PIPL - Parallel Image Processing Library

Agastya Sampath, Shijie Bian

Project Goal

Analyze benefits of parallel programming on image processing algorithms!

- ➔ Develop a comprehensive image processing library using parallel programming
- ➔ Utilize parallel programming tools such as OpenMP and CUDA to improve processing speed and optimize performance
- ➔ Conduct detailed profiling of library's performance under different scenarios to gain insight into how parallel programming can benefit image processing tasks

Project Summary

- ➔ Investigated the advantages of parallel programming in the field of image processing.
- ➔ Developed a C++ library for image processing named PIPL.
- ➔ Incorporated various image processing techniques, including denoising, color enhancement, and color conversion.
- ➔ Improved the performance of the library by utilizing parallelizing tools such as OpenMP and CUDA on GPU.
- ➔ Conducted a thorough analysis of the library's performance under various scenarios to provide insights into the benefits of parallel programming for image processing tasks.

Algorithms

Denoising	Color Enhancement	Color Conversion
Median Filtering	Histogram Equalization	RGB to YCbCr/Grayscale
Non-Local Means	CLAHE	YCbCr to RGB

Parallelization Frameworks

OpenMP

NVIDIA CUDA

INPUT

OUTPUT


and villages they have come.
Those became part of them.
MEMORIES HOTEL
ROAM AROUND IN WINTER
Memory Hotels,1858

Approach

Pseudocode

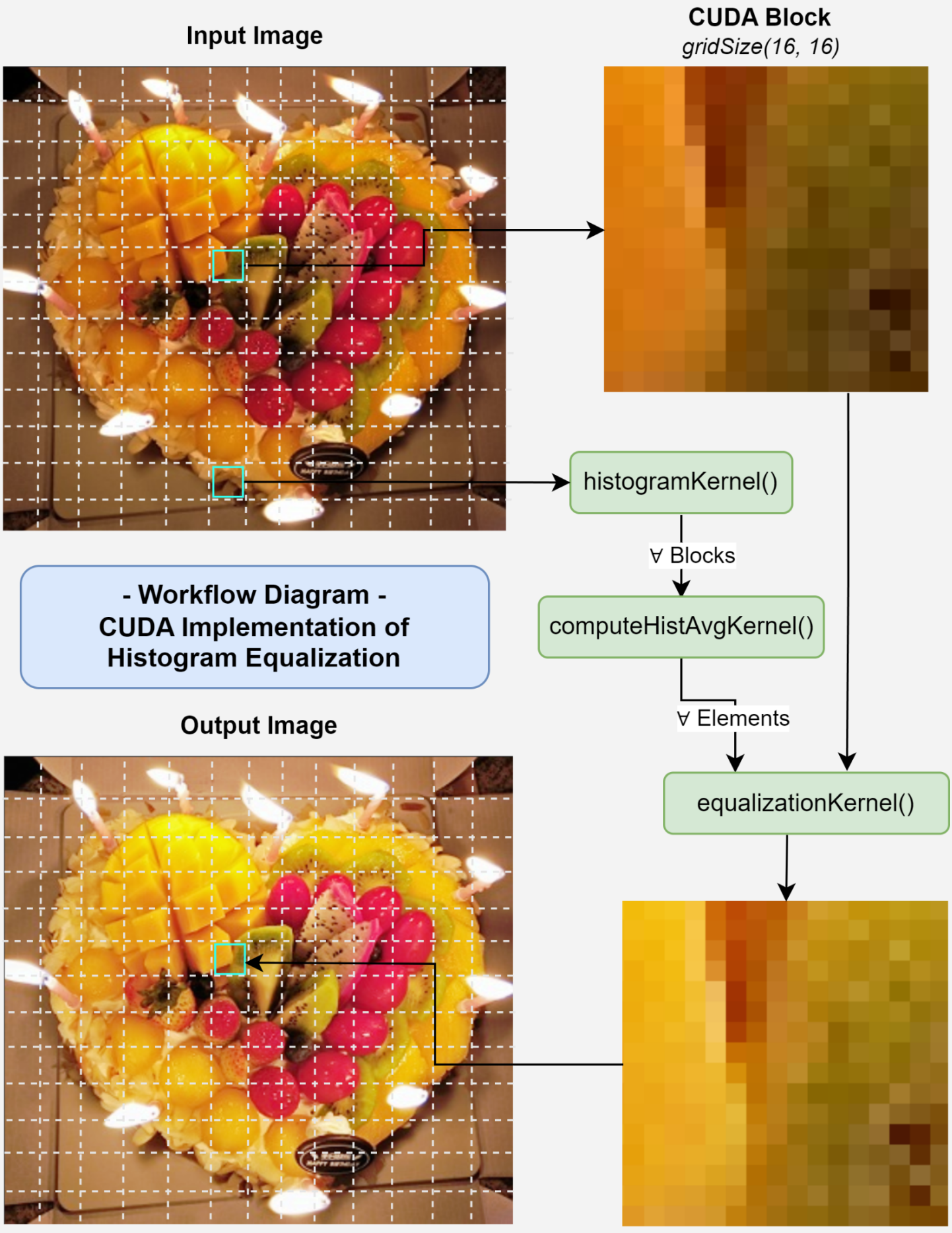
Enhancement - Histogram Equalization

```
Input: img (original image), res (resulting image) -> by reference
Output: res (resulting image) -> by reference

1. Set res equal to img
2. Set width equal to img.width()
3. Set height equal to img.height()
4. Set bins equal to 256 (for color channels)
5. Create empty histogram for each color channel: hist_r, hist_g, hist_b
6. Set each element of hist_r, hist_g, and hist_b to 0
7. For each pixel (x, y) in img:
8.      Increment the corresponding bin in hist_r, hist_g, and hist_b by 1
9.      Compute the average histogram, hist_avg, by averaging the corresponding values in hist_r, hist_g, and hist_b
10.     Create an empty cumulative histogram, cum_hist
11.     Set the first element of cum_hist to the first element of hist_avg
12. For each element in hist_avg (starting at index 1):
13.     Set the corresponding element in cum_hist to the sum of the current element in hist_avg and the previous element in cum_hist
14. For each pixel (x, y) in img:
15.     Get the R, G, and B values of the pixel
16.     Set the R, G, and B values of the corresponding pixel in res to the corresponding value in cum_hist scaled to the range [0, 255]
```

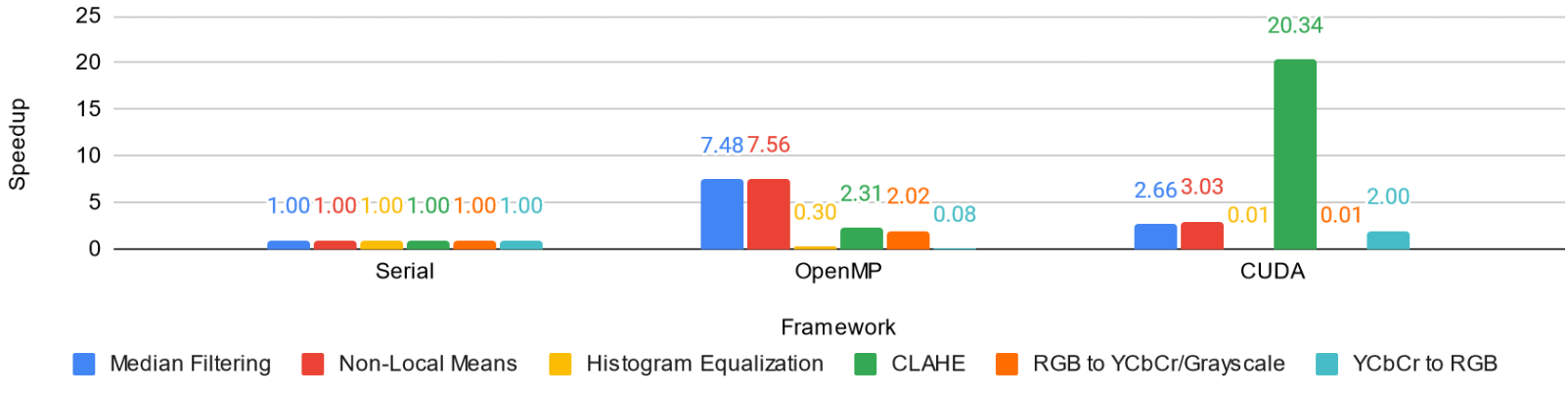
Parallelization Approach

OpenMP	CUDA
❖ Analyzed code to identify intensive sections ❖ Parallelized outer loops using #pragma omp parallel for directive (line 14) ❖ Used schedule clause to assign pixels dynamically to OpenMP threads ❖ Utilized SIMD approach to process multiple array elements in parallel (line 12) ❖ Used dynamic scheduling to distribute workload among threads based on resources	❖ Used kernel functions for better modularity and flexibility. ❖ Algorithm divided into 3 parts: histogram (line 7), cumulative histogram (line 12), equalization (line 14). ❖ 3 separate CUDA kernels implemented for each part. ❖ Parallelized computations across GPU threads. ❖ Mapped data structures to GPU architecture for efficient processing.

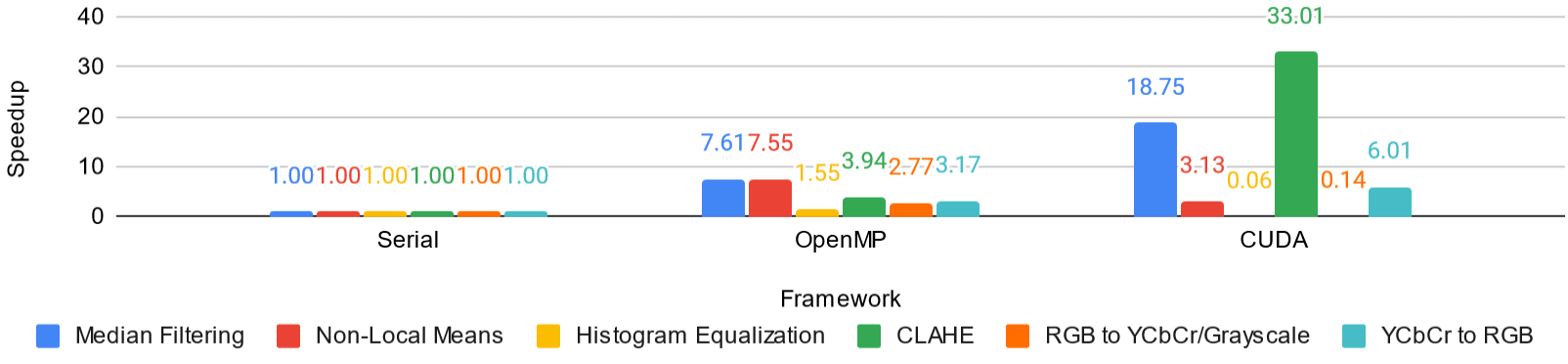

Input Image
CUDA Block
gridSize(16, 16)
histogramKernel()
∀ Blocks
computeHistAvgKernel()
∀ Elements
equalizationKernel()
- Workflow Diagram -
CUDA Implementation of
Histogram Equalization
Output Image

Problem Size Scaling

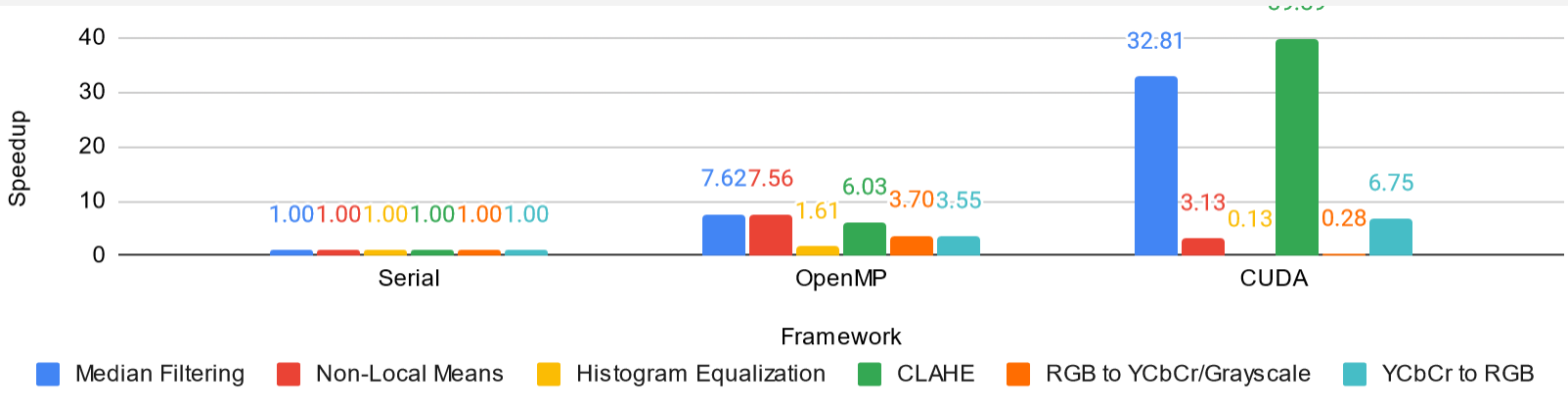
SD Images



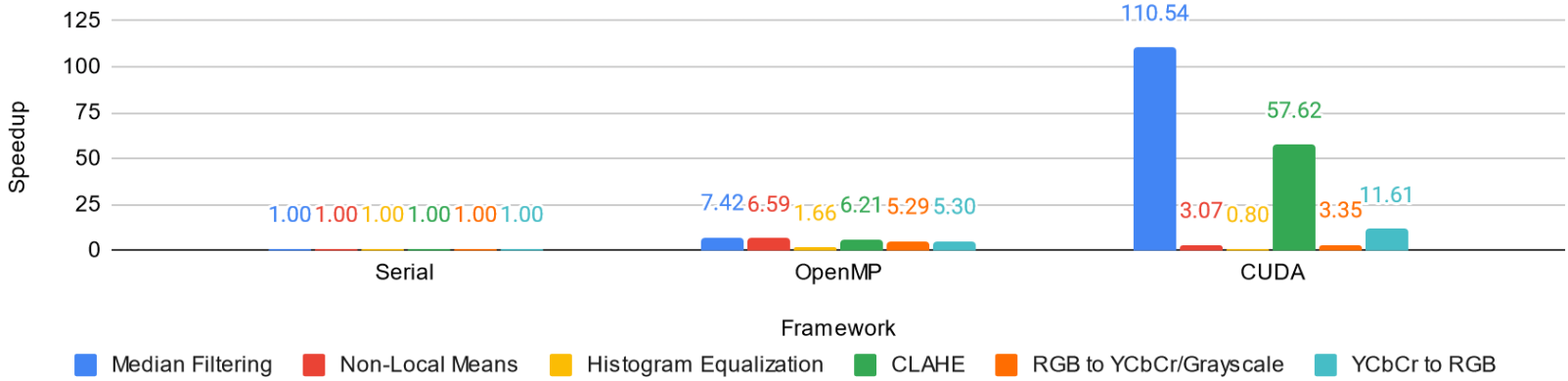
HD Images



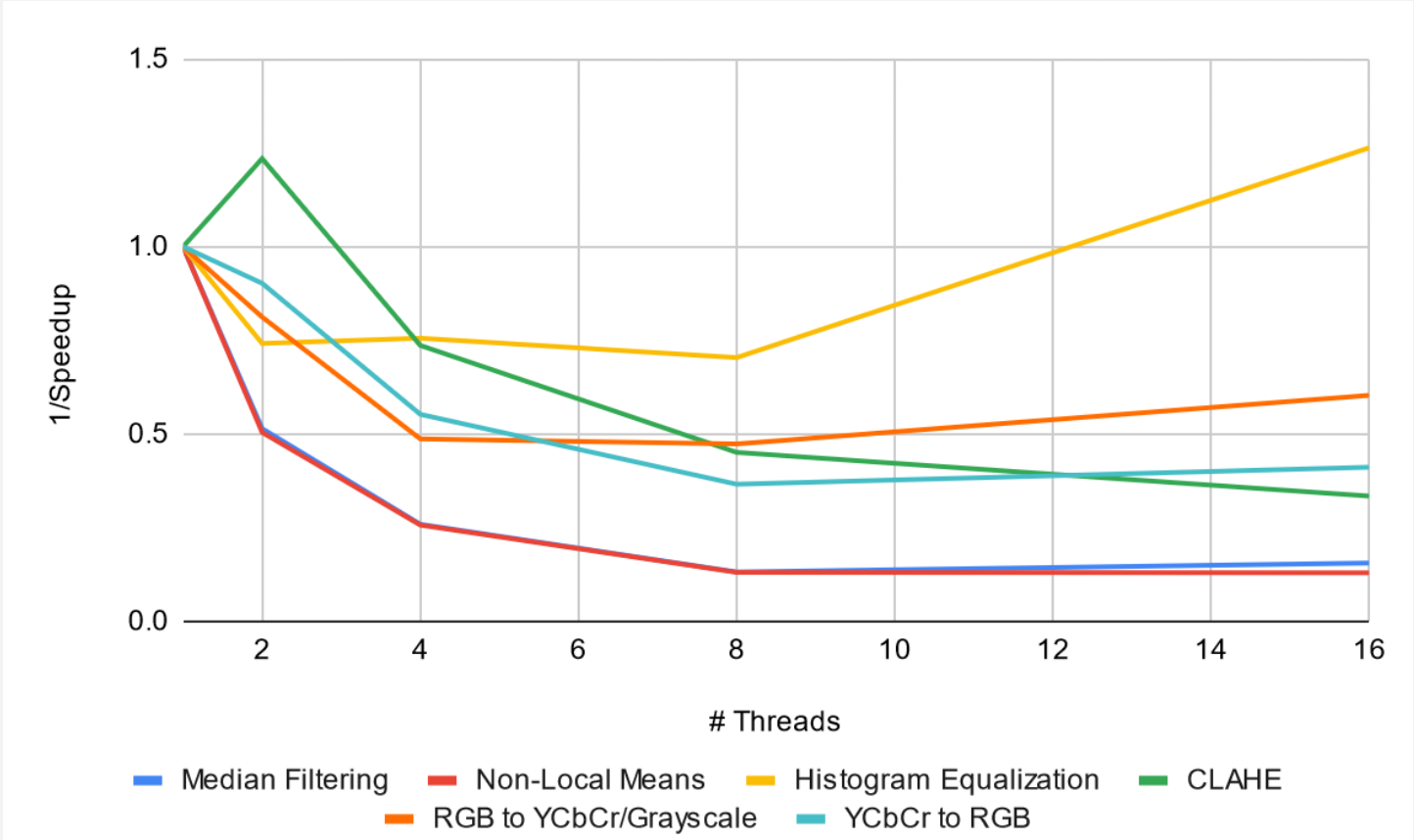
UHD Images



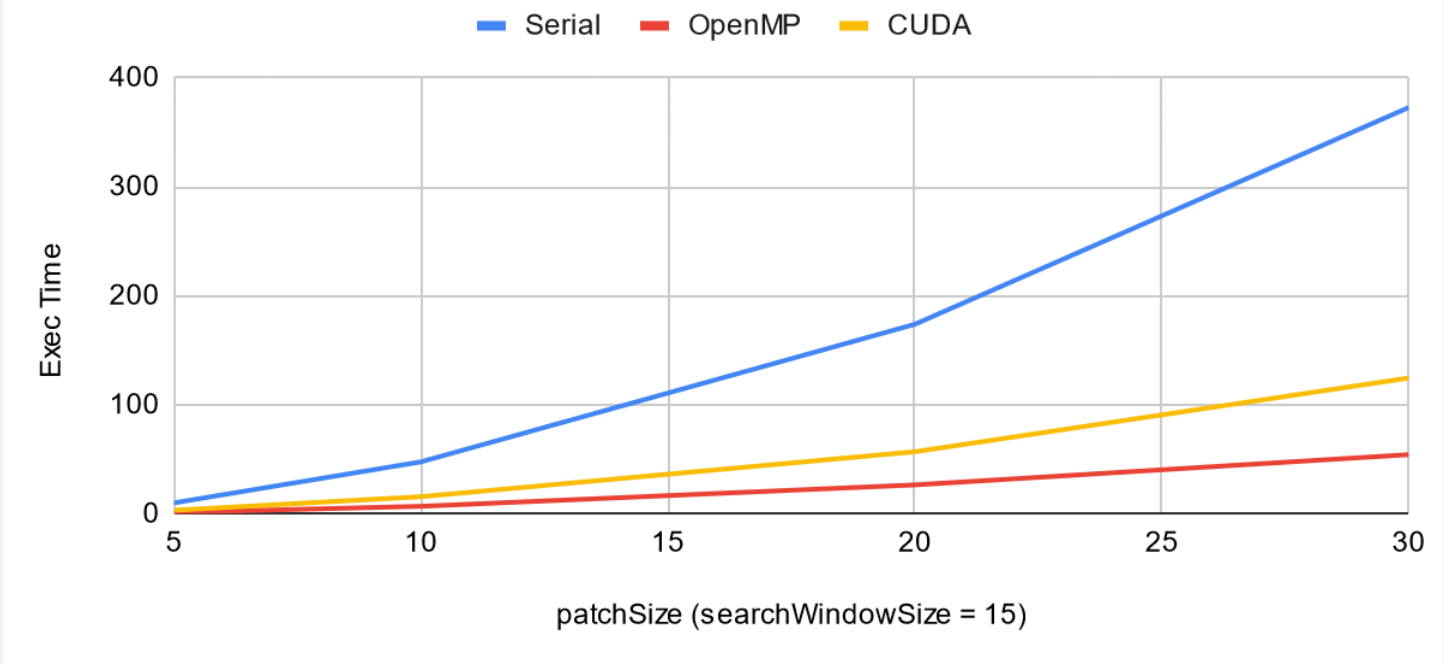
4K Images



OpenMP Thread Scaling



Parameter Scaling



Conclusions

- ★ Performed a comparison of OpenMP and CUDA frameworks for 6 image processing algorithms.
- ★ Choice of Framework heavily affects performance of most algorithms.
- ★ There are some algorithms and image sizes where parallelization is NOT useful!
- ★ CUDA implementations are great for 4K images and larger - OpenMP is comparable for every algorithm otherwise.
- ★ Image Processing algorithms scale very well with threads - the ones with more computation scale better.
- ★ Parameters affect scaling, but not choice of framework.

Possible Future Work

- ➔ Investigating the impact of optimizing work distribution on the GPU by tuning grid and block sizes to achieve better performance with GPU parallelization.
- ➔ Exploring more sophisticated parallelization techniques using the implemented library to identify even better-performing parallel code.
- ➔ Extending the analysis of parallelization techniques to other image-processing algorithms that operate on two-dimensional structures, including object detection and segmentation.

Final Resolution

Careful selection of the parallelization framework and algorithm parameters for the right image sizes is crucial for achieving significant performance improvements in image processing applications!!!!