#include &lt;stdio.h&gt;

#include &lt;gd.h&gt;

#include &lt;string.h&gt;

#include &lt;omp.h&gt;

int main(int argc, char \*argv[]) {

if (argc &lt; 4) {

printf(&quot;Usage: ./negative input.png output.png num\_threads\n&quot;);

return 1;

}

char \*input\_file = argv[1];

char \*output\_file = argv[2];

int num\_threads = atoi(argv[3]);

int color, x, y, i;

int red, green, blue;

FILE \*fp;

if((fp = fopen(input\_file, &quot;r&quot;)) == NULL) {

printf(&quot;Error opening file %s\n&quot;, input\_file);

return 1;

}

gdImagePtr img = gdImageCreateFromPng(fp);

int width = gdImageSX(img);

int height = gdImageSY(img);

double t1 = omp\_get\_wtime();

#pragma omp parallel for private(y, color, red, green, blue) num\_threads(num\_threads)

for(x=0; x&lt;width; x++) {

for(y=0; y&lt;height; y++) {

color = x + 0;

color = gdImageGetPixel(img, x, y);

red = 255 - gdImageRed(img, color);

green = 255 - gdImageGreen(img, color);

blue = 255 - gdImageBlue(img, color);

color = gdImageColorAllocate(img, red, green, blue);

gdImageSetPixel(img, x, y, color);

}

}

double t2 = omp\_get\_wtime();

if((fp=fopen(output\_file, &quot;w&quot;)) == NULL) {

printf(&quot;Error opening output file %s\n&quot;, output\_file);

return 1;

}

gdImagePng(img, fp);

gdImageDestroy(img);

fclose(fp);

printf(&quot;File Size: %dx%d\n&quot;, width, height);

printf(&quot;Time Taken: %.3lfms\n&quot;,(t2 - t1) \* 1000);

return 0;

}

Using Critical Section

#include &lt;stdio.h&gt;

#include &lt;gd.h&gt;

#include &lt;string.h&gt;

#include &lt;omp.h&gt;

int main(int argc, char \*argv[]) {

if (argc &lt; 4) {

printf(&quot;Usage: ./negative input.png output.png num\_threads\n&quot;);

return 1;

}

char \*input\_file = argv[1];

char \*output\_file = argv[2];

int num\_threads = atoi(argv[3]);

int color, x, y, i;

int red, green, blue;

FILE \*fp;

if((fp = fopen(input\_file, &quot;r&quot;)) == NULL) {

printf(&quot;Error opening file %s\n&quot;, input\_file);

return 1;

}

gdImagePtr img = gdImageCreateFromPng(fp);

int width = gdImageSX(img);

int height = gdImageSY(img);

double t1 = omp\_get\_wtime();

#pragma omp parallel for private(y, color, red, green, blue) num\_threads(num\_threads)

for(x=0; x&lt;width; x++) {

#pragma omp critical

{

for(y=0; y&lt;height; y++) {

color = x + 0;

color = gdImageGetPixel(img, x, y);

red = 255 - gdImageRed(img, color);

green = 255 - gdImageGreen(img, color);

blue = 255 - gdImageBlue(img, color);

color = gdImageColorAllocate(img, red, green, blue);

gdImageSetPixel(img, x, y, color);

}

}

}

double t2 = omp\_get\_wtime();

if((fp=fopen(output\_file, &quot;w&quot;)) == NULL) {

printf(&quot;Error opening output file %s\n&quot;, output\_file);

return 1;

}

gdImagePng(img, fp);

gdImageDestroy(img);

fclose(fp);

printf(&quot;File Size: %dx%d\n&quot;, width, height);

printf(&quot;Time Taken: %.3lfms\n&quot;,(t2 - t1) \* 1000);

return 0;

}