HPC Lab 4

Nishi Doshi (201601408)
Roshani (201601059)

Dhirubhai Ambani Institute of Information and Communication Technology 201601408@daiict.ac.in
201601059@daiict.ac.in

1 Overview

The various images used as input are created from 1024x1024 Lenna Image available online by resizing them to 32x32, 64x64, 128x128, 256x256 and 512x512 using python code.
The *serial fraction value* as well the *Number of Cores v/s Speedup* graphs are generated using python code. The formula used for calculating speedup in the graph is :
$for\ i\ in\ range(10):\ sum+= \frac{T_{serial}}{T_{parallel}}$
Hence, $speedup = \frac{sum}{10}$

Fig. 1. 32 ppm image

Fig. 2. 64 ppm image

Fig. 3. 128 ppm image

Fig. 4. 256 ppm image

Fig. 5. 512 ppm image

Fig. 6. 1024 ppm image

Image Rotation

Nishi Doshi (201601408)
Roshani (201601059)

Dhirubhai Ambani Institute of Information and Communication Technology

1 Implementation Details

1(a) Brief and clear description about the Serial implementation

In serial implementation, we go through the co-ordinates of every pixel of the input image and find the changed co-ordinates of that pixel in output image by a fixed rotation angle θ. After finding the changed co-ordinates, we place the pixel the original pixel at that position. If we find that the changed co-ordinates of the output image are out of bounds then we simple ignore them which results in black portion at the corner of the image and loss of some image pixels as well.

1(b) Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

In the parallel implementation of image rotation, we parallelize the image by decomposing the image into various columns. That is if p processors are available we divide the input image into p columns and process it by computing the changed co-ordinates on different processors and using the computed changed coordinates to generate the output image.

The images of obtained of various sizes after rotation are shown below :

Fig. 1. 32 ppm image

Fig. 2. 64 ppm image

Fig. 3. 128 ppm image

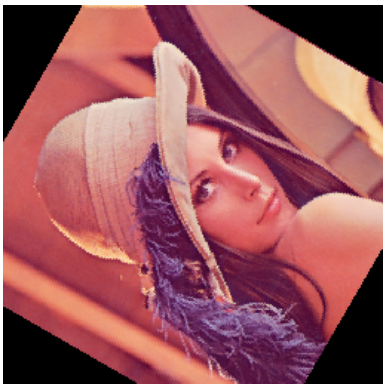


Fig. 4. 256 ppm image

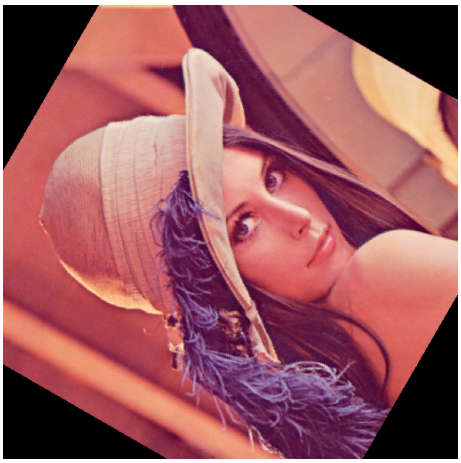


Fig. 5. 512 ppm image

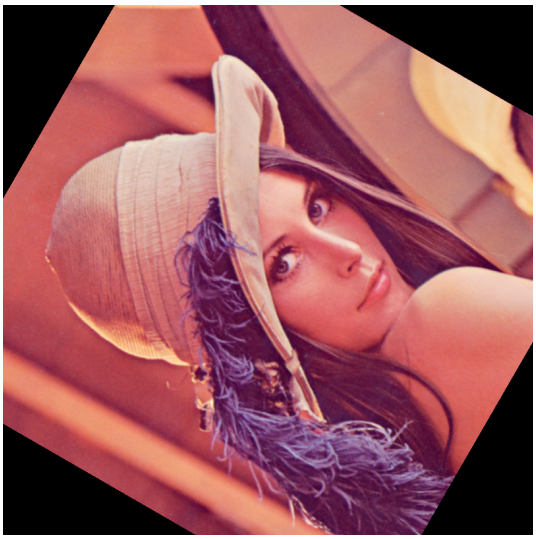


Fig. 6. 1024 ppm image

2 Complexity and Analysis Related

2(a) Complexity of serial code

The serial implementation includes reading the PPM formatted image and then traversing every co-ordinate in the read image. Thus if image has NxN dimension, then time complexity of serial code will be $O(N^2)$.

2(b) Complexity of parallel code (split as needed into work, step, etc.)

In case of parallel code, for every row the elements are divided into p groups where p represents number of processors. Hence the time complexity for traversing every row reduces from N elements to $\frac{N}{p}$. Thus one row traversal takes O($\frac{N}{p}$) time and N rows traversal takes O($N * \frac{N}{p}$) time complexity.

2(c) Cost of Parallel Algorithm

The cost of parallel Algorithm is calculated by :
$number\ of\ processors\ *\ time\ taken\ by\ code\ to\ run\ on\ one\ processor.$
In serial implementation, we find that the cost is $O(N^2)$ and in case of parallel implementation it is also of the same order. Hence, cost wise both serial and parallel implementation perform the same.

2(d) Theoretical Speedup (using asymptotic analysis, etc.)

Speedup is calculated using formula $Speedup = \frac{SerialTime}{ParallelTime}$. Hence, in this case theoretical the speedup should be equivalent to p.

2(e) Estimated Serial Fraction

The serial fraction is given by $Serial\ fraction = \frac{(end\ to\ end\ time - algo\ time)}{end\ to\ end\ time}$. Experimentally, for different problem sizes we find different values of serial fraction as shown below :

Problem Size	Serial Fraction
32	0.0021359
64	0.0031767
128	0.006838
256	0.0234074
512	0.0847734
1024	0.4166556

2(f) Number of memory accesses

The memory allocated per problem of NxN image size here is :

- $3N^2$ memory to store input image which has 3 pixel values R, G, B associated with it.
- $3N^2$ memory to store output image which has 3 pixel values R, G, B associated with it.
- Constant memory to store values of cx , cy and θ as described below.

For every pixel of input image a corresponding output image pixel is accessed. However a certain number of output image pixels are not accessed which result in black portion of the rotated image as shown above in figures. Hence a total of $2 * 3N^2$ memory access per NxN image are done.

2(g) Number of computations

For every $(i,j)^{th}$ pixel of the image, changed coordinates $(changed\_i, changed\_j)$ are calculated on the basis of previously calculated rotation value by using following formulas :

$$cx = \frac{image.width}{2}$$

$$cy = \frac{image.height}{2}$$

$$changed\_i = (i - cx)cos\theta + (j - cy)sin\theta + cx$$

$$changed\_j = (j - cy)cos\theta - (i - cx)sin\theta + cy$$

where $\theta = 60°$.
As we see that cx , cy and θ are constants. θ being a constant implies $cos\theta$ and $sin\theta$ values are also constant. Hence for every $(i,j)^{th}$ pixel we calculate values based on above equation. Every $(i,j)^{th}$ pixel requires 2 subtraction operations, 2 multiplication operations and 2 addition operations. Hence a total of 6 operations per pixel are done which total upto give $6N^2$ computations. For some border pixels as the calculated values go out of bounds assignment of colors to new image are not done. But such conditions do involve calculation of $changed\_i$ and $changed\_j$ coordinates. Hence both serial and parallel implementations perform $6N^2$ computations each.
The difference we observe in both implementations is that serial code does all these computations on one processor only and parallel code performs them all on different processors by dividing the input image column wise.

3 Curve Based Analysis

3(a) Time Curve related analysis (as no. of processor increases)

It is observed that as the number of processors on which the program runs increases the time taken for exexution decreases. This is because the execution time is given as : $\sigma(n) + \frac{\psi(n)}{p}$ where $\sigma(n)$ is the time taken for executing the serial code and $\frac{\psi(n)}{p}$ is the time taken for executing the parallel code, n is the problem size and p is the number of processors. Thus, as the number of processors increases so does the execution time decrease as they are inversely proportional.

3(b) Time Curve related analysis (as problem size increases, also for serial)

As the problem size increases the execution time will increase. The execution time of the serial code and the parallel code both will increase as more number of computations will be required.

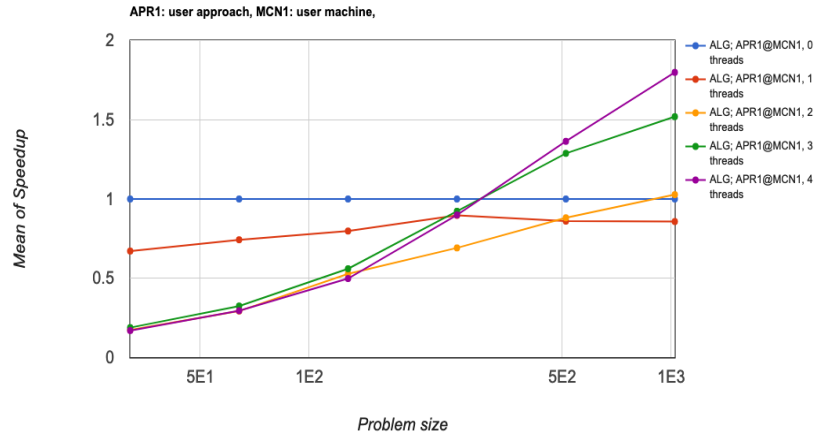
3(c) Speedup Curve related analysis (as problem size and no. of processors increase)

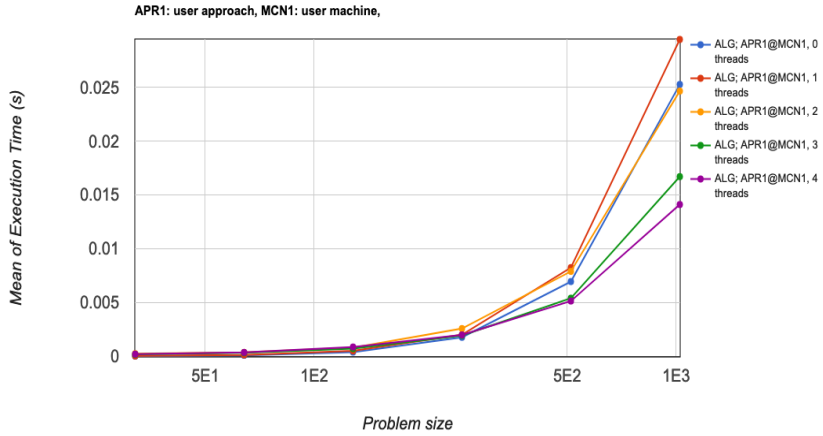
We observe that theoretically speedup value should be equal to the number of processors p. But in the graph shown below one observes that speedup maximum is obtained with 4 processor and image size 1024x1024 which is around 1.8. The reason for getting a less than 2 speed up is that the memory access of output image pixels that is $(changed\_i, changed\_j)^{th}$ pixels cannot be efficiently done as cache cannot guarantee the store of neighbourhood access in the output image. (θ being a constant in this case of rotating the image.)

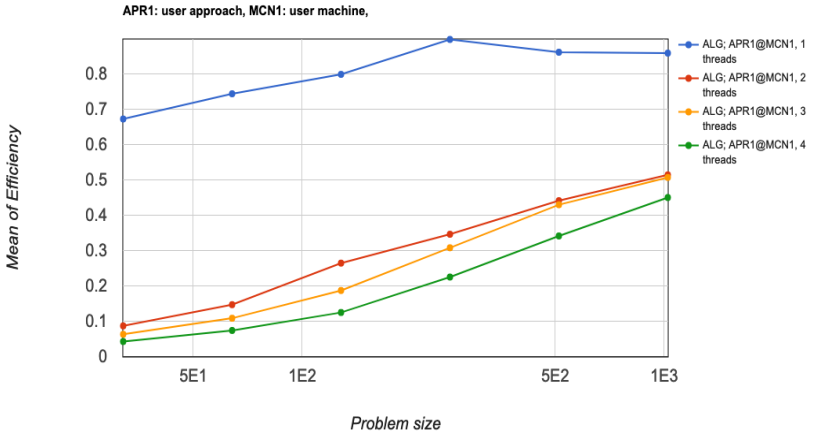
3(d) Efficiency Curve related analysis

We know that efficiency is calculated as the ratio of Speedup and the number of processors on which the code runs. Therefore, we note that as the number of processors increases the efficiency decreases. In our studies we obtained all the efficiencies to be less than 1.
The speedup when the code runs on 1 thread is obtained to be 1 and then slightly decreases. Thus it is observed that the efficiency touches 1 and then decreases. The speedup when the code runs on 2 threads is lesser than 1 at all times. Therefore, the efficiency obtained is lesser than 0.5. Similarly, we get the efficiency of 3 threads to be around 0.5. This trend is observed because the speedup when the code runs on 4 threads is obtained somewhat greater than 1.5 and lesser than 2. Thus the efficiency that will be obtained will be lesser than 0.5, which is observed in the graph.

4 Graphs






APR1: user approach, MCN1: user machine,
Mean of Efficiency
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
5E1
1E2
5E2
1E3
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

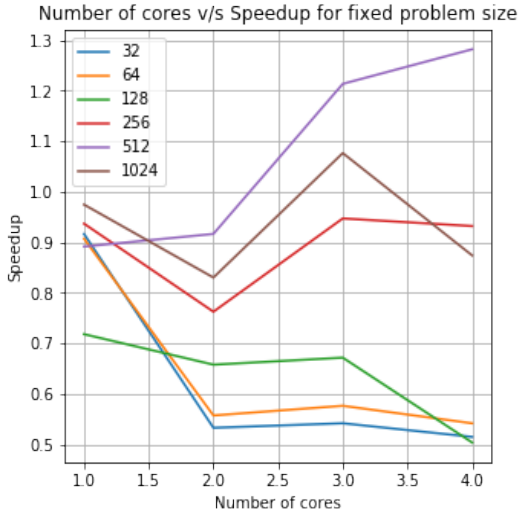

Number of cores v/s Speedup for fixed problem size
Speedup
Number of cores
32
64
128
256
512
1024
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Image Warping

Nishi Doshi (201601408)
Roshani (201601059)

Dhirubhai Ambani Institute of Information and Communication Technology

1 Implementation Details

1(a) Brief and clear description about the Serial implementation

The serial implementation of the code initially reads the image of PPM format and stores in PPMImage struct object *img*. After reading then image, every pixel co-ordinate of the image is traversed and changed coordinate in the output image is found by varying rotation angle θ (as opposed to being constant in the case of image rotation). After the changed coordinates are calculated the RGB values of input image's original coordinates are replicated in the output image.
Bilinear Interpolation technique is used here instead of discarding the coordinates outside the range of output image. In bilnear interpolation we map any co-ordinate greater than width or height of image to a particular position in image which is obtained by taking modulo of that outside point with the height or width. Hence, with this we can ensure that corresponding to every original coordinate a valid co-ordinate in the output image is obtained.
After observing the warped images, we see that there are still

1(b) Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

The parallel implementation of code involves dividing the image into p groups column wise and then applying the formula for warping the image. The algorithm for warping the image gets distributed on different cores present and output image pixels are then alloted R, G, B values. The images of obtained of various sizes after rotation are shown below :

Fig. 1. 32 ppm image

Fig. 2. 64 ppm image

Fig. 3. 128 ppm image

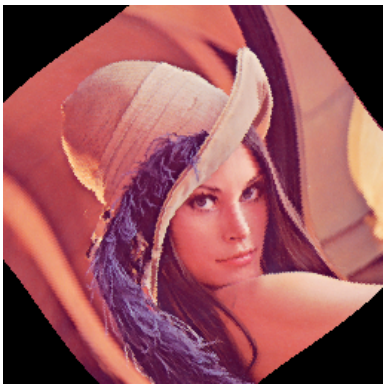


Fig. 4. 256 ppm image

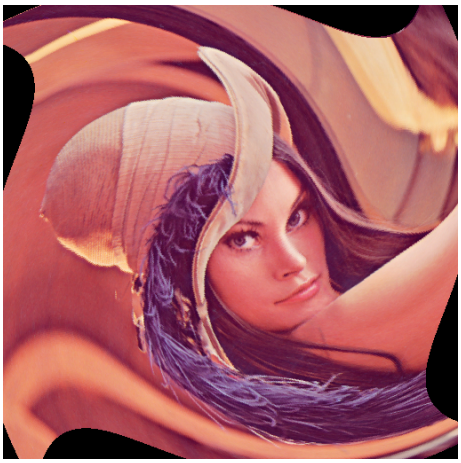


Fig. 5. 512 ppm image

Fig. 6. 1024 ppm image

2 Complexity and Analysis Related

2(a) Complexity of serial code

The serial code iterates through every $(i, j)^{th}$ pixel of the NxN image and hence time complexity of the serial code is $O(N^2)$.

2(b) Complexity of parallel code (split as needed into work, step, etc.)

The parallel code gets splits up the image into p columns. And hence computations get divided on different processors. Thus the time complexity of parallel code is $O(\frac{N^2}{p})$.

2(c) Cost of Parallel Algorithm

Cost of parallel algorithm is defined as time required for parallel code to run multiplied by the number of processors. Hence,

$$Cost = p * T_{parallel}$$

Thus, $cost = p * \frac{N^2}{p} = N^2$.
Here, we observe that cost for both parallel and serial codes is equal to the size of image. Thus, the algorithm developed is not cost optimal and only time complexity effectively decreases for parallel code.

2(d) Theoretical Speedup (using asymptotic analysis, etc.)

Speedup is calculated using formula $Speedup = \frac{SerialTime}{ParallelTime}$. Hence, in this case theoretical the speedup should be equivalent to p.

2(e) Estimated Serial Fraction

Problem Size	Serial Fraction
32	0.0024031
64	0.0048149
128	0.0150396
256	0.0558924
512	0.2531419
1024	1.1371359

2(f) Number of memory accesses

The memory allocated per problem of NxN image size here is :

- $3N^2$ memory to store input image which has 3 pixel values R, G, B associated with it.
- $3N^2$ memory to store output image which has 3 pixel values R, G, B associated with it.
- Constant memory to store values of cx and cy as described below.

For every pixel of input image a corresponding output image pixel is accessed. However a certain number of output image pixels are not accessed which result in black portion of the rotated image as shown above in figures. Hence a total of $2 * 3N^2$ memory access per NxN image are done.

2(g) Number of computations

The main equations involved here in computation are as follows :

$$cx = \frac{image.width}{2}$$

$$cy = \frac{image.height}{2}$$

$$\theta = 0.27 * \sqrt{((i - cx) * (i - cx) + (j - cy) * (j - cy))}$$

$$changed\_i = (i - cx)cos\theta + (j - cy)sin\theta + cx$$

$$changed\_j = (j - cy)cos\theta - (i - cx)sin\theta + cy$$

As can be seen the value of θ in case of warping always keeps on changing and hence an extra computation of θ is involved for each pixel which adds 3 multiplication operations, 4 subtraction operations and 1 addition operation to the previous calculations and hence a total of 6+3+4+1 = 14 computations are done per pixel. Hence a total of $14N^2$ computations are done per NxN image.
Here, the computations involved for calculating $sin\theta$ and $cos\theta$ are not constant as the value of θ changes. Hence, the exact number of computations cannot be calculated.

3 Curve Based Analysis

3(a) Time Curve related analysis (as no. of processor increases)

As the number of processors increase, the time required for the same image size decreases effectively as seen in graph. It is observed that as the number of processors on which the program runs increases the time taken for exexution decreases. This is because the execution time is given as : $\sigma(n) + \frac{\psi(n)}{p}$ where $\sigma(n)$ is the time taken for executing the serial code and $\frac{\psi(n)}{p}$ is the time taken for executing the parallel code, n is the problem size and p is the number of processors. Thus, as the number of processors increases so does the execution time decrease as they are inversely proportional.

3(b) Time Curve related analysis (as problem size increases, also for serial)

As the problem size increases the execution time will increase. The execution time of the serial code and the parallel code both will increase. as more number of computations will be required.

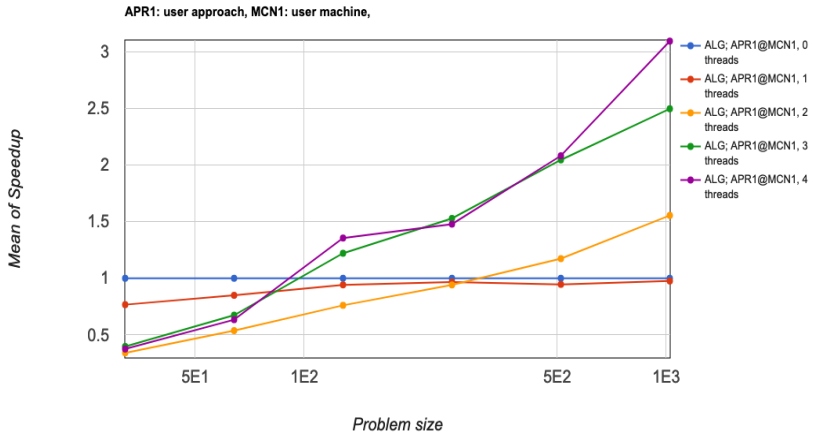
3(c) Speedup Curve related analysis (as problem size and no. of processors increase)

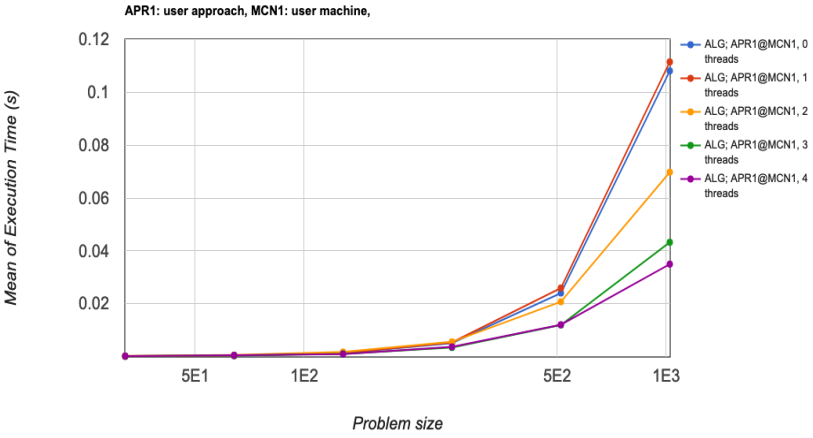
It can be observed from the graph that the speedup obtained for 4 threads is about 3, for 3 threads is 2.5, for 2 threads is 1.5 and for 1 thread is nearly 1 and then decreases. This trend is observed because in-case of image warping we access the new rotated pixels. The pixels are rotated by a θ that keeps on changing depending upon the distance of the pixel from the center. Thus it may so happen that the pixels that need to be loaded may not be present in the cache and thus delay in memory access reduces the speedup.

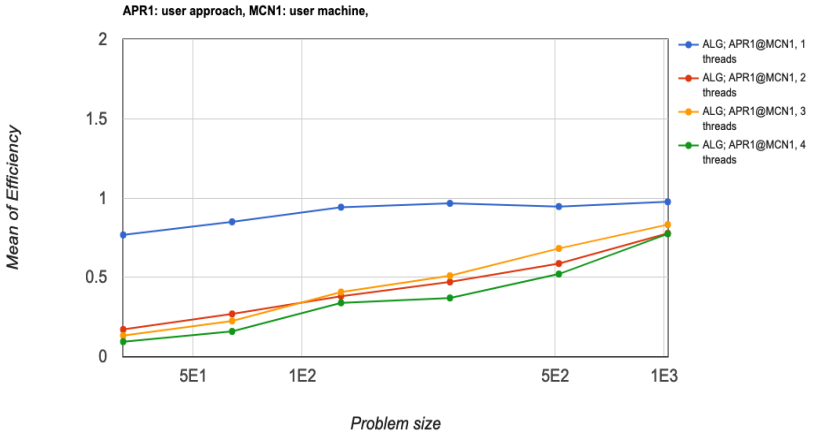
3(d) Efficiency Curve related analysis

We know that efficiency is calculated as the ratio of Speedup and the number of processors on which the code runs. Therefore, we note that as the number of processors increases the efficiency decreases. In our studies we obtained all the efficiencies to be less than 1.
The speedup when the code runs on 1 thread is obtained to be 1 and then slightly decreases. Thus it is observed that the efficiency touches 1 and then decreases. The speedup when the code runs on 2 threads is lesser than 2 at all times. Therefore, the efficiency obtained is lesser than 1 and at times greater than 0.5. Similarly, we get the efficiency of 3 threads to be around 0.5. This trend is observed because the speedup when the code runs on 4 threads is obtained around 3. Thus the efficiency that will be obtained will be lesser than 1, which is observed in the graph.

4 Graphs






APR1: user approach, MCN1: user machine,
Mean of Efficiency
Problem size
2
1.5
1
0.5
0
5E1
1E2
5E2
1E3
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads

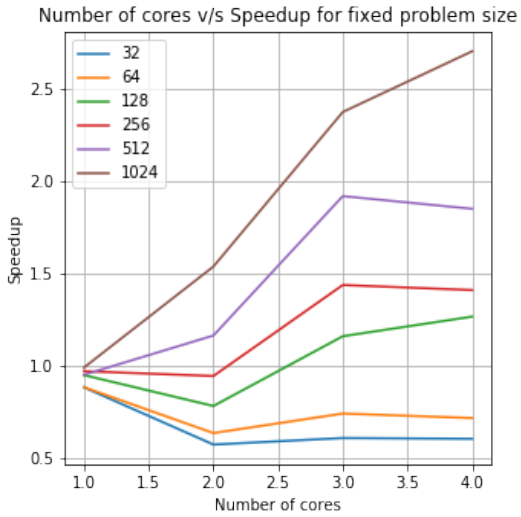

Number of cores v/s Speedup for fixed problem size
Speedup
Number of cores
32
64
128
256
512
1024
2.5
2.0
1.5
1.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Image Blurring

Nishi Doshi (201601408)
Roshani (201601059)

Dhirubhai Ambani Institute of Information and Communication Technology

1 Implementation Details

1(a) Brief and clear description about the Serial implementation

The images are initially read and stored in PPMImage struct data format. A half width *windowsize* (here 5) is selected. The logic of the code includes finding all the neighbours in the half width of the given pixel. Amongst all the neighbours, the pixel RGB values are sorted and median value of those values are taken and placed in the corresponding image in output. The sorting of the neighbours is done by putting all the R, G, B values in separate arrays and sorting them and then finding the median of the array.
If number of elements are even then median is equal to $\frac{arr[\frac{mid}{2}]+arr[\frac{mid}{2}+1]}{2}$
If number of elements are odd then median is equal to $arr[\frac{mid+1}{2}]$
Here $mid = \frac{number\ of\ elements}{2}$.
Median for all the three Red,Green and Blue pixels is calculated and in the output image corresponding $(i, j)^{th}$ pixel is replaced with median of these three values which are found in the boundary of image.
In case, the pixel lies on the boundary or does not have valid pixels that satisfy the window size of the pixel, then corresponding values are discarded and not taken into consideration.
For example, the pixel at position $(0, 0)$ will not have any neighbours to its right side or above it to be covered in the window and hence those values are not taken into consideration.

1(b) Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

In case of parallel implementation, the algorithm of serial implementation is followed. The image is divided into four parts on the basis of columns on different processors and output image is formulated.
The images of obtained of various sizes after blurring them are shown below. It can be seen in the following images that effect of blurring decreases as the size of image increases. The reason for this is that the when image size is smaller the neighbours included in blurring effect cover more amount of image as compared to images of larger size. Hence, the blur effect for same window size of 5 is observed more in case of images of smaller size compared to images of larger size.

Fig. 1. 32 ppm image

Fig. 2. 64 ppm image

Fig. 3. 128 ppm image

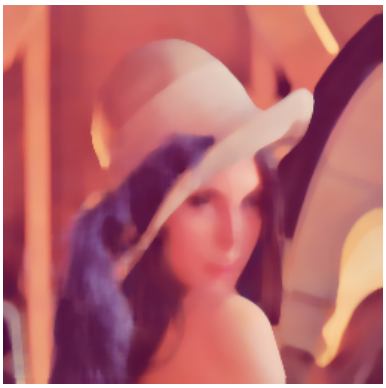


Fig. 4. 256 ppm image

Fig. 5. 512 ppm image

Fig. 6. 1024 ppm image

2 Complexity and Analysis Related

2(a) Complexity of serial code

The serial code involves use of neighbouring pixels of $(i,j)^{th}$ pixel to find the pixel values in output image. Hence depending on the size of window (here considered to be 5), the neighbours are iterated and their R,G,B values are added to array which is sorted in $36log_236$ time which can be considered to be a constant k for every pixel. (36 is selected for window size half width of 5). Hence, the time complexity of serial code is $O(N^2)$ where N is the size of image that is NxN.

2(b) Complexity of parallel code (split as needed into work, step, etc.)

While parallelizing we are only parallelizing the output image by columns and not parallelizing the sorting of array of R,G,B values of neighbours. Thus, complexity is $O(\frac{N^2}{p})$.

2(c) Cost of Parallel Algorithm

The cost of parallel algorithm is calculated as

$$Cost = p * \frac{N^2}{p} = N^2$$

2(d) Theoretical Speedup (using asymptotic analysis, etc.)

The theoritical speedup is calculated as
Speedup = $\frac{T_s}{T_p}$
where, T_s is the serial time and T_p is the parallel time.

2(e) Estimated Serial Fraction

Problem Size	Serial Fraction
32	0.1506703
64	0.6278714
128	2.5265148
256	10.0577404
512	39.6794599
1024	146.4220505

2(f) Number of memory accesses

The memory allocated per problem of NxN image size here is :

- $3N^2$ memory to store input image which has 3 pixel values R,G,B associated with it.
- $3N^2$ memory to store output image which has 3 pixel values R,G,B associated with it.
- Constant memory to store values of cx , cy and θ as described below.
- For every $(i,j)^{th}$ pixel an array of 1000 is allocated.

For every pixel of input image a corresponding output image pixel is accessed. However a certain number of output image pixels are not accessed which result in black portion of the rotated image as shown above in figures. Hence a total of $2 * 3N^2$ memory access per NxN image are done.

2(g) Number of computations

For every $(i, j)^{th}$ pixel of the image, changed coordinates $(changed\_i, changed\_j)$ are calculated on the basis of previously calculated rotation value by using following formulas :

$$cx = \frac{image.width}{2}$$

$$cy = \frac{image.height}{2}$$

$$changed\_i = (i - cx)cos\theta + (j - cy)sin\theta + cx$$

$$changed\_j = (j - cy)cos\theta - (i - cx)sin\theta + cy$$

where $\theta = 60°$.
As we see that cx , cy and θ are constants. θ being a constant implies $cos\theta$ and $sin\theta$ values are also constant. Hence for every $(i, j)^{th}$ pixel we calculate values based on above equation. Every $(i, j)^{th}$ pixel requires 2 subtraction operations, 2 multiplication operations and 2 addition operations. Hence a total of 6 operations per pixel are done which total upto give $6N^2$ computations. For some border pixels as the calculated values go out of bounds assignment of colors to new image are not done. But such conditions do involve calculation of $changed\_i$ and $changed\_j$ coordinates. Hence both serial and parallel implementations perform $6N^2$ computations each.
The difference we observe in both implementations is that serial code does all these computations on one processor only and parallel code performs them all on different processors by dividing the input image column wise.

3 Curve Based Analysis

3(a) Time Curve related analysis (as no. of processor increases)

As the number of processors increase, the time required for the same image size decreases effectively as seen in graph. It is observed that as the number of processors on which the program runs increases the time taken for exexution decreases. This is because the execution time is given as : $\sigma(n) + \frac{\psi(n)}{p}$ where $\sigma(n)$ is the time taken for executing the serial code and $\frac{\psi(n)}{p}$ is the time taken for executing the parallel code, n is the problem size and p is the number of processors. Thus, as the number of processors increases so does the execution time decrease as they are inversely proportional.

3(b) Time Curve related analysis (as problem size increases, also for serial)

As the problem size increases the execution time will increase. The execution time of the serial code and the parallel code both will increase. as more number of computations will be required.

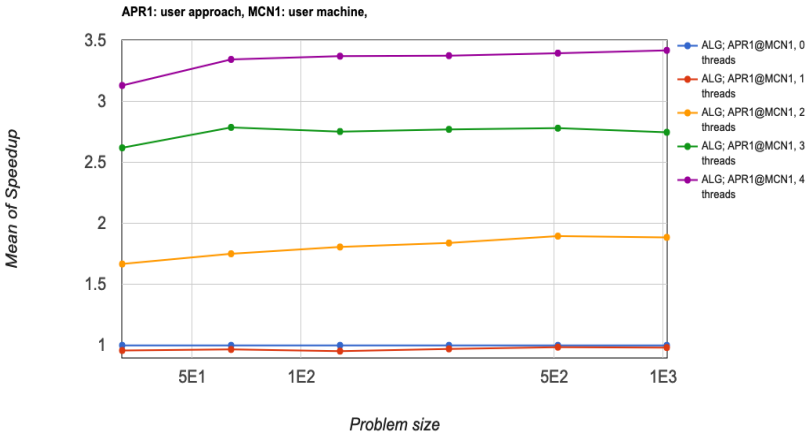
3(c) Speedup Curve related analysis (as problem size and no. of processors increase)

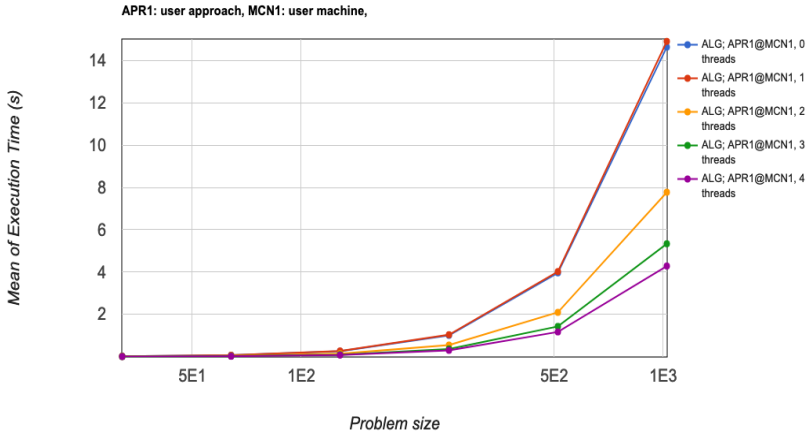
Speedup is calculated as the ratio of serial time is to parallel time. It is observed that the speedup in case of 4 threads is obtained to be around 3.5, for 3 threads is obtained to be lesser than 3 and slightly greater than 2.5, for 2 threads reaches around 2 and for 1 thread remains 1.

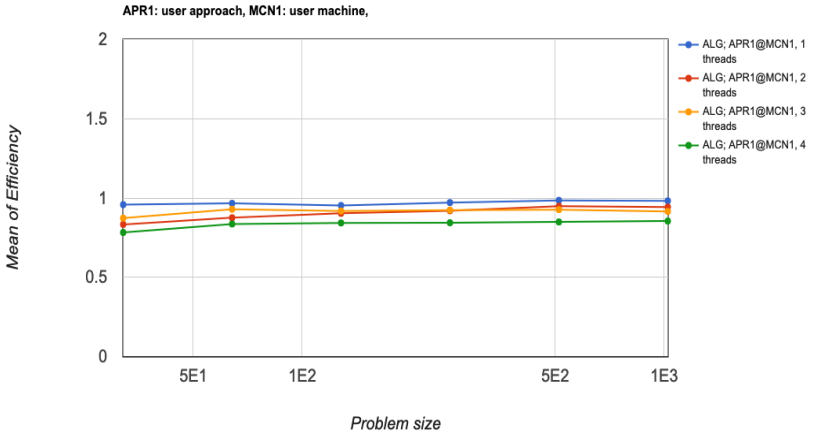
3(d) Efficiency Curve related analysis

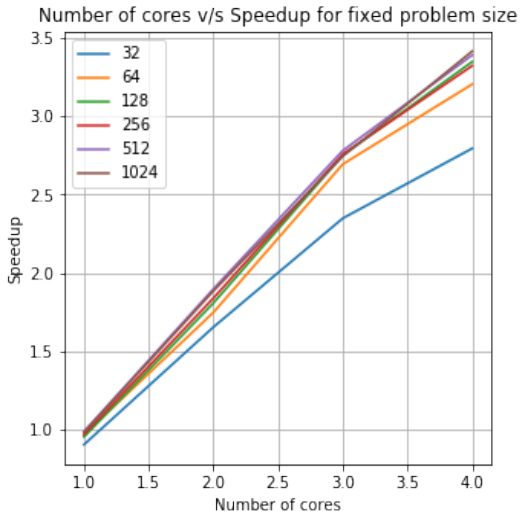
We know that efficiency is calculated as the ratio of Speedup and the number of processors on which the code runs. Therefore, we note that as the number of processors increases the efficiency decreases. In our studies we obtained all the efficiencies to be less than or around 1, and all efficiencies are greater than 0.5.
The speedup when the code runs on 1 thread remains 1 for all problem sizes. Thus it is observed that the efficiency touches 1. The speedup when the code runs on 2 threads is lesser than 2 at all times. Therefore, the efficiency obtained is lesser than 1 and at times greater than 0.5. Similarly, we get the efficiency of 3 threads to be around 1. In case of 4 threads the speedup obtained is around 3.5. Thus the efficiency that will be obtained will be lesser than 1 but greater than 0.5.

4 Graphs






APR1: user approach, MCN1: user machine,
Mean of Efficiency
Problem size
2
1.5
1
0.5
0
5E1
1E2
5E2
1E3
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


Number of cores v/s Speedup for fixed problem size
Speedup
Number of cores
32
64
128
256
512
1024
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Calculating the value of Pi - Using Monte Carlo Simulation

Nishi Doshi (201601408)
Roshani (201601059)

Dhirubhai Ambani Institute of Information and Communication Technology

1 Implementation Details

1(a) Brief and clear description about the Serial implementation

In case of the serial code we use the inbuilt function $rand\_r()$ and generate two random numbers x and y. In this case we consider 2 as the seed of the function $rand\_r()$. The random numbers generated are then put to a condition if

$$x^2 + y^2 <= 1$$

. This would ensure that the points lie inside the unit circle whose area we have taken into consideration.
Finally we divide the number of points that lie inside the circle with the total number of random points generated and multiply it by 4. This would give us an approximate value of the value of π i.e. 3.14.

1(b) Brief and clear description about the implementation of the approach (Parallelization Strategy, Mapping of computation to threads)

In case of the parallel approach we use the "#pragma omp parallel" to create different threads. Each thread generates its own set of random numbers and checks for the condition to be true. A count of points that lie within the unit circle for each thread is maintained and are added in the final count(which is a shared variable).

2 Complexity and Analysis Related

2(a) Complexity of serial code

The complexity of serial code is $O(N)$ as the loop runs for N iterations which is the problem size.

2(b) Complexity of parallel code (split as needed into work, step, etc.)

The complexity of the parallel code is given by $O(\frac{N}{p})$.

2(c) Cost of Parallel Algorithm

The cost of parallel algorithm is calculated as the product of the number of processors and time complexity of the parallel code. Therefore, $p * \frac{N}{p} = N$

2(d) Theoretical Speedup (using asymptotic analysis, etc.)

The theoritical speedup is calculated as
Speedup = $\frac{T_s}{T_p}$
where, T_s is the serial time and T_p is the parallel time.

2(e) Experimental Serial Fraction

The serial fraction is given by $Serial\ fraction = \frac{(end\ to\ end\ time - algo\ time)}{end\ to\ end\ time}$. Experimentally, for different problem sizes we find different values of serial fraction as shown below :

Problem Size	Serial Fraction
10	2.47e-05
100	3.33e-05
1000	0.0001658
10000	0.0014624
100000	0.0142974
1000000	0.1390354
10000000	1.3260173
100000000	13.1398037
50000	0.0073823
500000	0.0701721
5000000	0.6644158
50000000	6.5823775

2(f) Number of memory accesses

The memory allocated for all problem sizes will be $2N$ for input. As for all iterations two floating point variables are created and the random values that are generated are stored in these variables.

2(g) Number of computations

We have two divisions (rand_r()/RAND_MAX) in-order to generate the random variable., 2 multiplications(to calculate the square), 2 additions. The number of computations for each iteration would be $6N$.
Here, the computations involved for calculating $rand\_r()$ and $RAND\_MAX$ are not known to us as it may depend on the functions used. Hence, the exact number of computations cannot be calculated.

3 Curve Based Analysis

3(a) Time Curve related analysis (as no. of processor increases)

As the number of processors increase, the time required for the same image size decreases effectively as seen in graph. It is observed that as the number of processors on which the program runs increases the time taken for exexution decreases. This is because the execution time is given as : $\sigma(n) + \frac{\psi(n)}{p}$ where $\sigma(n)$ is the time taken for executing the serial code and $\frac{\psi(n)}{p}$ is the time taken for executing the parallel code, n is the problem size and p is the number of processors. Thus, as the number of processors increases so does the execution time decrease as they are inversely proportional.

3(b) Time Curve related analysis (as problem size increases, also for serial)

As the problem size increases the execution time will increase. The execution time of the serial code and the parallel code both will increase. as more number of computations will be required.

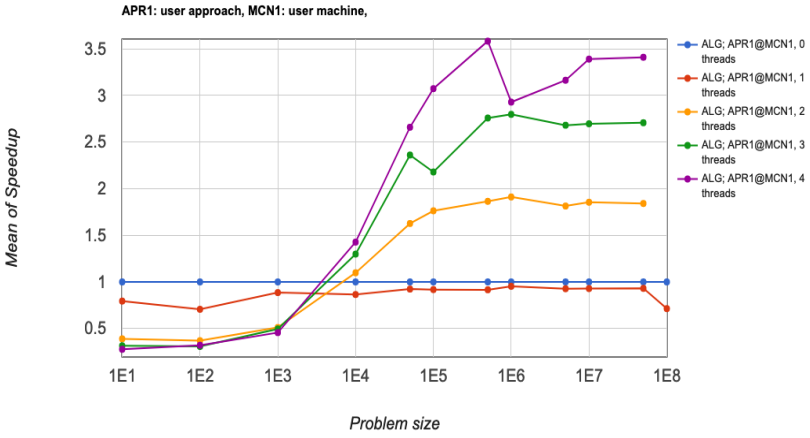
3(c) Speedup Curve related analysis (as problem size and no. of processors increase)

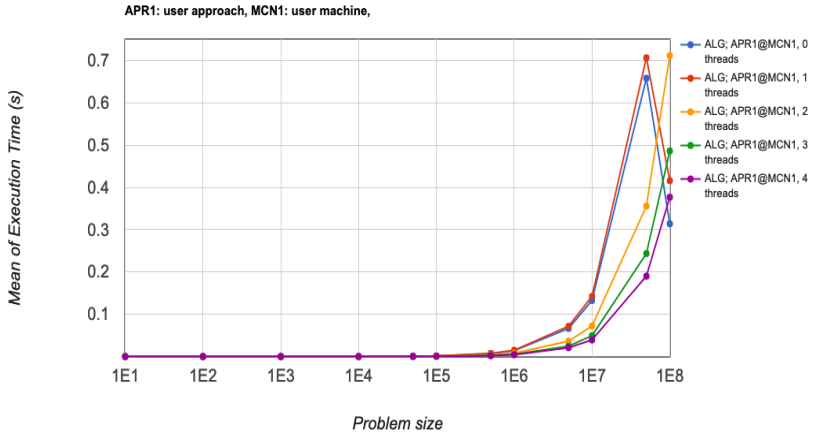
It can be observed from the graph that the speedup obtained for 4 threads is about 3.5 and then decreases, for 3 threads slightly lesser than 3, for 2 threads is greater than 1.5 and for 1 thread is nearly 1 and then decreases.

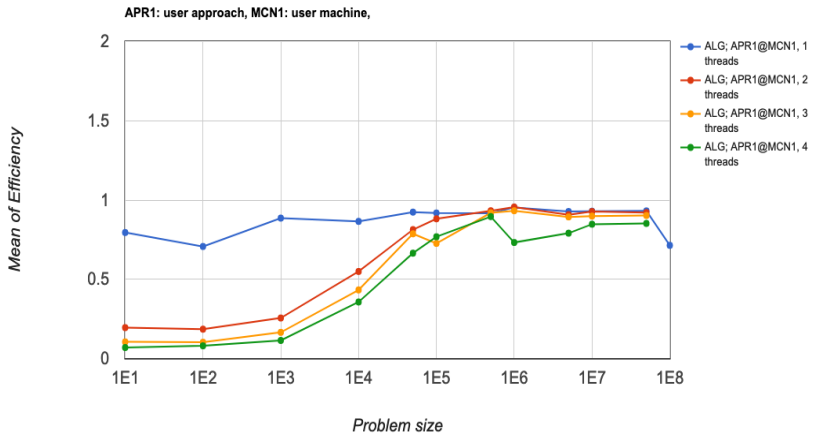
3(d) Efficiency Curve related analysis

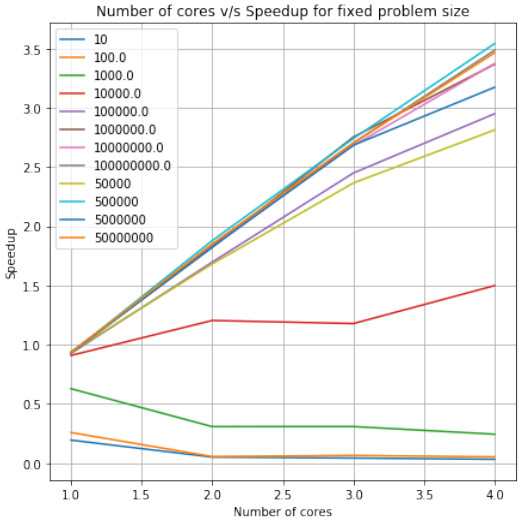
We know that efficiency is calculated as the ratio of Speedup and the number of processors on which the code runs. Therefore, we note that as the number of processors increases the efficiency decreases. In our studies we obtained all the efficiencies to be less than 1.
The speedup when the code runs on 1 thread is obtained to be 1 and then slightly decreases. Thus it is observed that the efficiency touches 1 and then decreases. The speedup when the code runs on 2 threads is lesser than 2 at all times. Therefore, the efficiency obtained is lesser than 1 and at times greater than 0.5. Similarly, we get the efficiency of 3 threads to be around 0.5. This trend is observed because the speedup when the code runs on 4 threads is obtained around 3.5. Thus the efficiency that will be obtained will be lesser than 1, which is observed in the graph.

4 Graphs




APR1: user approach, MCN1: user machine,
Mean of Execution Time (s)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 0 threads
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


APR1: user approach, MCN1: user machine,
Mean of Efficiency
2
1.5
1
0.5
0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
1E8
Problem size
ALG; APR1@MCN1, 1 threads
ALG; APR1@MCN1, 2 threads
ALG; APR1@MCN1, 3 threads
ALG; APR1@MCN1, 4 threads


Number of cores v/s Speedup for fixed problem size
10
100.0
1000.0
10000.0
100000.0
1000000.0
10000000.0
100000000.0
50000
500000
5000000
50000000
Speedup
Number of cores
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0