High Performance Computing
Assignment 4

Jahnavi Suthar(201301414) Deeksha Koul(201301435)

Hardware Details

CPU Model:INTEL(R) Core i5-4590
No. of cores: 16
Memory:7.6 GiB
Compiler:gcc
Optimization flags if used: None
Precision: double

IMAGE - WARPING

An image warp is a spatial transformation of an image, and is commonly found in photo-editing software as well as registration algorithms. In this example, we apply a "twist" transformation of the form

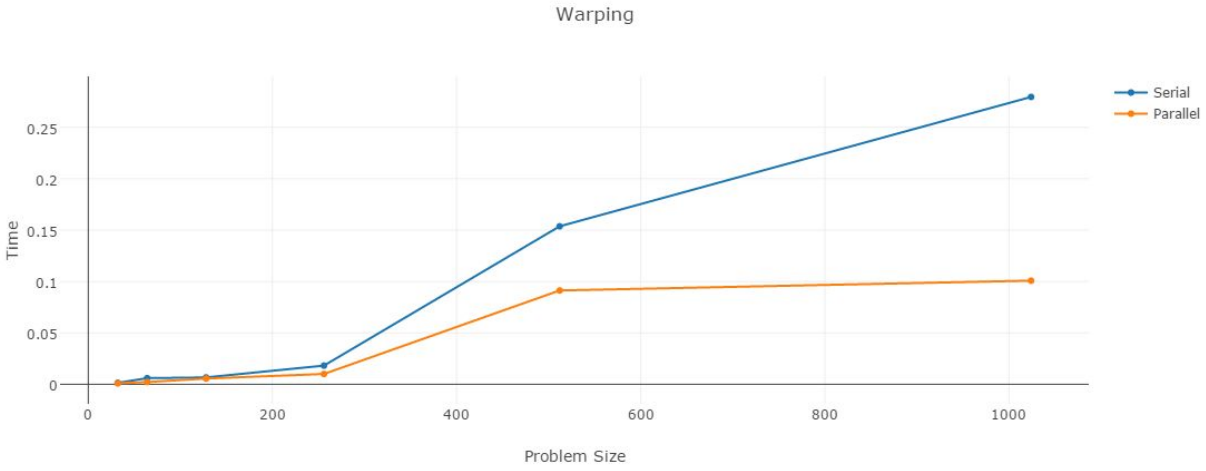
$x' = (x \square c_x)\cos q + (y \square c_y) \sin q + c_x$

$y' = \square(y \square c_y) \sin q + (y \square c_y)\cos q + c_y$ (1)

where $[cx;cy]^T$ is the rotation center, q = r=2 is a rotation angle that increases with radius r, $[x;y]^T$ is the point before transformation, and $[x';y']^T$ is the point after transformation. In the code, both the original image and transformed image are shared variables, along with the width and height of the image. The code loops over the pixels $[x',y']^T$ of the transformed image, mapping them back to the original image using the inverse transform of Equation 1. In the original image, the pixels are bilinearly interpolated. Each thread requires its own variables for x, y, index, radius, theta, xp, and yp. Since these variables are initialized within the parallelized code, we use the shared clause.OpenMP will multithread the outer loop (over yp) using static scheduling. The code, very easily multi-threaded,achieves an excellent speedup when executed on multiple cores.

Complexity of the algorithm: O(h * w),

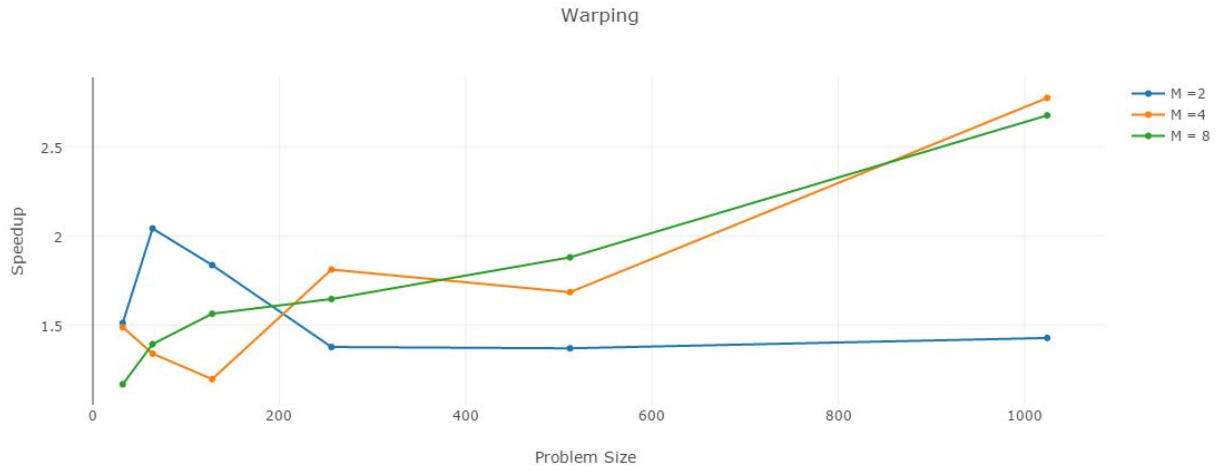
where h = height and w = width



Problem size vs Time

Observation:

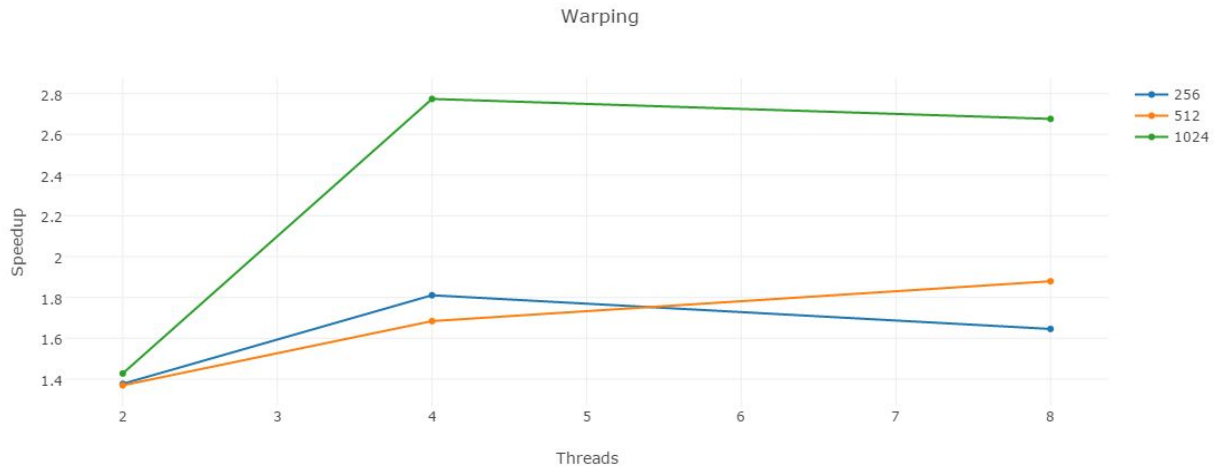
For all the problem sizes, parallel code takes lesser time than the serial code. But for the larger input sizes, it performs much better than the small input sizes.



Problem Size vs Speedup

Observation:

For problem size (the size of the full image matrix),that are small(less than 200) M= 2 number of cores performs better as there is no efficient parallelization required and as size grows due to concurrency the parallelization factor increases and thus M= 4 and M= 8 performs better than M= .After size increases from 1000 both M= 4 and M= 8 performs equally well tus increasing the speedup.



Number of threads vs Speedup

Observation:

For each core, as the problem size increases the speedup also tends to increase.For M= 4 , the problem size 256 works better than 512 this abnormal behaviour might be due to the space allocation that when we create and allocate image array.

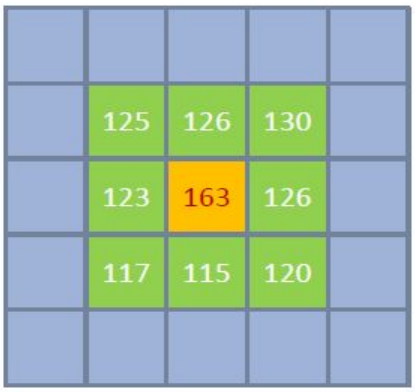
MEDIAN FILTERING

Median filtering is a commonly applied non-linear filtering technique that is particularly useful in removing speckle and salt and pepper noise [6]. Simply put, an image neighborhood surrounding each pixel is defined, and the median value of this neighborhood is calculated and is used to replace the original pixel in the output image:

Imed [x;y] = median (Iorig [i; j] ; i; j 2 nbor [x;y]) (3)

In this example we choose a square neighborhood around each pixel, defined using the halfwidth of the neighborhood, i.e., for a halfwidth of n, the number of pixels in the neighborhood would be $(2n+1)^2$ .At each pixel, the function GetNbors retrieves the neighbors; any neighbors that lie outside the image domain are assigned to be that of the nearest pixel within the image boundary. These neighbors are then sorted using the C sort function and the median selected.

- **Example: Median filter to soften colors in an image**



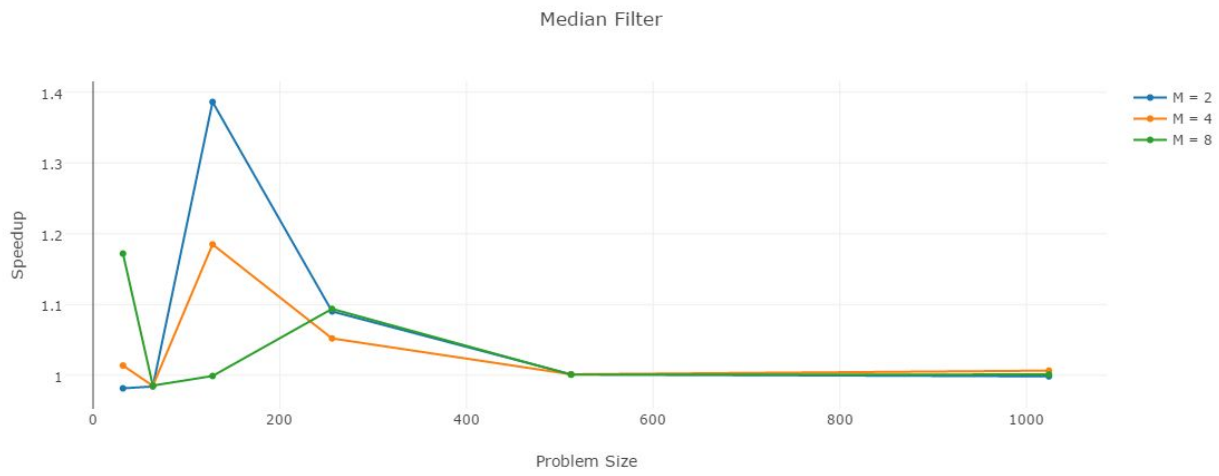
1. Get color values of neighboring pixels
   125, 126, 130, 123, 163, 126, 117, 115, 120
2. Sort color values (ascending)
   115, 117, 120, 123, 125, 126, 126, 130, 163
3. Choose new color value (median):
   115, 117, 120, 123, 125, 126, 126, 130, 163
4. Update color value

Complexity of the algorithm:

$O(h * w * \log(hw) * hw^2)$

where h = height, w = width of the image

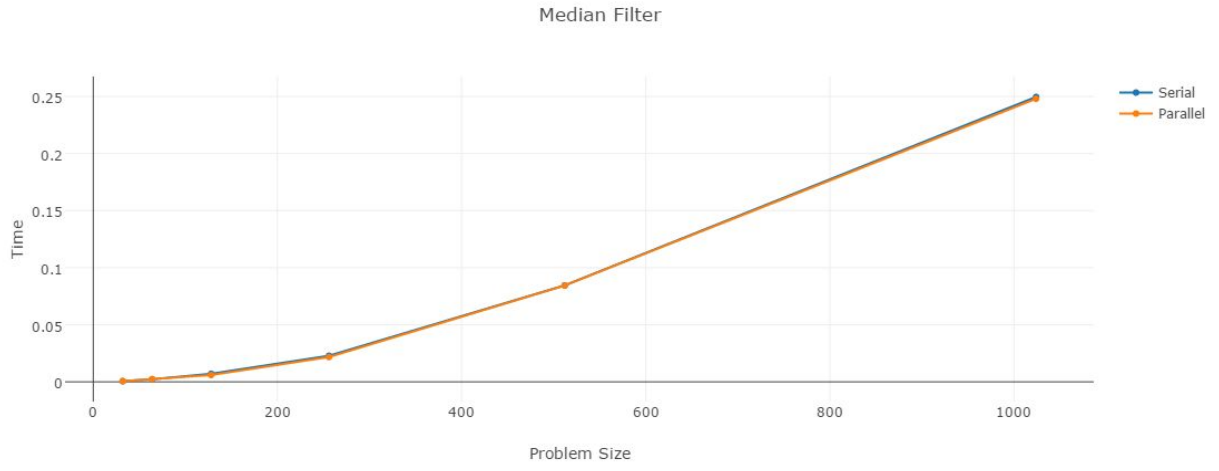
hw = halfwidth



Problem Size vs Speedup

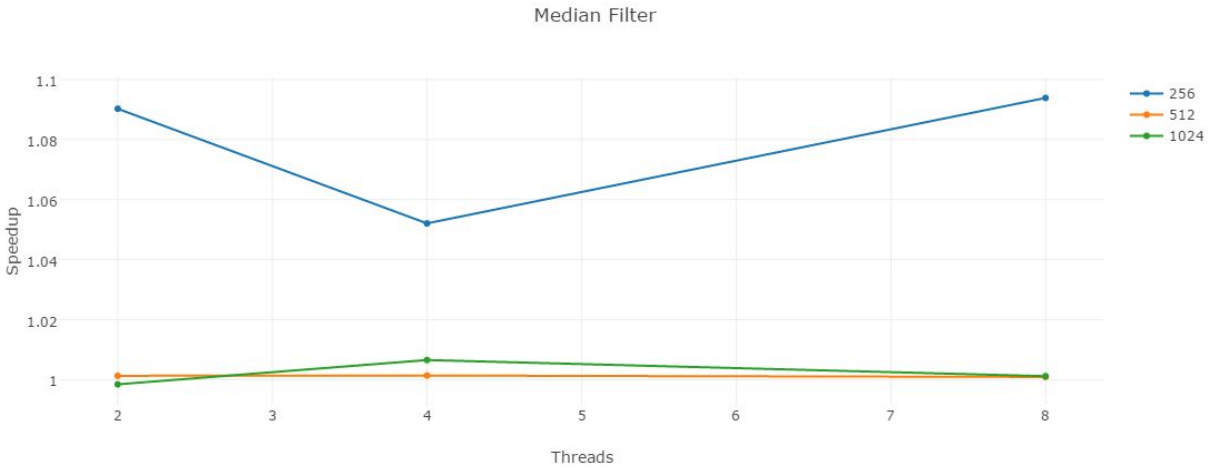
Observation:

For problem size less than 256 parallelization is not effective much but as the problem size increases the efficiency of parallel code increase and thus M= 4 and M= 8 outperforms M= 2.For greater prob sizes , the prob is efficient for parallel and thus speedup is nearly same for all the cores.



Observation:

Time taken for both the serial and parallel code is almost same, i.e. speedup is approximately 1.



Observation:

For each core , the speedup remains almost same i.e near to one but the speedup is somewhat greater for M= 2 as compared to M= 4 and M= 8 .As the number of cores increase , the speedup(although not much in difference as compared to previous)also increases.

Problems faced during parallelization:

1. Memory bound limitations: many a times the code wouldn't run stating segmentation fault(core dumped).
2. As we have allocated independent space for different arrays(that are large in size) time to time we need to free the space.
3. The image input used is primarily gray scale BMP format image that contains 8 bits per pixel .For other types of image inputs like jpeg,png the code becomes difficult to parallelize.