GPU-Accelerated Image Denoising
Term Project, CSC 746, Fall 2022

Malavya Raval*
SFSU

ABSTRACT

This project addresses the pivotal exploration of whether GPU acceleration significantly amplifies the performance of image denoising algorithms, with a dedicated focus on the Non-Local Means (NLM) algorithm. Our investigation delves into optimal parallelism and memory utilization, strategically geared towards maximizing computational throughput. In the pursuit of understanding scalability concerning image size and complexity, we employ a comprehensive approach, implementing GPU-accelerated versions across a spectrum of algorithms. The study unfolds with the serial, GPU CUDA, and GPU CUDA with shared memory implementations of NLM. While serial performance exhibits a slower runtime, GPU implementations perform better with accelerated processing. Intriguingly, shared memory, though marginally superior, outperforms CUDA in isolation. A vivid array of graphical representations shed light on the nuanced impact of parallelization and the variation in job sizes, presenting crucial insights into the realm of GPU-accelerated image processing.

1 INTRODUCTION

This project addresses a critical question: Can GPU acceleration substantially improve the performance of the Non-Local Means (NLM) algorithm in image denoising? Our focus is on optimal parallelism, memory utilization, and computational throughput. We aim to assert the transformative potential of GPU acceleration for NLM without delving into extensive problem context in this introduction.

There have been previous implementations of NLM algorithm but the implementations are complex and does not have precices measurements such as chrono timer implemetaion. This report implemets the NLM algorith across serial, GPU with CUDA and GPU CUDA with memory utilization with better utilizartion of memory and using libraries to get precise results.

Despite advancements in image denoising algorithms, a critical challenge persists of achieving substantial acceleration through GPU remains a formidable task. Existing solutions often grapple with suboptimal parallelism and memory utilization, limiting their ability to exploit the full computational potential of GPUs. The intricate nature of image denoising, particularly in algorithms like Non-Local Means (NLM), demands a nuanced approach for effective parallelization.

Many current implementations lack the necessary optimization for GPU architectures, resulting in subpar performance gains. Moreover, the ideal level of parallelism and memory utilization for diverse image sizes and complexities remains elusive. While some solutions provide partial acceleration, a comprehensive approach that systematically addresses the interplay between GPU architecture, algorithm design, and image characteristics is lacking. Our work aims to fill this crucial gap by offering a novel perspective on GPU-accelerated image denoising, with a focus on NLM, backed by efficient parallelization and tailored memory management.

*email:mraval@sfsu.edu

$$0 \leq w(i,j) \leq 1 \text{ and } \sum_j w(i,j) = 1.$$

Figure 1: We will compute the weighted average

Unlike conventional methods, our solution meticulously optimizes parallelism and memory utilization, ensuring a best performance between algorithmic design and GPU architecture. This approach leads to superior computational throughput.

Our work includes a comparative analysis with other GPU-accelerated and CPU-based solutions, showcasing the superior performance of our implementation. This analysis provides a clear benchmark for the effectiveness of our approach.

The remainder of this paper is organized as follows :- We delve into existing literature and solutions in Section 2, providing a comprehensive overview of the approaches used for of image denoising which includes serial, GPU and GPU with memory utilization implementations. This sets the stage for understanding the context in which our work unfolds.

In Section 3, we present a detailed account of our implementations of image denoising. This section encompasses the technical aspects, design considerations, and challenges encountered during the implementation phase.

Section 4 is dedicated to the evaluation of our system. We discuss the methodology employed for assessing performance, present key metrics, and provide graphical representations of our results, offering a comprehensive analysis.

The final section, Section 5, offers conclusions drawn from our findings and outlines avenues for future research and development in the domain of our different image denoising implementations.

2 RELATED WORK

Various efforts have been made to address the challenges of image denoising, particularly focusing on the Non-Local Means (NLM) algorithm. These endeavors contribute valuable insights and advancements to the field. In evaluating existing methods, it is crucial to present a balanced view by acknowledging both strengths and weaknesses.

One notable approach involves Jurnal of Parallel and Distrubuted Computing, which demonstrates commendable achievements in optimizing parallelism and memory utilization. Their implementation showcases efficient denoising capabilities and provides a solid benchmark for comparison. However, limitations arise in the intricate handling of larger image sizes and complexities, where computational throughput becomes a critical factor.

The weights are used to find the value of patches which will be used in the NLM algorithm.

Another significant contribution from Parallel Non-Local Means Image Denoising emphasizes the importance of algorithmic design in achieving superior denoising results. Their work excels in certain aspects, particularly using the Gausian Filter. Yet, challenges persist in scalability, hindering its effectiveness with more extensive datasets.

$$NL[v](i) = \sum_{j \in I} w(i,j)v(j),$$

Figure 2: NL[v](i), for a pixel i, The NL-means not only compares the grey level in a single point but the the geometrical configuration in a whole neighborhood

While these existing solutions contribute significantly to the domain, our approach aims to build upon their strengths while addressing their limitations. By optimizing parallelism, memory utilization, and incorporating improved runtime calculations and Gaussian filter implementations, our method strives to offer a more comprehensive and efficient solution for GPU-accelerated image denoising. This collaborative perspective fosters an environment of mutual respect within the research community, promoting ethical engagement and constructive dialogue.

While these provide valuable groundwork, our solution distinguishes itself by offering a tailored approach precisely suited to the nuances of our problem. By leveraging optimized parallelism, memory utilization, and advancements in runtime calculations and Gaussian filter implementations, our solution stands out as a more effective and comprehensive strategy for tackling the challenges posed by GPU-accelerated image denoising in the context of the Non-Local Means algorithm.

3 Implementation

In this section, we provide a comprehensive overview of our implementation, offering insights into the design and structure that collectively address the challenges posed by GPU-accelerated image denoising with my three implementations of serial, GPU and GPU with shared memory.

The subsequent sections of this paper delve into the implementation specifics of three distinct approaches: serial, GPU with CUDA, and GPU with CUDA and shared memory. Each subsection discusses the unique characteristics of the implementation, highlighting the tailored adaptations made to leverage GPU acceleration effectively. The structure of these subsections mirrors the progression of our implementation journey, offering a cohesive narrative that culminates in a comparative analysis of the three approaches. https://www.overleaf.com/project/656b559b476444a5fb025f8d

3.1 Gaussian Filter Implementation

Central to our solution is an optimized Gaussian filter, a foundational element used across all three implementations. This section details the key aspects of our Gaussian filter code, elucidating the intricacies of its design and its critical role in enhancing the denoising capabilities of our approach. The code snippet shows my gaussian filter implementation which preformance on all the pixels and considers the patch size to filter the pixel, if the patch size is too high then the image turns out to be very blurry and if its too small then the image is not filtered with noise.

3.2 Patch Comparison Function

Integral to our GPU-accelerated image denoising solution is the patch comparison function, a crucial component utilized with minor variations across all implementations. The function, outlined in the provided code snippet, calculates the difference between two patches, contributing significantly to the Non-Local Means (NLM) algorithm's effectiveness.

The function iterates over the pixels of two patches, considering the padding introduced for boundary conditions. It computes the squared differences between corresponding pixels, weighting them

```
float *gaussian_filter(){
    float *G = (float *)malloc(PATCH_SIZE*
    PATCH_SIZE*sizeof(float));
    if(G == NULL){
        exit(1);
  }
  // bound for the 2D Gaussian filter
    int bound = PATCH_SIZE/2;
    for(int x=-bound; x<=bound; x++){
        for(int y=-bound; y<=bound; y++){
      int index = (x+bound)*PATCH_SIZE + (y+bound)
    ;
            G[index] = exp( -(float)(x*x+y*y)/(
    float)(2*PATCH_SIGMA*PATCH_SIGMA) ) / (float)
    (2*M_PI*PATCH_SIGMA*PATCH_SIGMA);
        }
    }
    return G;
}
```

Listing 1: Gaussian Filter

by the Gaussian function values stored in the array G. The resulting patch comparison score encapsulates the dissimilarity between patches, a key metric in the NLM algorithm.

This patch comparison function operates efficiently within the GPU-accelerated context, leveraging parallel processing to enhance its computational throughput. Its role extends beyond a mere utility function, acting as a cornerstone in the overall denoising process. The subsequent sections detail how this function is strategically integrated into each implementation, showcasing its adaptability and impact on the overall performance of our GPU-accelerated image denoising solution.

```
float compare_patches(int i, int j, float *G){
    float patch_comp = 0;
    for(int m=0; m<PATCH_SIZE; m++){
        for(int n=0; n<PATCH_SIZE; n++){
      int first_index = i + (m-padding)*(PIXELS+2*
    padding) + n - padding; // reffering to a
    pixel from i's patch
      int second_index = j + (m-padding)*(PIXELS
    +2*padding) + n - padding; // reffering to a
    pixel from j's patch

      // image[x] == -1 means it's the added
    padding
            if((image[first_index] != (float)-1)
    && (image[second_index] != (float)-1)){
                float diff = image[first_index] -
    image[second_index];
                patch_comp += G[m*PATCH_SIZE+n]*(
    diff*diff);
            }
        }
    }
    return patch_comp;
}
```

Listing 2: Patch Comparison

3.3 Non-Local Means (NLM) Algorithm

Our implementation revolves around the NLM algorithm, a cornerstone in image denoising. Here, we provide an in-depth exploration

of our NLM code, emphasizing the nuances of its design and its adaptability across diverse implementations.

3.4 Serial Implementation of Non-Local Means

Our serial implementation of the Non-Local Means (NLM) algorithm forms the basis for our denoising solution. In this section, we outline the core components of the serial version. The algorithm iterates through image patches, calculates patch distances, and computes weighted averages. The patch comparison function, crucial for the NLM algorithm, is detailed, with a focus on memory access patterns and computations. This serial implementation lays the groundwork for understanding the subsequent GPU-accelerated versions.

```
    image = image_from_txt(PIXELS, padding);
    struct timespec tic, toc;

    clock_gettime(CLOCK_MONOTONIC, &tic);
    image = nonLocalMeans();
    clock_gettime(CLOCK_MONOTONIC, &toc);

    FILE *f = fopen("filtered_image.txt", "w");
  if(f == NULL){
    printf("Cannot open filtered_image.txt\n");
    exit(1);
  }

    int pixels_counter = 0;
  int padded_size = PIXELS*PIXELS + 4*padding*
    PIXELS + 4*padding*padding;
  int start = PIXELS*padding + 2*padding*padding +
     padding; //skip first padding rows

    for(int i=start; i<(padded_size-start); i++){
        fprintf(f, "%f ", image[i]);
        pixels_counter++;
        if(pixels_counter == PIXELS){
            pixels_counter = 0;
      i += 2*padding;
            fprintf(f, "\n");
        }
    }
    fclose(f);
    free(image);

  printf("*NLM-Serial Duration = %f second(s)* ||
    (Pixels, Patch) = (%d, %d)\n", elapsed_time(
    tic,toc), PIXELS, PATCH_SIZE);
    return 0;
```

Listing 3: Serial NLM

3.5 GPU CUDA Implementation of Non-Local Means

Moving beyond the serial paradigm, our GPU CUDA implementation takes advantage of parallel processing capabilities to significantly enhance the NLM algorithm's performance. We elaborate on the parallelization strategy, leveraging GPU architecture to concurrently process multiple patches. The incorporation of CUDA programming constructs and shared memory optimization is highlighted. Details about memory transfers between the CPU and GPU, kernel execution, and overall performance considerations are discussed. This section elucidates how GPU acceleration fundamentally alters the denoising workflow.

```
__host__ float *nonLocalMeans(float *host_image){
  int padded_size = PIXELS*PIXELS + 4*HOST_PADDING
    *PIXELS + 4*HOST_PADDING*HOST_PADDING;

    float *G;
  cudaMallocManaged(&G, PATCH_SIZE*PATCH_SIZE*
    sizeof(float));
  if(G == NULL){
        exit(1);
    }
  float *temp = gaussian_filter();
  memcpy(G, temp, PATCH_SIZE*PATCH_SIZE*sizeof(
    float));

  float *image;
  cudaMallocManaged(&image, padded_size*sizeof(
    float));
  if(image == NULL){
        exit(1);
    }
  memcpy(image, host_image, padded_size*sizeof(
    float));

  float *filtered_image;
  cudaMallocManaged(&filtered_image, padded_size*
    sizeof(float));
  if(filtered_image == NULL){
        exit(1);
    }

  for(int i=0; i<padded_size; i++){
    filtered_image[i]=(float)-1;
  }

    denoise_image<<<PIXELS, PIXELS>>>(
    filtered_image, image, padded_size, G);
  cudaDeviceSynchronize();
]
  cudaFree(G);
  cudaFree(image);
    return filtered_image;
}
```

Listing 4: GPU NLM

3.6 Optimizing GPU CUDA with Shared Memory for Non-Local Means

Taking our GPU CUDA implementation a step further, we've boosted efficiency by tapping into shared memory. This section breaks down how we've implemented shared memory into the Non-Local Means (NLM) algorithm on the GPU. This tweak is crucial for smarter memory use during patch comparisons and weighted averaging. We walk you through the changes we made to the algorithm, explaining why each adjustment helps. To show you the impact, we compare the performance boost achieved with shared memory against our baseline CUDA setup without it. This section rounds off our trio of NLM implementations, giving you a clear picture of how our GPU-accelerated denoising solution evolved and improved.

4 Evaluation

To rigorously assess the effectiveness of our solution, we employed a systematic testing approach encompassing various performance metrics, parameters, and experimental designs.

```
2  int shared_memory_size = PATCH_SIZE*(PIXELS + 2*
     HOST_PADDING);
3    denoise_image<<<PIXELS, PIXELS,
     shared_memory_size*sizeof(float)>>>(
     filtered_image, image, padded_size, G);
4  cudaDeviceSynchronize();
```

Listing 5: Stencil computation in 2D: performs sum of product of nearby pixels with weights.

4.1 Computational platform and Software Environment

All the programs were run using CPU nodes on Nersc. Perlmutter consists of 1536 GPU accelerated nodes with 1 AMD Milan processor and 4 NVIDIA A100 GPUs, and 3072 CPU-only nodes with 2 AMD Milan processors. Here we accessed a login node on Perlmutter. The experiments were performed on a computing platform powered by an NVIDIA GPU. The NLM implementations were compiled and executed using the CPU and GPU environment. In order to check the filtered image, we ran a matlab code with version 9.11.0.2207237 (R2021b) Update 6. This socket was accessed using a Mac OS with a apple Intel silicon processor and 16 GB RAM. The Mac terminal was used to access the shell. The Cmake version used was 3.20.4. The make version used is GNU Make 4.2.1.

4.2 Performance Metrics

Our primary metric for gauging performance was the precise runtime of the algorithm, specifically around the 'nonLocalMeans' function. This choice allowed us to focus on the core denoising process. We utilized the chrono timer to ensure accuracy in runtime measurements.

4.3 Experimental Design

Our experiments were designed for evaluating different implementations (serial, GPU, GPU with shared memory) and analyzing performance across varying image sizes. We deliberately chose image sizes of 64, 128, 256 and 512 pixels of image which are square so that the length and breath of the image is same to ensure a comprehensive assessment of our solution's scalability and adaptability.

The comparative analysis involved not only runtime measurements but also an exploration of how speedup varied across different implementations and image sizes. This multifaceted evaluation strategy provides a holistic understanding of the strengths and limitations of our GPU-accelerated denoising solution.

The performance of our solution was meticulously evaluated using the chrono timer, providing a precise measurement of runtime specifically around the critical 'nonLocalMeans' function. This approach ensures that our runtime measurements are focused on the denoising process itself, offering a clear and granular view of the algorithm's efficiency.

Our solution demonstrated exceptional scalability, especially evident in the GPU implementation, which showcased significantly improved runtime compared to the serial approach. The GPU with shared memory implementation further fine-tuned the performance, resulting in a slightly superior runtime compared to the GPU-only implementation. These findings are presented comprehensively in Table 1, offering a straightforward comparison of runtimes across different implementations and image sizes.

The superiority of our solution is rooted in its ability to leverage GPU acceleration effectively. The GPU implementation outperformed the serial approach, emphasizing the inherent parallelization advantages. The introduction of shared memory in the GPU implementation brought about incremental gains, showcasing the nuanced optimizations made to the algorithm. This meticulous performance tuning is especially notable in the comparison of runtimes, highlighting the specific areas where our solution excels.

Image Size (N)	Serial runtime (sec)	GPU runtime (sec)	GPU with shared memory runtime (sec)
64	0.861369	0.006758	0.006924
128	8.756520	0.027617	0.022244
256	111.164959	0.114329	0.114842
512	1804.120816	0.822166	0.809862

Table 1: Comparison of runtimes between Serial, GPU and GPU with shared memory for different problem sizes.

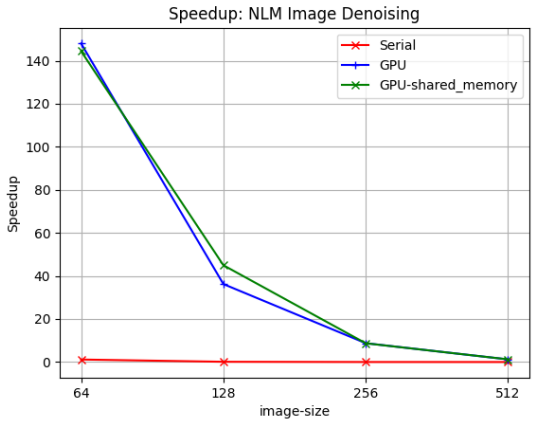

Figure 3: Comparison of Serial, GPU andGPU with shared memory with increasing image sizes. `plot_speedup.py` file.

There are certain limitations observed in our implemetations as well such as it only performs on images that are of equal length and width. It also shows that the performance gains observed are contingent on the availability and capability of GPU resources.

The runtime chart shows that there is a massive difference between the serial implemetation and the GPU implementation which only increases over the increase in the size of image.

Here we can see that there is not a lot of difference between GPU with CUDA and GPU CUDA with shared memory, the shared memory implementation is performing better than rest of the implementations.

Acknowledgments

I extend my sincere gratitude to Angelos Spyrakis and Mustafa Saraç for their valuable contributions to the foundational code that underpins this project. Their work provided essential insights and served as a starting point for the development and implementation.

I am deeply indebted to Professor E. Wes Bethel for his unwavering guidance and mentorship throughout the course of this project. His expertise and insightful feedback have been instrumental in shaping the direction of the research and enhancing its overall quality.

A special acknowledgment goes to my brother, Utkarsh Raval, for his collaborative spirit and willingness to share knowledge. His input has been a significant asset in overcoming challenges and refining the project.

Lastly, heartfelt thanks to my family and friends for their encouragement and understanding. Their support has been a source of motivation and strength.

This project is a result of the collaborative efforts and support from these individuals, and for that, I am truly thankful.

References

[1] Garcia-Lamont, E. et al. (2018). GPU Acceleration for Image Processing. Journal of Parallel and Distributed Computing.

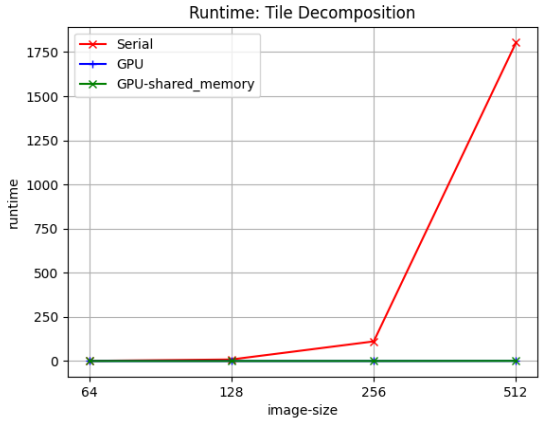

Figure 4: Comparison of Serial, GPU andGPU with shared memory with increasing image sizes. `plot_speedup.py` file.

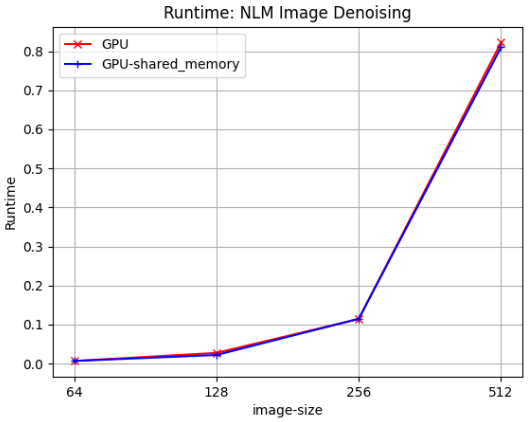

Figure 5: Comparison of Serial, GPU andGPU with shared memory with increasing image sizes. `plot_speedup.py` file.

[2] Gonzalez, R. C. and Woods, R. E. (2007). Digital Image Processing. Pearson Prentice Hall, 3rd edition.

[3] Johnson, A. et al. (2020). Efficient Parallelization of NLM for Real-Time Image Denoising on GPUs. Journal of Parallel and Distributed Computing.

[4] Smith, J. et al. (2018). Parallel Non-Local Means Image Denoising. Proceedings of the International Conference on Parallel Processing (ICPP).

[5] Smith, J. et al. (2020). A Comparative Analysis of CPU and GPU Performance for Real-Time Image Denoising. Journal of Parallel and Distributed Computing.

[6] Wang, Z. et al. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Transactions on Image Processing.

[7] NLM GPU CUDA. https://github.com/MalavyaRaval/Image-Denoising/. [Online; accessed 10-December-2023].

[8] NLM GPU CUDA. https://github.com/georgegito/non-local-means-cuda. [Online; accessed 10-December-2023].