```
/*correr el programa
gcc -fopenmp nombre.c
a.exe >> prueba.txt
*/

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <string.h>
#define NUM_THREADS 200 // 4 o 20

int main(){
  char image_name[] = "punk";
  int input;
  printf("Ingrese 1 para invertir horizontal, 2 para vertical o 3 para ambos: ");
  scanf("%d", &input);

  FILE *image, *outputImage, *lecturas;
  image = fopen(strcat(image_name,".bmp"),"rb");         //Imagen original a transformar
  if (input == 1)
    outputImage = fopen(strcat(image_name,"_horizontal.bmp"),"wb");    //Imagen transformada
  else if (input == 2)
    outputImage = fopen(strcat(image_name,"_vertical.bmp"),"wb");
  else
    outputImage = fopen(strcat(image_name,"_ambos.bmp"),"wb");
  unsigned char r, g, b;              //Pixel
  unsigned char original_image[54];

  for(int i=0; i<54; i++) {
    //fputc(fgetc(image), outputImage);
    original_image[i] = fgetc(image);
    fputc(original_image[i], outputImage);   //Copia cabecera a nueva imagen
  }

  long ancho = (long)original_image[20]*65536 + (long)original_image[19]*256 +(long)
original_image[18];
  long alto = (long)original_image[24]*65536 + (long)original_image[23]*256 +(long) original_image[22];
  int padding =ancho%4;

  printf("Padding= %d\n",padding);

  unsigned char* image_input = (unsigned char*) malloc((ancho*alto*3+alto*padding)*sizeof(unsigned
char));
  unsigned char* image_output = (unsigned char*) malloc(ancho*alto*3*sizeof(unsigned char));

  printf("Ancho= %ld\nAlto= %ld\n", ancho, alto);

  omp_set_num_threads(NUM_THREADS);
  int n = (3*(int)alto*(int)(ancho)) + (padding*(int)alto);

  printf("n= %d\n", n);


  const double startTime = omp_get_wtime();
  #pragma omp parallel
    #pragma omp single
      for(int i = 0; i<n; i++)
        *(image_input + i) = fgetc(image);

    if(input == 1){
        #pragma for collapse(2)
          for(int k = 0; k < alto; k++)
            for(int j = 0; j < ancho; j++){
              int pixel = *(image_input + 3*k*ancho + k*padding + 3*j +2)*0.21+ *(image_input +
3*k*ancho + k*padding + 3*j + 1)*0.72 + *(image_input + 3*k*ancho + k*padding + 3*j)*0.07;
              *(image_output + 3*k*ancho + k*padding + 3*(ancho-1) - 3*j) = pixel;
              *(image_output + 3*k*ancho + k*padding + 3*(ancho-1) - 3*j + 1) = pixel;
              *(image_output + 3*k*ancho + k*padding + 3*(ancho-1) - 3*j + 2) = pixel;
            }
        #pragma for collapse(2)
          for(int k = 0; k< alto; k++)
            for(int l=0; l<padding; l++)
              *(image_output + (3*k*ancho) + (k*padding)+l+(ancho*3)) = 0;
        #pragma omp single
          for(int i = 0; i<n; i++)
            fputc(*(image_output + i), outputImage);
        }
    else if(input == 2){
      #pragma for collapse(2)
        for(int k = 0; k < alto; k++)
          for(int j = 0; j < ancho; j++){
            int pixel = *(image_input + 3*k*ancho + k*padding + 3*j +2)*0.21+ *(image_input +
3*k*ancho + k*padding + 3*j + 1)*0.72 + *(image_input + 3*k*ancho + k*padding + 3*j)*0.07;
            *(image_output + 3*ancho*(alto-1) - 3*k*ancho + (alto-k-1)*padding + 3*j) = pixel;
            *(image_output + 3*ancho*(alto-1) - 3*k*ancho + (alto-k-1)*padding + 3*j + 1) = pixel;
            *(image_output + 3*ancho*(alto-1) - 3*k*ancho + (alto-k-1)*padding + 3*j + 2) = pixel;
          }
      #pragma for collapse(2)
        for(int k = 0; k< alto; k++)
          for(int l=0; l<padding; l++)
            *(image_output + (3*k*ancho) + (k*padding)+l+(ancho*3)) = 0;
      #pragma omp single
        for(int i = 0; i<n; i++)
          fputc(*(image_output + i), outputImage);
      }
    else{
      #pragma for collapse(2)
        for(int k = 0; k < alto; k++)
          for(int j = 0; j < ancho; j++){
            int pixel = *(image_input + 3*k*ancho + k*padding + 3*j +2)*0.21+ *(image_input + 3*k*ancho + k*padding + 3*j + 1)*0.72 + *(image_input + 3*k*ancho + k*padding + 3*j)*0.07;
            *(image_output + 3*k*ancho + k*padding + 3*j) = pixel;
            *(image_output + 3*k*ancho + k*padding + 3*j + 1) = pixel;
            *(image_output + 3*k*ancho + k*padding + 3*j + 2) = pixel;
          }
      #pragma for collapse(2)
        for(int k = 0; k< alto; k++)
          for(int l=0; l<padding; l++)
            *(image_output + (3*k*ancho) + (k*padding)+l+(ancho*3)) = 0;
      #pragma omp single
        for(int i = 0; i<n; i++)
          fputc(*(image_output + n -i), outputImage);
      }
  const double endTime = omp_get_wtime();
          printf("Ha demorado (%lf) segundos en ejecutar.\n", (endTime - startTime));
  fclose(image);
  fclose(outputImage);
  return 0;

}
```