Quadtree/Octree Distribution of Data using p4est

Ana C. Perez-Gea

May 11, 2017

In this project we see the advantages of using mesh refinement to process massive amounts of data. When there are a large number of data points that a computer needs to process, it can be problematic when the computer does not have enough memory to load and process all data points at the same time. In this project I simulate a large number of data points and use quadtrees or octrees to partition the data and have different processors working with different parts of the data. In particular, I use the package p4est in the C programming language to do the mesh refinement and load data into different processors.

1 Data

I simulated the data using a file of the NYU logo as seen in Figure 1. I will cover the 2-dimensional case in detail and then extend the explanations to the 3-dimensional case. Inside of the torch, many points are generated and outside just a few. The x-coordinates of the figure are stored in one vector and the y-coordinates in another. At each image point, we repeat the storing of the row (x-coordinate) and column (y-coordinate) location of that data point but with random noise added each time. This is that for image row i we loop for $0 \leq j < m_i$ such that

$$x_{i,j} = r_i + u_{i,j}$$

where $u_{i,j}$ is a uniform random variable between 0 and 1 and m_i is the number of repetitions for that image point. If we are at an inside point we repeat more times so m_i will be larger than if we are at an outside point.

2 p4est

The package p4est does all the handling of the splitting of the quadtrees and octrees for us. I took the example file p4est_step1.c and modified it for my purposes. In particular, it has a function that defines the condition for splitting. This function is a boolean that, if we have more than a certain number of points in a quadrant, it will return 1.

3 Quadtree

I first implemented the 2-dimensional case. In table 1 we can see the minimum and maximum row and column values as well as the total number of points generated for a single run. Since

Figure 1: Image for Data Simulation

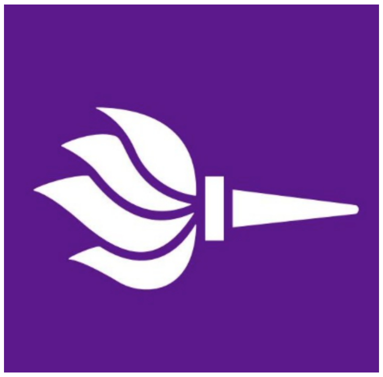


Table 1: Data

Min row:	0.000008
Max row:	139.999893
Min col:	4.048210
Max col:	155.932297
Total points:	869993

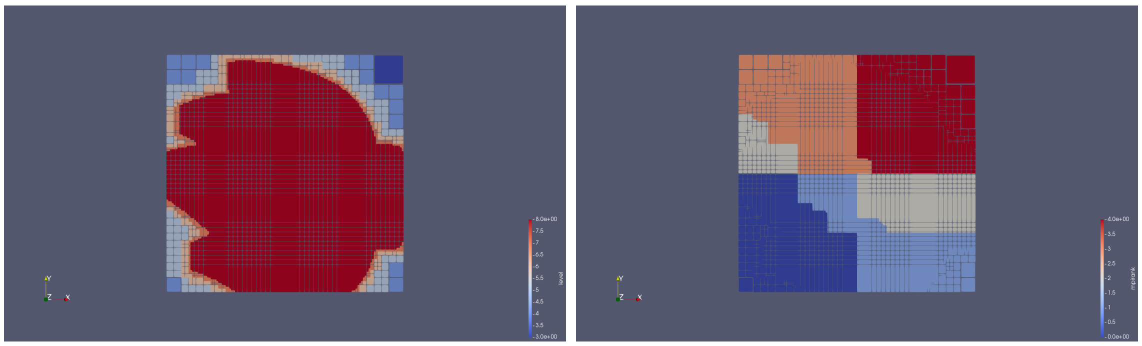
a positive noise was added to the row and column values, we can see that the values go a little above zero or the number of rows and columns. The minimum column value starts at a little over four because there are no points to be simulated in the first four columns.

I ran the code using 5 processors with MPI and it took about 35 seconds. Figure 2a shows the levels of partitioning. The red in the middle shows small areas and the blue outside shows large quadtrees. Figure 2b shows the points that are being processed by the different processors. Each color represents a different processor, an MPI rank.

```
mpirun -np 5 p4est_data
```

Figure 2: Quadtree

(a) Levels (b) Rank



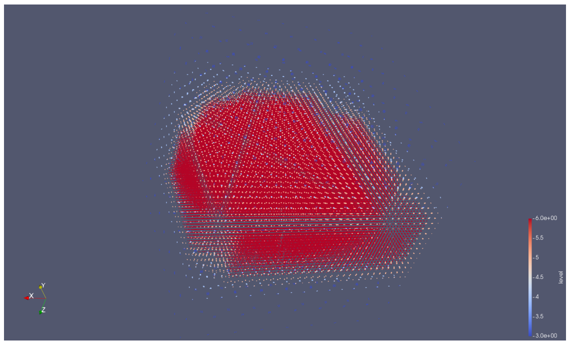
4 Octree

Octrees are 3-dimensional representation of the quadtrees. In my case, I repeated the data in the z-direction 5/8 of the way up and 5/8 down. I repeated the above run with 5 MPI ranks for this 3-dimensional case. Now Figure 3 shows the data points themselves to be able to view the shape of the volume. We can see in Figure 3a the levels of refinement by color. The red in the center is fine grid and the blue in the outside is coarse. In Figure 3b the colors represent a different rank, where we can see each processor got about the same number of points.

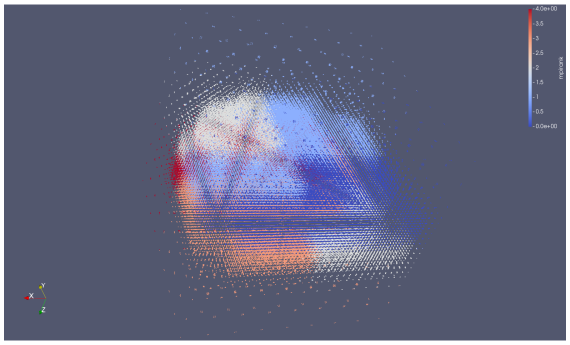
```
mpirun -np 5 p8est_data
```

Figure 3: Octree

(a) Levels



(b) Rank



5 C Code

```
/*
Datatree
Ana C. Perez-Gea
Code adapted from p4est example file p4est_step1.c
*/
#ifndef P4_TO_P8
#include <p4est_vtk.h>
#else
#include <p8est_vtk.h>
#endif

#ifdef P4_TO_P8
static const p4est_qcoord_t eighth = P4EST_QUADRANT_LEN (3);
#endif

/* Parameters to adjust */
#define max_pow0 7 // no deeper than this level (power of 2)
int max_size = 500; // keep splitting if we have more than this many data points
int data_base = 1000; // data point power base (for simulating data)
int max_data = 10000000; //max points to simulate, adjust so it covers all your points

/* Parameters not to touch */
float *array, *cols, *rows, max_col, min_col, max_row, min_row;
int count = 0; // actual number read from file
#define max_quad0 (1<<max_pow0)
```

```c
static const int max_pow = max_pow0;
static const int max_quad = max_quad0;

/** Callback function to decide on refinement. */
static int refine_fn (p4est_t * p4est, p4est_topidx_t which_tree, p4est_quadrant_t *
    quadrant){
  int tilelen, inside, ql;
  int i, j, offsi, offsj;
  float dx, dy, qx, qy;

  /* We do not want to refine deeper than a given maximum level. */
  if (quadrant->level > max_pow) {
    return 0;
  }
  #ifdef P4_TO_P8
  /* In 3D we extrude the 2D image in the z direction between [3/8, 5/8]. */
  if (quadrant->level >= 3 && (quadrant->z < 3 * eighth || quadrant->z >= 5 * eighth)) {
    return 0;
  }
  #endif

  inside = 0;
  qx = quadrant->x;
  qy = quadrant->y;
  ql = quadrant->level;
  offsi = quadrant->x / P4EST_QUADRANT_LEN (max_pow);
  offsj = quadrant->y / P4EST_QUADRANT_LEN (max_pow);
  qx = (float)offsi / (float)max_quad;
  qy = (float)offsj / (float)max_quad;
  for(i=0; i<count; i++){
    dx = (rows[i]-min_row) / (max_row-min_row);
    dy = (cols[i]-min_col) / (max_col-min_col);
    //printf("dx in rows[%d]=%f\n", i, dx);
    //printf("dy in cols[%d]=%f\n", i, dy);
    if(qx<dx && dx<(qx+1.0/(float)(ql+1))){
      //printf("condition in rows < %f + %f\n", qx,1.0/(float)(ql+1));
      if(qy<dy && dy<(qy+1.0/(float)(ql+1))){
        //printf("condition in cols < %f + %f\n", qy,1.0/(float)(ql+1));
        inside++;
        if(inside > max_size){
          //printf("WIN rows[%d]=%f\n", i, rows[i]);
          //printf("WIN cols[%d]=%f\n", i, cols[i]);
          return 1;
        }
      }
    }
  }
  return 0;
}

/* Main function creates a connectivity and forest, refines it, and writes a VTK file. */
int main (int argc, char **argv) {
  int                 mpiret, i;
  int                 recursive, partforcoarsen, balance;
  sc_MPI_Comm         mpicomm;
  p4est_t            *p4est;
  p4est_connectivity_t *conn;
  double t1, t2;

  /* Open file we will use to simulate data points */
  FILE *fp;
  int row,col,inc;
  float data, u1, u2;
  char ch;
  //array = (float*)malloc(sizeof(float)*ple*ple);
  cols = (float*)malloc(sizeof(float)*max_data);
  rows = (float*)malloc(sizeof(float)*max_data);
  fp = fopen("example/steps/logo.txt","r");
```

```
 row = col = 0;
 while(EOF!=(inc=fscanf(fp,"%f%c", &data, &ch)) && inc == 2){
   //array[count] = data;
   for(i=0;i<(int)pow(data_base,(0.6549-data));i++){
    u1 = (float)rand() / (float)RAND_MAX ;
    rows[count] = row+u1;
    if(rows[count]<min_row || 0==count){
     min_row = rows[count];
    }
    if(rows[count]>max_row || 0==count){
     max_row = rows[count];
    }
    u2 = (float)rand() / (float)RAND_MAX ;
    cols[count] = col+u2;
    if(cols[count]<min_col || 0==count){
     min_col = cols[count];
    }
    if(cols[count]>max_col || 0==count){
     max_col = cols[count];
    }
    ++count;
    if(count==max_data){
     goto exit;
    }
  }
  ++col;
   if(ch == '\n'){
     ++row;
     col = 0;
   } else if(ch != ','){
     fprintf(stderr, "Different␣separator␣(%c)␣of␣row␣at␣%d␣\n", ch, row);
     goto exit;
   }
 }
 exit:
  fclose(fp);

 printf("Min␣row:␣%f\n", min_row);
 printf("Max␣row:␣%f\n", max_row);
 printf("Min␣col:␣%f\n", min_col);
 printf("Max␣col:␣%f\n", max_col);
 printf("Total␣points:␣%d\n", count);

 /* Initialize MPI */
 mpiret = sc_MPI_Init (&argc, &argv);
 SC_CHECK_MPI (mpiret);
 mpicomm = sc_MPI_COMM_WORLD;
 // Get the rank of the process
 int world_rank;
 sc_MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
 t1 = sc_MPI_Wtime();

 /* Store the MPI rank as a static variable so subsequent global p4est log messages are
   only issued from processor zero.  */
 sc_init (mpicomm, 1, 1, NULL, SC_LP_ESSENTIAL);
 p4est_init (NULL, SC_LP_PRODUCTION);
 P4EST_GLOBAL_PRODUCTIONF
  ("This␣is␣the␣p4est␣%dD␣data\n", P4EST_DIM);

 /* Create a forest that consists of just one quadtree/octree. */
#ifndef P4_TO_P8
 conn = p4est_connectivity_new_unitsquare ();
#else
 conn = p8est_connectivity_new_unitcube ();
#endif

 /* Create a forest that is not refined; it consists of the root octant. */
 p4est = p4est_new (mpicomm, conn, 0, NULL, NULL);
```

```
  /* Refine the forest recursively in parallel. */
  recursive = 1;
  p4est_refine (p4est, recursive, refine_fn, NULL);

  /* Partition: The quadrants are redistributed for equal element count. */
  partforcoarsen = 1;
  p4est_partition (p4est, partforcoarsen, NULL);

  /* If we call the 2:1 balance we ensure that neighbors do not differ in size by more than
      a factor of 2.  */
  balance = 1;
  if (balance) {
    p4est_balance (p4est, P4EST_CONNECT_FACE, NULL);
    p4est_partition (p4est, partforcoarsen, NULL);
  }
  printf("Hi␣this␣is␣processor␣%d\n", world_rank);
  /* Write the forest to disk for visualization, one file per processor. */
  p4est_vtk_write_file (p4est, NULL, P4EST_STRING "_data");

  /* Destroy the p4est and the connectivity structure. */
  p4est_destroy (p4est);
  p4est_connectivity_destroy (conn);

  /* Verify that allocations internal to p4est and sc do not leak memory. */
  sc_finalize ();

  t2 = sc_MPI_Wtime();
  printf( "Elapsed␣time␣is␣%f\n", t2 - t1 );

  /* This is standard MPI programs.  Without --enable-mpi, this is a dummy. */
  mpiret = sc_MPI_Finalize ();
  SC_CHECK_MPI (mpiret);
  return 0;
}
```