CHAPTER 1

INTRODUCTION

1.1 BASICS OF DIGITAL IMAGE

1.1.1 Digital Image

A digital image is a numeric representation of (normally binary) a two-dimensional image. It is a 2D discrete signal or an N x N array of elements. Each element in an array is a number which represents the sampled intensity.

A digital image consists of picture elements called pixels. Pixels are the smallest sample of an image. A pixel represents the brightness at one point.

1.1.2 Types of Images

1.1.2.1 Binary Image

Fig 1.1: Example of Binary Image

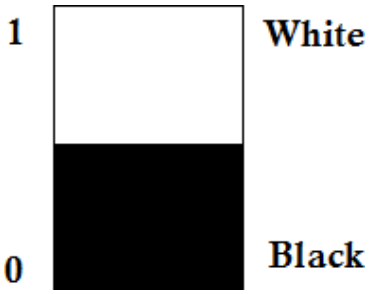


Fig 1.2: Intensity of Binary Image

As the name suggests, these are the images in which each pixel is represented by two values (0 and 1). It is also known as Black and White images. A binary image can be obtained by applying suitable threshold to a grayscale image. These images often arise as a result of certain operations in digital image processing and also in some input/output devices.

1.1.2.2 Gray-scale Image

The gray-scale image carries only intensity information and consists of only gray tones of colors. Intensity information is stored as an 8-bit integer which results in 256 possible shades of gray.

Fig1.3: Example of Grayscale Image

Color intensity	DEC	HEX	BIN
	0	0x00	00000000
	16	0x10	00010000
	32	0x20	00100000
	48	0x30	00110000
	64	0x40	01000000
	80	0x50	01010000
	96	0x60	01100000
	112	0x70	01110000
	128	0x80	10000000
	144	0x90	10010000
	160	0xA0	10100000
	176	0xB0	10110000
	192	0xC0	11000000
	208	0xD0	11010000
	224	0xE0	11100000
	255	0xFF	11111111

Fig1. 4: Intensity of Grayscale Image

1.1.2.3 Color Image

A color image consists of three values per pixel which give information about the intensity and chrominance of light. Common color spaces are RGB (Red, Green, and Blue), HSV (Hue, Saturation, and Value), and CMYK (Cyan, Magenta, Yellow, and Black).

Fig1.5: Example of Color Image

1.1.3 Image Smoothing Techniques

Smoothing, also called blurring, is a simple and frequently used image processing operation. There are many reasons for smoothing. Removal of noise is one of them.

Smoothing operation can be done by applying a filter to our image. Most common filters are Normalized Box Filter, Gaussian Filter, and Median Filter.

1.1.3.1 Normalized Box Filter

This filter is the simplest of all. Each output pixel is the *mean* of its kernel neighbors (all of them contribute with equal weights)

1.1.3.2 Gaussian Filter

Gaussian filter is probably the most useful filter (although not the fastest). Gaussian filtering is done by convolving each point in the input array with a *Gaussian kernel* and then summing them all to produce the output array.

1.1.3.3 Median Filter

The median filter run through each element of the signal (in this case the image) and replaces each pixel with the median of its neighboring pixels (located in a square neighborhood around the evaluated pixel).

CHAPTER 2

OVERVIEW

2.1 Types of Noise [1]

1. Gaussian Noise:

Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors. Source: thermal vibration of atoms and discrete nature of radiation of warm objects.

Fig.2.1 Plot of Probability Distribution Function

The side image is a bell-shaped probability distribution function which have mean 0 and standard deviation (sigma) 1.

2. Impulse Noise:

Impulse Function: In the discrete world impulse function on a value of 1 at a single location and in continuous world impulse function is an idealised function having unit area.

Fig2.2: Impulse function in discrete world and continuous world

2.1 Types of Impulse Noise:

There are three types of impulse noises. Salt Noise, Pepper Noise, Salt and Pepper Noise.

Salt Noise: Salt noise is added to an image by addition of random bright (with 255-pixel value) all over the image.

Pepper Noise: Salt noise is added to an image by addition of random dark (with 0-pixel value) all over the image.

Salt and Pepper Noise: Salt and Pepper noise is added to an image by addition of both random bright (with 255-pixel value) and random dark (with 0-pixel value) all over the image. This model is also known as data drop noise because statistically it drops the original data values.

3. Poisson Noise:

The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise.

4. Speckle Noise

A fundamental problem in optical and digital holography is the presence of speckle noise in the image reconstruction process. Speckle is a granular noise that inherently exists in an image and degrades its quality. Speckle noise can be generated by multiplying random pixel values with different pixels of an image**.**

2.2 Literature Survey

In this section we shall look at some published literature, so that we can formulate our work

Author	Title	Key points
[8]Prateek Kumar Garg, Pushpneel Verma, Ankur Bhardwaj	A survey paper on Various Median Filtering Techniques for Noise Removal from Digital Images	Median based filter reduces both bipolar and unipolar impulse noise they replace centre pixel with a value equal to median of all pixel in window using these filters will help to reduce least and highest intensity value pixels generally represented by impulse noise.
[9]S. Deivaalkshmi S. Sarath P.Palanisamy	Detection and removal of Salt and Pepper noise in images by improved median filter	The methodology for the removal of salt and pepper noise followed by median filtering in both binary and grey level images to overcome the limitations of linear and non-linear filters
[10]Sukhdev Singh Ghuman	Comparison of Single Core and Multicore Processor	Single core executes single instruction and multicore can execute multiple instructions at a time. So, multicore performs better when compared to single core.
[11]Ola Surakhi, Mohammad Khanafseh, Sami Sarhan	A survey on Parallel Multicore computing: Performance and improvement	Multicore processor combines multiple cores to offer good performance in terms of execution time.

[12]Marc Moreno Maza	Introduction to Multicore Programming	Programming directly on processor core is painful and error prone, hence OpenMp is used to abstract cores, handles synchronization & perform load balancing
[13]Narasimha Kaulgud Sharmila B S	Comparison of Time complexity in median filtering on multicore architecture	Time consumption is less in multicore when compared to that of single Core and Python takes less time for processing when compared with C programming because of built in libraries.
[14]Alan Edelman	Julia: A Fresh Approach to Parallel Programming	Julia shows the fascinating link between specialization and abstraction. Specialization allows for custom treatment and abstraction allows code reuse.
[15]Tobias Knopp	Experimental Multi-threading Support for the Julia Programming Language.	Julia provides asynchronous programming facilities in the form of tasks as well as distributed multi-process parallelism.

We observe that median filtering is time consuming and is a good candidate for implementing on multicore

2.3 Median filter

Median filtering is a nonlinear neighbourhood technique which is widely used to remove noise in the image especially salt and pepper noise. Furthermore, median filtering techniques is efficient technique compared to other linear techniques which preserve the edges of the image. However, in the median filtering majority of the time will be spent on calculating a median of each window which results in high computation and time**.** In median filtering technique, the pixel values in the neighbourhood window are ranked according to intensity, and the middle

value becomes the output for the pixel under evaluation. Using median filtering there are many algorithms implemented to reduce salt and pepper noise also called as impulse noise but which results in high computational time because to find the median it has to sort all the pixel values in increasing order which is relatively slow even with fast algorithm like quick sort.

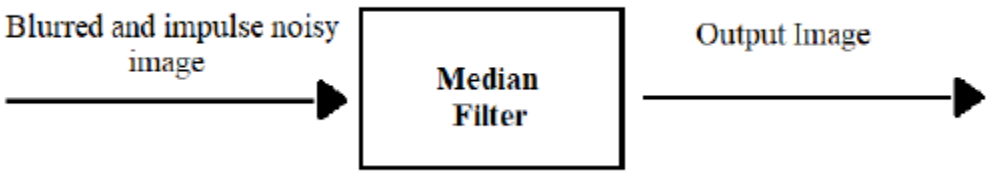


Fig2.3: block diagram of median filter

2.4 Problem statement and formulation

2.4.1 Problem statement

To design an efficient algorithm for median filtering using multicore processor architecture in general purpose operating system

2.4.2 Problem formulation

Processing time is one of the main criteria in any of the process. However, in the median filtering majority of the time will be spent on calculating a median of each window which results in high computation time. So, this complexity can be overcome using all the cores present in the processor that is by the use of multicore architecture. So, the above process can be implemented depending on the number of cores present in the CPU the image is spliced. In this work we have considered 1, 2 and 4 cores of CPU, so we split the 256x256 image into 1 part or 2 part or 4 part according to the number of cores that we are working with .each parts of image is converted into array and it is given to the cores and then median filtering is performed parallelly and here the parallelism is achieved by using multiprocessing module, PyMP module and Julia.

(a) Median filter

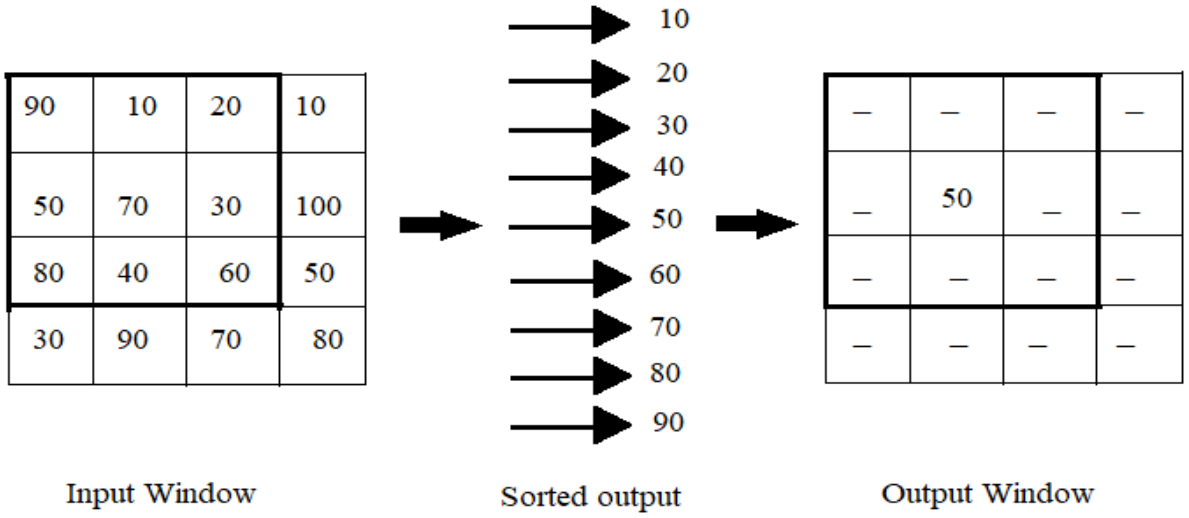


Fig: 2.4 Median Filter working

- Input image which is affected by salt and pepper noise is read.
- The image is converted to array using NumPy.
- For a window size of 3 x 3, it considers the 9 elements which is show in the above figure.
- These elements are stored in an empty array.
- Then a sorting algorithm (*ex.* Bubble sort or Quick sort) is applied to that stored empty array.
- The middle element of the window is replaced by the middle element of the sorted empty array.
- Similarly, the window slides throughout the image and removes the noise.
- If an Image is a 3D array then the median filter is applied separately to the RGB channels.

(b) Single Core

- An image which is affected by impulse noise is taken as input.
- Here we are considering single core system.
- The core is assigned with a task of removing noise from the given input image using median filter.
- The Denoised image from the core is taken.

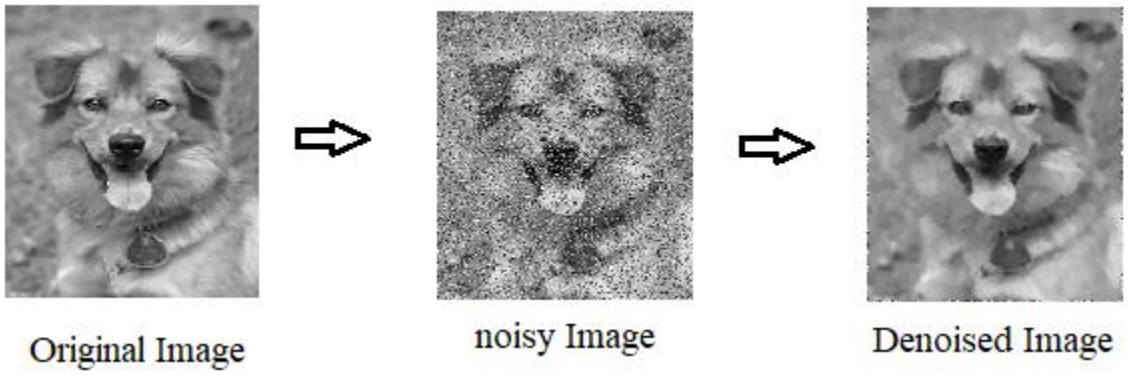


Fig 2.5: Single core working of median filter.

(c) Dual Core

- An image which is affected by impulse noise is taken as input.
- Here we are considering dual core system, so the images are split into 2 equal parts and given as input to two cores.
- Both the core is assigned with a task of removing noise from the given input image using median filter.
- Here both core work parallelly.
- The Denoised image from each core is merged together.

Fig 2.6: Dual core working of median filter.

(d) Quad Core

- An image which is affected by impulse noise is taken as input.
- Here we are considering quad core system, so the images are split into 4 equal parts and given as input to each of the cores.
- Each core is assigned with a task of removing noise from the given input image using median filter.
- Here each core work parallelly.
- The Denoised image from each core is merged together.

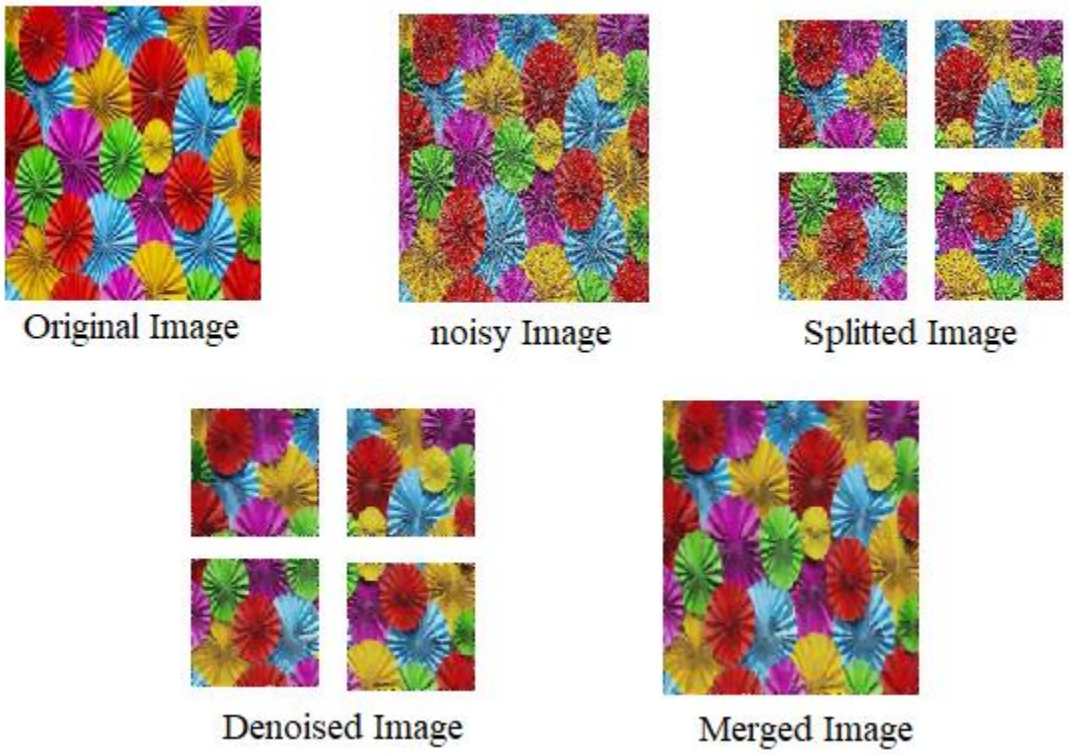


Fig 2.7: Quad core working of median filter.

2.5 TOOL SET

2.5.1 [1]Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system Conda. The

Anaconda distribution is used by over 13 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. Anaconda distribution comes with more than 1,500 packages as well as the Conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI). The big difference between Conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason Conda exists. Pip installs all Python package dependencies required, whether or not those conflict with other packages you installed previously. So, your working installation of, for example, Google Tensor Flow, can suddenly stop working when you pip install a different package that needs a different version of the NumPy library. More insidiously, everything might still appear to work but now you get different results from your data science, or you are unable to reproduce the same results elsewhere because you didn't pip install in the same order.

2.5.2 [2]Python

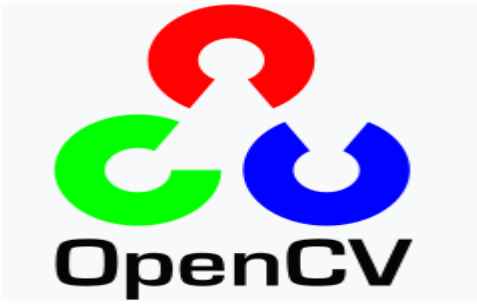
Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A

non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development

2.5.3 [3]OpenCV



OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep learning frameworks Tensor Flow, Torch/PyTorch and Caffe OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, and OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo, BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake**.**

2.5.4 [4]Spyder

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open source software. It is released under the MIT license. Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the

community. Spyder is extensible with first- and third-party plugins, includes support for interactive tools for data inspection and embeds Python-specific code quality assurance and introspection instruments, such as Pyflakes, Pylint and Rope. It is available cross-platform through Anaconda, on Windows, on macOS through MacPorts, and on major Linux distributions such as Arch Linux, Debian, Fedora, Gentoo Linux, openSUSE and Ubuntu. Spyder uses Qt for its GUI, and is designed to use either of the PyQt or PySide Python bindings. QtPy, a thin abstraction layer developed by the Spyder project and later adopted by multiple other packages, provides the flexibility to use either backend

2.5.5 [5]Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default, Jupyter Notebook ships with the IPython kernel. As of the 2.3 release (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell.

2.5.6 [6]Julia

Julia is a high-level, high-performance, dynamic programming language. While it is a general purpose language and can be used to write any application, many of its features are well-suited for high-performance numerical analysis and computational science.

Julia is garbage-collected,[24] uses eager evaluation, and includes efficient libraries for floating-point calculations, linear algebra, random number generation, and regular expression matching. Many libraries are available, including some (e.g., for fast Fourier transforms) that were previously bundled with Julia and are now separate.

Several development tools support coding in Julia, such as IDEs (e.g. Juno, and Microsoft's Visual Studio Code, with extensions adding Julia support, both providing debugging support);

[1, 2,3,4,5,6] Copyright of all the logos belongs to respective organizations

CHAPTER 3

THREADING AND FUNCTION CALL

3.1 INTRODUCTION

Threading in python is used to run multiple threads (tasks, function calls) at the same time. This does not mean that they are executed on different CPUs. Python threads will NOT make your program faster if it already uses 100 % CPU time [2].

Python threads are used in cases where the execution of a task involves some waiting. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

Function Calls: A callable object is an object that can accept some arguments (also called parameters) and possibly return an object. A function is the simplest callable object in Python and it is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing [3].

3.2 WORKING

(a) Read the image using imread().

(b) Convert the image to arrays using NumPy.

(c) Add salt and pepper noise to the image

(d) Now split the noisy image and assign to the each of the threads in case of threading and assign to a function in case of function call

(e) The noisy image is denoised using median filter in each of the threads in case of threading and it is denoised in a function in case of function call

(f) Finally, the denoised image are merged together

3.3 TIME COMPARISON

- The processing time required for filtering technique was less when multiple threads were used. But when function call method was used for the same process it took more time.
- Images of larger size took more processing time when median filtering technique was applied when compared to that of smaller images
- The time is estimated by using “Time module” in python

Table3.1: Time comparison between threads and function call

Image name	Size of Image	Threads Processing time	Function Call Processing time
Image1	256X256	0.6363s	3.0259s
Image2	512X512	0.7161s	3.9434s
Image3	1024X1024	0.7390s	5.9979

From the below tabular column we can infer that the processing time taken by the multi-threads is less than that of the single threads. Hence the processing time will be less when multi threads are used

Table3.2: Time comparison between single and multithread with respect to window size

For image size of 256 x 256,

Window Size	Processing time Of multithread	Processing time of Single thread
3 x 3	0.6142s	0.9213s
5 x 5	0.6812s	0.9454s
7 x 7	0.7232s	0.9953s

CHAPTER 4

IMPLEMENTATION USING PYTHON

4.1 MULTIPROCESSING MODULE

4.1.1 Introduction

Multiprocessing refers to the ability of a system to support more than one processor at the same time. Applications in a multiprocessing system are broken to smaller routines that run independently. The operating system allocates these threads to the processors improving performance of the system.

Consider a computer system with a single processor. If it is assigned several processes at the same time, it will have to interrupt each task and switch briefly to another, to keep all of the processes going. Hence using multiprocessing module will ease our task [4].

4.1.2 Working

(a) Import process in multiprocessing module.

(b) Read the image.

(c) Image is converted into array using NumPy.

(d) Salt and pepper noise are added to this array by invoking a function called noisy.

(e) Now the array is divided into equal parts.

(f) Now median filter functions will,

1) Accepts the splitted noisy image as input.
2) Removes the noise (denoise).

(g) The process will define the target with the function name (*ex*. Median filter, noisy) and it will also pass the arguments to the function.

(h) The process is started using process_name.start(), and ended using process_name.join().

(i) Once all the processes are executed the denoised image from all the process are merged together

(j) Processing time is noted.

4.1.3 Time Comparison

Table4.1: Time comparison of single core color image of multiprocessing module wrt window size and image size.

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Single Core Color Image (3 x 3)	256 x 256	2.6276	5.81144	10.3516
	512 x 512	9.81279	23.0434	41.4452
	768 x 768	21.8997	50.5472	97.4883
	1024 x1024	38.5431	88.9790	167.4003

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4 times of that of 3 x 3 window size.
- The time taken by 5 x 5 window size is 2.3 times of that of 3 x 3 window size.
- The time taken by 7 x 7 window size is 1.9 times of that of 5 x 5 window size.

Table4.2: Time comparison of quad core color image of multiprocessing module wrt window size and image size.

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Quad Core Color Image (3 x 3)	256 x 256	1.17632	2.6514	4.7532
	512 x 512	4.8085	10.7497	19.3705
	768 x 768	10.3027	23.8032	43.7283
	1024 x1024	18.23519	42.4893	78.3706

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4.2 times of that of 3 x 3 window size.
- The time taken by 5 x 5 window size is 2.3 times of that of 3 x 3 window size.
- The time taken by 7 x 7 window size is 1.8 times of that of 5 x 5 window size.

Table4.3: Time comparison of color image of multiprocessing module wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Color image (3 x 3)	256 x 256	2.6276	1.2797	1.1763
	512 x 512	9.8127	4.9684	4.8086
	768 x 768	21.899	10.571	10.302
	1024 x 1024	38.543	18.737	18.2351

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 2.05 times of that of dual Core.
- The time taken by Single core is 2.041 times of that of quad Core.
- The time taken by dual core is 1.0275 times of that of quad Core.

Table4.4: Time comparison of monochrome image of multiprocessing module wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Monochrome Image (3 x 3)	256 x 256	2.5797	1.3744	1.1378
	512 x 512	9.6517	4.9453	4.5952
	768 x 768	21.2275	10.239	10.099
	1024 x 1024	36.8967	18.546	17.556

Observation:

(a) For a monochrome Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.877 times of that of dual Core.
- The time taken by Single core is 2.10 times of that of quad Core.
- The time taken by dual core is 1.05 times of that of quad Core.

Table4.5: Time comparison of Grayscale image of multiprocessing module wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Gray Scale image (3 x 3)	256 x 256	2.5897	1.3739	1.1573
	512 x 512	9.7843	4.9364	4.8341
	768 x 768	21.3223	10.602	10.449
	1024 x 1024	38.2877	18.538	18.329

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.8849 times of that of dual Core.
- The time taken by Single core is 2.2377 times of that of quad Core.
- The time taken by dual core is 1.0114 times of that of quad Core.

4.2 PyMP MODULE

4.2.1 Introduction

This package brings OpenMP-like functionality to Python. It takes the good qualities of OpenMP such as minimal code changes and high efficiency and combines them with the Python Zen of code clarity and ease-of-use. The module is configurable by environment variables as well as at runtime. The PYMP variables are used with preference. At runtime, the configuration values can be set at any time by using: pymp.config. nested, pymp.config. thread_limit and pymp.config.num_threads [5].

4.2.2 Working

When entering a parallel context, processes are forked as necessary. That means that child processes are started, which are in (nearly) exactly the same state as the creating process. The memory is not copied, but referenced. Only when a process writes into a part of the memory it gets its own copy of the corresponding memory region. This keeps the processing overhead low (but of course not as low as for OpenMP threads).

Once the parallel region is left, child processes exit and only the original process 'survives'. The 'shared' data structures from the corresponding sub module are synchronized either via shared memory or using a manager process

(a) Import PyMP.
(b) Read the image.
(c) Image is converted into array using NumPy.
(d) Salt and pepper noise are added to this array.
(e) Now the array is divided into equal parts, for example consider 4 equal parts.
(f) Parallel sections are created to assign the task for the cores.
(g) When Core_id's are invoked it works on the task assigned for it, parallelly.
(h) And the task assigned for each core is to apply median filtering technique on the given input noise image.
(i) Finally, when the denoised images is obtained as the output from each of the core it is merged into a single Image.
(j) Processing time is noted.

4.2.3 Time Comparison

Table4.6: Time comparison of single core color image of PyMP module wrt window size and image size.

	Size	3x3	5x5	7x7
Single Core Color Image	256x256	1.9322	4.3804	8.5678
	512x512	7.7812	17.6028	35.7011
	768x768	17.5276	51.0755	80.4680
	1024x1024	31.3554	76.8289	147.2965

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4.4342 times of that of 3 x 3 window size.
- The time taken by 5 x 5 window size is 2.2670 times of that of 3 x 3 window size.
- The time taken by 7 x 7 window size is 1.9559 times of that of 5 x 5 window size.

Table4.7: Time comparison of dual core color image of PyMP module wrt window size and image size.

	Size	3x3	5x5	7x7
Dual Core Color image	256x256	1.0637	2.4322	4.4173
	512x512	4.3111	9.6252	17.6738
	768x768	9.5796	21.4869	39.7726
	1024x1024	16.9071	38.3955	70.8145

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4.15 times of that of 3 x 3 window size.
- The time taken by 5 x 5 window size is 2.29times of that of 3 x 3 window size.
- The time taken by 7 x 7 window size is 1.82 times of that of 5 x 5 window size.

Table4.8: Time comparison of quad core color image of PyMP module wrt window size and image size.

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Quad Core Color image	256 x 256	1.0409	2.3150	4.2549
	512 x 512	4.1587	9.1333	17.5388
	768 x 768	9.3335	20.8799	39.5432
	1024 x1024	16.4326	36.9002	70.7911

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4.09 times of that of 3 x 3 window size
- The time taken by 5 x 5 window size is 2.22 times of that of 3 x 3 window size
- The time taken by 7 x 7 window size is 1.84 times of that of 5 x 5 window size

Table4.9: Time comparison of color image of PyMP module wrt number of cores and image size.

Color Image (3 x 3)	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
	256 x 256	1.9322	1.0637	1.0409
	512 x 512	7.7812	4.3111	4.1587
	768 x 768	17.5276	9.5796	9.3335
	1024 x 1024	31.3554	16.9071	16.4326

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.82 times of that of dual Core.
- The time taken by Single core is 1.86 times of that of quad Core.
- The time taken by dual core is 1.03 times of that of quad Core.

Table4.10: Time comparison of monochrome image of multiprocessing module wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Monochrome Image (3 x 3)	256 x 256	1.8616	1.1051	1.1677
	512 x 512	7.5193	4.2563	4.1734
	768 x 768	17.3069	9.4626	9.2845
	1024 x 1024	30.6614	16.8353	16.2891

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.68 times of that of dual Core.
- The time taken by Single core is 1.59 times of that of quad Core.
- The time taken by dual core is 1.03 times of that of quad Core.

Table4.11: Time comparison of grayscale image of multiprocessing module wrt number of cores and image size

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Gray Scale Image (3 x 3)	256 x 256	1.9013	1.058	1.029
	512 x 512	7.7803	4.2797	4.18162
	768 x 768	17.3629	9.6295	9.2536
	1024 x1024	31.5574	16.8062	16.3545

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.8 times of that of dual core.
- The time taken by Single core is 1.85 times of that of quad Core.
- The time taken by dual core is 1.03 times of that of quad Core.

CHAPTER 5

IMPLEMENTATION USING JULIA

5.1 INTRODUCTION

Julia is a high-level, high-performance, dynamic programming language. While it is a general-purpose language and can be used to write any application, many of its features are well-suited for high-performance numerical analysis and computational science. Julia uses dynamic typing, resembles scripting, and has good support for interactive use. Julia supports high-level syntax which makes it an efficient language for programmers. Julia offers rich language of descriptive data types. Julia supports multiple dispatch which makes it easy to compile object-oriented and functional programming code patterns [6].

5.2 SINGLE CORE v/s MULTICORE

5.2.1 Introduction

A **single-core** processor is a microprocessor with a single core on a chip, running a single thread at any one time. Most microprocessors prior to the multi-core era are single-core. The class of many-core processors follows on from multi-core, in a progression showing increasing parallelism over time. Processors remained single-core until it was impossible to achieve performance.

A **multi-core processor** is a computer processor integrated circuit with two or more separate processing units, called cores, each of which reads and executes program instructions, as if the computer had several processors. Multi-core processors are widely used across many application domains, including general-purpose, embedded, network, digital signal processing (DSP), and graphics (GPU).

5.2.2 Working

1) Single-core

(a) Read the noisy image which is affected using salt and pepper noise.

(b)Single Core works on removal of noise in the image.

(c) Removal of noise in the image is done using median filtering technique.

(d)So, when the core completes the assigned task a denoised image is Obtained.

(e)Processing time is noted.

2) Multi-core

(a) Read the noisy image.

(b)Input image is splitted into equal parts.

(c) Julia supports parallel loops using the threads.@threads macro.

(d) This macro is affixed in front of a for loop to indicate that the loop is Multithreaded region.

(e)Threads are assigned to each core.

(f)With the help of Thread_id cores are invoked to do assign the task.

(g)And the task assigned for each core is to apply median filtering technique on the given input noise image.

(h)Finally when the denoised images is obtained as the output from each of the core it is merged into a single Image.

(i) Processing time is noted.

5.2.3 Time Comparison

Table5.1: Time comparison of single core color image of Julia wrt window size and image size

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Single Core Color Image	256x256	0.4595	0.9776	1.8875
	512x512	1.1721	3.4237	6.9268
	768x768	2.3546	7.3317	15.7351
	1024x1024	4.0359	12.8299	27.5997

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 4.12 times of that of 3 x 3 window size
- The time taken by 5 x 5 window size is 2.12 times of that of 3 x 3 window size
- The time taken by 7 x 7 window size is 1.93 times of that of 5 x 5 window size

Table5.2: Time comparison of Dual core color image of Julia wrt window size and image size.

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Dual Core Color Image	256x256	0.3379	0.5622	1.047
	512x512	0.7107	1.7019	3.3383
	768x768	1.3466	3.5395	7.4477
	1024x1024	2.0182	6.2565	13.2128

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 3.1 times of that of 3 x 3 window size
- The time taken by 5 x 5 window size is 1.66 times of that of 3 x 3 window size
- The time taken by 7 x 7 window size is 1.86 times of that of 5 x 5 window size

Table5.3: Time comparison of quad core color image of Julia wrt window size and image size

	Size	3x3 (sec)	5x5 (sec)	7x7 (sec)
Quad Core Color Image	256 x 256	0.3381	0.6007	0.9852
	512 x 512	0.7088	1.6936	3.2803
	768 x 768	1.291	3.5530	7.2116
	1024 x1024	2.1288	6.1237	12.7950

Observation:

(a) For an Image of size n x n:

- The time taken by 7 x 7 window size is 2.91 times of that of 3 x 3 window size.
- The time taken by 5 x 5 window size is 1.78 times of that of 3 x 3 window size.
- The time taken by 7 x 7 window size is 1.64 times of that of 5 x 5 window size.

Table5.4: Time comparison of color image of Julia wrt number of cores and image size

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Color Image (3 x 3)	256x256	0.4595	0.3379	0.3380
	512x512	1.1721	0.7107	0.7088
	768x768	2.3546	1.3466	1.2918
	1024x1024	4.0359	2.0182	2.1288

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.36 times of that of dual core.
- The time taken by Single core is 1.36 times of that of quad Core.
- The time taken by dual core is 1.04 times of that of quad Core.

Table5.5: Time comparison of monochrome image of Julia wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Monochrome Image (3 x 3)	256x256	0.4551	0.3417	0.3855
	512x512	1.1214	0.6517	0.6984
	768x768	2.2427	1.2275	1.2532
	1024x1024	3.8175	2.00886	2.0982

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.33 times of that of dual core.
- The time taken by Single core is 1.18 times of that of quad Core.
- The time taken by dual core is 1.0 times of that of quad Core.

Table5.6: Time comparison of grayscale image of Julia wrt number of cores and image size.

	Size	Single Core (sec)	Dual Core (sec)	Quad Core (sec)
Gray Scale Image (3 x 3)	256x256	0.4505	0.3406	0.39042
	512x512	1.1978	0.6997	0.7249
	768x768	2.2998	1.2807	1.3264
	1024x1024	3.9388	2.0983	2.1377

Observation:

(a) For a Color Image of size n x n and a window size of 3 x 3:

- The time taken by Single core is 1.3 times of that of dual core.
- The time taken by Single core is 1.15 times of that of quad Core.
- The time taken by dual core is 1.0 times of that of quad Core.

5.3 MULTICORE

5.3.1 Split image is given to 12 Cores

5.3.1.1 Working

(a) Read the noisy image.

(b)Input image is splitted into 12 equal parts.

(c) Julia supports parallel loops using the Threads.@threads macro.

(d) This macro is affixed in front of a for loop to indicate that the loop is Multithreaded region.

(e)Threads are assigned to each core.

(f)With the help of Thread_id cores are invoked to do assign the task.

(h)And the task assigned for each core is to apply median filtering technique on the given input noise image.

(i)Finally when the denoised images is obtained as the output from each of the core it is merged into a single Image.

5.3.1.2 Time Comparison

(*Intermediate results)

Table 5.7: Time comparison of grayscale image of Julia wrt number of cores and window size.

	Window Size	6 core (sec)	8 core (sec)	12 core (sec)
Color Image 250X250	3 x 3	0.244	0.236	0.2
	5 x 5	0.381	0.34	0.28
	7 x 7	0.66	0.521	0.36

Observation:

(a) For a Color Image of size n x n :

- With the increase in window size the time taken by core to complete the assigned task increases.
- With the increase in number of cores in the system the time taken by it complete the given task decreases.

CHAPTER 6

COMPARISON OF PERFORMANCE BETWEEN PYTHON AND JULIA

6.1 Introduction

Table6.1: comparison between Julia and Python [7]:

Feature	Julia	python
Speed	Julia is much faster than Python as it has execution speed very close to that of C.	Python on the other hand is fast but is slower in comparison to C.
Community	Julia being a new language holds a community of very small size, hence resources for solving doubts and problem are not much.	Python has been around for ages, and it has a very large community of programmers. So, it becomes much easier to get your problems resolved online.
Code Conversion	Julia codes can easily be made by converting C or Python codes.	It is very difficult to make python codes by converting C codes or the other way around.
Array Indexing	Julia arrays are 1-indexed, i.e. arrays start with 1-n not 0-n. It might cause a problem With programmers having habit of using other languages.	Python arrays are 0-indexed. Generally, every language has 0-indexing for arrays.
Libraries	Julia has limited libraries to work upon. However, it can interfere with libraries of C and Fortran to handle plots.	Python on the other hand has plenty of libraries, Hence it becomes easier to perform multiple additional tasks.
Dynamically Typed	Julia is dynamically typed language, it helps Developers to create variables without Specifying their types. Julia also provides a benefit of static typing.	Python is also dynamically typed and helps in creation of variables without type declaration. It is different from Julia just because it is not statically typed.

6.2 Single Core

6.2.1 Time Comparison between python and julia

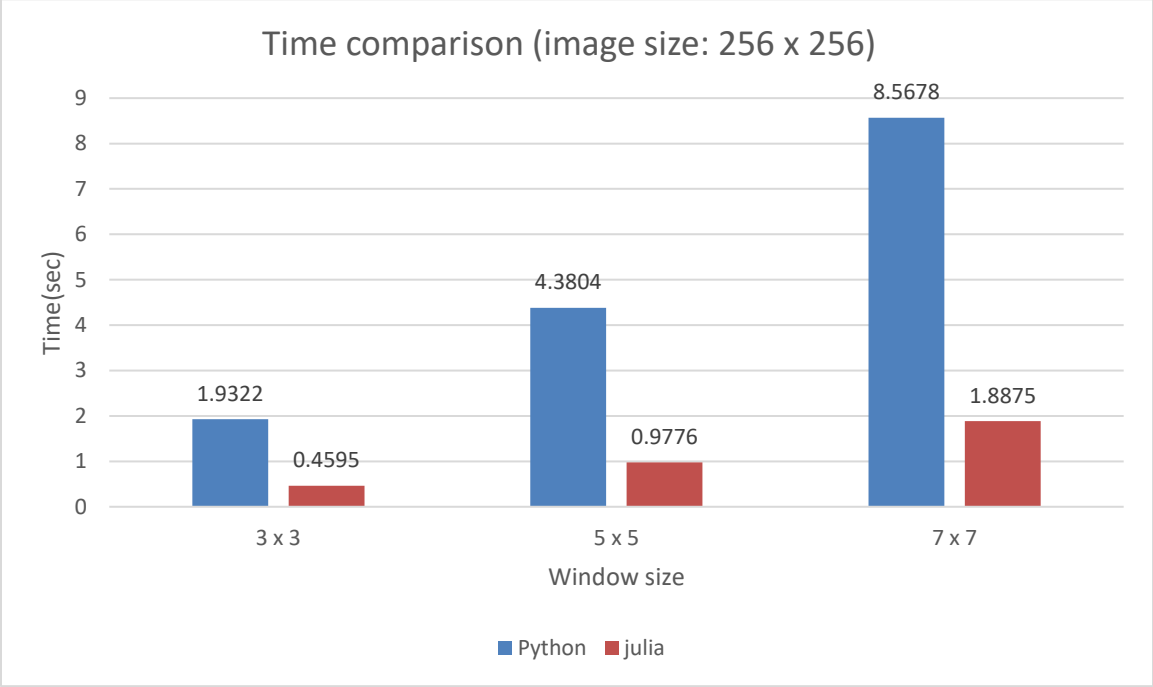


Fig6.1: time comparison between julia and python in single core

Observation:

(a) For a Color Image of size n x n:

(1) Window size 3 x 3

- The time taken by Python is 4.20 times of that of Julia.

(2) Window size 5 x 5

- The time taken by Python is 4.48 times of that of Julia.

(3) Window size 7 x 7

- The time taken by Python is 4.54 times of that of Julia.

6.3 Multi Core

6.3.1 Time Comparison between python and julia

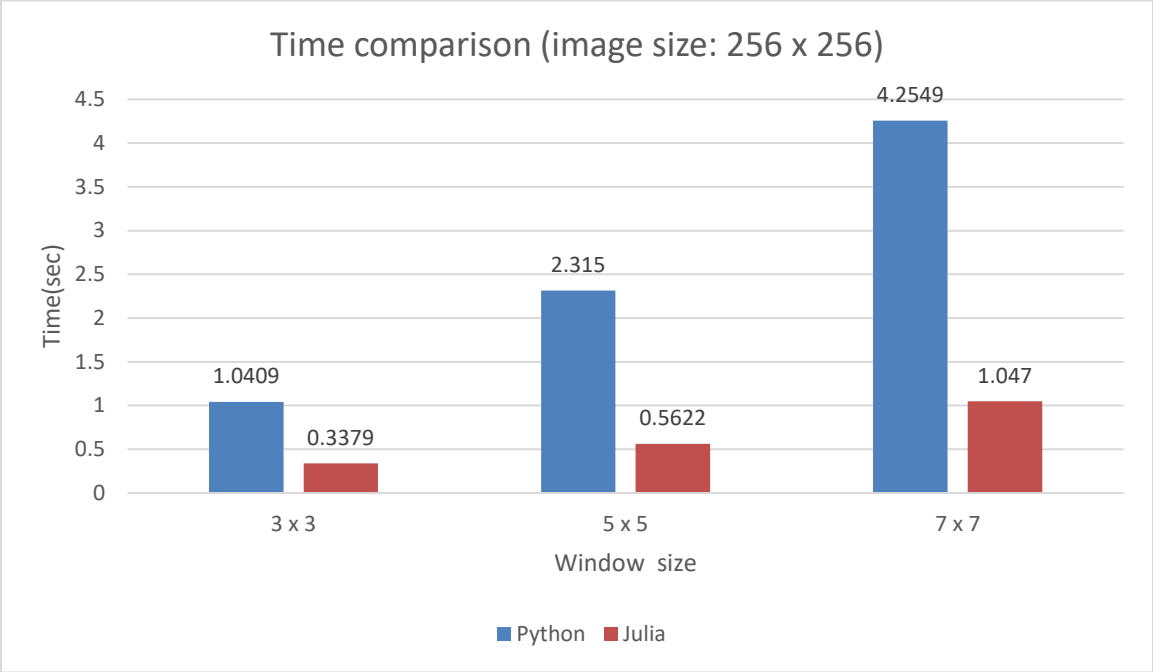


Fig6.2: time comparison between julia and python in multicore

Observation:

(a) For a Color Image of size n x n:

(1) Window size 3 x 3

- The time taken by Python is 3.08 times of that of Julia.

(2) Window size 5 x 5

- The time taken by Python is 4.12 times of that of Julia.

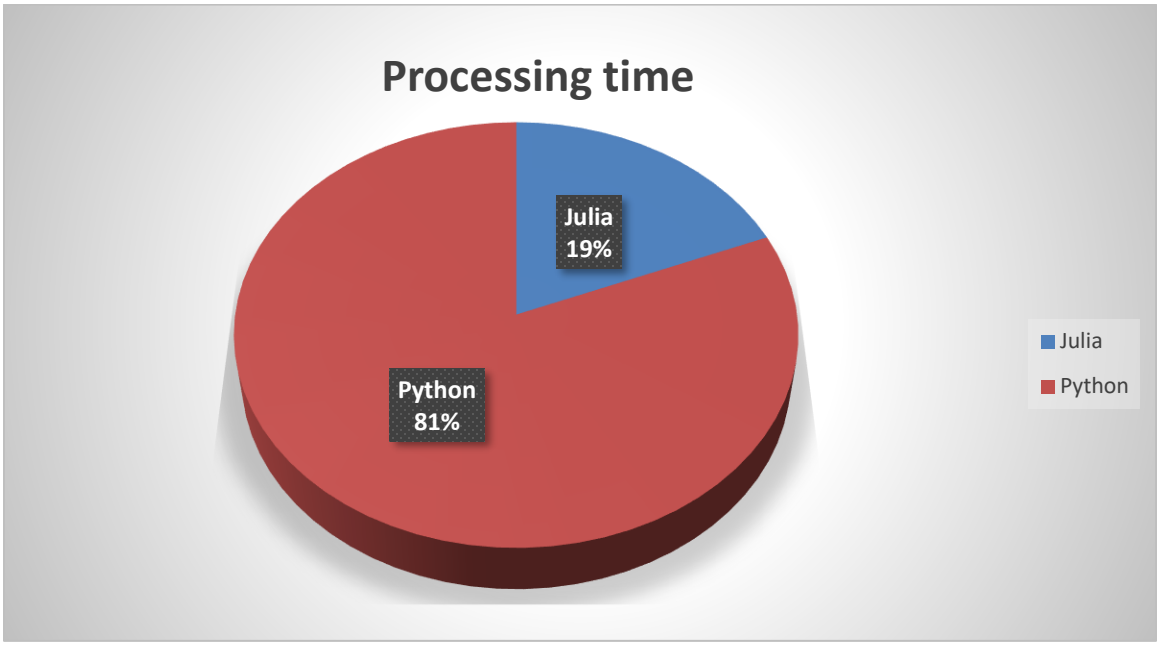
(3) Window size 7 x 7

- The time taken by Python is 4.06 times of that of Julia.

CHAPTER 7

RESULTS AND SUMMARY

Digital images are often corrupted by impulse noise (salt and pepper noise) Due to errors generated in noisy sensors, errors that occur in the process of converting signals from analog to digital converters and also errors that are generated in communication channels. In order to remove impulse noise and enhance the affected image quality, we have implemented median filtering techniques. Experimental results indicate that the proposed method of using Julia programming language performs significantly better in preserving image details and also preserving image edge information the proposed method achieves better results when applied to images corrupted by impulse noise than in the images corrupted by the Gaussian noise



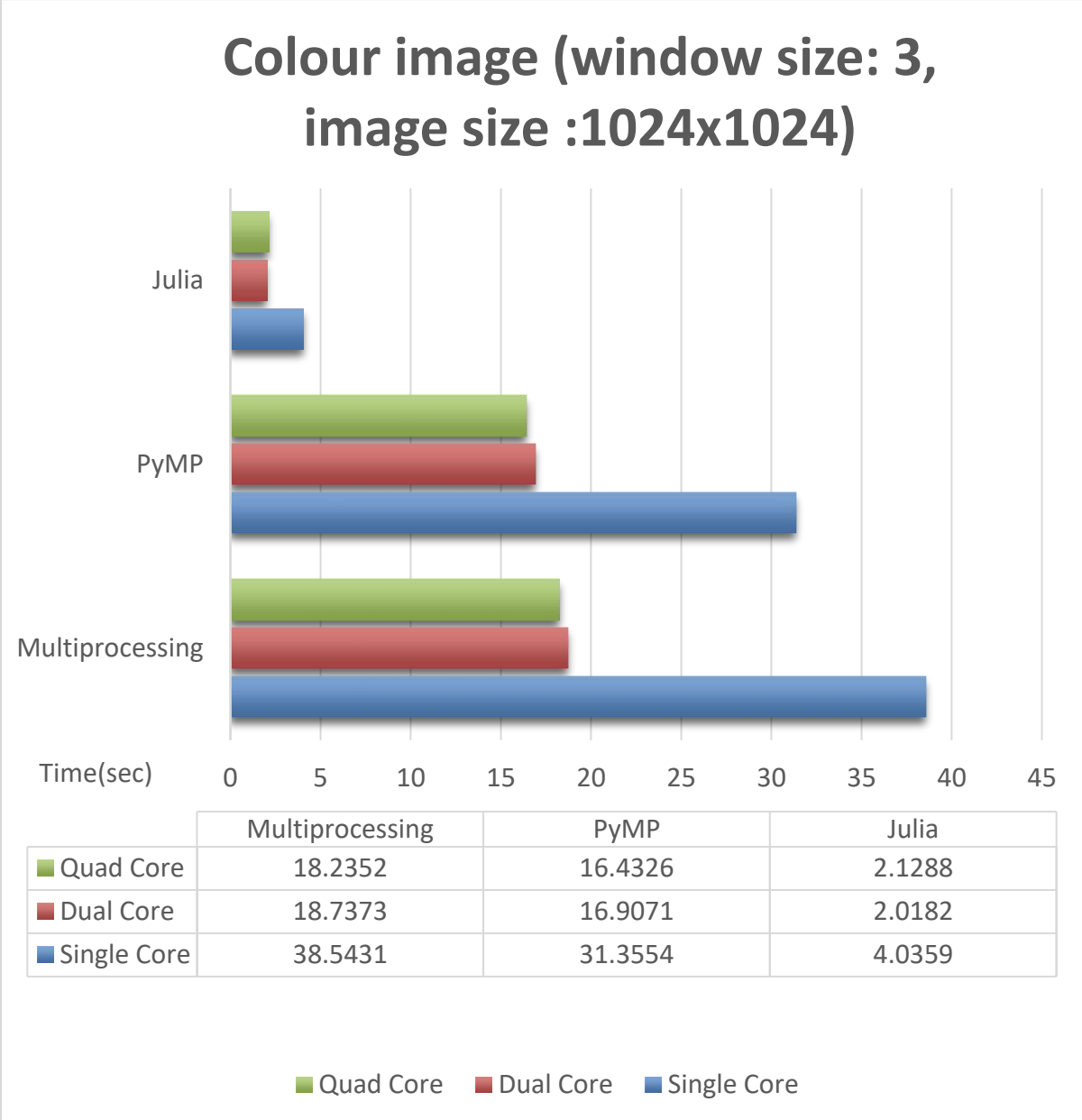
Here considering a particular reading, Julia spent only 19% of the total time in completing that task on the other hand python took 81% of the total time to do the same task. By this analysis we can infer that Julia has a better performance than Python when high computation is needed. (Note: This observation is not ideal, it may vary for different readings)

CHAPTER 8

CONCLUSION AND FUTURE WORK

Let us plot the time taken by the algorithm for different tools. As a sample we have shown one of the results (we can include other results too, if needed).

8.1 CONCLUSION



	Multiprocessing	PyMP	Julia
Quad Core	18.2352	16.4326	2.1288
Dual Core	18.7373	16.9071	2.0182
Single Core	38.5431	31.3554	4.0359

.

From, the above figure we can conclude that Julia performs better when compared to python. It takes less processing time to perform a given task than python because it is designed to quickly implement math concepts.

And also the when the user needs to deal with large data as input which also requires high computation time it is better to go with Multicore Architecture. Or else it is fine to work with a system with a single core, because of overhead and scheduling reasons.

8.2 Application

- Digital Image Processing
- Median Filter works better for small to moderate levels Gaussian noise. But for high noise Gaussian Filter is better. For speckle noise and salt and pepper noise, it is particularly effective.
- Along with reducing the noise in a signal, it is important to preserve edges which are of critical importance to the visual importance of images and this is done by median filter.
- Median filter can remove noise and speckles without blurring the picture like removing artifacts from imperfect data acquisition.
- Horizontal stripes sometimes produced by optical scanners are also successfully removed using median filter.
- Median filters are also used in radiographic systems and in many commercial tomographic scan systems.
- Median filters are also employed for processing electroencephalogram (EEG) signal and blood pressure recordings.
- This type of filter is also likely to be found in new commercial digital televisions because of the very good cost-to-performance ratio.
- Different types of noises in the fingerprint images pose greater difficulty for recognizers. So, this can be improved by median filtering.
- Various industrial areas such as military, transportation, automation and control etc work with multicore systems because of their intensive advantages.
- New multicore processors used in embedded devices provide more functionality, increase system performance and run at lower temperatures.

- The multicore operating environment introduces a new software paradigm where general purpose and real time operating systems and applications need to run concurrently.

8.3 CHALLENGES FACED

- Random module in Julia didn't work like python's Random module so it was difficult to generate salt and pepper noise.
- While dividing the images equally the variable which was defined as int used to consider round off to lower scale, because of that the image was not divided equally.
- In Julia even the monochrome image will be read as 3D array.
- Due to scheduling, overhead and other reason the quad core took more time than the dual core.
- To merge the monochrome images using OpenCV module was not possible
- In order to preserve the local details of the image, median filter should only change the intensity of corrupted pixels on the damaged image. However, it is very difficult to detect the corrupted pixels from this image correctly.

8.4 FUTURE WORK

- To use multicore processor systems with high specifications like increase in number of cores, less power consumption and with large size memory.
- To explore different modules in other programming language and check whether there is further reduction in the computation time.
- Since median filtering techniques tend to destroy lines, other fine image details and perform poorly in the presence of signal-dependent noise, this problem should be overcome using effective algorithms.
- To check the processing time by taking large amount of data as input to the median filter and with the help of multicore processors.

REFERENCES

[1] https://medium.com/image-vision/noise-in-digital-image-processing-55357c9fab71

[2] https://en.m.wikibooks.org/wiki/Python_Programming/Threading

[3] https://en.m.wikibooks.org/wiki/Python_Programming/Functions

[4] https://docs.python.org/3/library/multiprocessing.html

[5] https://github.com/classner/pymp

[6] https://en.m.wikipedia.org/wiki/Julia_(programming_language)

[7] https://www.geeksforgeeks.org/julia-vs-python/amp/#referrer=https://www.google.com

[8] A survey paper on Various Median Filtering Techniques for Noise Removal from Digital Images by Prateek Kumar Garg, Pushpneel Verma, Ankur Bhardwaj

[9] Detection and removal of Salt and Pepper noise in images by improved median filter By S. Deivaalkshmi ,S. Sarath, P.Palanisamy

[10] Comparison of Single Core and Multicore Processor by Sukhdev Singh Ghuman

[11] A survey on Parallel Multicore computing: Performance and improvement by Ola Surakhi, Mohammad Khanafseh, Sami Sarhan

[12] Introduction to Multicore Programming by Marc Moreno Maza

[13] Comparison of Time complexity in median filtering on multicore architecture by Narasimha Kaulgud, Sharmila B S

[14] Julia: A Fresh Approach to Parallel Programming by Alan Edelman

[15] Experimental Multi-threading Support for the Julia Programming Language by Tobias Knopp