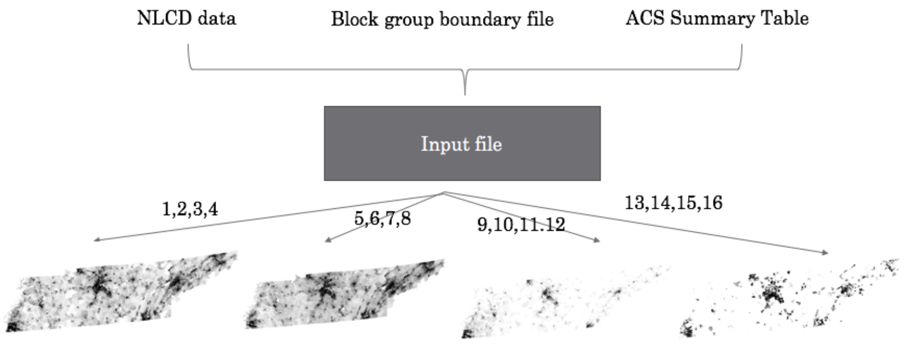

Figure 8: Work distribution(2)

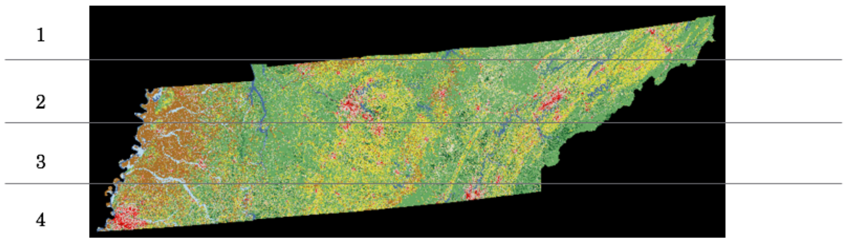

Figure 9: Work distribution(3)

3 Penalized Maximum Entropy Dasymetric Mapping(P-MEDM)

3.1 Introduction

P-MEDM Method is proposed by Dr. Nagle. The main motivation for proposing this method is that the role of uncertainty in dasymetric modeling has not been fully addressed as of yet. Uncertainty is usually present because most population data are themselves uncertain, or the geographic processes that connect population and the ancillary data layers are not precisely known. The P–MEDM propagates uncertainty through the model and yields fine-resolution population estimates with associated measures of uncertainty. For detailed explanation, read Nagle's paper in http://dx.doi.org/10.1080/00045608.2013.843439. Also, programs have been written by Nagle and April to explain this method.
There is one major difference between this program and what we are going to do. We want to map population in a block group into 30m*30m grids while Nagle wants to map population in a PUMA(A2) into block groups. However, we can still learn a lot from the program.

3.2 Steps

The steps are the same as Regression Weighting Method, as shown in figure 2. However, the last step: calculation is different between these two methods. For Regression Weighting Method, we use this easy formula:

$$\widehat{Pop_t} = Pop_s \frac{w_t}{\sum_{t \in s} w_t}$$

Here s is the source region(block group), t is the target region(30m*30m grid). w_t is the weights we have assigned to each target region by regression model. While for P-MEDM, we will use this formula to determine population in each grid:

$$\max - \sum_{it} \frac{n}{N} \frac{w_{it}}{d_{it}} \log(\frac{w_{it}}{d_{it}}) - \sum_{k} \frac{e_k^2}{2\sigma_k^2}$$

subject to the relaxed pycnophylactic constraints

$$\sum_{it \in k} w_{it} = \widehat{Pop_k} + e_k$$

for each constraint k. W_{it} is the number of individuals like sample record i in target region t. $\widehat{Pop_k}$ will be the total population Where n is the size of the microdata sample, N is the population size, σ_k^2 is the variance of the uncertainty e_k, and d_{it} is a prior estimate of the population w_{it}. The term $\sum_k \frac{e_k^2}{2\sigma_k^2}$ is a penalty factor; it penalizes solutions with large errors. This is an optimization problem with 2 unknowns and one constraint. We can solve this problem using trust region method(A3).

3.3 Implementation Details

Implementation has not yet been done. But I can still give some advice on that. The first few parts should be the same as the program for Regression Weighting Method. The only thing we need to do is to change the calculation formula to our new P-MEDM formula and use trust region to solve the problem. (Here we will use the regression coefficient as d_it.) However, it will take extremely long time to run trust region directly compared to using Regression Weighting Method. Thus, we have to do some transformation on the input data (from dense matrix to sparse matrix, for example). Also, preconditioning is needed for the optimization problem. We can trace Nagle's code for more reference. Again, we cannot copy Nagle's code directly. Firstly, because we want to map population in a block group into 30m*30m grids while Nagle wants to map population in a PUMA(A2) into block groups. Secondly, because his code is written is R and C++, which will be hard to parallelize. We have to translate them into C. After we successfully implement P-MEDM Method in serial, we will try to parallelize it. This is worth doing because it will take a long time to run P-MEDM if we want to map population into 30m*30m grids.

4 Future Work

4.1 Regression Coefficients

As I have mentioned before, I use non-negative regression method to estimate the population density of each 30m*30m grid. However, the coefficients I got is not satisfactory. There are 3 possible ways to solve this problem suggested by Dr. Nagle:
1. Clever subsetting of data
2. Use constrained regression to force the regression coefficients to be positive
3. Aggregation of negative land use classes with other non-negative classes

There isn't a well-established literature here. The most widely used method is clever subsetting to respond that are mostly of one land use class. And we prefer to choose Poisson regression as our regression model:

$$E(P_S|A_S) = \exp(\alpha + \beta A_S)$$

. Because the regression program is written in R, we only need to modify on nnls.r.

4.2 Increasing Scale

We want to do the same process for the whole United States. The main method will be no different. However, we have to modify the data preparation part. The main problem is, for the block group boundary file, they are all in state level. There is no block group boundary file for the whole United States. So if we want to rasterize them separately using QGIS by ourselves, it will take a long time. One possible way to solve this problem is that we can use GDAL to do rasterization instead of QGIS. And we can run the program in parallel to save time. Then, we can combine these matrices together, or we can remain them in each parallel processes without combining them. We can choose the better method from these two. If we want to do dasymetric mapping for the whole world, there will be more difficulties.

4.3 Cutting Data

There are many ways for cutting data. We don't necessarily have to cut the data by rows. For example, we can cut the data by columns or by boxes. Though for me, cutting by rows will be a better choice because GDAL read data by rows. Another thing that can be improved is we can choose how much to read by each process. Now we assign equal amount of data to every process. However, for processes which are responsible for processing data on the boundary–They don't need to do much calculation compared to processes which are responsible for processing data in the center. Thus we can assign more data for the first kind of processes so that they will not wait for the second kind of processes. This will save some time. We have to determine how much data should be assigned to each process.

4.4 P-MEDM

First of all, we have to have a deeper understanding of Prof. Nagle's code. A lot of the contents in the code are not revealed in the paper. We can ask him for more reference. Socond, Nagle's code for P-MEDM method only contains half of the story: it only uses ACS Summary Table and PUMS data to predict the number of population in a block group. However, it doesn't use the ancillary level data(NLCD) to perform dasymetric mapping. We need to implement the code for performing dasymetric mapping by ourselves(Nagle seems to miss the codes for that part). Third, we can try our best to find a possible way to parallelize P-MEDM Method. But that will be a lot more difficult because the original codes are written in R and C++. Possibly we have to change the codes into C codes first.

5 Conclusion

Dasymetric mapping, as a commonly used method in geographic and demographic study, is our researching target in this project. There are two main methods to do dasymetric mapping: Regression Weighting Method and P-MEDM Method. In this project, I have implemented Regression Weighting Method in serial as well as in parallel. This seems unnecessary because the serial code is fast enough already. However, this gives us an idea on how to parallelize P-MEDM Method, which is more useful in practice. The following work will be fixing some small problems in Regression Weighting Method, implementing P-MEDM method in C and parallelizing it.

Acknowledgement

Support from City University of Hong Kong, Oak Ridge National Laboratory, Joint Institute for Computational Sciences and University of Tennessee are gratefully acknowledged. Our mentors Dr. Cheng Liu, Dr. Kwai Wong, Dr Lonnie Crosby and Dr. Nicholas Nagle, as well as other students in this CSURE program, are of great help.

References

[1] Nicholas N. Nagle, Barbara P. Buttenfield, Stefan Leyk, Seth Spielman(2014) *Dasymetric Modeling and Uncertainty*, Annals of the Association of American Geographers, 104:1, 80-95, DOI: 10.1080/00045608.2013.843439.

[2] Andrey Petrov (2012) *One hundred years of dasymetric mapping: Back to the origin*, The Cartographic Journal, 49:3, 256-264, DOI: 10.1179/1743277412Y.0000000001

[3] Jeremy Mennis (2003) *Generating surface models of population using dasymetric mapping*, The Professional Geographer, 55:1, 31-42

[4] Mitchel Langford (2006) *Obtaining population estimates in non-census reporting zones: An evaluation of the 3-class dasymetric method*, Computers, Environment and Urban Systems 30, 161-180

[5] Cory L. Eicher and Cynthia A. Brewer (2011) *Dasymetric mapping and areal interpolation: Implementation and evaluation*, Cartography and Geographic Information Science, Vol.28, No.2, pp.125-138

[6] Rachel Sleeter *Dasymetric mapping techniques for the San Francisco Bay Region, California*

Appendices

A Terms explanation

A.1 Block Group

A Census Block Group is a geographical unit used by the United States Census Bureau which is between the Census Tract and the Census Block. It is the smallest geographical unit for which the bureau publishes sample data, i.e. data which is only collected from a fraction of all households.

A.2 PUMA

PUMA is a larger census unit than block group. They are the only sub-state geographic identifiers on the PUMS records. But now that these areas are being used to publish summary tables based on the American Community Survey data, they should become a lot more widely used. Because they are required to have a minimum population of 100,000 all PUMA areas exceed the 65,000 population threshold, thus insuring that there will be single-year ACS data for them published each year.

A.3 Trust Region Method

Trust region method is a commonly used method in optimization problems. Suppose the target function is f. For trust region method, the information gathered about f is used to construct a model function m_k whose behavior near the current point x_k is similar to that of the actual objective function f. Because the model m_k may not be a good approximation of f when x is far from x_k, we restrict the search for a minimizer of m_k to some region around x_k. In other words, we find the candidate step p by approximately solving the following subproblem:
min $m_k(x_k+p)$, where x_k+p lies insidethe trust region. If the candidate solution does not produce a sufficient decrease in f, we conclude that the trust region is too large, and we shrink it and re-solve the subproblem above. Usually, the trust region is a ball defined by $\|(p) \leq \delta$, where the scalar δ ¿ 0 is called the trust-region radius. Elliptical and box-shaped trust regions may also be used. The model m_k is usually defined to be a quadratic function of the form

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p$$

where f_k,∇f_k,and B_k are a scalar, vector,and matrix, respectively.As the notation indicates, f_k and ∇f_k are chosen to be the function and gradient values at the point x_k, so that m_k and f are in agreement to first order at the current iterate x_k. The matrix B_k is either the Hessian $\nabla^2 f_k$ or some approximation to it. For more reference, read J.Wright's *Numerical Optimization.*

B Tips on running the code

Some packages should be installed to run the program successfully. First of all, you should install GDAL. If you want to do regression in C rather than in

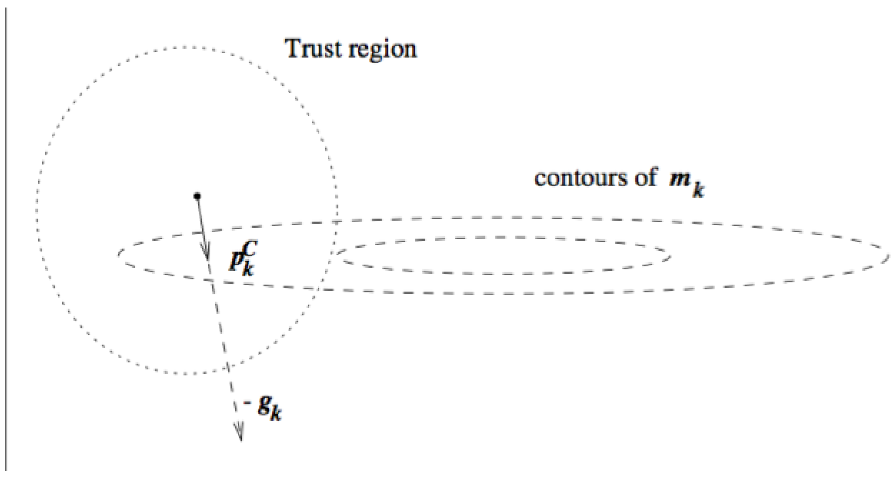

Figure 10: Trust Region Method

R, you should install gsl or some other statistical packages in C. Also, another library called REmbedded should also be installed. But that will be installed automatically when you install R. These packages should be included in your program, also should be linked with your program in the make file.
Another thing to notice is that you have to define your LD_LIBRARY_PATH to the path to libraries above. For example, in my case, I have to type these commands in Linux:

```
LD_LIBRARY_PATH=/usr/local/lib:
            /home/zzhang22/nlcd/GSL-INSTALL/lib:
                        /home/zzhang22/R/R-3.2.4/lib
export LD_LIBRARY_PATH
```

We also have to define our R_HOME by ourselves like this:

```
R_HOME=/home/zzhang22/R/R-shared-install/lib64/R
export R_HOME
```

C Serial Code

```
#include "gdal.h"
#include "cpl_string.h"
#include "cpl_conv.h" /* for CPLMalloc() */
#include "stdlib.h"
#include "mpi.h"
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include <Rembedded.h>
#include <Rinternals.h>
#include <ogr_api.h>
```

```
//These two methods are not needed now
void create_csv(int a[][4125],int m,int n){
        FILE *fp;
        int i,j;
        fp=fopen("test.csv","w+");
        for(i=0;i<m;i++){
                for(j=0;j<n-1;j++){
                        fprintf(fp,"%d,",a[i][j]);
                }
                fprintf(fp,"%d\n",a[i][n-1]);
        }
}

void read_csv(int bg_pop[4125][4]){
        FILE *fp = fopen("../pop_by_race.csv","r");
        if(fp!=NULL){
                char line[40];
                int i=0;
                while(fgets(line,sizeof line,fp)!=NULL){
                        if(line[0]!='\n'&&line[0]!='\r'){
                                char* pch;
                                pch=strtok(line,",");
                                for(int j=0;j<4;j++){
                                        bg_pop[i][j]=atoi
                                            (pch);
                                        pch=strtok(NULL,"
                                            , \n\r");
                                }
                                i++;
                        }
                }
                fclose(fp);
        }
        else{
                perror("user.dat");
        }
}

void source(const char *name){
        SEXP e;

        PROTECT(e=lang2(install("source"),mkString(name))
            );
        R_tryEval(e,R_GlobalEnv,NULL);
        UNPROTECT(1);
}

void read_bg(int pop_by_race[4125][4]){
        int r_argc = 2;
        char *r_argv[] = { "R", "--silent" };
```

```
    Rf_initEmbeddedR(r_argc, r_argv);
    OGRRegisterAll();
    OGRDataSourceH hDS;
    hDS = OGROpen("/home/zzhang22/nlcd/Input/
        bg_boundary/cb_2015_47_bg_500k.shp", FALSE,
        NULL);
    if( hDS == NULL ){
            printf( "Open failed.\n" );
            exit( 1 );
    }
    OGRLayerH hLayer;

hLayer = OGR_DS_GetLayer( hDS, 0 );
    OGRFeatureH hFeature;

OGR_L_ResetReading(hLayer);
    int i=0;
    int geolen=4125;
    char *GEOID[4125];
    while( (hFeature = OGR_L_GetNextFeature(hLayer))
        != NULL ){
    OGRFeatureDefnH hFDefn = OGR_L_GetLayerDefn(
        hLayer);
    int iField=5;
    GEOID[i]=OGR_F_GetFieldAsString( hFeature, iField
        ) ;
    i++;
    }
    source("/home/zzhang22/nlcd/Rcodes/ACS.r");
    SEXP arg;
    PROTECT(arg=allocVector(STRSXP, geolen));
    for(i=0;i<geolen;i++){
            SET_STRING_ELT(arg,i,Rf_mkChar(GEOID[i]))
                ;
    }
    SEXP ACS_call;
    PROTECT(ACS_call=lang2(install("ACS"),arg));
    int errorOccurred;
    SEXP ret=R_tryEval(ACS_call,R_GlobalEnv,&
        errorOccurred);
    double *pop;
    if(!errorOccurred){
            pop=REAL(ret);
            for(int i=0;i<geolen;i++){
                    for(int j=0;j<4;j++){
                            pop_by_race[i][j]=(int)
                                pop[i+j*geolen];
                    }
            }
    }
```

```
        UNPROTECT(2);
}

int main(int argc,char **argv)
{
        //Get Array for the population of each block
            group
        int bg_pop[4125][4]={0};

        //read block group population by already-prepared
             csv files
        //read_csv(bg_pop);

        //read block group population from original
            dataset
        read_bg(bg_pop);

        //load Tennessee's NLCD data, and rasterized
            block group boundary data
        GDALDatasetH  hDataset;
        GDALDatasetH  hDataset2;
        GDALDatasetH  hDataset3;
        GDALDatasetH  hDataset4;
        GDALAllRegister();
        hDataset = GDALOpen("../NLCD_TN.tif", GA_ReadOnly
             );
        hDataset2 = GDALOpen("../REMAINDER.tif",
            GA_ReadOnly);
        hDataset3 = GDALOpen("../QUOTIENT.tif",
            GA_ReadOnly);
        hDataset4 = GDALOpen("../result.tif",GA_Update);

        GDALRasterBandH hBand;
        GDALRasterBandH hBand2;
        GDALRasterBandH hBand3;
        GDALRasterBandH hBand4;
        hBand = GDALGetRasterBand(hDataset,1);
        hBand2 = GDALGetRasterBand(hDataset2,1);
        hBand3 = GDALGetRasterBand(hDataset3,1);
        hBand4 = GDALGetRasterBand(hDataset4,1);

        int relation[96]={-1};
        relation[11]=0;
        relation[21]=1;
        relation[22]=2;
        relation[23]=3;
        relation[24]=4;
        relation[31]=5;
        relation[41]=6;
        relation[42]=7;
```

```
relation[43]=8;
relation[52]=9;
relation[71]=10;
relation[81]=11;
relation[82]=12;
relation[90]=13;
relation[95]=14;

unsigned char *land;
unsigned char *remainder;
unsigned char *quotient;
int ASC[15][4125]={0};
int nXSize = GDALGetRasterBandXSize( hBand );
land = (unsigned char *) CPLMalloc(sizeof(
   unsigned char)*nXSize);
remainder = (unsigned char *) CPLMalloc(sizeof(
   unsigned char)*nXSize);
quotient = (unsigned char *) CPLMalloc(sizeof(
   unsigned char)*nXSize);

time_t t_start,t_end;
t_start =time(NULL);

for(int j=0;j<9477;j++){
        GDALRasterIO( hBand, GF_Read, 0, j,
            nXSize, 1,
                land, nXSize, 1, GDT_Byte,
                0, 0 );
        GDALRasterIO( hBand2, GF_Read, 0, j,
            nXSize, 1,
                remainder, nXSize, 1, GDT_Byte,
                0, 0 );
        GDALRasterIO( hBand3, GF_Read, 0, j,
            nXSize, 1,
                quotient, nXSize, 1, GDT_Byte,
                0, 0 );
        for(int i=0;i<26038;i++){
                int intLand=(int) land[i];
                if(relation[intLand]!=-1){
                        int bg_id=(int) (quotient
                            [i]*256+remainder[i]);
                        ASC[relation[intLand]][
                            bg_id-1]++;
                }
        }
}

double *coeff;
//getCoeff(coeff,ASC,bg_pop);
//create_csv(ASC,15,4125);
```

```
int bg_arr[4125*4];
int ASC_arr[15*4125];
for(int i=0;i<4;i++){
        for(int j=0;j<4125;j++){
                bg_arr[i*4125+j]=bg_pop[j][i];
        }
}
for(int i=0;i<4125;i++){
        for(int j=0;j<15;j++){
                ASC_arr[i*15+j]=ASC[j][i];
        }
}

source("/home/zzhang22/nlcd/Rcodes/nnls.r");
SEXP bgp,ascp;
PROTECT(bgp=allocVector(INTSXP,4125*4));
memcpy(INTEGER(bgp),bg_arr,4125*4*sizeof(int));
PROTECT(ascp=allocVector(INTSXP,15*4125));
memcpy(INTEGER(ascp),ASC_arr,15*4125*sizeof(int))
    ;

SEXP rgg_call;
PROTECT(rgg_call=lang3(install("coeff"),bgp,ascp)
    );
int errorOccurred;
SEXP ret=R_tryEval(rgg_call,R_GlobalEnv,&
    errorOccurred);
if(!errorOccurred){
        coeff=REAL(ret);
}
UNPROTECT(3);
Rf_endEmbeddedR(0);

int totalWeight[4125]={0};
for(int i=0;i<4125;i++){
        for(int j=0;j<15;j++){
                if(j==1) totalWeight[i]=
                    totalWeight[i]+14*ASC[j][i];
                else if(j==2) totalWeight[i]=
                    totalWeight[i]+49*ASC[j][i];
                else if(j==3) totalWeight[i]=
                    totalWeight[i]+91*ASC[j][i];
                else if(j==4) totalWeight[i]=
                    totalWeight[i]+126*ASC[j][i];
                else if(j==6||j==7||j==8)
                    totalWeight[i]=totalWeight[i
                    ]+10*ASC[j][i];
                else if(j==11||j==12) totalWeight
                    [i]=totalWeight[i]+20*ASC[j][i
                    ];
        }
}
unsigned char *result;
int weight;
result = (unsigned char *) CPLMalloc(sizeof(
    unsigned char)*nXSize);
for(int j=0;j<9477;j++){
        GDALRasterIO( hBand, GF_Read, 0, j,
            nXSize, 1,
                land, nXSize, 1, GDT_Byte,
                0, 0 );
        GDALRasterIO( hBand2, GF_Read, 0, j,
            nXSize, 1,
                remainder, nXSize, 1, GDT_Byte,
                0, 0 );
        GDALRasterIO( hBand3, GF_Read, 0, j,
            nXSize, 1,
               quotient, nXSize, 1, GDT_Byte,
                0, 0 );
        for(int i=0;i<26038;i++){
                int bg_id=(int) (quotient[i]*256+
                    remainder[i]);
                if(bg_id!=0&&totalWeight[bg_id
                    -1]!=0){
                        int intLand=(int) land[i
                            ];
                        if(relation[intLand]==1)
                            weight=14;
                        else if(relation[intLand
                            ]==2) weight=49;
                        else if(relation[intLand
                            ]==3) weight=91;
                        else if(relation[intLand
                            ]==4) weight=126;
                        else if(relation[intLand
                            ]==6||relation[intLand
                            ]==7||relation[intLand
                            ]==8) weight=10;
                        else if(relation[intLand
                            ]==11||relation[
                            intLand]==12) weight
                            =20;
                        else weight=0;
                        float temp=(float)(bg_pop
                            [bg_id -1][1])/
                            totalWeight[bg_id -1]*
                            weight;
                        result[i]=(int)(sqrt(temp
                            )*100)-1;
```

```
                                if(result[i]==255) result
                                    [i]=0;
                        }
                        else result[i]=0;
                }
                GDALRasterIO( hBand4, GF_Write, 0, j,
                    nXSize, 1, result, nXSize, 1, GDT_Byte
                    , 0, 0);
        }
        t_end = time(NULL) ;
        printf("time:%.0f s\n",difftime(t_end,t_start));
        return 0;
}
```

D Parallel Code

```
#include "gdal.h"
#include <math.h>
#include "cpl_string.h"
#include "cpl_conv.h" /* for CPLMalloc() */
#include "stdio.h"
#include "stdlib.h"
#include "mpi.h"
#include <time.h>
#include <Rembedded.h>
#include <Rinternals.h>
#include <ogr_api.h>

void create_csv(int a[][4125],int m,int n){
        FILE *fp;
        int i,j;
        fp=fopen("test.csv","w+");
        for(i=0;i<m;i++){
                for(j=0;j<n-1;j++){
                        fprintf(fp,"%d,",a[i][j]);
                }
                fprintf(fp,"%d\n",a[i][n-1]);
        }
}

void read_csv(int *bg_pop){
        FILE *fp = fopen("../BG_POP.csv","r");
        if(fp!=NULL){
                char line[4];
                int i=0;
                while(fgets(line,sizeof line,fp)!=NULL){
                        if(line[0]!='\n'&&line[0]!='\r'){
                                bg_pop[i]=atoi(line);
                                i++;
                        }
                }
                                fclose(fp);
        }
        else{
                perror("user.dat");
        }
}

void source(const char *name){
        SEXP e;

        PROTECT(e=lang2(install("source"),mkString(name))
            );
        R_tryEval(e,R_GlobalEnv,NULL);
        UNPROTECT(1);
}

void read_bg(int **pop_by_race){
        int r_argc = 2;
        char *r_argv[] = { "R", "--silent" };
        Rf_initEmbeddedR(r_argc, r_argv);
        OGRRegisterAll();
        OGRDataSourceH hDS;
        hDS = OGROpen( "/home/zzhang22/nlcd/Input/
            bg_boundary/cb_2015_47_bg_500k.shp", FALSE,
            NULL);
        if( hDS == NULL ){
                printf( "Open failed.\n" );
                exit( 1 );
        }
        OGRLayerH hLayer;

        hLayer = OGR_DS_GetLayer( hDS, 0 );
        OGRFeatureH hFeature;

        OGR_L_ResetReading(hLayer);
        int i=0;
        int geolen=4125;
        char *GEOID[4125];
        while( (hFeature = OGR_L_GetNextFeature(hLayer))
            != NULL ){
                OGRFeatureDefnH hFDefn =
                    OGR_L_GetLayerDefn(hLayer);
                int iField=5;
                GEOID[i]=OGR_F_GetFieldAsString( hFeature
                    , iField) ;
                i++;
        }
        source("/home/zzhang22/nlcd/Rcodes/ACS.r");
        SEXP arg;
        PROTECT(arg=allocVector(STRSXP, geolen));
        for(i=0;i<geolen;i++){
                SET_STRING_ELT(arg,i,Rf_mkChar(GEOID[i]))
                    ;
        }
        SEXP ACS_call;
        PROTECT(ACS_call=lang2(install("ACS"),arg));
        int errorOccurred;
        SEXP ret=R_tryEval(ACS_call,R_GlobalEnv,&
            errorOccurred);
        double *pop;
        if(!errorOccurred){
                pop=REAL(ret);
                for(int i=0;i<geolen;i++){
                        for(int j=0;j<4;j++){
                                pop_by_race[i][j]=(int)
                                    pop[i+j*geolen];
                        }
                }
        }
        UNPROTECT(2);
}

int malloc2dint(int ***array, int n, int m) {

        /* allocate the n*m contiguous items */
        int *p = (int *)calloc(n*m,sizeof(int));
        if (!p) return -1;

        /* allocate the row pointers into the memory */
        (*array) = (int **)malloc(n*sizeof(int*));
        if (!(*array)) {
                free(p);
                return -1;
        }

        /* set up the pointers into the contiguous memory
             */
        for (int i=0; i<n; i++)
                (*array)[i] = &(p[i*m]);

        return 0;
}

int free2dint(int ***array) {
        /* free the memory - the first element of the
            array is at the start */
        free(&((*array)[0][0]));

        /* free the pointers into the memory */
        free(*array);

        return 0;
}

int main(int argc,char **argv)
{
        //process id and number of cores used
        int myrank, nprocs;
        //to calculate time used
        time_t t_start,t_end;

        //block group population of all features
        int **bg_pop_all;

        int feature_num=4;//total features to analyze in
           ACS Summary Table
        int part;//Which part of the map will this
           process analyze
        int feature;//Which feature will this process
           analyze

        //block group population of a certain feature
        int bg_pop_one[4125];

        //load Tennessee's NLCD data, and rasterized
           block group boundary data
        GDALDatasetH  hDataset;
        GDALDatasetH  hDataset2;
        GDALDatasetH  hDataset3;
        GDALDatasetH  hDataset4;
        GDALAllRegister();
        hDataset = GDALOpen("../NLCD_TN.tif", GA_ReadOnly
            );
        hDataset2 = GDALOpen("../REMAINDER.tif",
           GA_ReadOnly);
        hDataset3 = GDALOpen("../QUOTIENT.tif",
           GA_ReadOnly);//remainder.tif and quotient.tif
           together represent the block group id

        GDALRasterBandH hBand;
        GDALRasterBandH hBand2;
        GDALRasterBandH hBand3;
        GDALRasterBandH hBand4;
        hBand = GDALGetRasterBand(hDataset,1);
        hBand2 = GDALGetRasterBand(hDataset2,1);
        hBand3 = GDALGetRasterBand(hDataset3,1);

        //allocate buffer for reading/writing a band
        unsigned char *land;
        unsigned char *remainder;
        unsigned char *quotient;
        unsigned char *result;

        int nXSize = GDALGetRasterBandXSize( hBand );
        int nYSize = GDALGetRasterBandYSize( hBand );
        land = (unsigned char *) CPLMalloc(sizeof(
           unsigned char)*nXSize);
        remainder = (unsigned char *) CPLMalloc(sizeof(
           unsigned char)*nXSize);
        quotient = (unsigned char *) CPLMalloc(sizeof(
           unsigned char)*nXSize);
        result = (unsigned char*) CPLMalloc(sizeof(
           unsigned char)*nXSize);

        //ASC is the number of 30m*30m grids in a block
           group for a specific land type.
        //Becasue there are 15 different land types and
           4125 different block groups, the array is in
           15*4125
        int **ASC;
        int **real_ASC;

        //To store the total weights for every block
           group
        int totalWeight[4125]={0};

        //weight for each 30m*30m grid
        int weight;

        //relation between the number in NLCD data and
           landtype
        int relation[96]={-1};
        relation[11]=0;
        relation[21]=1;
        relation[22]=2;
        relation[23]=3;
        relation[24]=4;
        relation[31]=5;
        relation[41]=6;
        relation[42]=7;
        relation[43]=8;
        relation[52]=9;
        relation[71]=10;
        relation[81]=11;
        relation[82]=12;
        relation[90]=13;
        relation[95]=14;

        //Initialize MPI
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);//get the
    rank of each process
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);//get the
    number of cores used

part=myrank%feature_num;
feature=myrank/feature_num;

if(feature==0)
        hDataset4 = GDALOpen("../result1.tif",
            GA_Update);
else if(feature==1)
        hDataset4 = GDALOpen("../result2.tif",
            GA_Update);
else if(feature==2)
        hDataset4 = GDALOpen("../result3.tif",
            GA_Update);
else if(feature==3)
        hDataset4 = GDALOpen("../result4.tif",
            GA_Update);
hBand4 = GDALGetRasterBand(hDataset4,1);

//4125 is the number of block groups in Tennessee
    , 4 is the number of features in ACS Table
malloc2dint(&bg_pop_all,4125,4);
malloc2dint(&ASC,15,4125);
malloc2dint(&real_ASC,15,4125);
//We use this function because we want all
    elements of the 2-d array lies contiguously
//so that we can broadcast it to other processes
    using MPI.

if(myrank==0){
        read_bg(bg_pop_all);
}
//pass this value to every other process
MPI_Bcast(&(bg_pop_all[0][0]),4125*4,MPI_INT,0,
    MPI_COMM_WORLD);

for(int i=0;i<4125;i++){
        bg_pop_one[i]=bg_pop_all[i][feature];
}

t_start=time(NULL);
//Each process will be responsible for read
    several rows in th map like this:
//----------------------------------------
//        process 1
//
```

```
//----------------------------------------
//          process 2
//
//----------------------------------------
//          process 3
//
//----------------------------------------
//          process 4
//
//----------------------------------------
if(feature==0){
        for(int j=part*(nYSize/ceil((float)nprocs
            /feature_num)+1);j<(part+1)*(nYSize/
            ceil((float)nprocs/feature_num)+1);j
            ++){
                if(j>=nYSize) break;
                //Read the land type into 'land'
                GDALRasterIO( hBand, GF_Read, 0,
                    j, nXSize, 1,
                        land, nXSize, 1, GDT_Byte
                            ,
                        0, 0 );
                //Read the block group ids into '
                    remainder' and 'quotient'
                GDALRasterIO( hBand2, GF_Read, 0,
                    j, nXSize, 1,
                        remainder, nXSize, 1,
                            GDT_Byte,
                        0, 0 );
                GDALRasterIO( hBand3, GF_Read, 0,
                    j, nXSize, 1,
                        quotient, nXSize, 1,
                            GDT_Byte,
                        0, 0 );
                for(int i=0;i<nXSize;i++){
                        int intLand=land[i];
                        if(relation[intLand]!=-1)
                            {
                                int bg_id=(
                                    quotient[i
                                    ]*256+
                                    remainder[i]);
                                ASC[relation[
                                    intLand]][
                                    bg_id-1]++;
                        }
                }
        }
}
MPI_Allreduce(&(ASC[0][0]),&(real_ASC[0][0])
            ,4125*15,MPI_INT,MPI_SUM,MPI_COMM_WORLD);

        double *coeff;

        //spread the 2d array to a 1d array so that it is
             easy to be tranformed into an R vector
        int bg_arr[4125*4];
        int ASC_arr[15*4125];

        for(int i=0;i<4;i++){
                for(int j=0;j<4125;j++){
                        bg_arr[i*4125+j]=bg_pop_all[j][i
                            ];
                }
        }

        for(int i=0;i<4125;i++){
                for(int j=0;j<15;j++){
                        ASC_arr[i*15+j]=real_ASC[j][i];
                }
        }

        source("/home/zzhang22/nlcd/Rcodes/nnls.r");
        SEXP bgp,ascp;
        PROTECT(bgp=allocVector(INTSXP,4125*4));
        memcpy(INTEGER(bgp),bg_arr,4125*4*sizeof(int));
        PROTECT(ascp=allocVector(INTSXP,15*4125));
        memcpy(INTEGER(ascp),ASC_arr,15*4125*sizeof(int))
            ;

        SEXP rgg_call;
        PROTECT(rgg_call=lang3(install("coeff"),bgp,ascp)
            );
        int errorOccurred;
        SEXP ret=R_tryEval(rgg_call,R_GlobalEnv,&
            errorOccurred);
        if(!errorOccurred){
                coeff=REAL(ret);
        }
        UNPROTECT(3);
        Rf_endEmbeddedR(0);

/*Here we get the weights for each cell(stored in double*
     coeff) by non-negative regression method. However,
    the coefficients determined are not very satisfactory.
     So we use some pre-determined weights in this program
    . Still, we have built the basic structure of doing
    regression using embedded R. In the next step, we only
     need to change the codes in R to help us get better
    weights, and use these weights to replace the
```

```
arbitrarily assigned weights below.*/

     for(int i=0;i<4125;i++){
              for(int j=0;j<15;j++){
                       if(j==1) totalWeight[i]=
                           totalWeight[i]+14*real_ASC[j][
                           i];//14, 49, 91... here will
                           be replaced by regression
                           coefficients
                       else if(j==2) totalWeight[i]=
                           totalWeight[i]+49*real_ASC[j][
                           i];
                       else if(j==3) totalWeight[i]=
                           totalWeight[i]+91*real_ASC[j][
                           i];
                       else if(j==4) totalWeight[i]=
                           totalWeight[i]+126*real_ASC[j
                           ][i];
                       else if(j==6||j==7||j==8)
                           totalWeight[i]=totalWeight[i
                           ]+10*real_ASC[j][i];
                       else if(j==11||j==12) totalWeight
                           [i]=totalWeight[i]+20*real_ASC
                           [j][i];
              }
     }

     for(int j=part*(nYSize/ceil((float)nprocs/
         feature_num)+1);j<(part+1)*(nYSize/ceil((float
         )nprocs/feature_num)+1);j++){
              if(j>=nYSize) break;
              GDALRasterIO( hBand, GF_Read, 0, j,
                  nXSize, 1,
                       land, nXSize, 1, GDT_Byte,
                       0, 0 );
              GDALRasterIO( hBand2, GF_Read, 0, j,
                  nXSize, 1,
                       remainder, nXSize, 1, GDT_Byte,
                       0, 0 );
              GDALRasterIO( hBand3, GF_Read, 0, j,
                  nXSize, 1,
                     quotient, nXSize, 1, GDT_Byte,
                       0, 0 );
              for(int i=0;i<nXSize;i++){
                       int bg_id=(quotient[i]*256+
                           remainder[i]);
                       if(bg_id!=0&&totalWeight[bg_id
                           -1]!=0){
                                int intLand= land[i];
                                if(relation[intLand]==1)
```

```
                                        weight=14;
                                    else if(relation[intLand
                                        ]==2) weight=49;
                                    else if(relation[intLand
                                        ]==3) weight=91;
                                    else if(relation[intLand
                                        ]==4) weight=126;
                                    else if(relation[intLand
                                        ]==6||relation[intLand
                                        ]==7||relation[intLand
                                        ]==8) weight=10;
                                    else if(relation[intLand
                                        ]==11||relation[
                                        intLand]==12) weight
                                        =20;
                                    else weight=0;
                                    float temp=(float)(
                                        bg_pop_one[bg_id-1])/
                                        totalWeight[bg_id-1]*
                                        weight;
                                    result[i]=(int)(sqrt(temp
                                        )*100)-1;
                                    if(result[i]==255) result
                                        [i]=0;
                            }
                            else result[i]=0;
                    }
                    GDALRasterIO( hBand4, GF_Write, 0, j,
                        nXSize, 1,
                            result, nXSize, 1, GDT_Byte,
                            0, 0);
            }
            t_end = time(NULL) ;
            printf("time: %.0f s\n", difftime(t_end,t_start))
                ;

            free2dint(&bg_pop_all);
            free2dint(&ASC);
            free2dint(&real_ASC);

            MPI_Finalize();
            return 0;
}
```

E R Code

I have written two R functions. The first one is responsible for extracting ACS Summary Table data. The second one is responsible for doing regression estimation.

```
ACS<-function(z){
        A<-read.csv('/home/zzhang22/nlcd/Input/ACS/aff_
            download-4/ACS_14_5YR_B02001_with_ann.csv')
        myvars<-c("GEO.id2","HD01_VD01","HD01_VD02","HD01
            _VD03","HD01_VD05")
        A<-A[myvars]
        B<-as.data.frame(z)
        B$GROUPID<-seq.int(nrow(B))
        C<-merge(x=A,y=B,by.x="GEO.id2",by.y="z")
        myvars2<-c("GROUPID","HD01_VD01","HD01_VD02","
            HD01_VD03","HD01_VD05")
        C<-C[myvars2]
        attach(C)
        C<-C[order(GROUPID),]
        detach(C)
        myvars3<-c("HD01_VD01","HD01_VD02","HD01_VD03","
            HD01_VD05")
        C<-C[myvars3]
        for(i in 1:4)
                C[,i]<-as.numeric(as.character(C[,i]))
        C<-as.matrix(C)
        D<-as.vector(C)
        return(D);
}

coeff<-function(pop,land){
        library(nnls)
        A<-matrix(pop,nrow=4125,ncol=4)
        B<-matrix(land,nrow=15,ncol=4125)
        B<-t(B)
        A1=A[,1]
        A2=A[,2]
        A3=A[,3]
        A4=A[,4]
        b1=nnls(B,A1)
        b2=nnls(B,A2)
        b3=nnls(B,A3)
        b4=nnls(B,A4)
        b=c(b1$x,b2$x,b3$x,b4$x)
        return(b)
}
```