Parallel Processing with OpenCv

Youssef Mahmoud Hafez

Youssef Mohamed Al-Sayed

Youssef Essam Mohamed

Adham Hamdy Mohamed

Moaaz Ehab Mohamed

Mustafa Mohamed Al-Sayed

Eng. Mahmoud Al-Badry

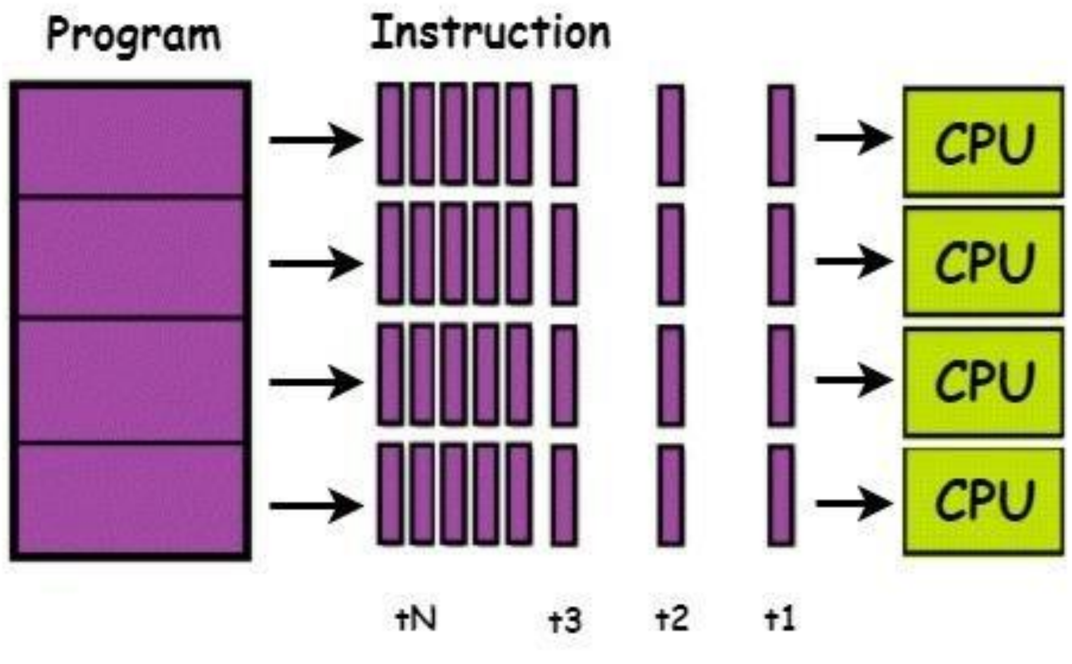
Eng. Abdulrahman Salam

Dr. Masoud Ismael

Parallel Image Processing with MPI: A Practical Guide

1. Introduction

Parallel Image Processing with MPI is an application designed to perform common image processing operations in parallel on a cluster or multi-core system using the Message Passing Interface (MPI). The objective of this application is to achieve faster execution times by distributing the image processing tasks among multiple MPI processes.



2. Design

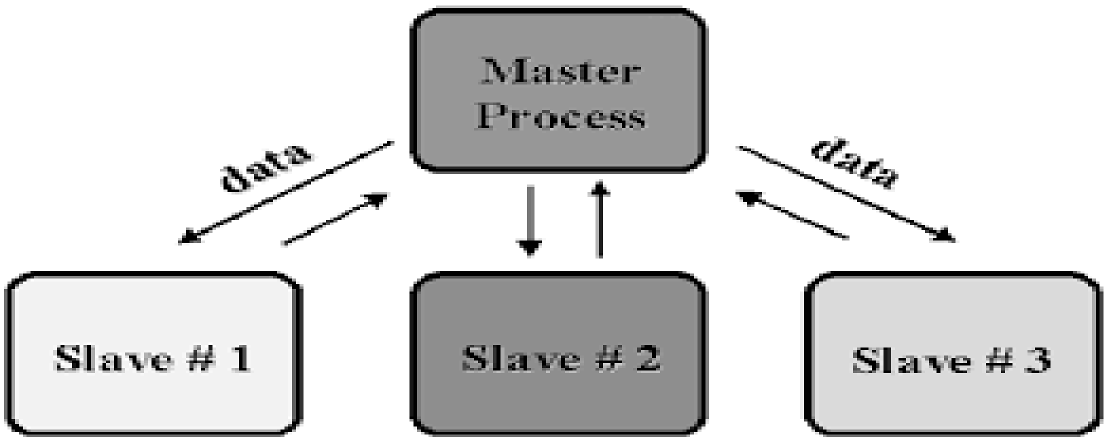
2.1 Functionality

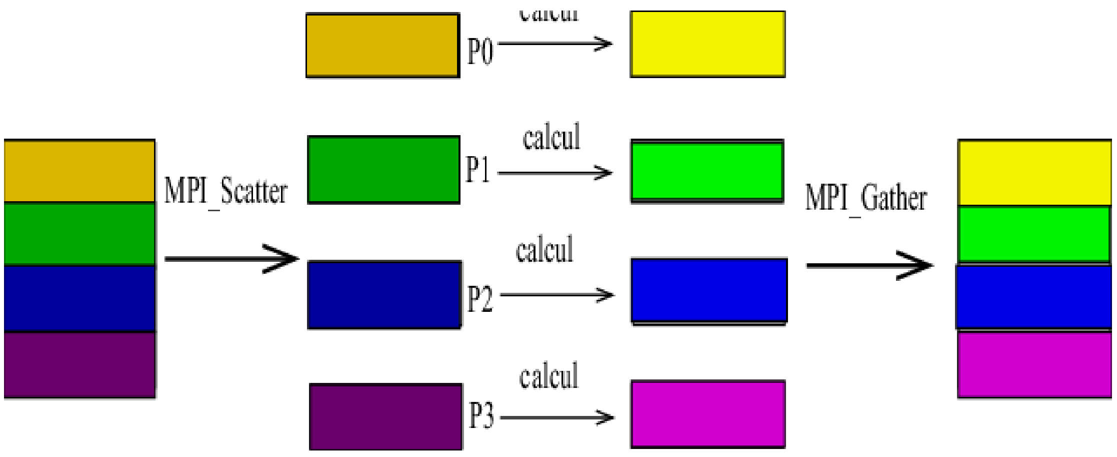
The application supports the following features:

- Loading an input image (in JPEG or PNG format) on the master node.
- Distributing the image across multiple MPI processes for parallel processing.
- Implementing various image processing algorithms in parallel, including Gaussian Blur, Edge Detection, Image Rotation, Image Scaling, Histogram Equalization, Color Space Conversion, Global Thresholding, Local Thresholding, Image Compression, and Median Filtering.
- Performance analysis to measure execution times, scalability, and identify performance bottlenecks.

2.2 Architecture

The application follows a master-worker architecture, where the master node coordinates the image loading and distribution tasks, and each worker node performs a specific image processing operation in parallel. MPI communication primitives such as **MPI_Scatter** and **MPI_Gather** are used to distribute image data and collect processed results.





2.3 Algorithm Parallelization

Each image processing algorithm is parallelized by dividing the image into chunks and distributing them among MPI processes. Each process operates independently on its portion of the image, and MPI communication is used to exchange necessary information between processes if required.

3. Implementation

3.1 Language and Libraries

The application is implemented in C++ and utilizes the OpenCV library for image loading, processing, and I/O operations. MPI functions are used for parallelization and communication among MPI processes.

3.2 Image Processing Algorithms

The following image processing algorithms are implemented in the application with there steps:

1. Gaussian Blur

- **Initialization**: It initializes variables for measuring time and disables parallel execution if required.

- **Broadcasting**: The blur radius is broadcasted from the root process to all other processes.

- **Image Processing**: Each process applies Gaussian blur to its local portion of the image.

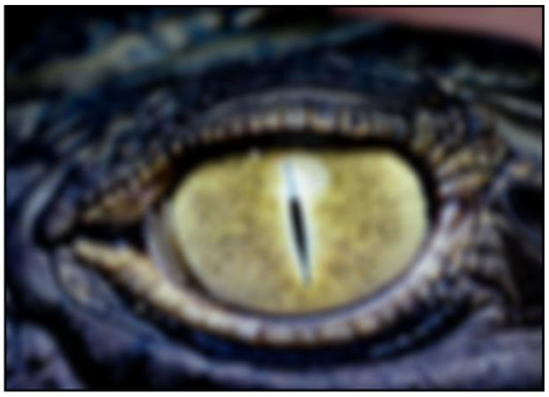
- **Saving**: Processed parts of the image are saved to individual files with names based on the process rank.

- **Gathering**: The processed image parts are gathered on the root process to reconstruct the final image.

- **Completion**: If the process is the root (rank 0), it saves the final image and prints a completion message with the processing time.

- **Finalization**: MPI is finalized to release resources.

Original image


Gaussian Blur filter applied

2. Edge Detection

- **Initialization**: It initializes variables for measuring time.

- **Edge Detection**: Sobel edge detection is applied to the local portion of the image. Sobel operator computes the gradient magnitude along the x-axis.

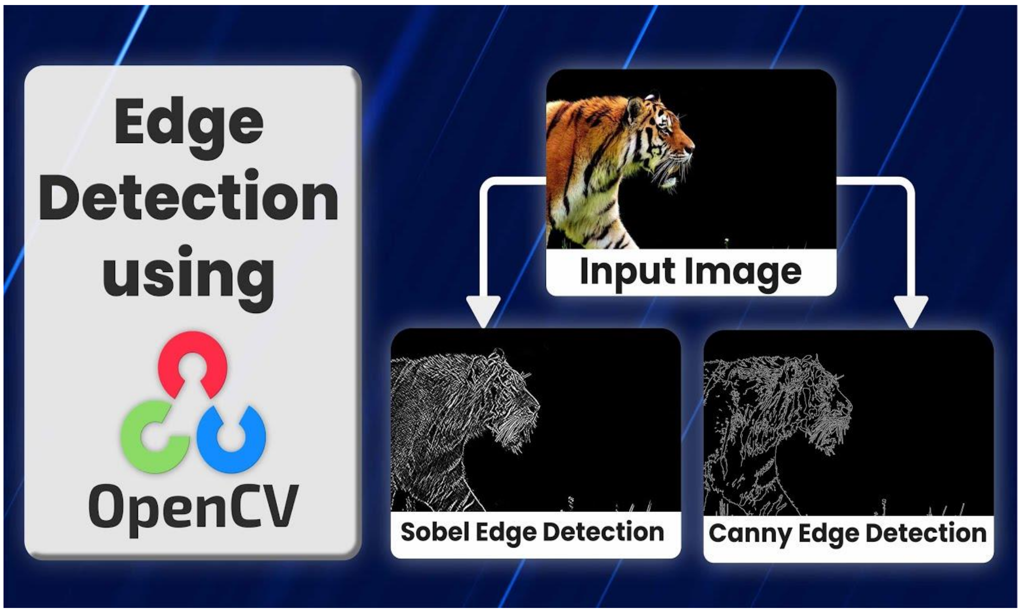
- **Conversion**: The resulting edges are converted back to an 8-bit unsigned integer format using convertScaleAbs.()

- **Saving**: Processed edges are saved to individual files with names based on the process rank.

- **Gathering**: The processed edge parts are gathered on the root process to reconstruct the final image.

- **Completion**: If the process is the root (rank 0), it saves the final image and prints a completion message with the processing time.

- **Finalization**: MPI is finalized to release resources.



3. Image Rotation

- **Initialization**: Variables are initialized, including a fixed rotation angle of 90 degrees.

- **Broadcasting**: The rotation angle is broadcasted to all processes.

- **Time Measurement**: Variables for measuring execution time are initialized.

- **Image Rotation**: Each process rotates its local portion of the image by 90 degrees clockwise

.

- **Saving**: Rotated image parts are saved to individual files, named based on the process rank.
- **Size Gathering**: Sizes of rotated image parts from all processes are gathered on the root process.

- **Displacements Calculation**: Displacements for gathering rotated image parts are calculated.

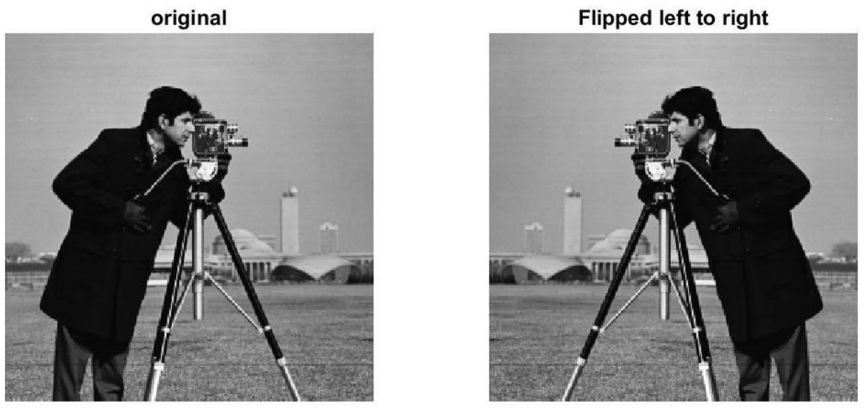
- **Image Gathering**: Rotated image parts from all processes are gathered into a single buffer on the root process.

- **Concatenation**: On the root process, rotated image parts are concatenated horizontally to form the final image.

- **Saving Final Image**: The final rotated image is saved with the specified filename.

- **Completion**: A completion message with the processing time is printed on the root process.

- **Finalization**: MPI is finalized to release resources.



4. <u>Image Scaling</u>

- **<u>Initialization</u>**: Variables are initialized, including start time for performance measurement.

- **<u>Scaling</u>**: Each process scales its local portion of the image.

- **<u>Saving</u>**: Scaled images are saved with filenames based on process rank.

- **<u>Gathering Sizes</u>**: Sizes of scaled image parts are gathered on the root process.

- **<u>Displacements Calculation</u>**: Displacements for gathering are calculated based on gathered sizes.

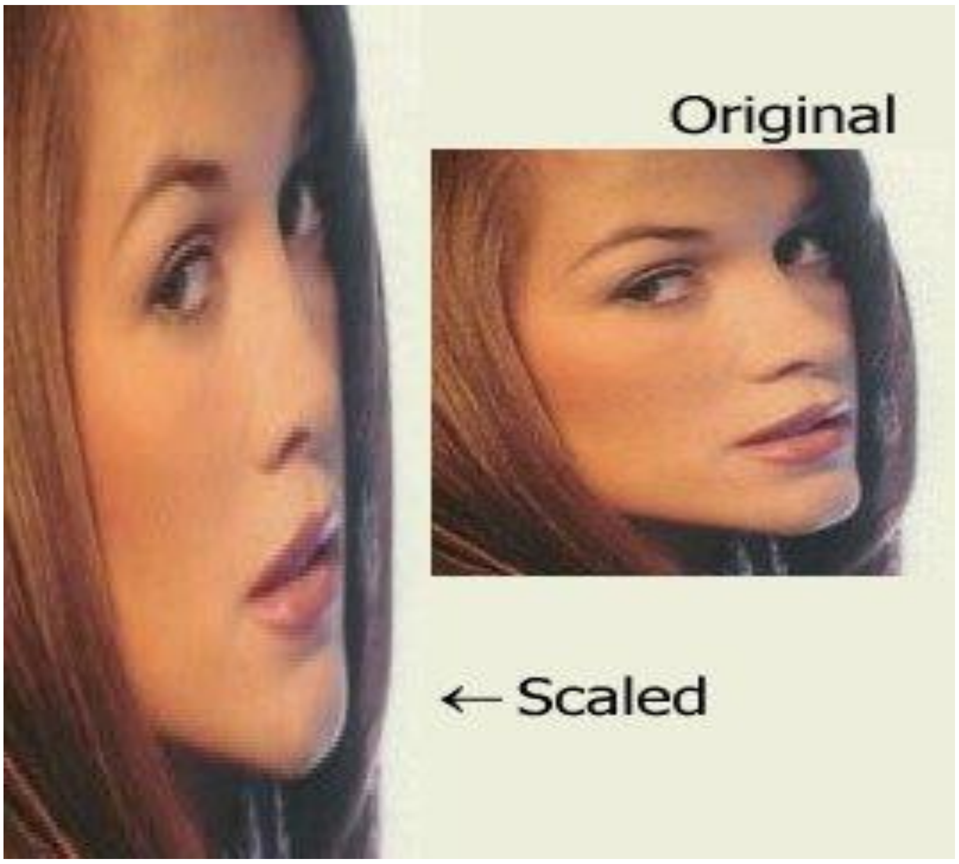
- **<u>Gathering Image Parts</u>**: Scaled image parts are gathered into a single buffer on the root process

  .

- **Concatenation**: On the root process, scaled image parts are concatenated vertically to form the final image.

- **Saving and Completion**: Final scaled image is saved, and a completion message with processing time is printed.

- **MPI Finalization**: MPI is finalized to release resources.



5. Histogram Equalization

- **Initialization**: Start time is recorded.

- **Equalization**: Each process converts its image portion to grayscale and equalizes its histogram.

- **Saving**: Equalized image parts are saved with filenames based on process rank.

- **Gathering Sizes**: Sizes of equalized image parts are gathered on the root process.

- **Displacements Calculation**: Displacements for gathering are computed based on gathered sizes

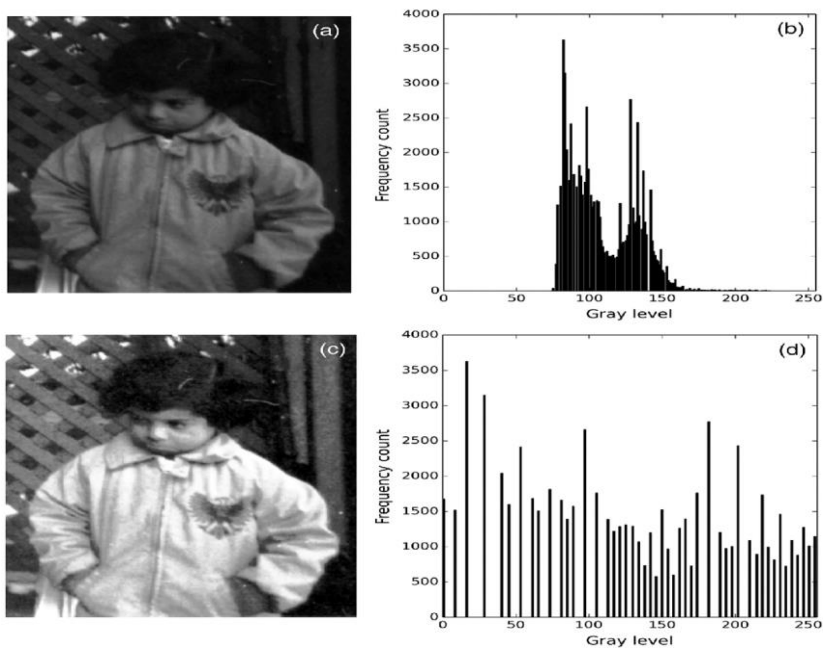
.

- **Gathering Image Parts**: Equalized image parts are gathered into a single buffer on the root process.

- **Concatenation**: On the root process, equalized image parts are merged vertically to form the final image.

- **Saving and Completion**: Final image is saved, and a completion message with processing time is displayed.

- **MPI Finalization**: MPI is finalized



6. <u>**Color Space Conversion**</u>

- **<u>Initialization</u>**: Start time is recorded

- **<u>Conversion</u>**: Each process converts its image portion to a different color space, such as BGR to HSV.

- **<u>Saving</u>**: Converted image parts are saved with filenames based on process rank.

- **<u>Gathering Sizes</u>**: Sizes of converted image parts are gathered on the root process.

- **<u>Displacements Calculation</u>**: Displacements for gathering are computed based on gathered sizes.

- **<u>Gathering Image Parts</u>**: Converted image parts are gathered into a single buffer on the root process.

- **Concatenation**: On the root process, converted image parts are merged vertically to form the final image.

- **Saving and Completion**: Final image is saved, and a completion message with processing time is displayed.

- **MPI Finalization**: MPI is finalized.

7. Global Thresholding

- **Initialization**: Start time is recorded.

- **Thresholding**: Each process converts its image portion to grayscale and applies a global thresholding algorithm.

- **Saving**: Thresholded image parts are saved with filenames based on process rank.

- **Gathering Sizes**: Sizes of thresholded image parts are gathered on the root process.

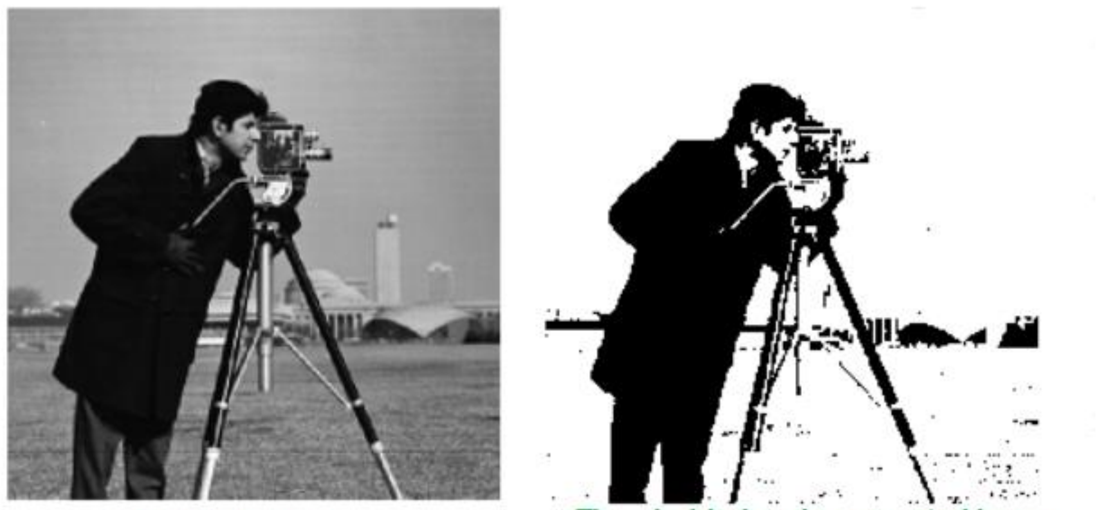
- **Displacements Calculation**: Displacements for gathering are computed based on gathered sizes.

- **Gathering Image Parts**: Thresholded image parts are gathered into a single buffer on the root process.

- **Concatenation**: On the root process, thresholded image parts are merged vertically to form the final image.

- **Saving and Completion**: Final image is saved, and a completion message with processing time is displayed.

- **MPI Finalization**: MPI is finalized.



Original Image Thresholded and segmented Image

8. Local Thresholding

- **Initialization**: Start time is recorded.

- **Grayscale Conversion**: The local image is converted to grayscale.

- **Local Thresholding Parameters**: Local variables for block size and threshold value are defined.

- **Local Thresholding**: Adaptive thresholding is applied to the grayscale image using the defined parameters.

- **Saving**: Thresholded image parts are saved with filenames based on process rank.

- **<u>Gathering Sizes</u>**: Sizes of thresholded image parts are gathered on the root process.

- **<u>Displacements Calculation</u>**: Displacements for gathering are computed based on gathered sizes

.

- **<u>Gathering Image Parts</u>**: Thresholded image parts are gathered into a single buffer on the root process.

- **<u>Concatenation</u>**: On the root process, thresholded image parts are merged vertically to form the final image.

- **<u>Saving and Completion</u>**: Final image is saved, and a completion message with processing time is displayed.

- **<u>MPI Finalization</u>**: MPI is finalized.

9. <u>Median Filtering</u>

- **<u>Initialization</u>**: Start MPI timing for performance measurement.

- **<u>Median Filtering</u>**: Apply median filtering to the local image using a specified kernel size (in this case, 5). Median filtering is a nonlinear image processing technique used to remove noise while preserving edges.

- **<u>Saving Results</u>**: Save the filtered image locally with a filename based on the process rank.

- **<u>Gathering Results</u>**: Gather the processed parts from all processes onto the root process (rank 0).

- **Concatenating Results**: On the root process, concatenate the processed image parts to form the final image.

- **Saving Final Image**: Save the final image with a specified output filename.

- **Completion Message**: Print a completion message with the processing time.

- **MPI Finalization**: Finalize MPI after processing is complete.

And each algorithm is implemented as a separate function and parallelized using MPI.

4. Performance Analysis Results

4.1 Execution Times

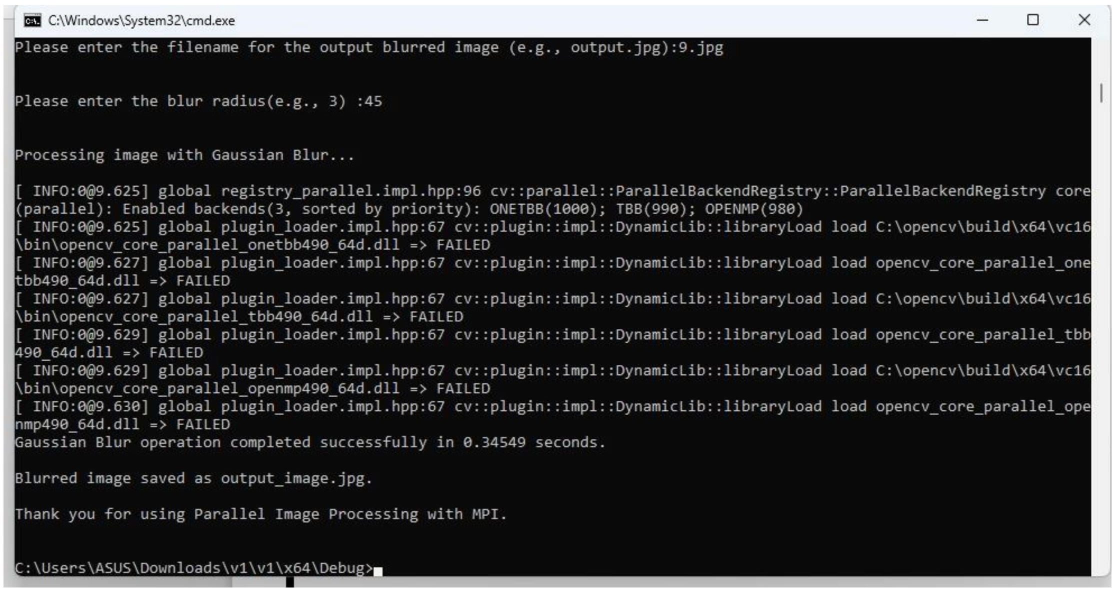
- The application demonstrates significant speedup when using multiple MPI processes compared to sequential execution.
- Execution times vary depending on the image size, algorithm complexity, and number of MPI processes.

4.2 Scalability

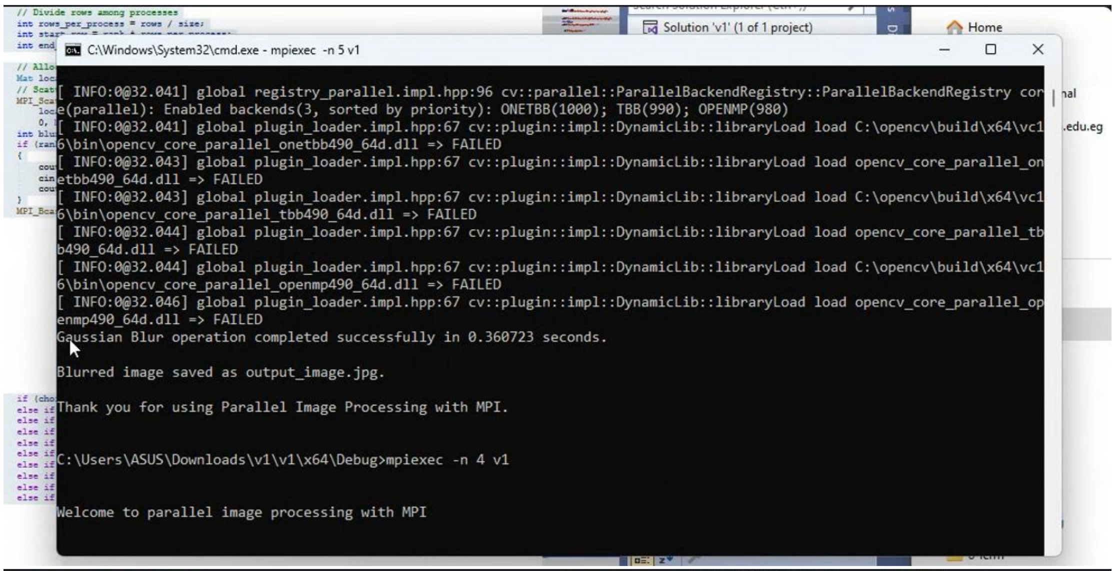
- Scalability analysis shows that the application scales well with an increasing number of MPI processes for certain algorithms and image sizes.
- However, scalability might be limited by factors such as communication overhead and load imbalance.

4.3 Performance Bottlenecks

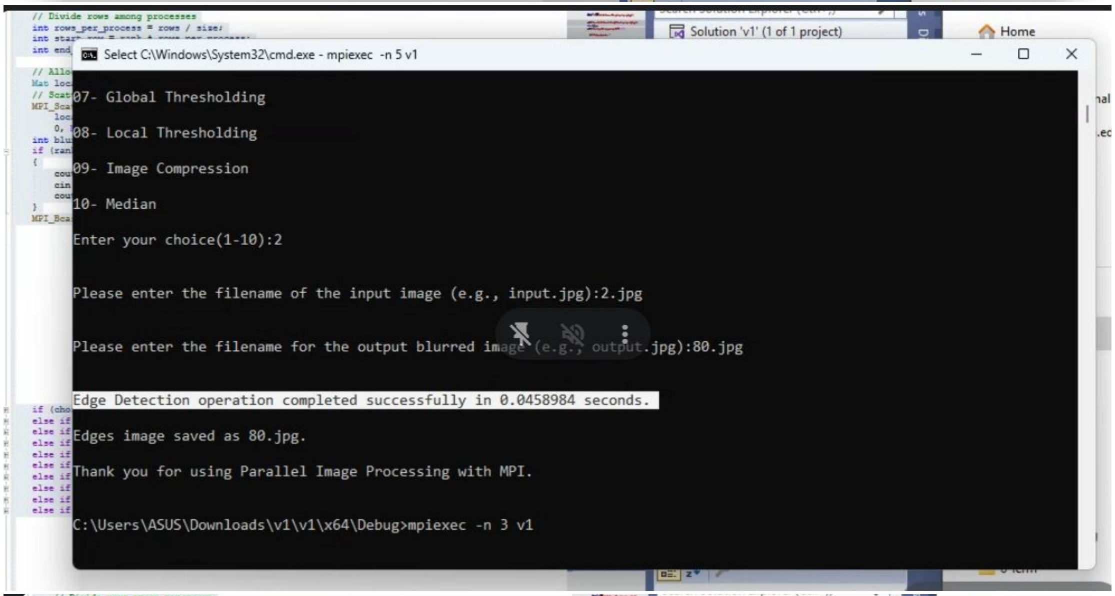
- MPI profiling tools reveal potential performance bottlenecks such as uneven workload distribution, inefficient communication patterns, and high synchronization overhead.



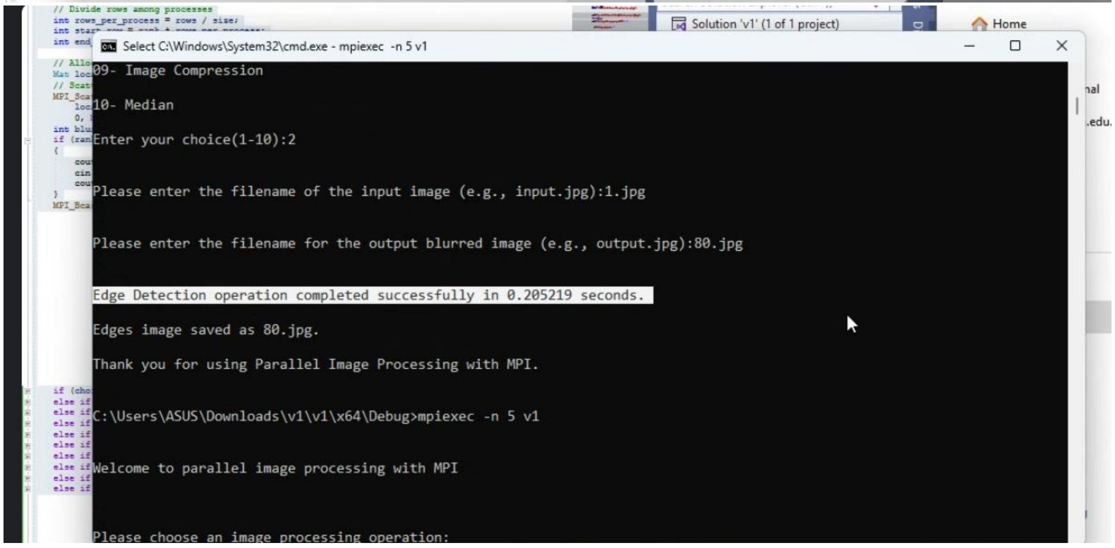
```
Please enter the filename for the output blurred image (e.g., output.jpg):9.jpg


Please enter the blur radius(e.g., 3) :45


Processing image with Gaussian Blur...

[ INFO:0@9.625] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry core
(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@9.625] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@9.627] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_one
tbb490_64d.dll => FAILED
[ INFO:0@9.627] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@9.629] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb
490_64d.dll => FAILED
[ INFO:0@9.629] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@9.630] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_ope
nmp490_64d.dll => FAILED
Gaussian Blur operation completed successfully in 0.34549 seconds.

Blurred image saved as output_image.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>
```



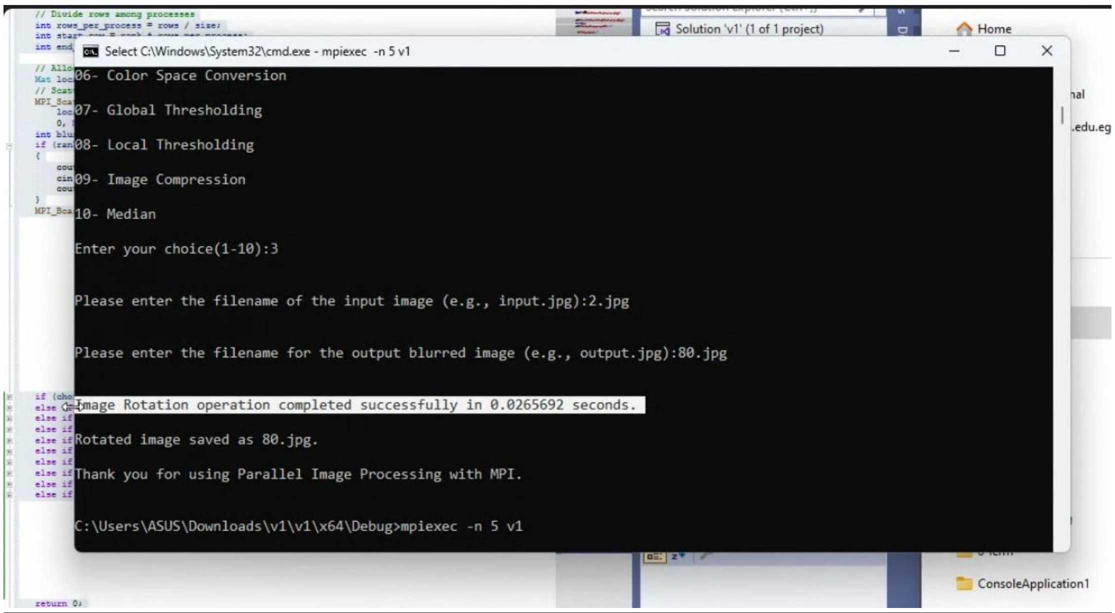
```
[ INFO:0@32.041] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry cor
e(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@32.041] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@32.043] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_on
etbb490_64d.dll => FAILED
[ INFO:0@32.043] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@32.044] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tb
b490_64d.dll => FAILED
[ INFO:0@32.044] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@32.046] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_op
enmp490_64d.dll => FAILED
Gaussian Blur operation completed successfully in 0.360723 seconds.

Blurred image saved as output_image.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 4 v1


Welcome to parallel image processing with MPI
```



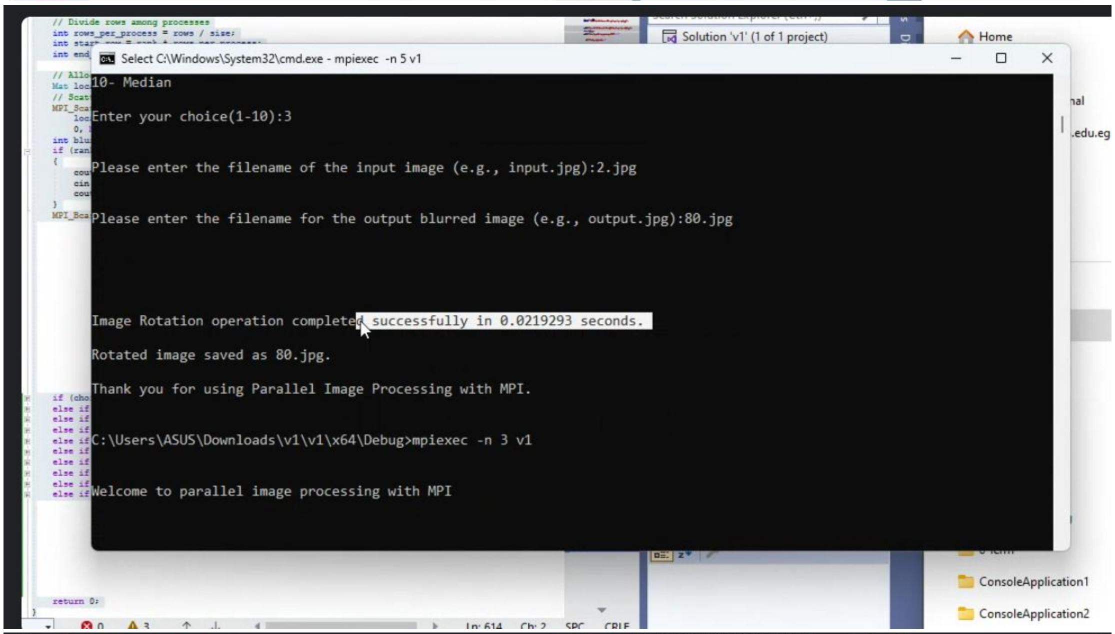
```
07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):2


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Edge Detection operation completed successfully in 0.0458984 seconds.

Edges image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 3 v1
```



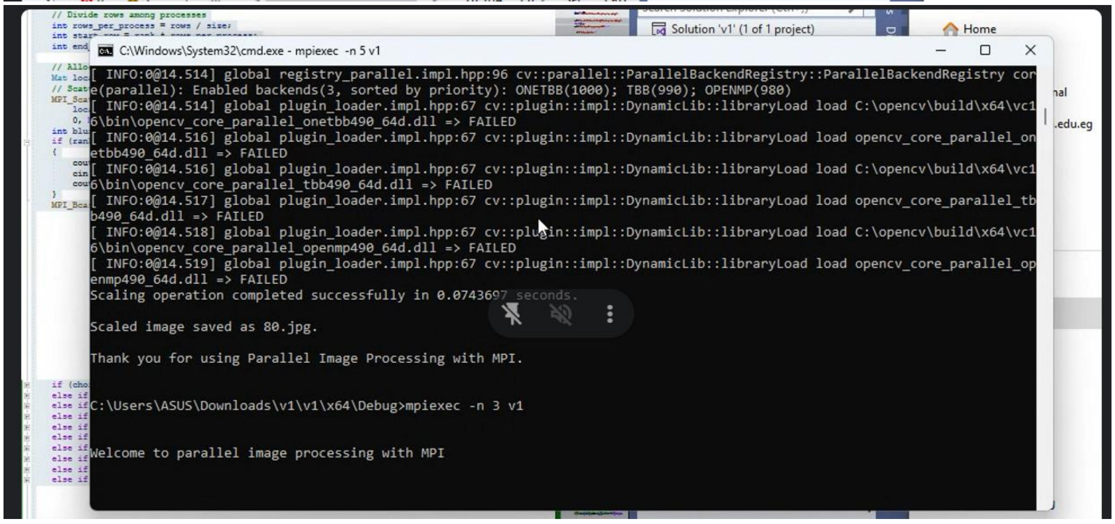
```
09- Image Compression

10- Median

Enter your choice(1-10):2


Please enter the filename of the input image (e.g., input.jpg):1.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Edge Detection operation completed successfully in 0.205219 seconds.

Edges image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 5 v1


Welcome to parallel image processing with MPI


Please choose an image processing operation:
```



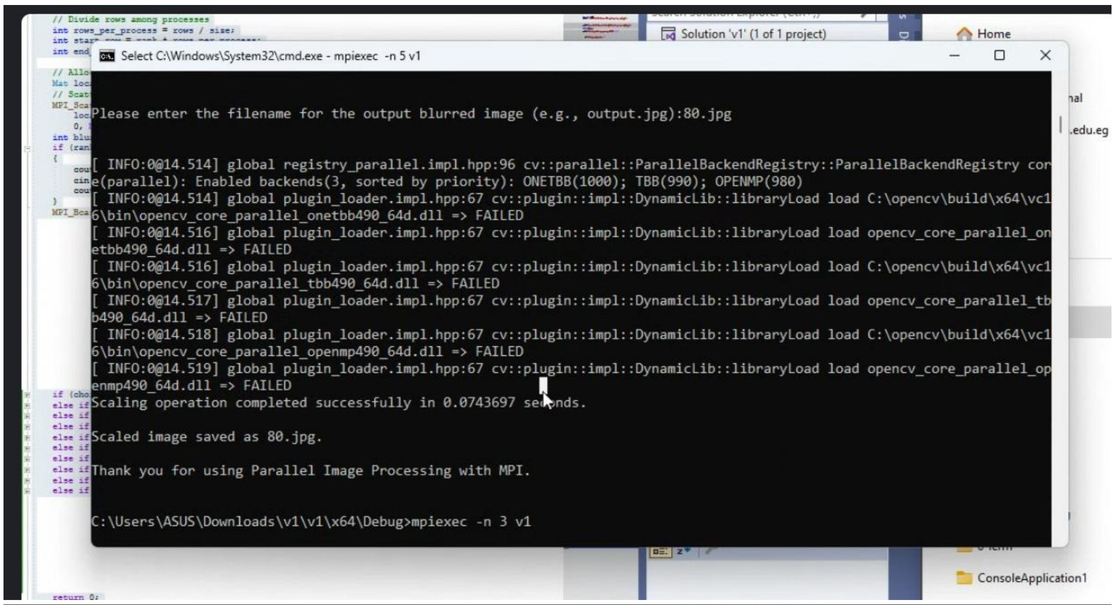
```
06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):3

Please enter the filename of the input image (e.g., input.jpg):2.jpg

Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg

Image Rotation operation completed successfully in 0.0265692 seconds.

Rotated image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.

C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 5 v1
```



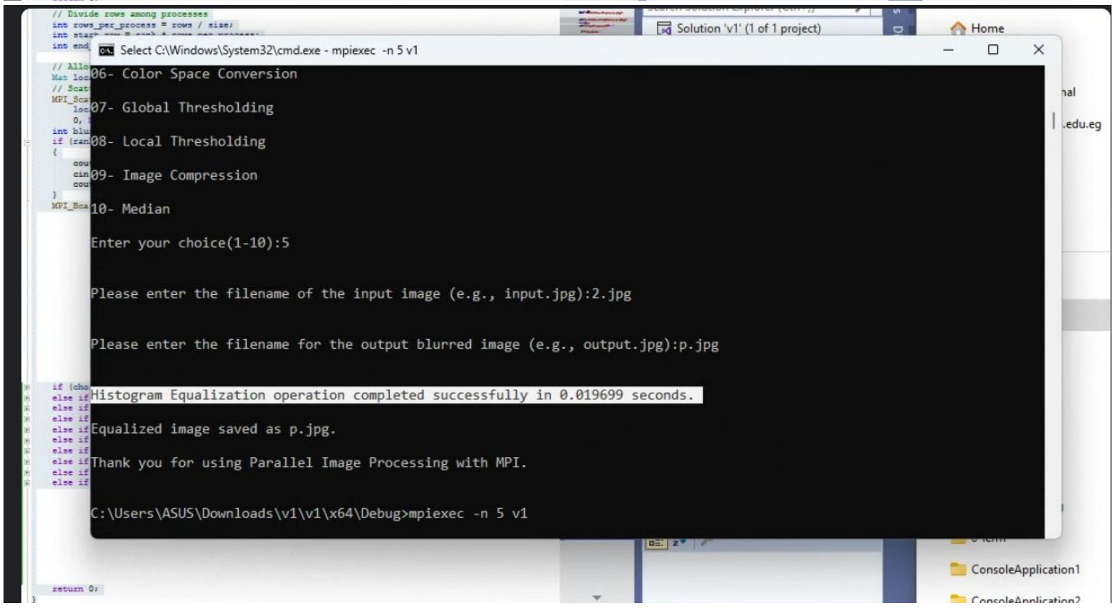
```
10- Median

Enter your choice(1-10):3

Please enter the filename of the input image (e.g., input.jpg):2.jpg

Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg



Image Rotation operation completed successfully in 0.0219293 seconds.

Rotated image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.

C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 3 v1

Welcome to parallel image processing with MPI
```



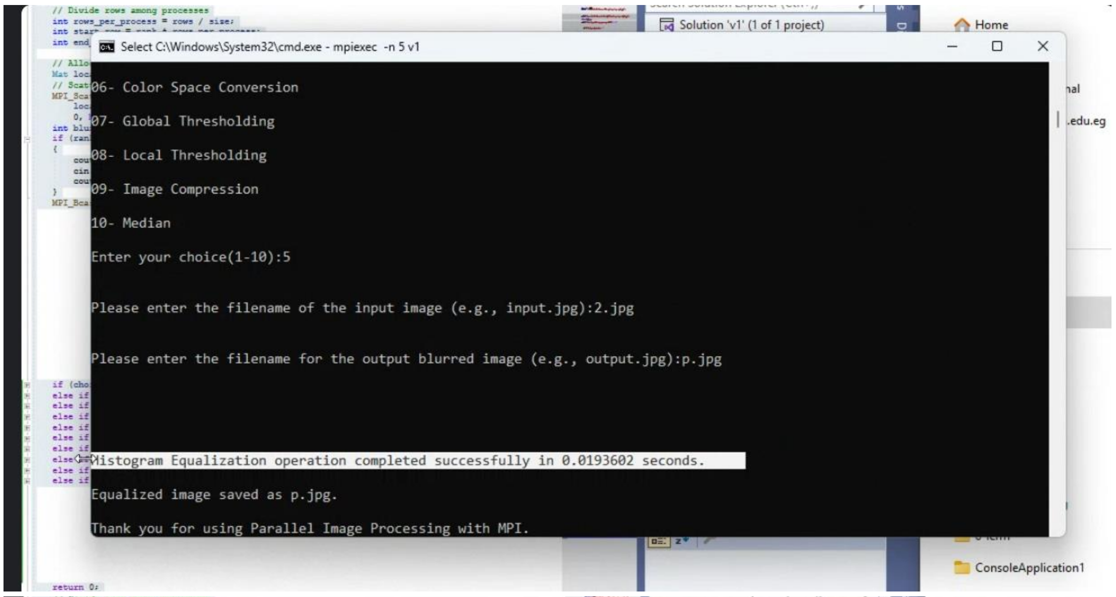
```
[ INFO:0@14.514] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry cor
e(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@14.514] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@14.516] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_on
etbb490_64d.dll => FAILED
[ INFO:0@14.516] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@14.517] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tb
b490_64d.dll => FAILED
[ INFO:0@14.518] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@14.519] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_op
enmp490_64d.dll => FAILED
Scaling operation completed successfully in 0.0743697 seconds.

Scaled image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 3 v1


Welcome to parallel image processing with MPI
```



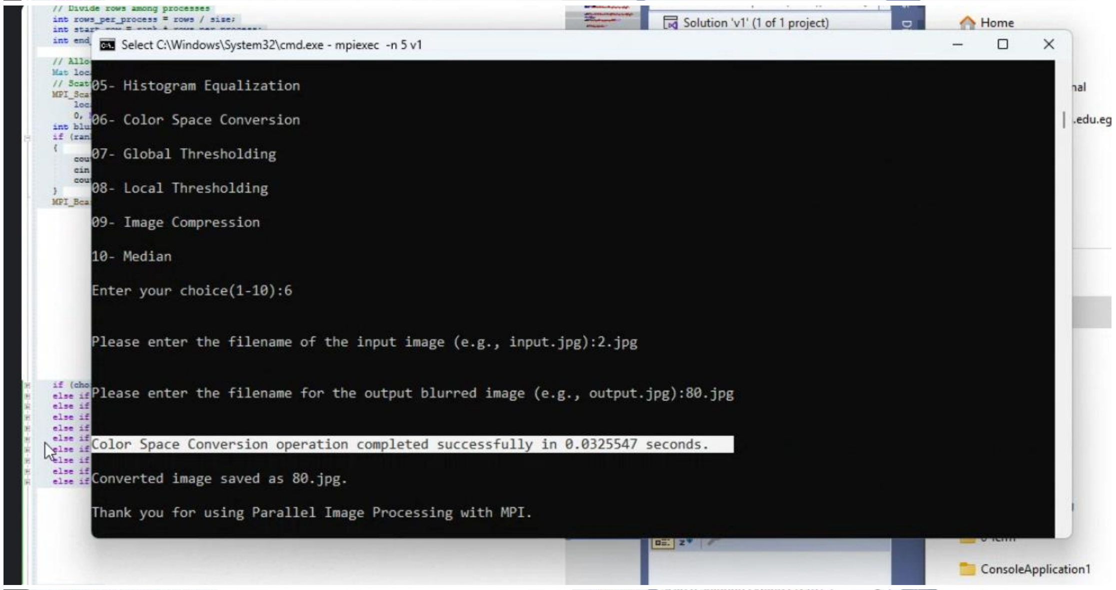
```
Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


[ INFO:0@14.514] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry cor
e(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@14.514] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@14.516] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_on
etbb490_64d.dll => FAILED
[ INFO:0@14.516] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@14.517] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tb
b490_64d.dll => FAILED
[ INFO:0@14.518] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc1
6\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@14.519] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_op
enmp490_64d.dll => FAILED
Scaling operation completed successfully in 0.0743697 seconds.

Scaled image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 3 v1
```



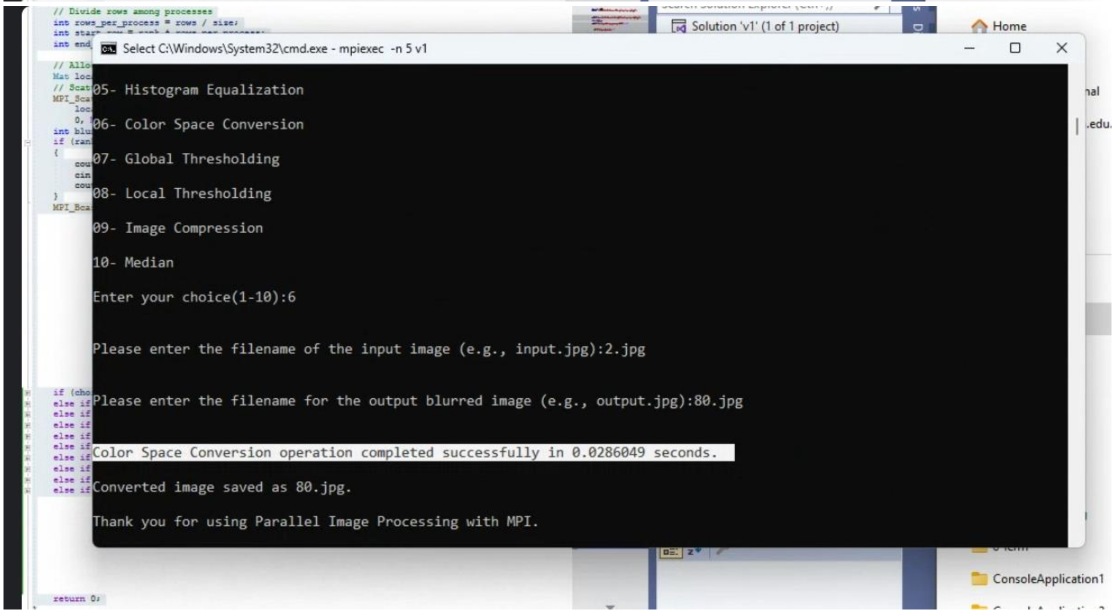
```
06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):5


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):p.jpg


Histogram Equalization operation completed successfully in 0.019699 seconds.

Equalized image saved as p.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n 5 v1
```



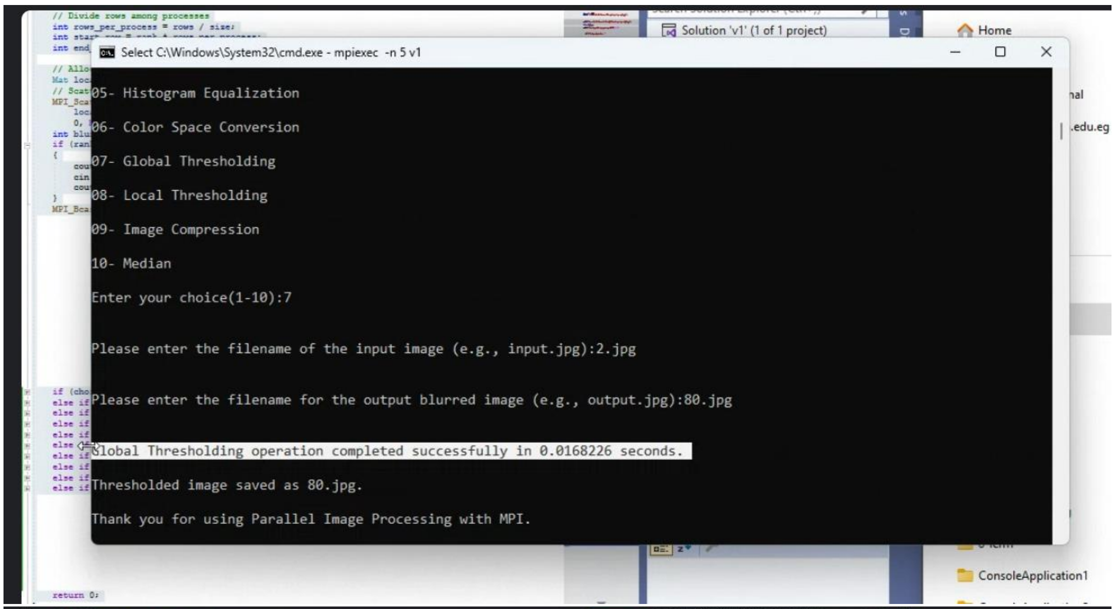
```
Select C:\Windows\System32\cmd.exe - mpiexec -n 5 v1

06- Color Space Conversion
07- Global Thresholding
08- Local Thresholding
09- Image Compression
10- Median
Enter your choice(1-10):5

Please enter the filename of the input image (e.g., input.jpg):2.jpg

Please enter the filename for the output blurred image (e.g., output.jpg):p.jpg



Histogram Equalization operation completed successfully in 0.0193602 seconds.
Equalized image saved as p.jpg.
Thank you for using Parallel Image Processing with MPI.
```



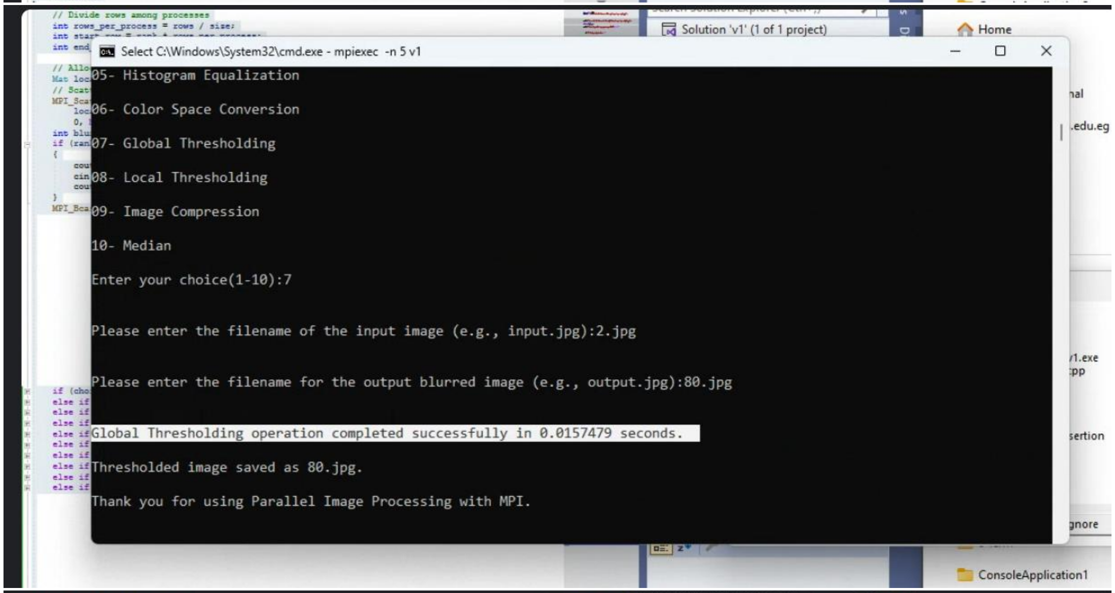
```
Select C:\Windows\System32\cmd.exe - mpiexec -n 5 v1

05- Histogram Equalization
06- Color Space Conversion
07- Global Thresholding
08- Local Thresholding
09- Image Compression
10- Median
Enter your choice(1-10):6

Please enter the filename of the input image (e.g., input.jpg):2.jpg

Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg

Color Space Conversion operation completed successfully in 0.0325547 seconds.
Converted image saved as 80.jpg.
Thank you for using Parallel Image Processing with MPI.
```



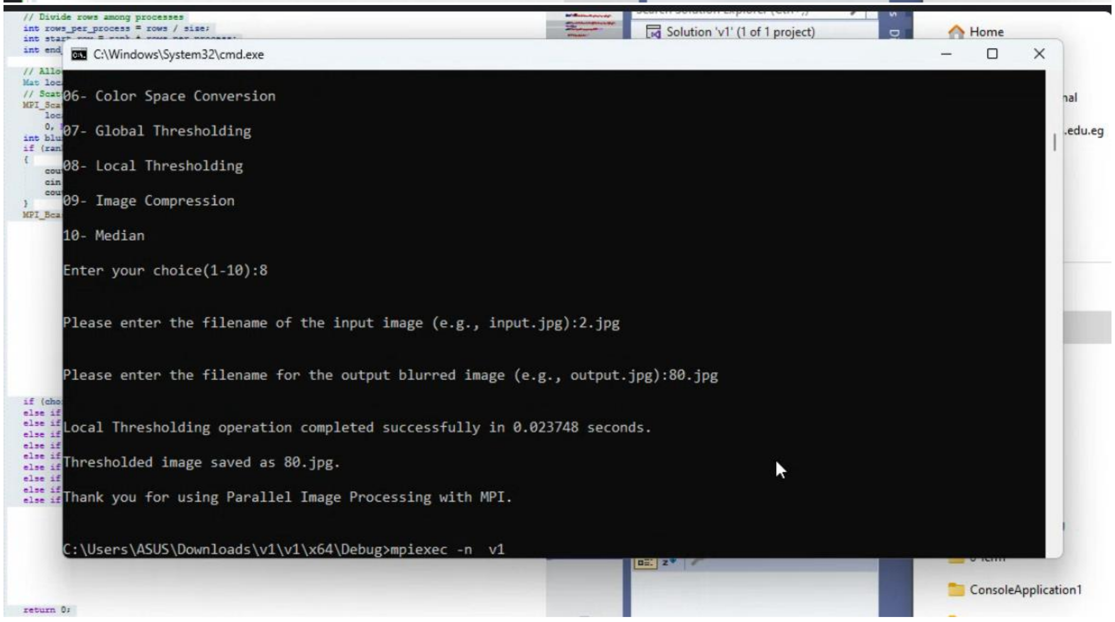
```
Select C:\Windows\System32\cmd.exe - mpiexec -n 5 v1

05- Histogram Equalization
06- Color Space Conversion
07- Global Thresholding
08- Local Thresholding
09- Image Compression
10- Median
Enter your choice(1-10):6

Please enter the filename of the input image (e.g., input.jpg):2.jpg

Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg

Color Space Conversion operation completed successfully in 0.0286049 seconds.
Converted image saved as 80.jpg.
Thank you for using Parallel Image Processing with MPI.
```


Select C:\Windows\System32\cmd.exe - mpiexec -n 5 v1
Solution 'v1' (1 of 1 project)
Home
ConsoleApplication1

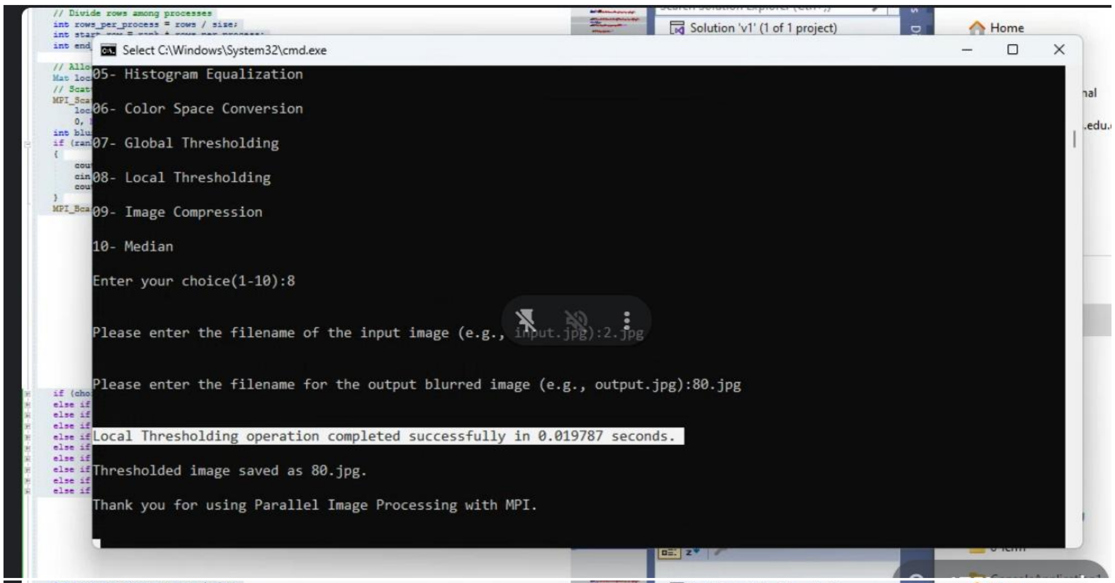
```
05- Histogram Equalization

06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):7


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Global Thresholding operation completed successfully in 0.0168226 seconds.

Thresholded image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.
```


Select C:\Windows\System32\cmd.exe - mpiexec -n 5 v1
Solution 'v1' (1 of 1 project)
Home
ConsoleApplication1

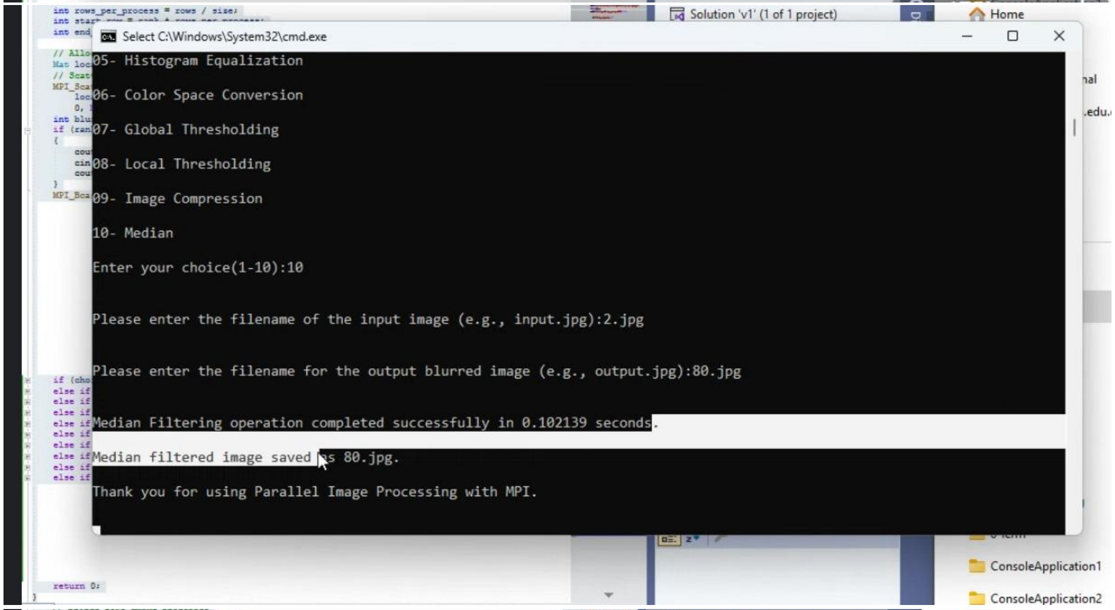
```
05- Histogram Equalization

06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):7


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Global Thresholding operation completed successfully in 0.0157479 seconds.

Thresholded image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.
```



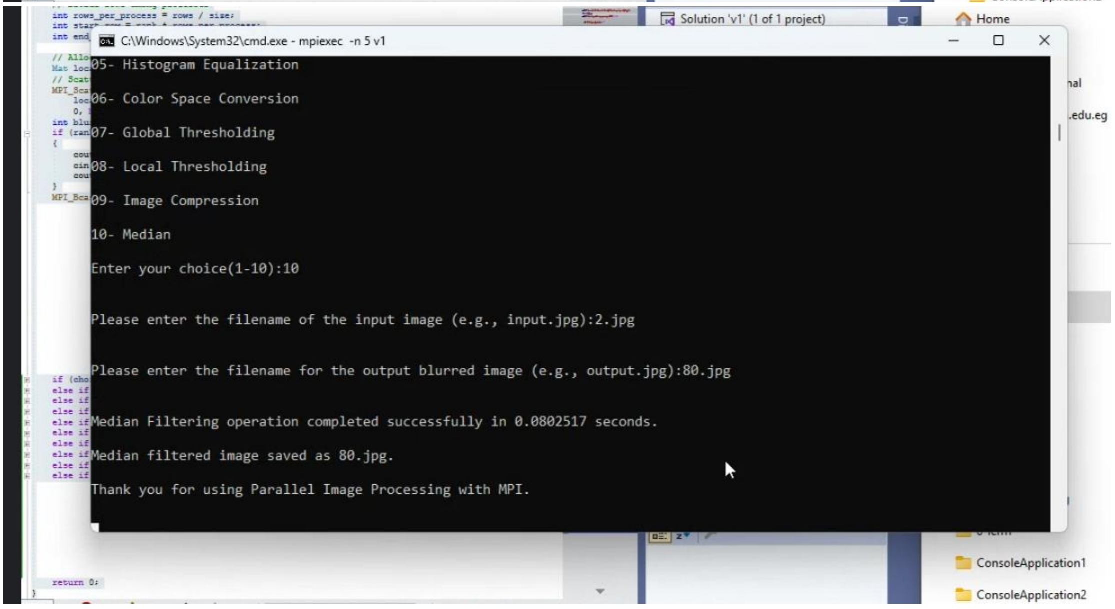
```
06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):8


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Local Thresholding operation completed successfully in 0.023748 seconds.

Thresholded image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.


C:\Users\ASUS\Downloads\v1\v1\x64\Debug>mpiexec -n  v1
```



```
05- Histogram Equalization

06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):8


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Local Thresholding operation completed successfully in 0.019787 seconds.

Thresholded image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.
```



```
05- Histogram Equalization

06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):10


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Median Filtering operation completed successfully in 0.102139 seconds.

Median filtered image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.
```



```
05- Histogram Equalization

06- Color Space Conversion

07- Global Thresholding

08- Local Thresholding

09- Image Compression

10- Median

Enter your choice(1-10):10


Please enter the filename of the input image (e.g., input.jpg):2.jpg


Please enter the filename for the output blurred image (e.g., output.jpg):80.jpg


Median Filtering operation completed successfully in 0.0802517 seconds.

Median filtered image saved as 80.jpg.

Thank you for using Parallel Image Processing with MPI.
```

5. Conclusion

Parallel Image Processing with MPI provides an efficient approach to perform image processing tasks in parallel on distributed systems. By leveraging MPI for parallelization and OpenCV for image processing, the application achieves faster execution times and improved scalability. Further optimization and tuning based on performance analysis results can enhance the overall efficiency of the application.