I found that the OpenMP parallel for loop was faster in all cases than the OpenMP tasks. For the parallel for, going from 1 to 2 to 4 threads all sped up the running time. Only the final step up to 8 threads decreased the running time (but only by one second. On the other hand, each increase of threads on the OpenMP tasks code resulted in an increased running time. I did not expect such a drastic increase in running time for the tasks code, but I guess the added overhead outweighed the potential benefits.

```
/* ISABEL HILTON
THREADS    TIME
2          34.09 s
4          50.82 s
8          85.43 s
*/

/*
  This program is an adaptation of the Mandelbrot program
  from the Programming Rosetta Stone, see
  http://rosettacode.org/wiki/Mandelbrot_set

  Compile the program with:

  gcc -o mandelbrot -O4 mandelbrot.c

  Usage:

  ./mandelbrot <xmin> <xmax> <ymin> <ymax> <maxiter> <xres> <out.ppm>

  Example:

  ./mandelbrot 0.27085 0.27100 0.004640 0.004810 1000 1024 pic.ppm

  The interior of Mandelbrot set is black, the levels are gray.
  If you have very many levels, the picture is likely going to be quite
  dark. You can postprocess it to fix the palette. For instance,
  with ImageMagick you can do (assuming the picture was saved to pic.ppm):

  convert -normalize pic.ppm pic.png

  The resulting pic.png is still gray, but the levels will be nicer. You
  can also add colors, for instance:

  convert -negate -normalize -fill blue -tint 100 pic.ppm pic.png

  See http://www.imagemagick.org/Usage/color_mods/ for what ImageMagick
  can do. It can do a lot.
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <stdint.h>
#include <sys/time.h>

#define PIX_NUM 6

/* Return the current time in seconds, using a double precision number.       */
double when() {
    struct timeval tp;
    gettimeofday(&tp, NULL);
    return ((double) tp.tv_sec + (double) tp.tv_usec * 1e-6);
}

int compute_offset(int x, int y, int z, int width) {
  return ((x * PIX_NUM) + (y * width * PIX_NUM) + z);
}

void write(char *data, char *dest, int numItems, int startIndex) {
  for (unsigned int i = 0; i < numItems; i++) {
    dest[i + startIndex] = data[i];
  }
}

int main(int argc, char* argv[])
{
  /* Parse the command line arguments. */
  if (argc != 8) {
    printf("Usage:   %s <xmin> <xmax> <ymin> <ymax> <maxiter> <xres> <out.ppm>\n",
argv[0]);
    printf("Example: %s 0.27085 0.27100 0.004640 0.004810 1000 1024 pic.ppm\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  /* The window in the plane. */
  const double xmin = atof(argv[1]);
  const double xmax = atof(argv[2]);
  const double ymin = atof(argv[3]);
  const double ymax = atof(argv[4]);

  /* Maximum number of iterations, at most 65535. */
  const uint16_t maxiter = (unsigned short)atoi(argv[5]);

  /* Image size, width is given, height is computed. */
  const int xres = atoi(argv[6]);
  const int yres = (xres*(ymax-ymin))/(xmax-xmin);

  /* The output file name */
  const char* filename = argv[7];

  /* Open the file and write the header. */
  FILE * fp = fopen(filename,"wb");
  char *comment="# Mandelbrot set";/* comment should start with # */

  /*write ASCII header to the file*/
  fprintf(fp,
          "P6\n# Mandelbrot, xmin=%lf, xmax=%lf, ymin=%lf, ymax=%lf, maxiter=%d\n%d\n%d\n
%d\n",
          xmin, xmax, ymin, ymax, maxiter, xres, yres, (maxiter < 256 ? 256 : maxiter));

  int array_size = xres * yres * PIX_NUM;
  unsigned char *pixels = (unsigned char *) malloc(array_size * sizeof(unsigned char));
  int index = 0;

  /* Precompute pixel width and height. */
  double dx=(xmax-xmin)/xres;
  double dy=(ymax-ymin)/yres;

  double x, y; /* Coordinates of the current point in the complex plane. */
  double u, v; /* Coordinates of the iterated point. */
  int i,j; /* Pixel counters */
  int k; /* Iteration counter */
  double start = when();
  for (j = 0; j < yres; j++) {
    y = ymax - j * dy;
#pragma omp parallel firstprivate(x)
{
#pragma omp single private(i)
{
    for(i = 0; i < xres; i++) {
      double u = 0.0;
      double v= 0.0;
      double u2 = u * u;
      double v2 = v*v;
      x = xmin + i * dx;

      /* iterate the point */
#pragma omp task private(k)
{
      for (k = 1; k < maxiter && (u2 + v2 < 4.0); k++) {
            v = 2 * u * v + y;
            u = u2 - v2 + x;
            u2 = u * u;
            v2 = v * v;
      };

      /* compute  pixel color and write it to file */
      int offset = compute_offset(i, j, 0, xres);
      if (k >= maxiter) {
        /* interior */
        unsigned char black[] = {0, 0, 0, 0, 0, 0};
        //fwrite (black, 6, 1, fp);
        write(black, pixels, 6, offset);
      }
      else {
        /* exterior */
        unsigned char color[6];
        color[0] = k >> 8;
        color[1] = k & 255;
        color[2] = k >> 8;
        color[3] = k & 255;
        color[4] = k >> 8;
        color[5] = k & 255;
        //fwrite(color, 6, 1, fp);
        write(color, pixels, 6, offset);
      };
}
    }
}
}
}
  double end = when();
  double man_time = end - start;
  printf("TIME = %f\n", man_time);

  fwrite(pixels, 1, array_size, fp);

  fclose(fp);
  return 0;
}
```