# Parallel and Distributed Computing Spring 2024

## Parallelising and Distributing Image Denoising Using NLM

**Hiba Mallick - 24015 & Muhammad Musab Iqbal - 24495**

The Non-Local Means (NLM) algorithm is a popular technique for image denoising, a crucial task in computer vision, aiming to remove noise from images to obtain the original image for improved quality and performance in applications like visual tracking and image classification. However, its sequential implementation becomes computationally expensive for large images, hindering real-time applications.

With this project, we aimed to parallelize and distribute the NLM algorithm using MPI and CUDA C. These approaches significantly reduced processing time by leveraging parallel and distributed computing.

To analyze performance, we used images of different pixel sizes 200×200, 270×277, and 400×600. The limited nature of these is due to the limited processing power of equipment.

Running our sequential MPI and CUDA implementations, we compared performances based on execution time with different image sizes, the speedup in execution time achieved, and the overall scalability of our accelerated implementations.

For all our sequential and MPI implementations, we used the stb_image.h library in C for image processing and used image pixel values in CUDA.

We kept the parameters for denoising consistent:

- PATCH_SIZE 7
- SEARCH_WINDOW_SIZE 21
- H 10.0

The mathematical equation for NLM algorithms is:

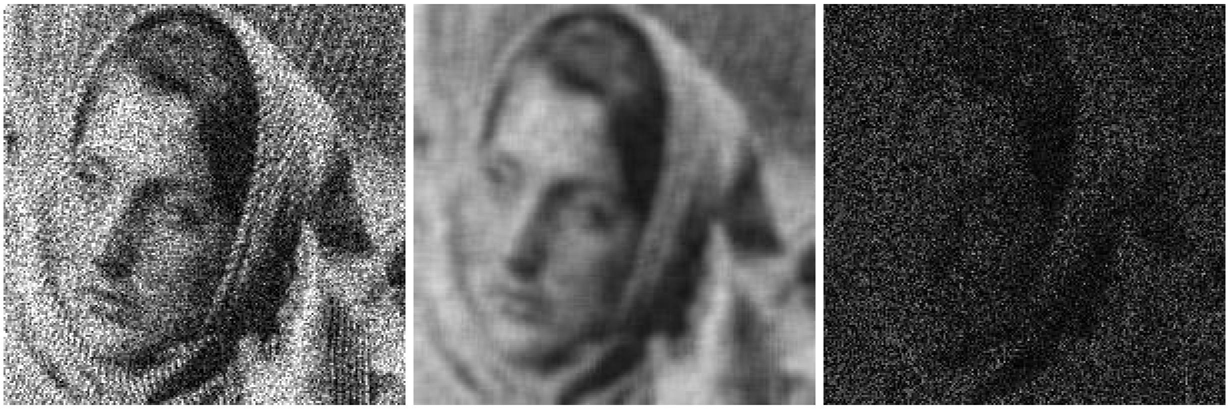
$$\hat{f}(x)| = \sum_{y \in \Omega} w(x,y) f(y), \quad \forall x \in \Omega$$

### Sequential Implementation

In the sequential implementation, we will implement the NLM algorithm by giving a noisy input image file. The code iterates over each pixel in the image, calculating a weighted average of surrounding pixels to denoise the image and outputs an image file. We ran this implementation over different image sizes and averaged the time across 10 runs.

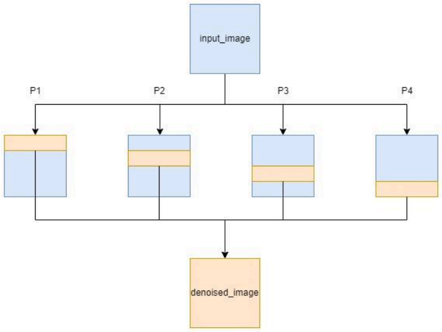
| Image Size | Total Pixels | Time (in seconds) |
|---|---|---|
| 200×200 | 40,000 | 0.0705393 seconds |
| 270×277 | 74,790 | 0.116629 seconds |
| 400×600 | 240,000 | 0.3535246 seconds |

The sequential implementation is very slow, considering how big image sizes can get today and how many images need to be denoised in real-life uses like visual tracking.



### PRAM Model

The NLM algorithm can be parallelized using a CREW PRAM model in the following way:

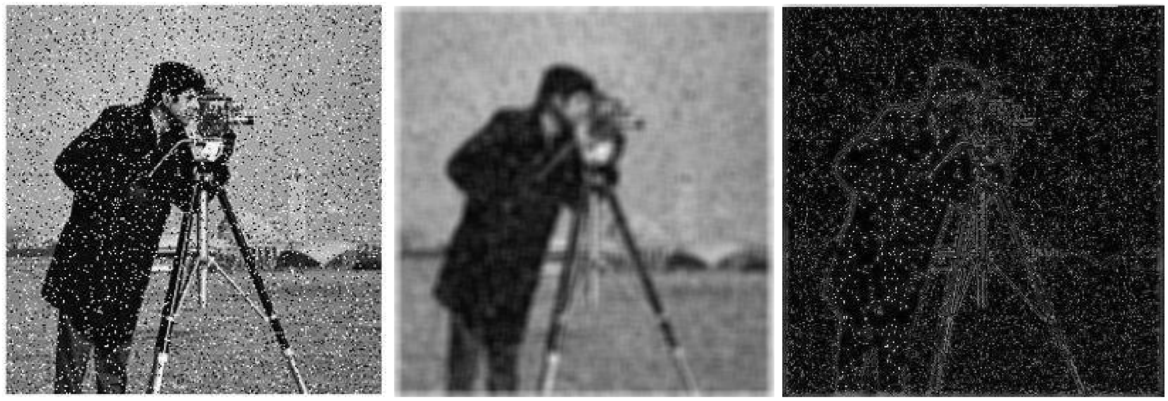


### MPI Implementation

We will implement a parallel NLM algorithm using MPI to denoise an input image by distributing the computation across multiple processes.

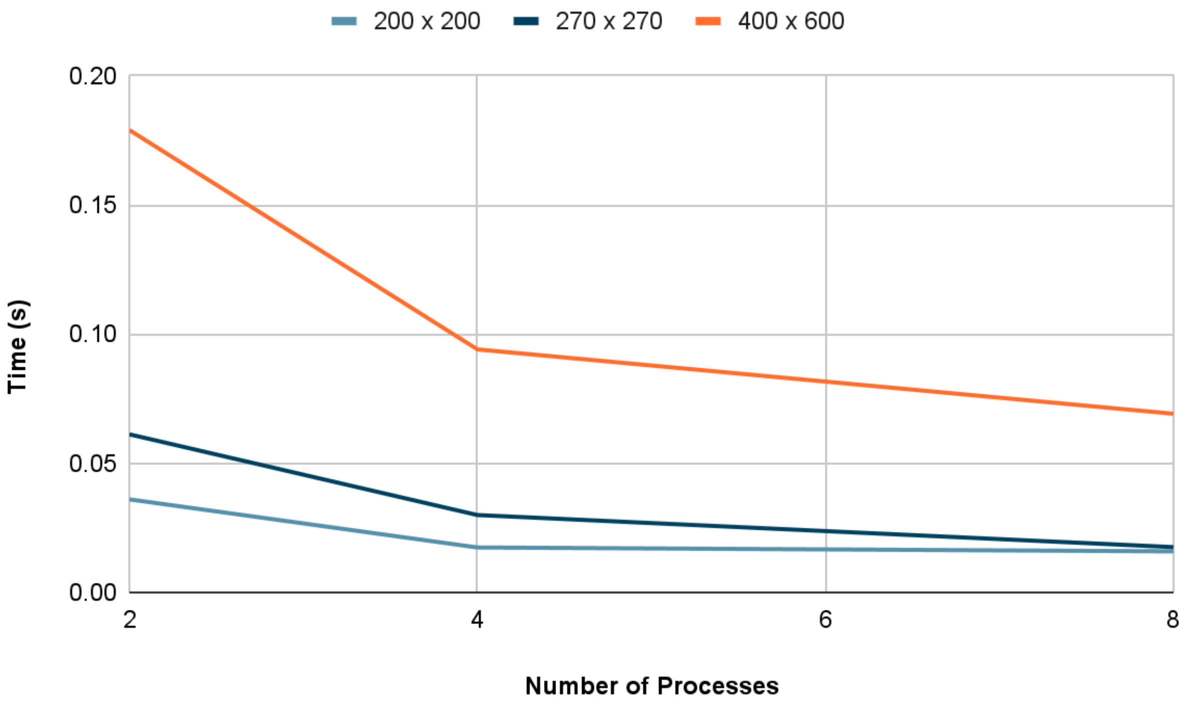
This is implemented by writing a non_local_means_denoise function that denoises a segment of the image. Each portion calculates the denoise value for its assigned rows of the image and stores the result.

For the MPI section, the dimensions and pixel data are broadcast to all the processes, and they each denoise their portion of the image using the function independently. MPI communication primitives will be used to exchange information needed for calculating weighted averages across process boundaries. Finally, the denoised image chunks from all processes will be gathered and combined to form the final denoised image.



Here is the time analysis of non-local means compared over 3 image sizes and varying numbers of processes on the MPI cluster:

| | 200 x 200 | 270 x 270 | 400 x 600 |
|---|---|---|---|
| 2 processes | 0.036015 | 0.061196 | 0.179008 |
| 4 processes | 0.017392 | 0.029949 | 0.094096 |
| 8 processes | 0.015890 | 0.017526 | 0.069170 |

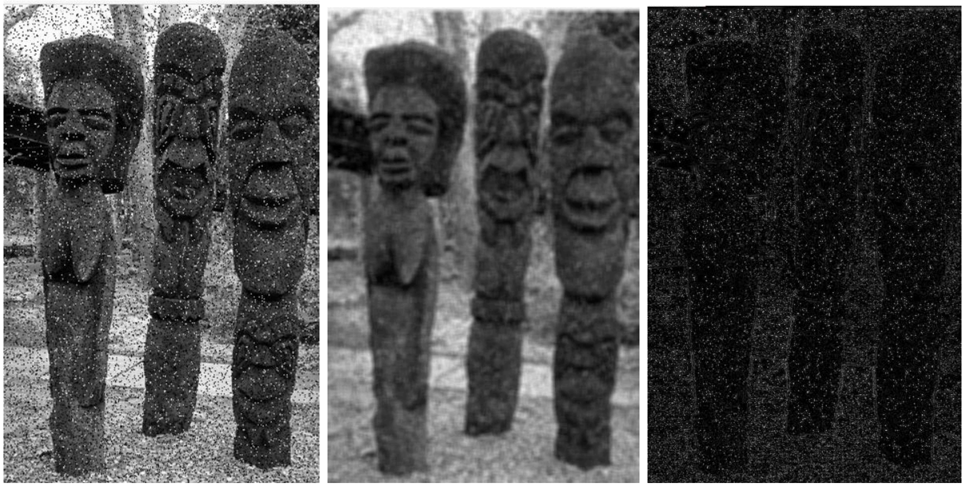


### CUDA C Implementation

Finally, we implemented the parallel NLM algorithm using CUDA C to leverage the parallel processing power of GPUs to achieve even higher performance than MPI on a cluster.

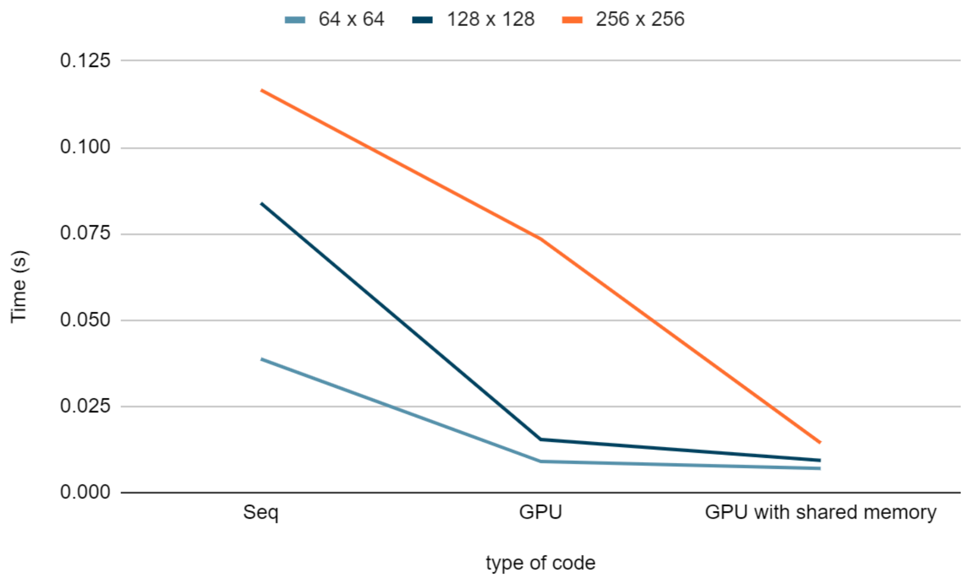
In the GPU with local memory implementation, each thread calculates the filtered value for a pixel using local memory. The kernel reads image data, computes patch distances, and then applies the NLM algorithm to obtain the filtered value and updates its local variable.

In the GPU (shared memory with global variable)implementation of the NLM algorithm, we used shared memory for improved performance. Each thread loads relevant parts of the image into shared memory to reduce global memory accesses. It uses shared memory to calculate the filtered value more efficiently than the global memory approach. This is similar to the global memory version, but the kernel configuration includes shared memory allocation.



Here is the time analysis of non-local means compared over 3 image sizes and sequential, GPU with local memory and GPU (shared memory with global variable):

| | 64X64 | 128X128 | 256X256 |
|---|---|---|---|
| Seq | 0.03869 | 0.08389 | 0.116629 |
| GPU (local var) | 0.00901126 | 0.015387 | 0.033493 |
| GPU (shared memory with global var) | 0.004987 | 0.009328 | 0.014376 |



### Performance Evaluation

We will evaluate the performance of the parallel NLM algorithm using MPI across various image sizes and data set sizes using the speedup factor of MPI and CUDA GPU programming to accelerate the sequential NLM algorithm.

**MPI Speedup:**
200 x 200: **443.9%**
270 x 277: **665.5%**
400 x 600: **511.1%**

**CUDA C Speedup (local variable)**:
64 x 64: **429.3%**
128 x 128: **545.2%**
256 x 256: **348.2%**

**CUDA C Speedup (shared memory global variable):**
64 x 64: **775.8%**
128 x 128: **899.3%**
256 x 256: **811.3%**

### GitHub Repository

https://github.com/mallickhiba/Image-Denoising-with-Parallel-Distributed-Non-Local-Means.git