Assignment 3

201401222 : Vaibhav Patel

201401409 : Tanmay Patel

Hardware details: CPU model, memory information details, no of cores, compiler, optimization flags if used, precision used:

CPU modelIntel
® Core™ i54200U
CPU @ 3.30GHz × 4
Memory information7.7Gb
Graphics Intel
No. of cores4
OS type: 64bit
Compiler gcc
L1 Cache size : 128 KBytes, 8 way associative, 64 byte line size.
L2 Cache size : 512 KBytes, 8 way associative, 64 byte line size.
L3 Cache size : 3 MBytes, 12 way associative, 64 byte line size.

Question-1

Write a serial code for twist transformation/ image warping of an image. Write a parallel version of this using openMP. Change the image dimension (same image) and compare the serial vs. parallel implementation.

Complexity:

O(m*n)
Where
m = number of pixels in a row of given image
n = number pixels in a column

Possible Speedup:

Theoretically, It can go upto 4 (**Which is the number of cores**). Because after the profiling the part which can be parallelized is taking more than 99% of the time.

Profiling Information:

After profiling the function BilinearlyInterpolate took maximum time (approx. 90%). But Is not the right way. We want the main calculation function which is binary interpolate which is taking 100% time.

Optimization strategy:

We just need to use pragma omp directives to parallelize the outer loop. And we tried Dynamic and static scheduling which yielded nothing important. The catch was to avoid any variable shared between threads if it is not necessary.

Problems faced in parallelization:

The problem was straight forward. Nested loops and we just need to parallelize the outer one.

Input:

Image (Which we want to warp) (in ppm format)

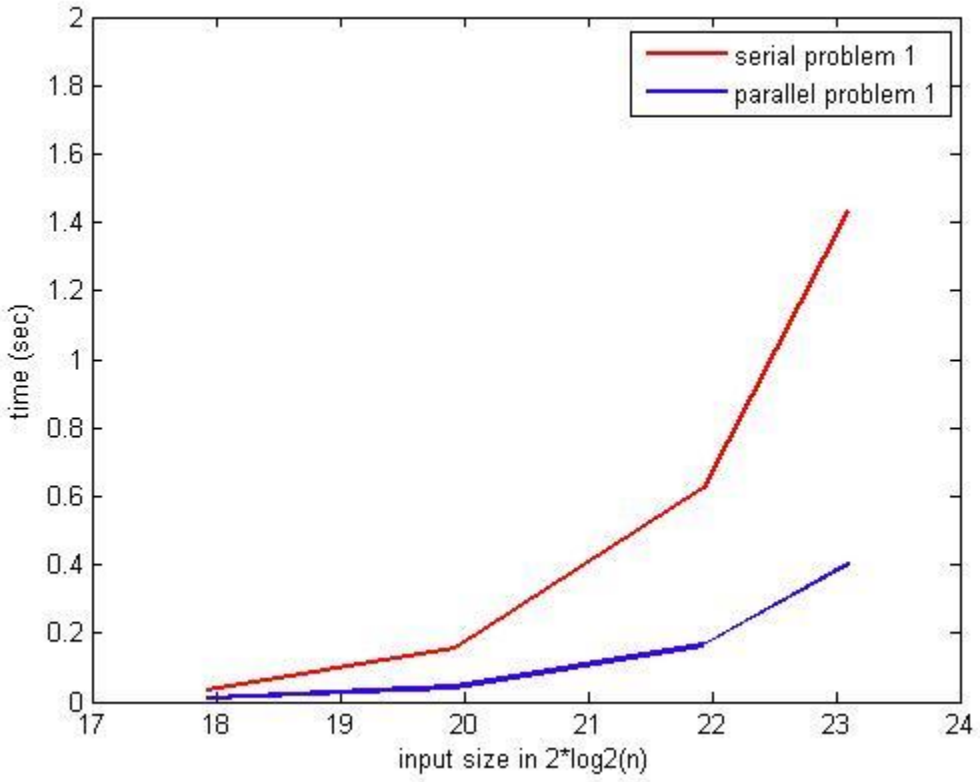
Output:

Warped Image

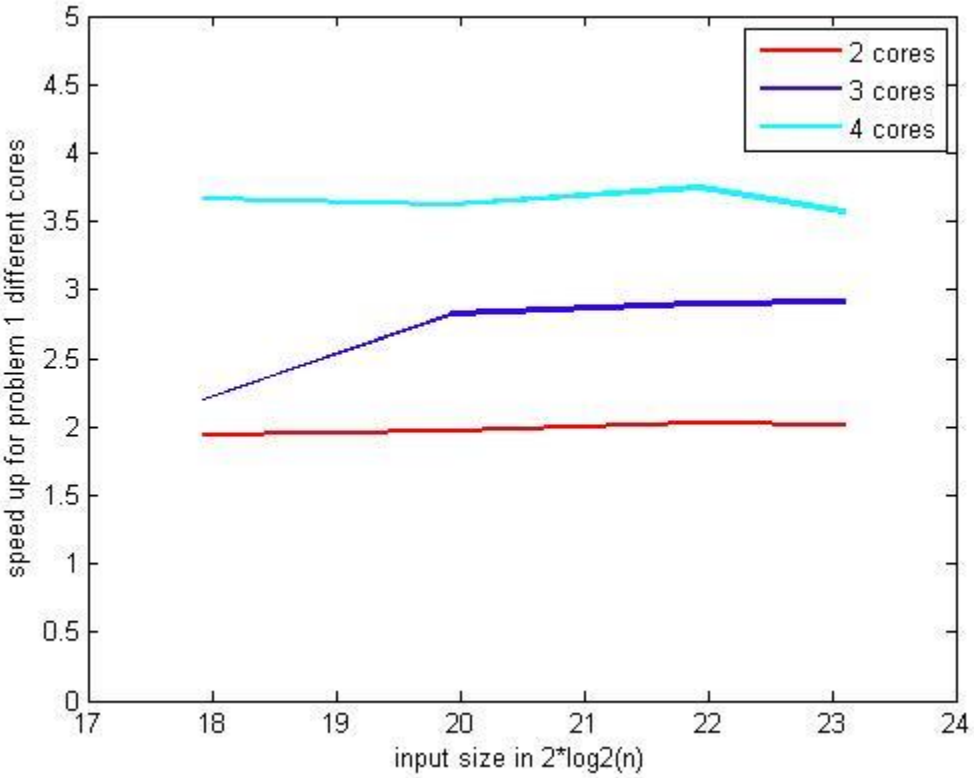
Parallel overhead time:

0.025390625 seconds

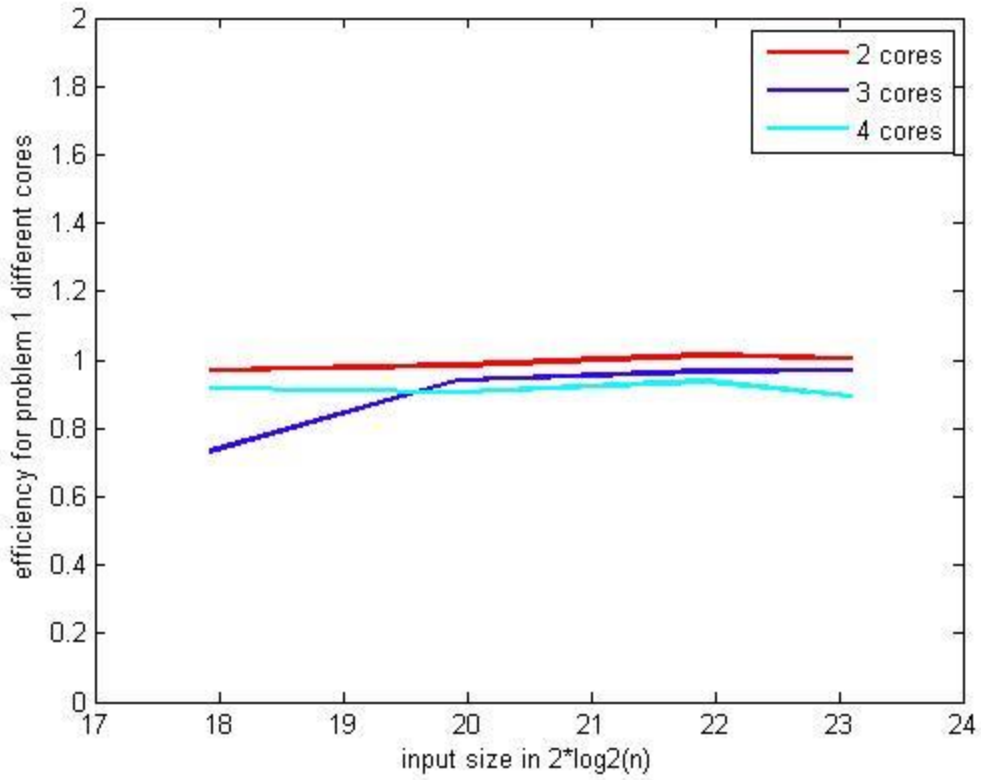
Problem Size vs Time (Serial, parallel):



Problem size vs. Speedup curve For Different Number of Threads:



Efficiency vs. Problem Size:



Efficiency for two threads is highest. Obvious.

Observations and comments:

1. Speed up increases with problem size. Just because the parallel overhead is less.
2. As we increase the number of cores it simply has to do more communication compared to the serial one. So the efficiency is decreasing over increase in number of cores.

Question 2:

Filtering: Consider an image neighborhood surrounding each pixel is defined, and the median value (filter in this case) of this neighborhood is calculated and is used to replace the original pixel in the output image by using the following filter: Imed [x,y] = median (Iorig [i, j] ; i, j belongs to nbor [x,y])

If you 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. Any neighbors that lie outside the image domain are assigned to be that of the nearest pixel within the image boundary for calculation of median.

Complexity:

$O((m*n*hw^2) * (hw^{2\log(hw^2)}))$

Where

m = number of pixels in a row of given image

n = number pixels in a column

hw= half-width

Possible Speedup:

Theoretically, It can go upto 4 (**Which is the number of cores**). Because after the profiling the part which can be parallelized is taking more than 99% of the time.

Profiling Information:

After profiling the function cmpfunc() took maximum time (approx. 60%). And the other time taking function is getneighbours().

Problems faced in parallelization:

The problem was straight forward. Nested loops and we just need to parallelize the outer one.

Input:

Image (Which we want to warp) (in ppm format)

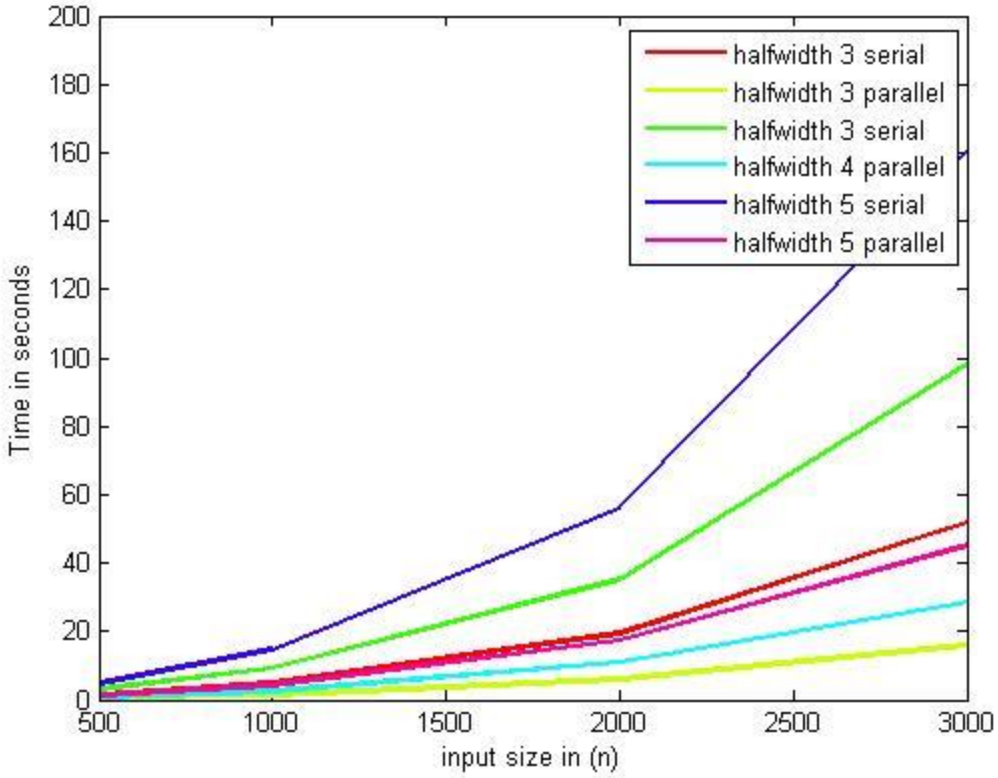
Output:

Blurred Image

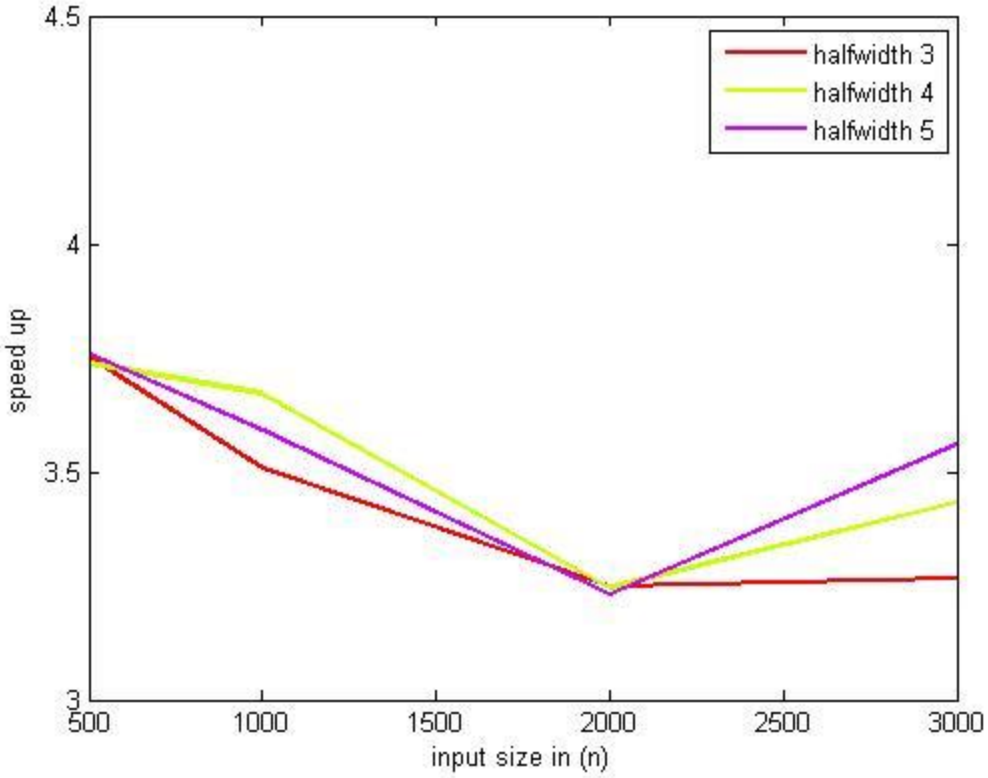
Parallel overhead time:

0.03124528 seconds

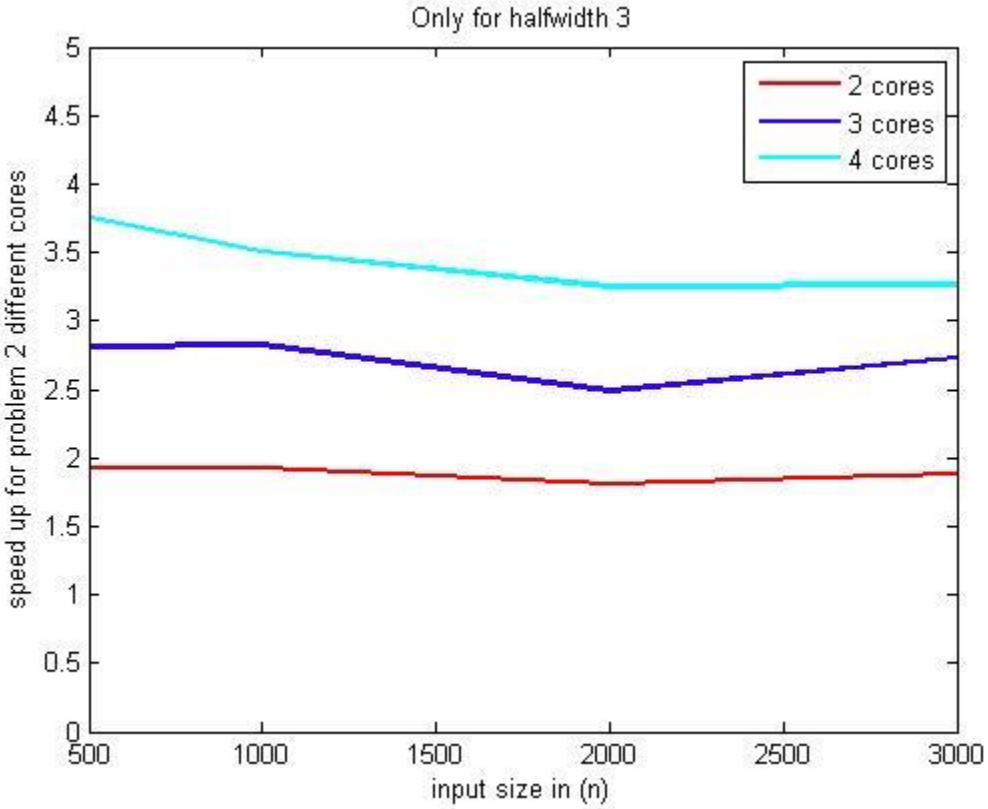
Problem Size vs Time (Serial, parallel) For Different Halfwidths:



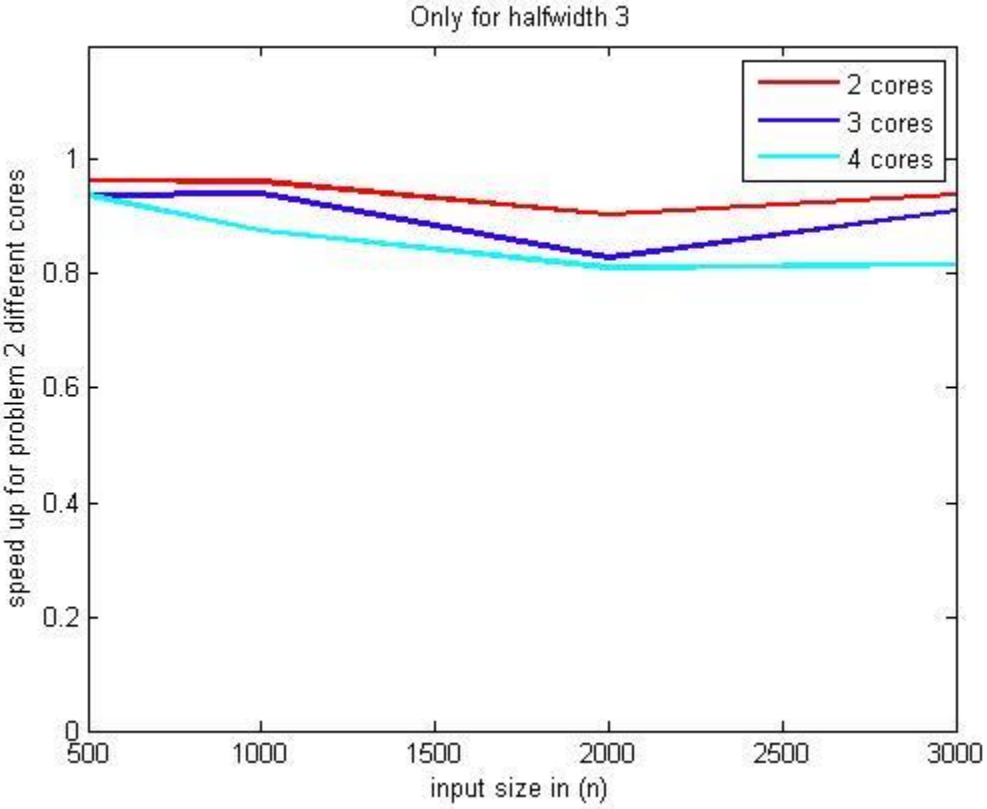
Problem size v. Speedup curve For Different Halfwidths:



Number of cores Vs speedup:



Efficiency vs. Problem Size:



Observations and comments:

1. Speed up increases with problem size. Just because the parallel overhead is less.
2. As we increase the number of cores it simply has to do more communication compared to the serial one. So the efficiency is decreasing over increase in number of cores.