Image and Video Processing in Java

Done by:
Mohammad Karaki 6393
Mohammad Atwi 6328

Presented to:
Dr. Mohammad Aoude

1.Introduction:

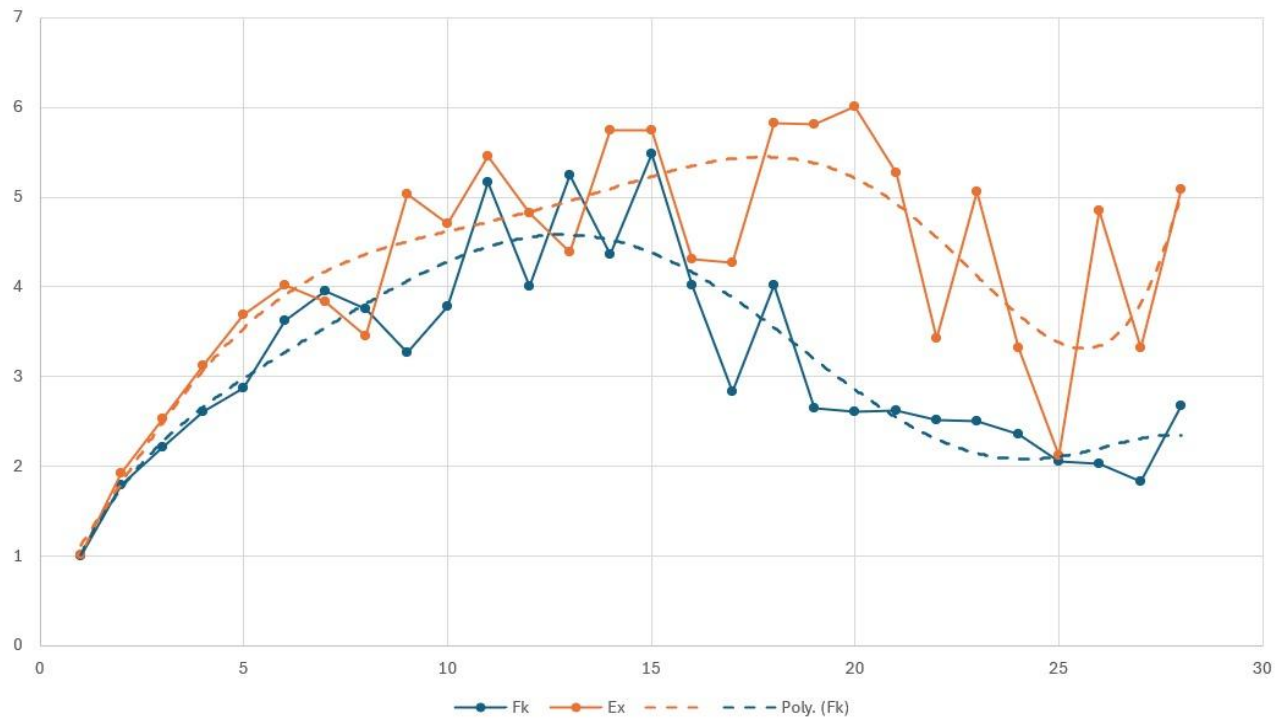
In today’s digital era, images and videos play a central role in how we communicate, consume information, and express creativity. As the quality of media continues to increase—especially with the rise of 4K and higher resolutions—the demand for efficient processing becomes critical. This project addresses that challenge by implementing image and video filtering using Java through the Fork/Join framework and ExecutorService. Since videos are essentially sequences of images, and images can be represented as large matrices of pixel values, we leverage a tile-based parallel processing approach to significantly improve performance and scalability.

2.Design and implementation

Our project employed a tile-based approach for image processing, dividing the image into smaller blocks to facilitate parallel processing. We utilized both ExecutorService and ForkJoinPool to manage threads and process these tiles concurrently. Each thread was responsible for looping through a tile, both vertically and horizontally, to apply the chosen filter. This approach allowed us to leverage multi-core processors and significantly improve processing times.

For video processing, we used JavaCV to extract individual frames from the video. We then applied our parallel image processing pipeline to each frame, processing them concurrently to apply the desired filter. After processing all frames, we combined them back together.

One of the key challenges we faced was achieving a satisfactory speedup. Initially, our implementation yielded a speedup of maximum around 2x but was more often than not dropping to around 1x or sometimes below, which was below our target of 3x. To overcome this, we experimented with varying block sizes and adjusting the number of threads used in the thread pool and ended up defining the block size dynamically depending on the image dimensions and the number of threads. Through iterative testing and refinement, we were able to identify the optimal configuration that yielded an average speed up of 3x, with some configurations achieving even higher speedups of up to 4x, 5x, or 6x (2 times). By optimizing our design, we were able to achieve significant performance gains and demonstrate the effectiveness of parallel processing for image and video applications.



This graph shows the speedup vs number of threads 15 was around the peak speedup we could get in both executor and fork/join.

3.parallel vs sequential

Applying image filters using parallel processing with ExecutorService or ForkJoinPool can significantly improve performance compared to sequential processing as shown in the graph above. By dividing the image into smaller blocks and processing them concurrently, you can leverage multi-core processors to reduce processing time. ExecutorService can handle this by assigning each section to a separate thread, while ForkJoinPool can recursively divide the task into smaller sub-tasks, utilizing a work-stealing algorithm to optimize thread utilization. This parallel approach enables faster application of filters, making it ideal for real-time image or video processing or batch processing large image datasets. In contrast, sequential processing would handle the entire image in a single thread, leading to longer processing times and potential performance bottlenecks. By harnessing parallel processing, you can achieve faster and more efficient image filtering, enhancing overall system performance.

4.Conclusion and Future Work

In conclusion our project showcases the power of parallel processing in image and video processing, using ExecutorService and ForkJoinPool to significantly boost performance. By leveraging multi-core processors, we can efficiently process large media datasets, making it ideal for applications like real-time video editing or image analysis.

Future directions include exploring additional performance optimization techniques, such as:
- GPU acceleration using CUDA
- Utilizing SIMD (Single Instruction, Multiple Data) instructions to process multiple data elements simultaneously
- Leveraging Java's Vector API to explicitly vectorize computations and further enhance performance

By refining these techniques, we can further enhance the efficiency and effectiveness of image and video processing.

5.Individual Contribution

This project was a collaborative effort, with each team member making significant contributions. Mohammad Atwi was responsible for developing the image processing component, designing and implementing the tile-based approach for parallel processing. Mohammad Karaki focused on video processing, successfully integrating JavaCV to split videos into frames and feeding them into the image processing pipeline, as well as designing a user-friendly GUI for the application.We both worked together to optimize the system and achieve high speed up.