HPC PROJECT: Parallelizing Several Image Segmentation Techniques

Zubair Bulbulia and Naeem Docrat

University of the Witwatersrand

June 23, 2019

Overview

Introduction

Image segmentation is the application of image processing techniques to alter an image such that meaningful information can be obtained. This can be done by finding objects,lines or contours in an image, or by grouping pixels with shared or similar characteristics.

This Project aims to exploit the highly parallelizable aspect of image processing using OpenMPI and CUDA kernels.

Background: K-Means Clustering

The K-Means clustering algorithm is an unsupervised machine learning algorithm that is often used to find patterns or structure in unlabelled data. Here it is used to segment a colour image into a specified number (K) of colours. This is useful for image compression, and is known as vector quantization.
The K-Means algorithm works by first creating K random cluster centres. Then, by looping through each pixel in an image, pixels are assigned to clusters that they are similar to (in terms of colour). After this, each cluster centre is updated to the mean value of all pixels in the respective cluster. The process is repeated until convergence, and thereafter the pixels in the output image are assigned the same value as the cluster centre value of their respective cluster.

Background: Nick Thresholding

Nick Thresholding is an adaptive thresholding algorithm which makes use of a given window size and k.
The window size and k should be image specific.
It works as follows:
for each pixel intensity calculate the sum and square sum of all the neighbouring pixels which fall within the given window size. Then calculate the mean of all the pixels in the window using the sum. Then use the following formula to calculate the threshold:

$$Threshold(x, y) = M(x, y) + K * \sqrt{(E(x, y) - M(x, y)^2)/WS}$$

where M(x,y) is the mean of all the neighbouring pixels of x and y, K is given and image specific, E(x,y) is the sum of the square of each neighbouring pixel and WS is the Window Size given.
Threshold is then compared to the pixel intensity at (x,y) if threshold is less than (x,y) then the pixel is set to 1 else its set to 0.

Background: Canny Edge Detection

The Canny Edge detection algorithm is often used in image segmentation to accurately detect edges, and consists of four steps. These are as follows:

- Gaussian smoothing - This is done to reduce noise in the image, by using convolution and a gaussian filter.
- Find Gradient - Sobel filters are applied to get the vertical and horizontal gradients, and thereafter the gradient magnitude and direction is calculated.
- Non-Maximum Suppression - Used to find thin edges in image. The pixels from the magnitude image are used to find the local maximum in the direction of the gradient(ridge pixels), and sets non maximum values to 0.
- Hysteresis thresholding - Thresholding is done using two values T1 and T2, where ridge pixels are classified as 'strong' or 'weak' edge pixels. Thereafter, the algorithm attempts to link edges by using the weak pixels that are connected to strong edge pixels.

Methodology: K-Means

K-Means Clustering was implemented using the following approaches:

- Serial approach
- CUDA parallelisation
- MPI parallelisation

The algorithm was tested on the following 3 colour images:

- storm.jpg, size 1910*1000 = 1 910 000 pixels
- cat.jpg, size 800*800 = 640 000 pixels
- suburbs.jpg, size 702*336 = 235 872 pixels

The values of K used are 2, 8, 64 and 128.

Serially, K-means has asymptotic time complexity of $O(K * N^2)$ for an image of size N (N = width*height). The CUDA implementation reduces this by a factor of at least $\frac{imagesize}{2}$, without taking into account overhead costs.

Methodology: K-Means

The CUDA implementation was done using a kernel function, where each thread assigns a pixel to its respective cluster. This was then run using the same number of threads as the pixels in the image. Updating the cluster centres could only be done after an all threads had assigned pixels to clusters (synchronization is needed).
The assignment and update processes are repeated until convergence; the former was done on the gpu and the latter serially. After convergence, the new image was created by assigning pixel values based on the cluster centres values on the gpu.
The MPI Implementation splits up the work done for assigning the pixels to clusters similarly to how it is done in CUDA, however the cluster updates are performed diffrerently. An MPI barrier is called before the updates, and MPI AllReduce is used. This does a reduction operation, and then broadcasts it to the other processes. Therefore most of the code is parallelised in the MPI implementation.

Methodology: Nick Thresholding

Nick Thresholding was implemented using 3 different approaches:

- Serial approach
- Cuda kernel Shared approach
- MPI Approach

Nick Thresholding was tested on 3 different Images:

- MRI of size 225x225
- Lena of size 512x512
- Newspaper of size 1080x1080

using the three images Nick thresholding was also tested using 4 diferent filter sizes:

- 20x20, 40x40, 60x60 and 80x80

Methodology: Nick Thresholding Cont...

The Serial approach is extremely slow. It yields $O(W^2N^2)$ for a W*W Window Size and A N*N image.
In the Cuda approach each pixel was loaded from global memory to shared memory by a thread, then each thread calculates its threshold for the image using its neighbours which it can get by accessing that blocks shared memory, thus reducing access time significantly. The complexity is $O(\frac{W^2N^2}{NumOfThreads})$
In the OpenMPI approach The image is scattered across all nodes. Then each node performs thresholding on its respective sub image, then the image is gathered back to the root node.The complexity is $O(\frac{W^2N^2}{NUMOFNODES})$

Methodology: Canny Edge Detection

Canny Edge Detection was implemented using 3 different approaches:

- Serial approach
- Cuda kernel Shared approach
- OpenMpi Approach

Canny Edge Detection was tested on 3 different Images:

- MRI of size 225x225
- Lena of size 512x512
- Newspaper of size 1080x1080

The Sobel and Averaging Filter used was of size 3x3.

Methodology: Canny Edge Detection Cont...

In the Serial Approach of canny Edge Detection a convolution of the image with a blur kernel is done then, three more convolutions of the image with edge detector kernels then Computation of the gradient direction and finally Non-maximum suppression, and thresholding with hysteresis. This leads to a complexity of $O(m * n * W^2)$, for a NxM size image and WxW size filter.
The CUDA approach parallelises the convolution operation. The kernels are also stored in constant memory to reduce memory access costs. The complexity is thus $O(m * n * W^2)/p$, where p is the number of threads. Storing the image in texture/shared memory can be done as well.
In the OpenMPI Approach The image was scattered in Multiple nodes Each Node performed its respective calculations on each sub-image getting its neighbours from previous nodes and next nodes(Unless its the root or last node) then the image was gathered back to the root node. Thus leads to a Complexity of $O(\frac{W^2 * N * M}{NUMOFNODES})$, for a NxM size image and WxW size filter.

Results: K-Means Clustering

The following shows image storm.jpg of resolution 1910*1000, as well as the 4 output images after clustering when K = 2,8,64 and 128 respectively.

Figure: Original image, K=2, K=8, K=64 & K=128

Results: K-Means Clustering

The following shows the input image cat.jpeg of resolution 800*800, as well as the 4 output images after clustering when K = 2,8,64 and 128 respectively.

Figure: Original image, K=2, K=8, K=64 & K=128

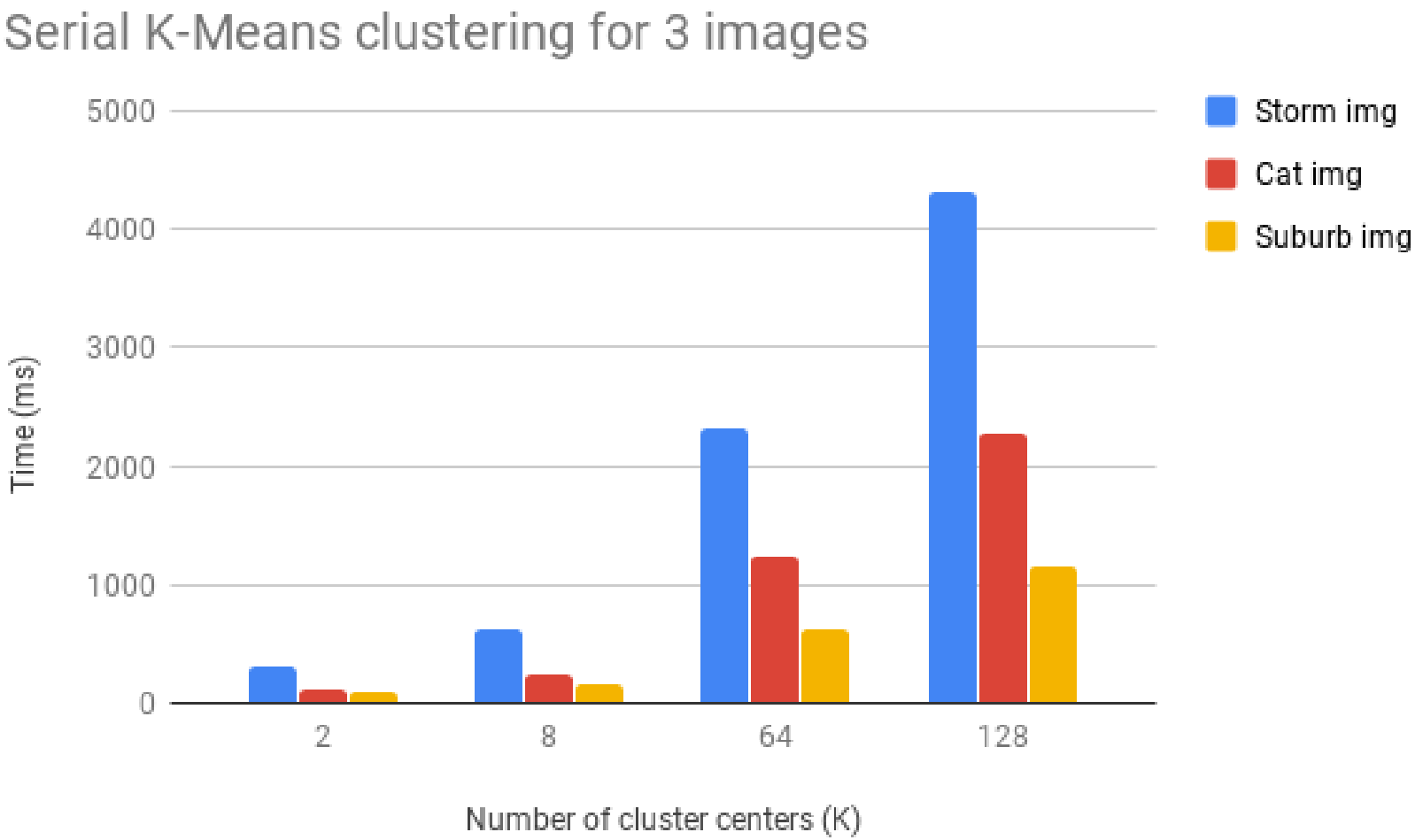
Results: K-Means Clustering

The following shows the input image suburb.jpeg of resolution 702*336, as well as the 4 output images after clustering when K = 2,8,64 and 128 respectively.

Figure: Original image, K=2, K=8, K=64 & K=128

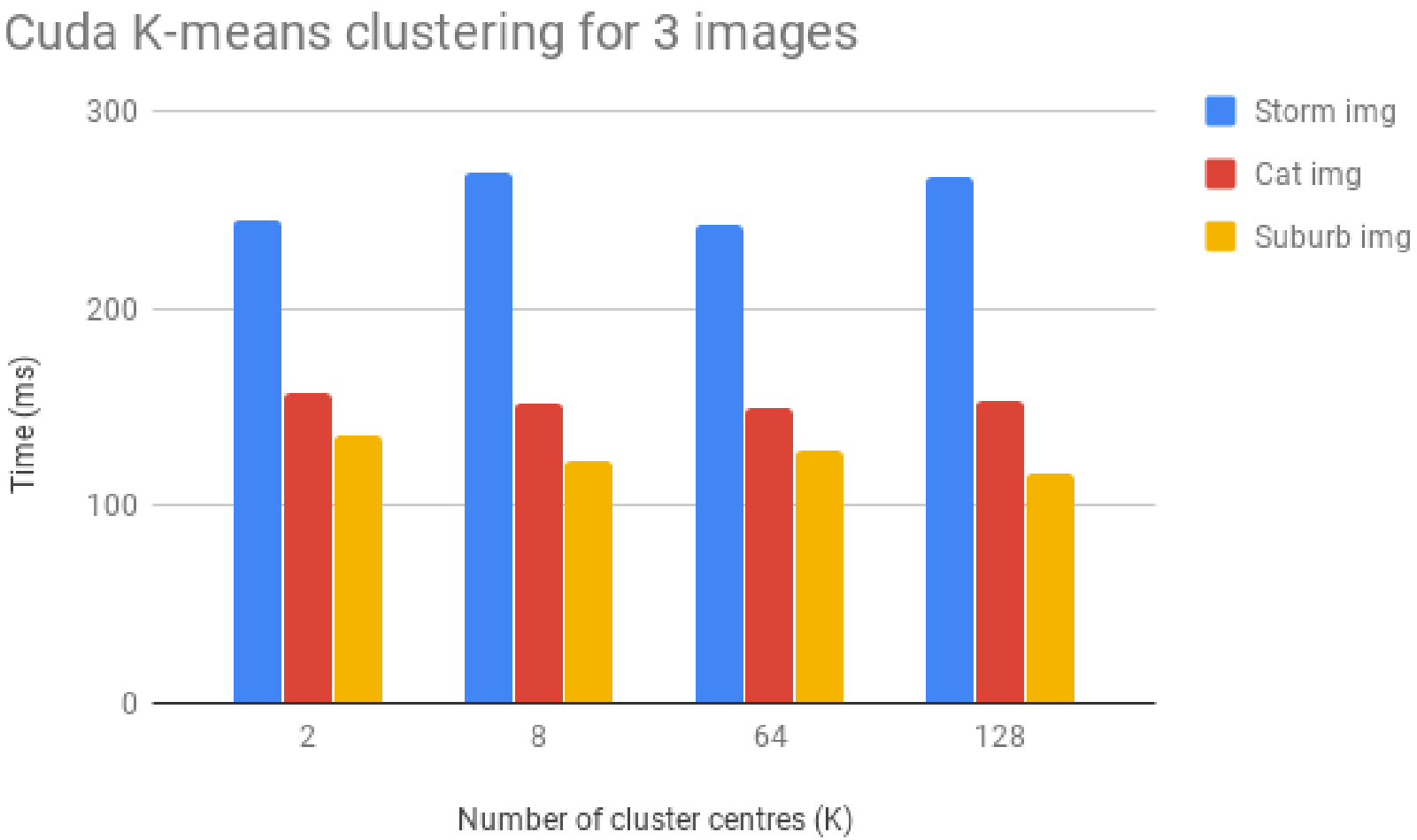
Results: K-Means Clustering

The graph below shows run times obtained when running the serial K-Means clustering algorithm on various colour images for multiple K values.



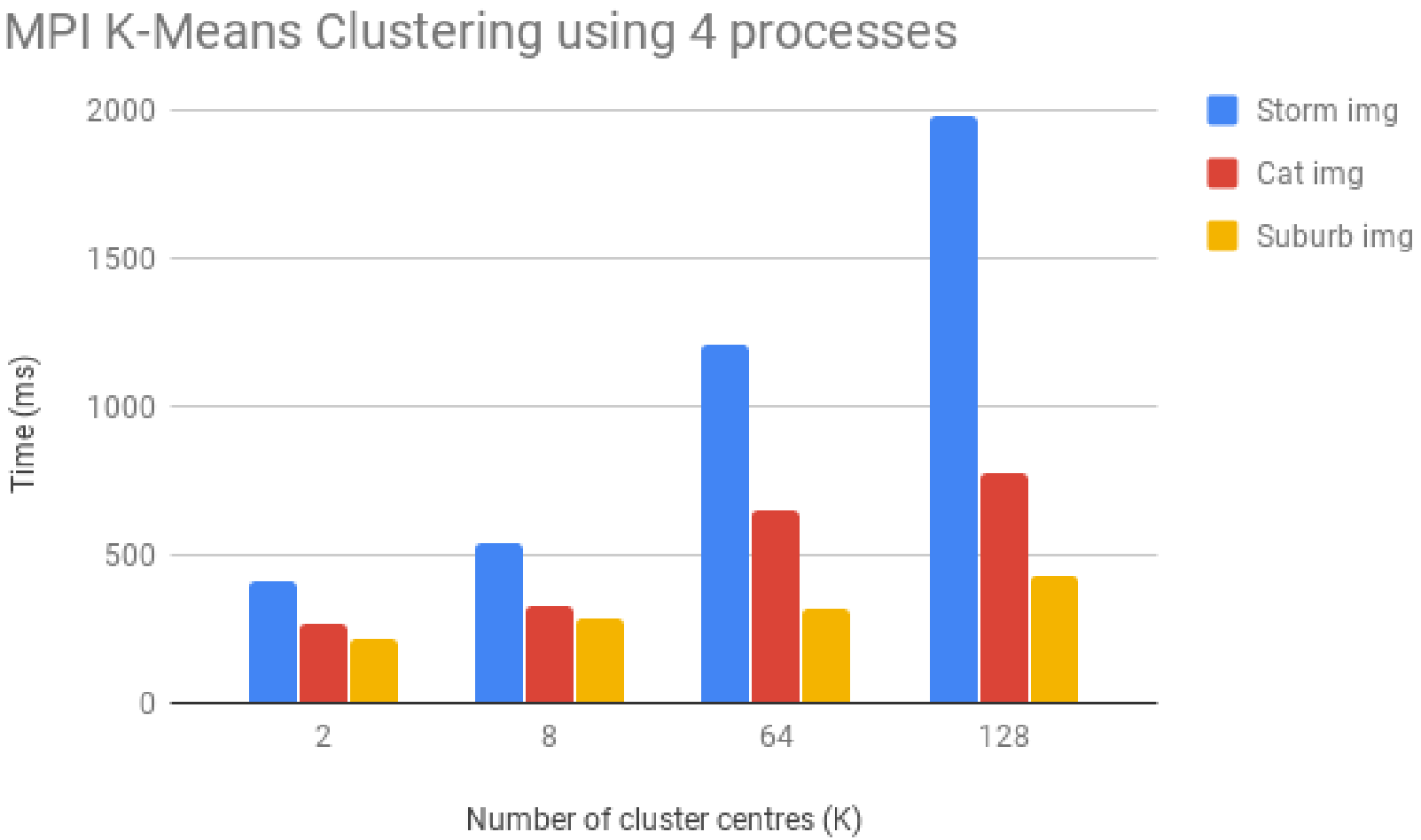
Results: K-Means Clustering

The following depicts the results of running the Cuda implementation of the K-Means clustering algorithm on various colour images for multiple K values.



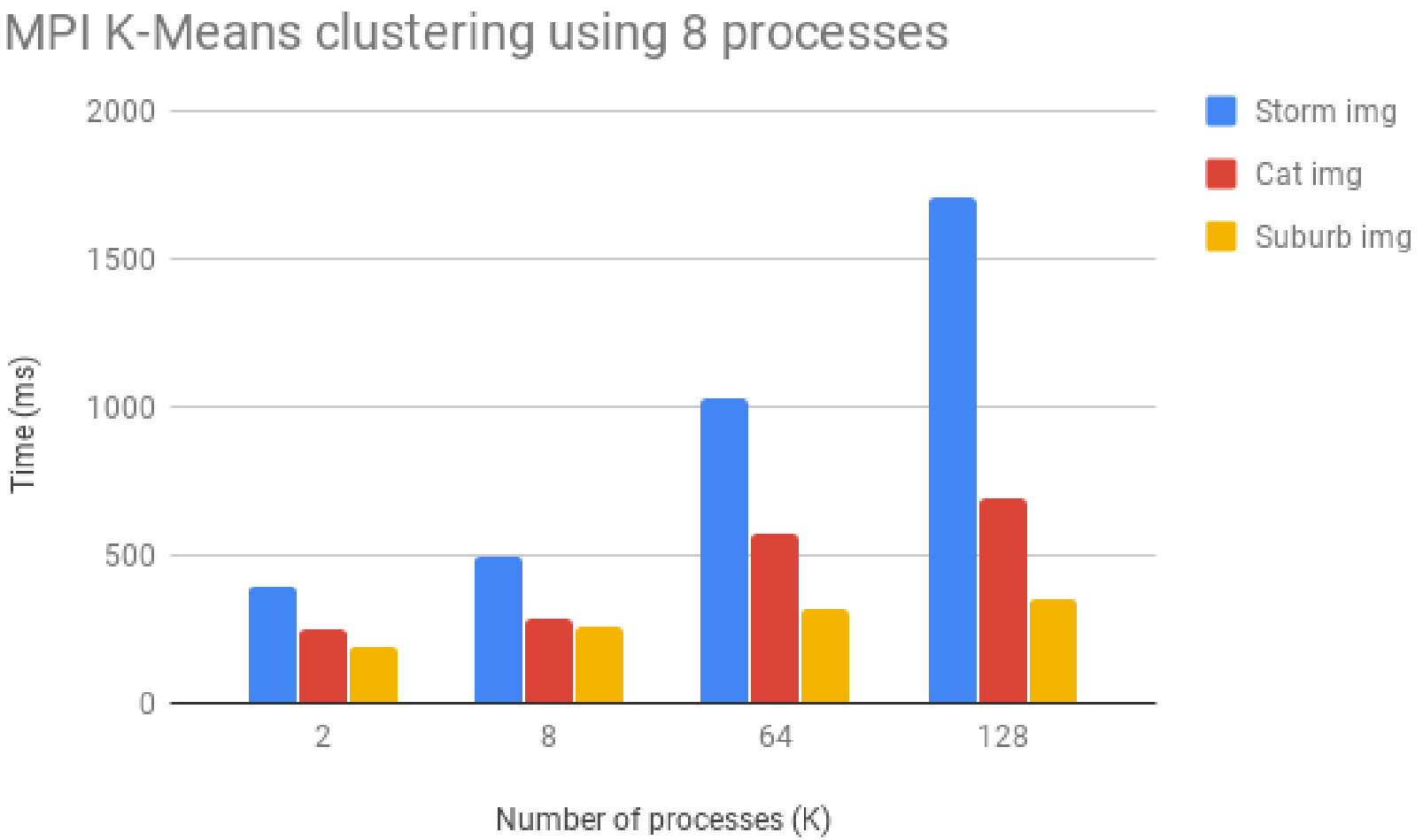
Results: K-Means Clustering

This graph shows the run times when running the MPI implementation of the K-Means clustering algorithm with 4 processes for multiple K values.



Results: K-Means Clustering

The following shows run times when running the MPI implementation of the K-Means clustering algorithm with 8 processes for multiple K values.



Evaluation: K-Means Clustering

The results for the serial K-Means clustering implementation shows that as the number of cluster centres increases for a given image, the run time increases as well. As expected, run time increases with increasing image size for the same K values.
Similar trends hold for the MPI implementations. The overall MPI run times decreased when 8 processes were used instead of 4. This decrease in total run time occurs despite an increase in the overhead costs as number of processes increases. MPI vs serial on the storm, cat and suburb images yield a speedup of 2.18, 2.96 and 2.69 respectively at K=128.
However with the CUDA implementation, for increasing K values the run times remain similar. The changes in run time are negligible for the K values as GPUs are very efficient for processing images in parallel. The CUDA implementation on storm, cat and suburb images yield a speedup of 16.21, 14.88 and 9.88 respectively at K=128.

Results: Nick Thresholding

Figure: MRI Image of size 225x225

Results: Nick Thresholding Cont...

Figure: Lena Image of size 512x512

Results: Nick Thresholding Cont...

The Call—Chronicle—Examiner

SAN FRANCISCO, THURSDAY, APRIL 19, 1906.

EARTHQUAKE AND FIRE: SAN FRANCISCO IN RUINS

NO HOPE LEFT FOR SAFETY OF ANY BUILDINGS

BLOW BUILDINGS UP TO CHECK FLAMES

WHOLE CITY IS ABLAZE

CHURCH OF SAINT IGNATIUS IS DESTROYED

MAYOR CONFERS WITH MILITARY AND CITIZENS

The Call—Chronicle—Examiner

SAN FRANCISCO, THURSDAY, APRIL 19, 1906.

EARTHQUAKE AND FIRE: SAN FRANCISCO IN RUINS

NO HOPE LEFT FOR SAFETY OF ANY BUILDINGS

BLOW BUILDINGS UP TO CHECK FLAMES

WHOLE CITY IS ABLAZE

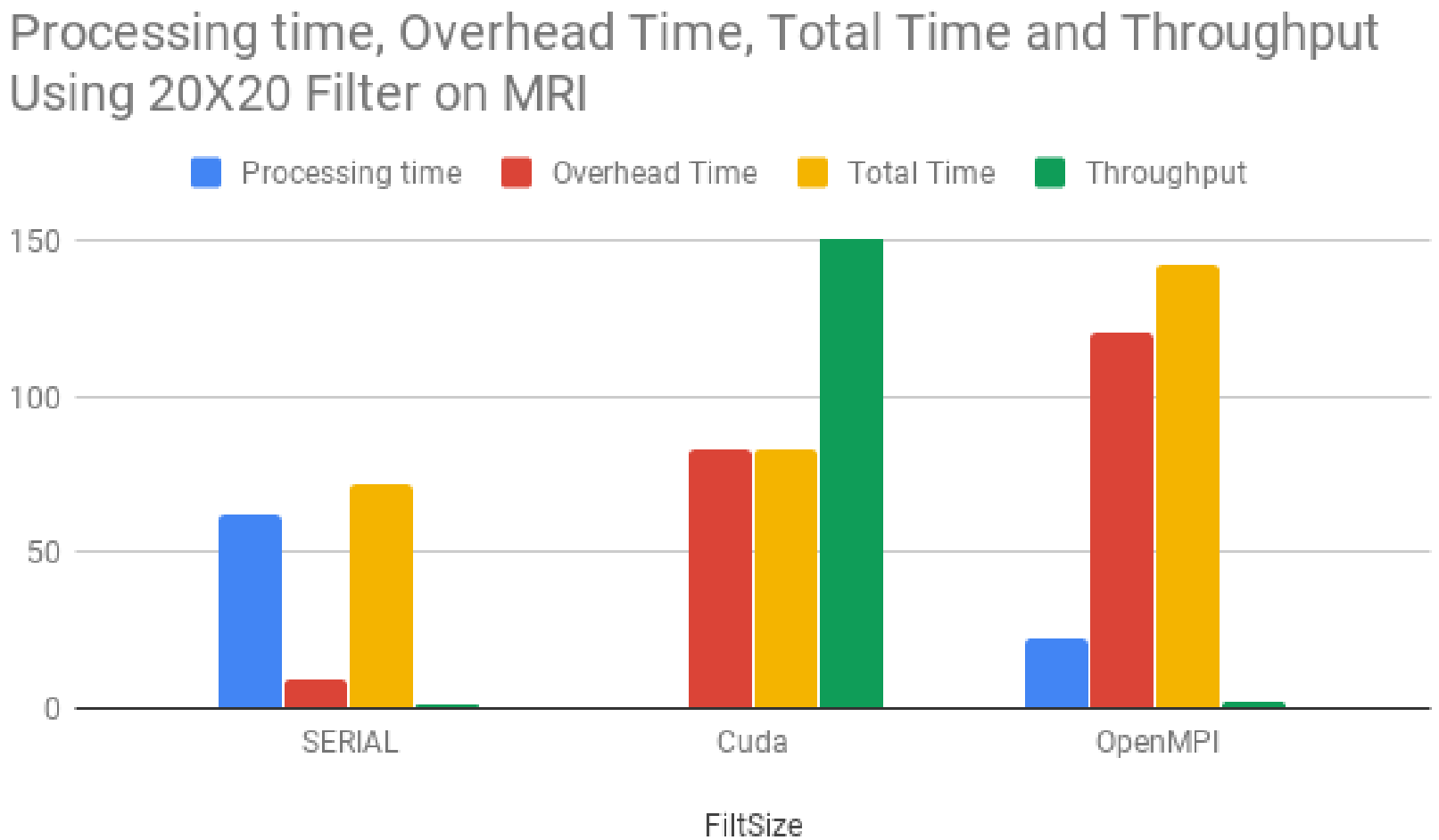
CHURCH OF SAINT IGNATIUS IS DESTROYED

MAYOR CONFERS WITH MILITARY AND CITIZENS

Figure: Newspaper Image of size 1080x1080

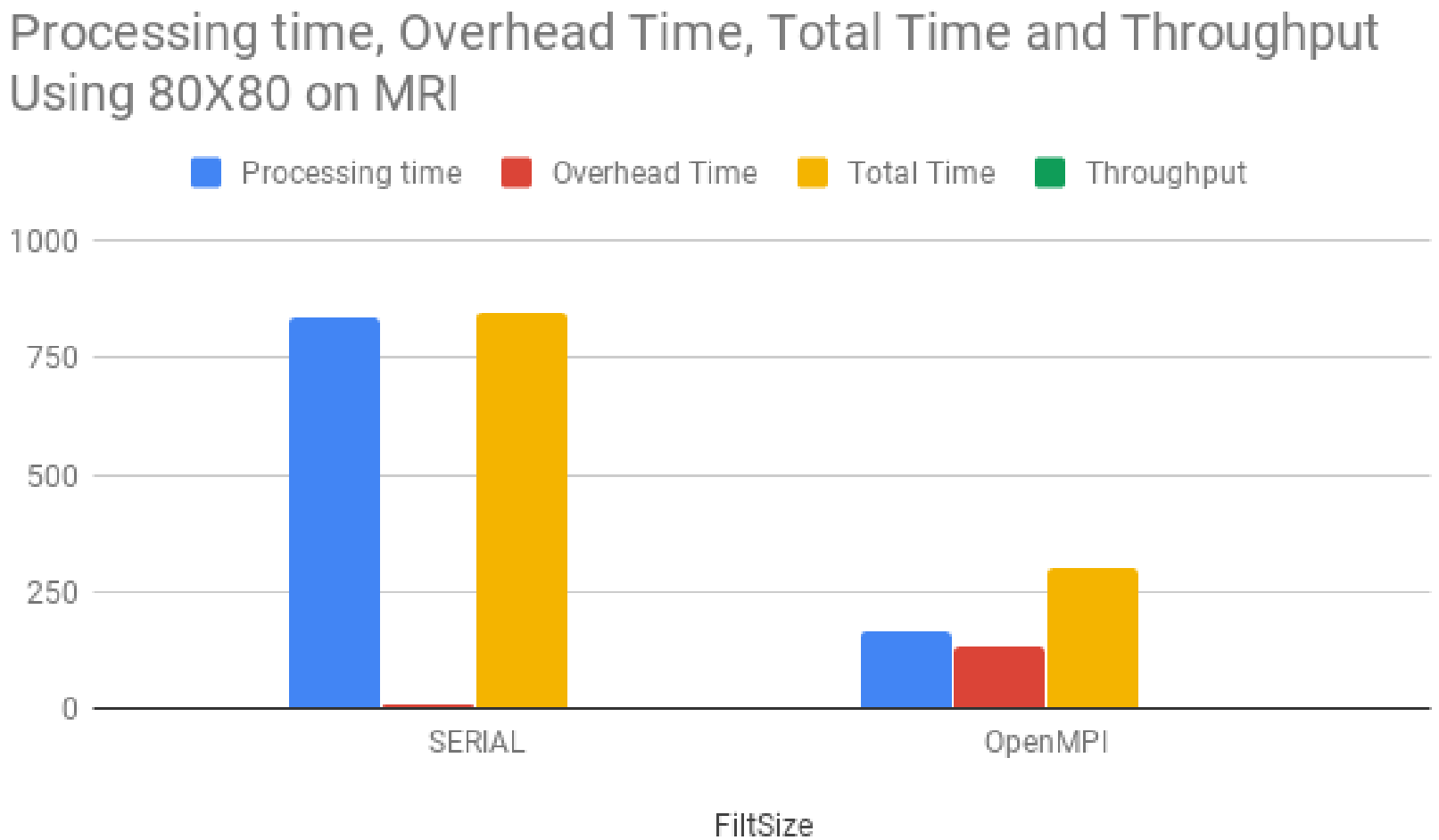
Results: Nick Thresholding Cont...

The Following Graph below is a result of Running Nick Thresholding on MRI Which is of size 225x225 using a 20x20, 40x40, 60x60 and 80x80 filter.



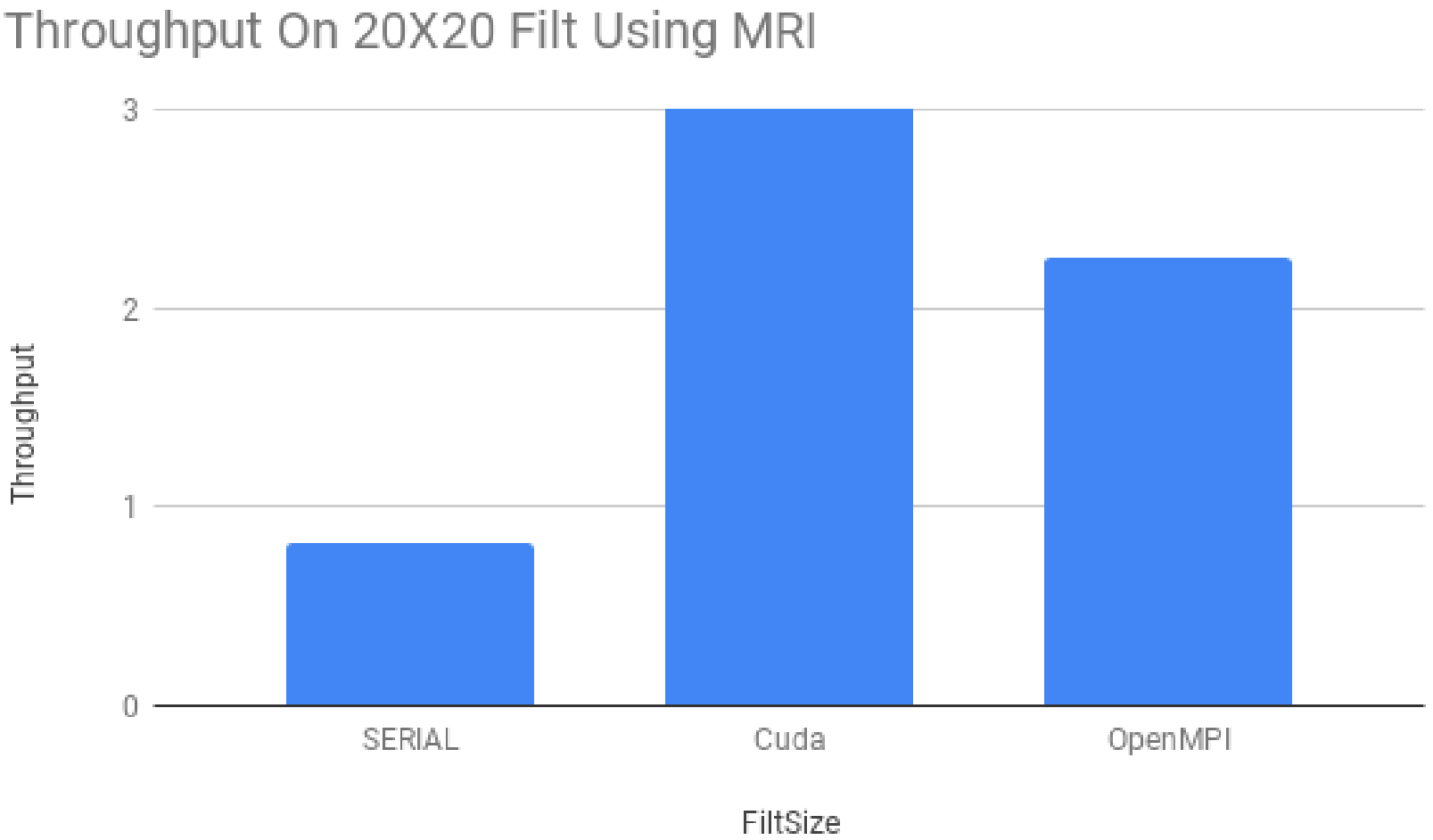
Results: Nick Thresholding Cont...

The Following Graph below is a result of Running Nick Thresholding on MRI Which is of size 225x225 using a 20x20, 40x40, 60x60 and 80x80 filter.



Results: Nick Thresholding Cont...

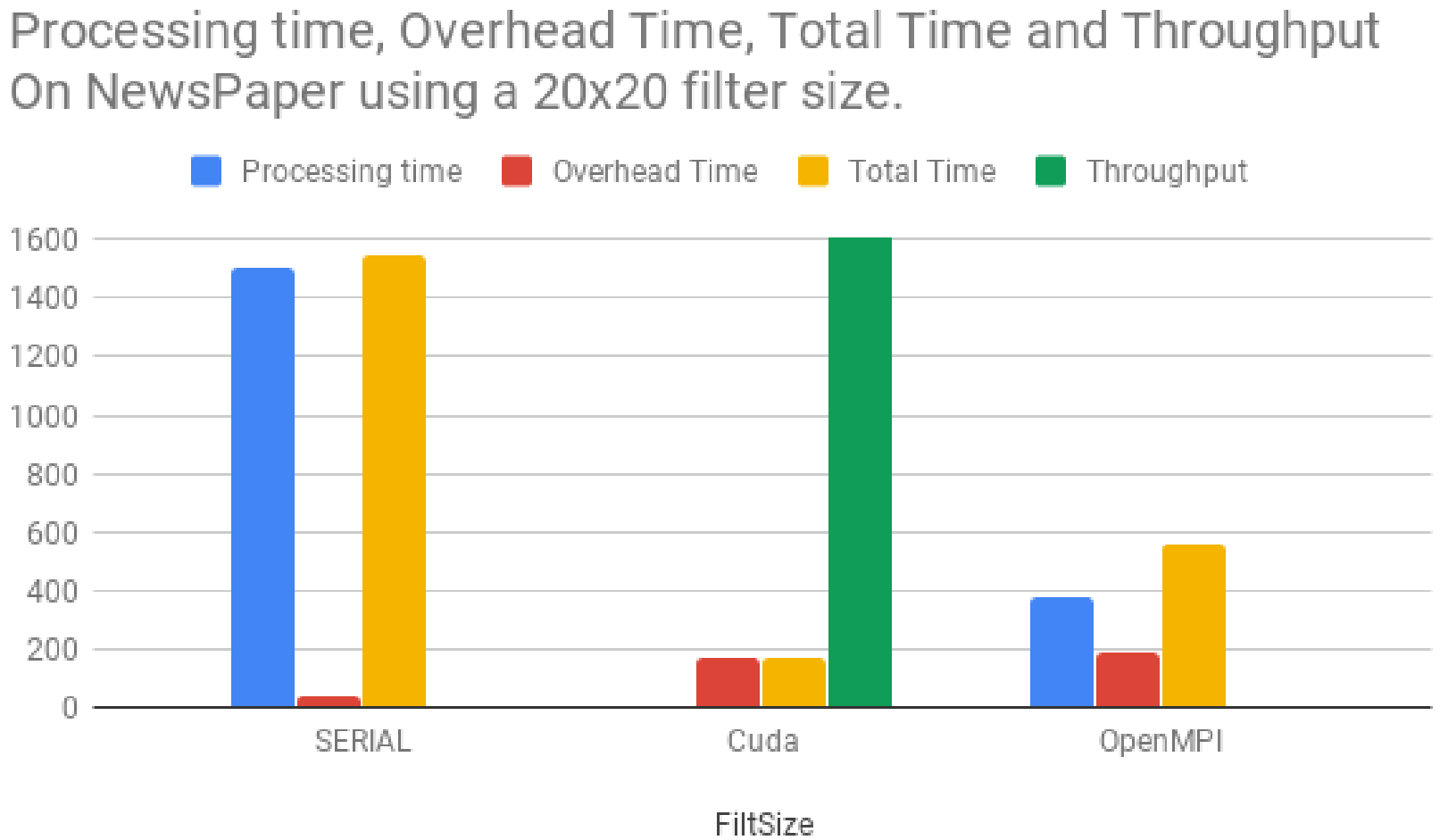
The Following is a comparison of throughput, in MPIXELS/sec, using the 20x20 Filter



The same trend follows using a 40x40,60x60 and 80x80 filter.

Results: Nick Thresholding Cont...

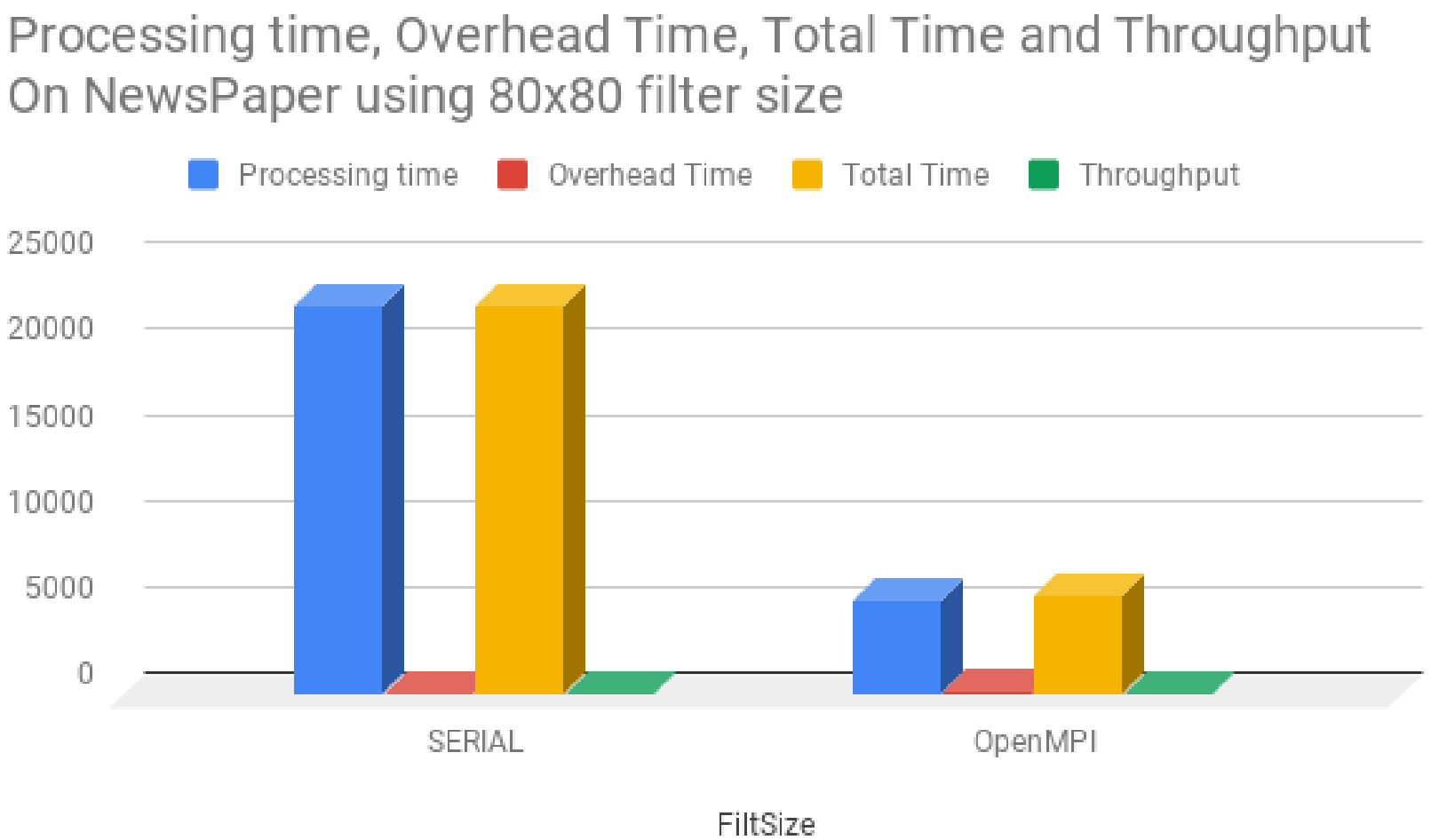
The following graph below is a result of running Nick Thresholding on Image: NewsPaper of size 1080x1080. using a 20x20 filter



A similar trend follows on a 80x80 filter.

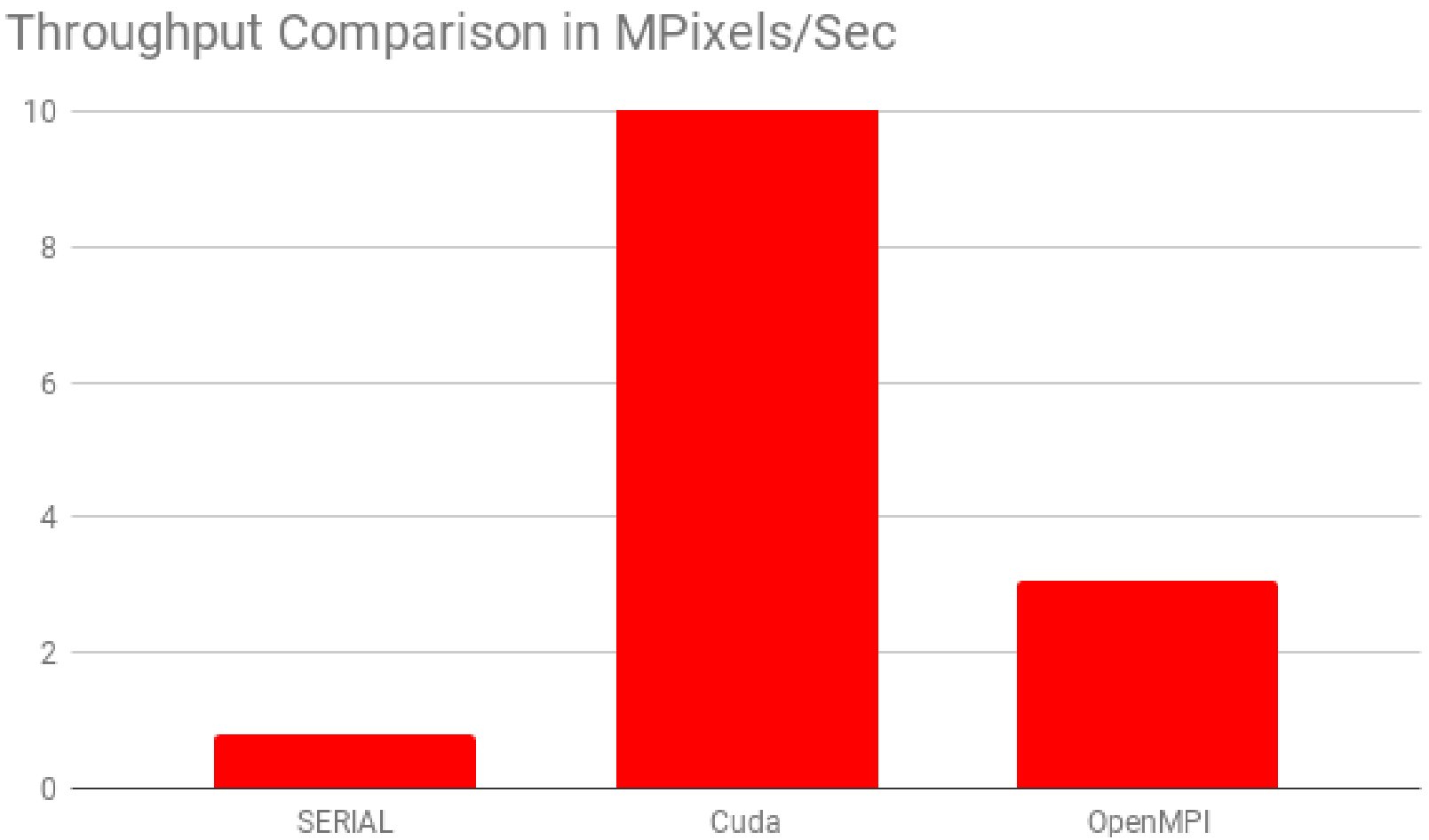
Results: Nick Thresholding Cont...

The following graph below is a result of running Nick Thresholding on Image: NewsPaper of size 1080x1080. using a 80x80 filter



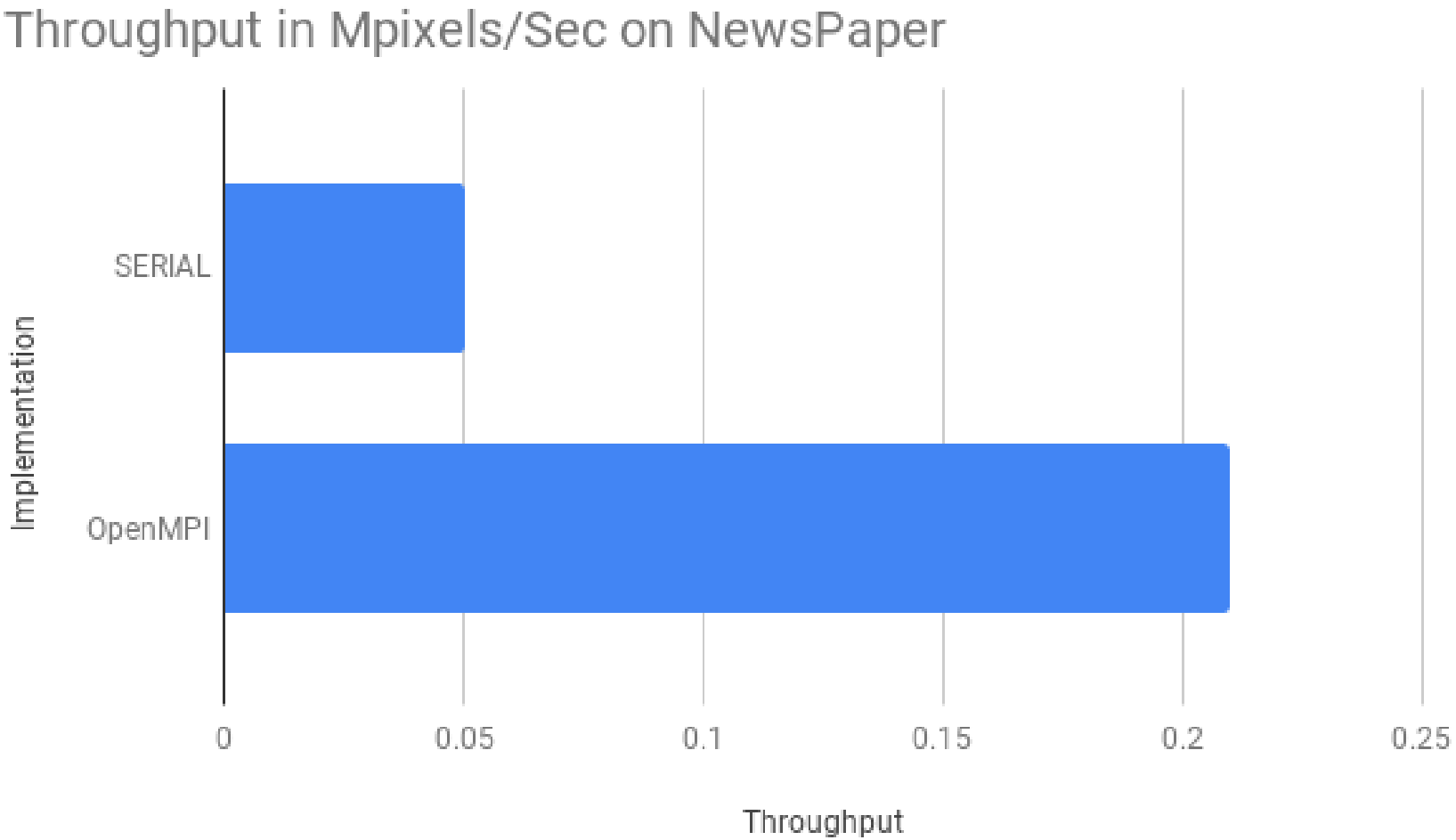
The next slide depicts the throughput in MPIXELS/SEC.

Results: Nick Thresholding Cont...



The same trend follows using a 80x80 filter

Results: Nick Thresholding Cont...



Evaluation: Nick Threshold

In the Graphs below, the shared memory implementation of the cuda kernel outperforms the serial and openMpi implementations. This makes it the most efficient implementation, however, the kernel is implemented such that each thread loads in one element from global memory to shared memory. Since the max threads for a block is 1024, $BLOCKSIZE + WINDOWSIZE <= 32$, Since its $(BLOCKSIZE + WINDOWSIZE) * (BLOCKSIZE + WINDOWSIZE)$. For window sizes bigger than the one mentioned above an OpenMPI immplementation can be used. Analyzing the results above we can see that OpenMPI does give a significant speedup. However on window sizes less 32 a cuda-shared memory approach is the most efficient, with a throughput far greater than serial and OpenMpi. Processing times for Cuda is extremely small since global memory accesses are greatly reduced by using the shared memory approach
Furthermore, the Speedup compared to serial using OpenMPI using a 20x20 Filter size on NewsPaper image is 2.75x. Using Cuda On the NewsPaper image, using 20x20 Filter size the speedup compared to serial is 9.07x

Results: Canny Edge Detection

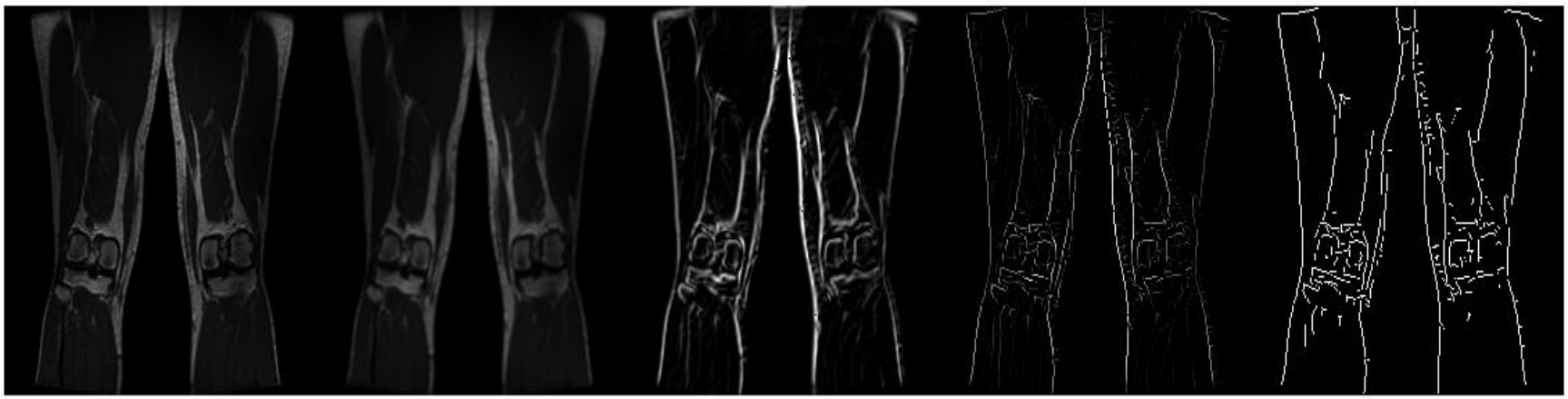


Figure: Canny Edge Detection on image MRI, from left, Original,Gaussian,Gradient Magnitude,NMS,hysteresis

Results: Canny Edge Detection Cont...

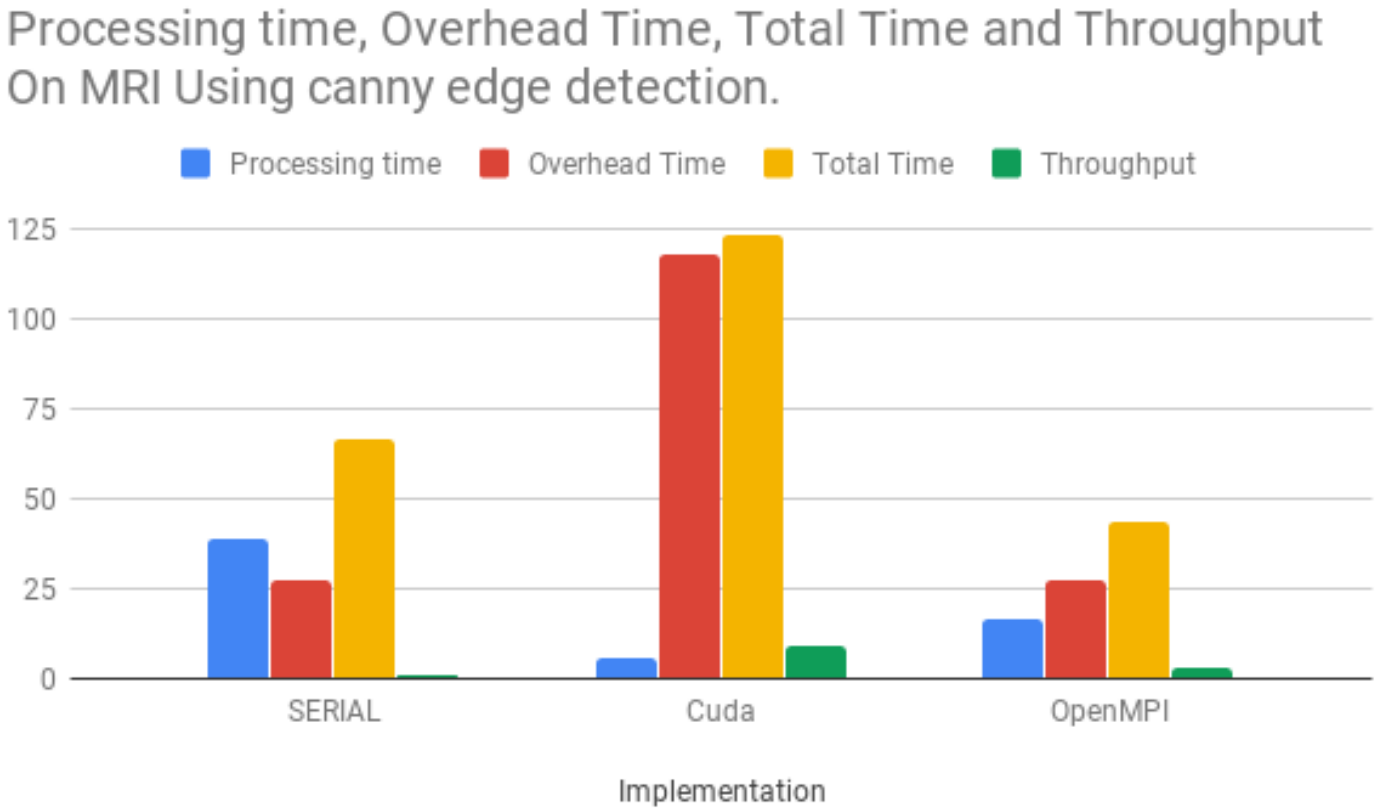
Figure: Canny Edge Detection on image Lena, from left, Original,Gaussian,Gradient Magnitude,NMS,hysteresis

Results: Canny Edge Detection Cont...

Figure: Canny Edge Detection on image NewsPaper, from left, Original,Gaussian,Gradient Magnitude,NMS,hysteresis

Results: Canny Edge Detection Cont...

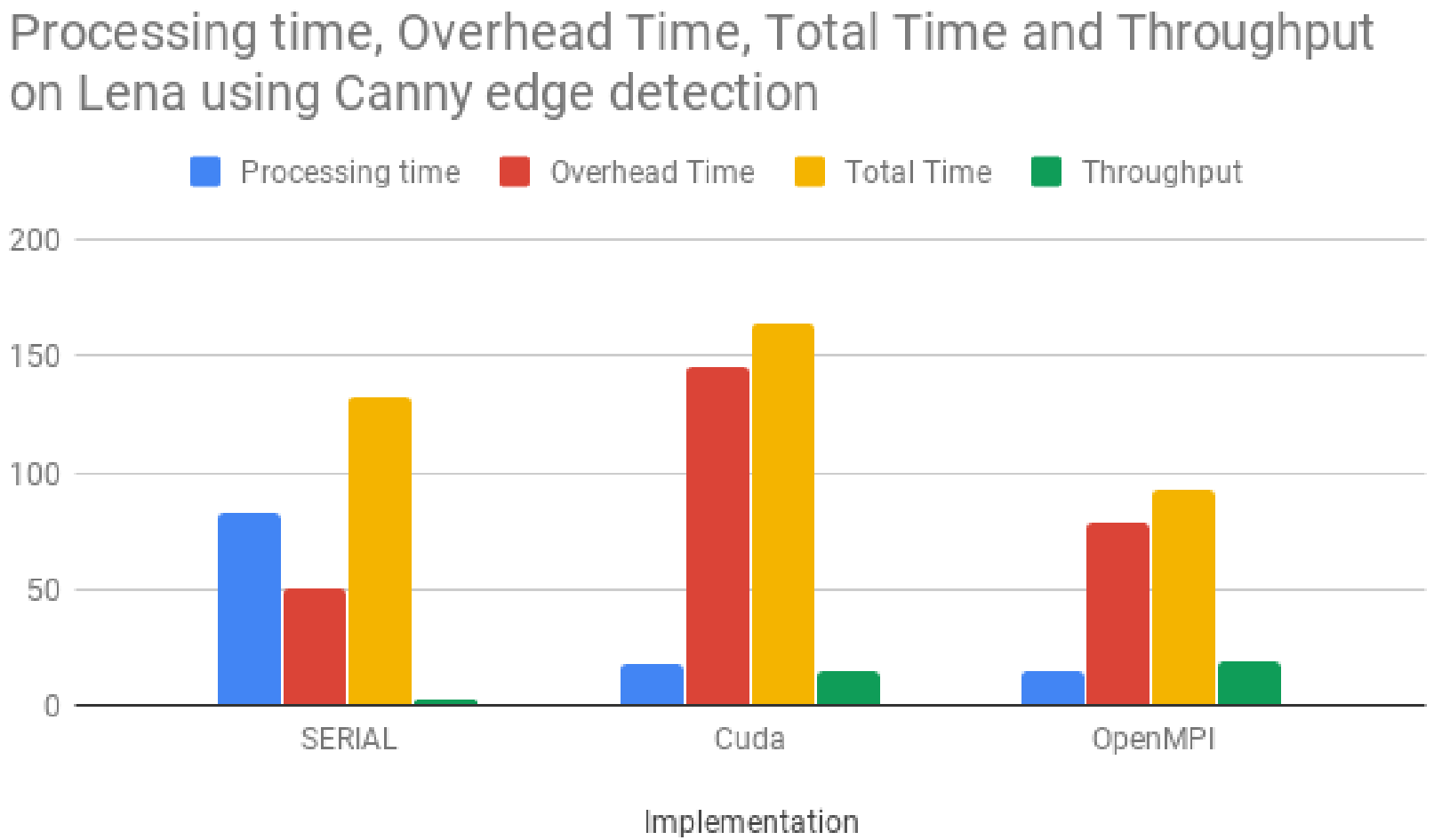
The following depicts Canny edge detection on MRI of size 225X225.



As seen Above, Cuda has the greatest overhead time because of the multiple images that need to be copied from and to the kernel. On a relatively small image the serial implementation outperforms cuda but not MPI.

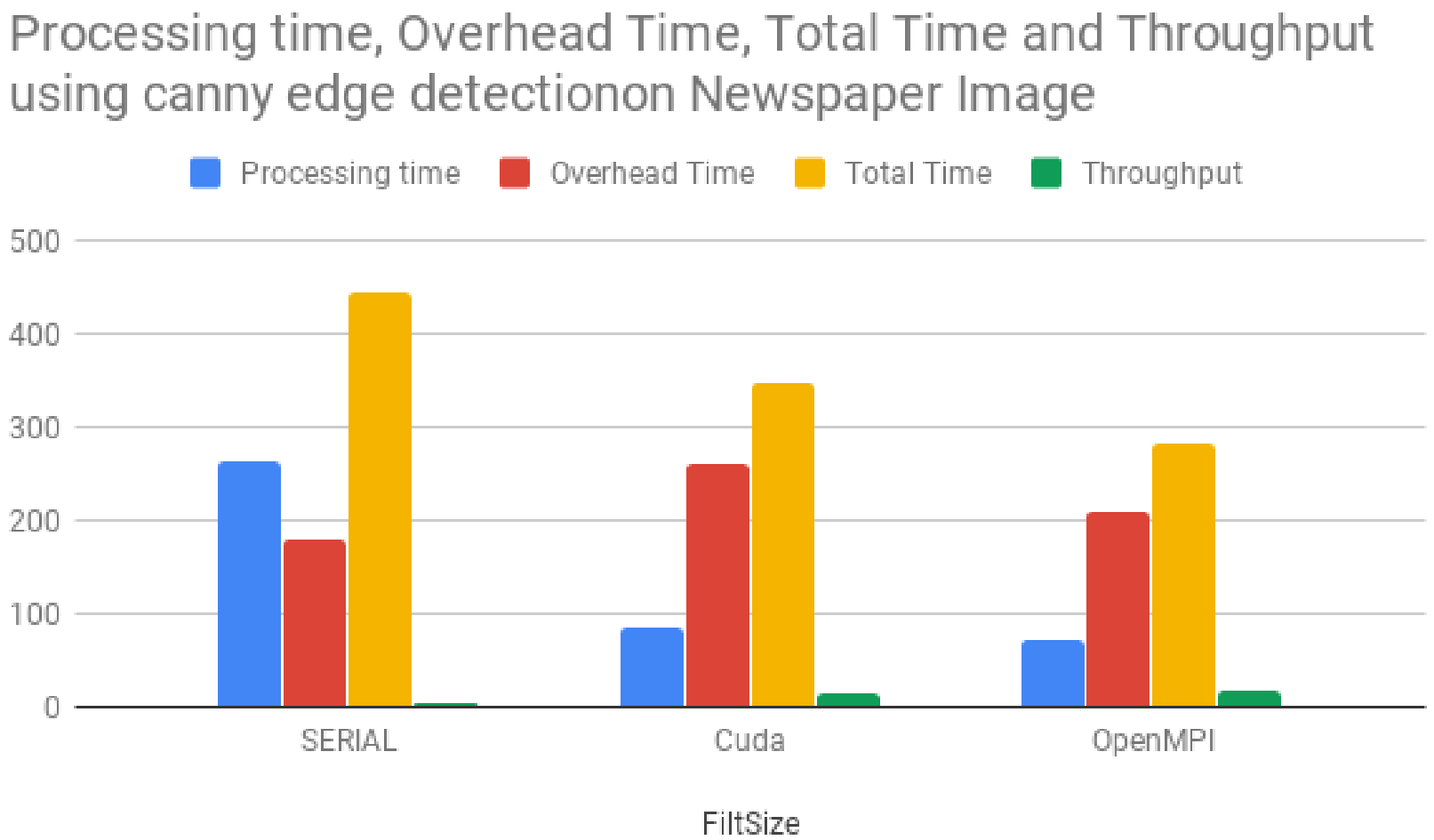
Results: Canny Edge Detection Cont...

The following depicts Canny edge detection on LENA of size 512x512.



Results: Canny Edge Detection Cont...

The following depicts Canny edge detection on NewsPaper of size 1080x1080.



Evaluation: Canny Edge Detection

As seen in the trend, while the cuda kernel outperforms Serial on larger images smaller images may run faster in serial. The Cuda kernel can be used when the processing time of the serial is much higher than the overhead time with its processing time for Cuda.
MPI is the fastest implementation on the specific images since it has a low overhead cost as well as a processing time. However this time scales up as the image become larger. On larger images or Filter sizes (Averaging Filter) Mpi may become slower than cuda
Furthermore The speedup using cuda on NewsPaper Image is 1.28x and using OpenMPI on NewsPaper Image is 1.58x