Quadtree/Octree Distribution of Data using p4est

Ana C. Perez-Gea

May 22, 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 differents 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 C Code

```
/*
Datatree
Ana C. Perez-Gea
Code adapted from p4est example file p4est_step1.c
*/

/* p4est_to_p8est.h #define's the 2D names to the 3D names such that most code
 * only needs to be written once. */
#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

float *array, *cols, *rows, max_col, min_col, max_row, min_row;
int count = 0; // actual number read from file
int max_size = 15; // keep splitting if we have more than this many data points
int dmult = 1000; // data point multiplier (for simulating data)
int max_data = 10000000; //max points to read from file
int max_level = 10; // no deeper than this level

/** Callback function to decide on refinement.
 *
 * Refinement and coarsening is controlled by callback functions.
 * This function is called for every processor-local quadrant in order; its
 * return value is understood as a boolean refinement flag.
 */
static int refine_fn (p4est_t * p4est, p4est_topidx_t which_tree, p4est_quadrant_t *
    quadrant)
{
  int                 tilelen, inside;
```

```
  int                 offsi, offsj;
  int                 i, j;
  float dx, dy;

  /* The connectivity chosen in main () only consists of one tree. */
  P4EST_ASSERT (which_tree == 0);

  /* We do not want to refine deeper than a given maximum level. */
  if (quadrant->level > max_level) {
    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;
  //printf("quadrant level:%d\n", quadrant->level);
  //printf("quadrant x:%f\n", quadrant->x);
  //printf("quadrant y:%f\n", quadrant->y);
  for(i=0; i<count; i++){
    //printf("rows[%d]=%f\n", i, rows[i]);
    //printf("cols[%d]=%f\n", i, cols[i]);
    //printf("cond %f\n", (max_row-min_row)/(float)(quadrant->level+1));
    dx = (rows[i]-min_row) / (max_row-min_row);
    dy = (cols[i]-min_col) / (max_col-min_col);
    if(quadrant->x<dx && dx<quadrant->x+1.0/(float)(quadrant->level+1)){
      if(quadrant->y<dy && dy<quadrant->y+1.0/(float)(quadrant->level+1)){
        inside++;
        if(inside > max_size){
          return 1;
        }
      }
    }
  }
  return 0;
}

/** The main function of the step1 example program.
 *
 * It 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;

  /* 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)(data*dmult);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("Data␣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);

  /* 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 ();

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