Indian Institute of Information Technology, Raichur

Major Project Proposal (8th Semester)

Parallel Implementation of Image Processing algorithms on GPU platform. (Tentative)

Guide:

Dr. Debasish Mukherjee

Submitted By:

Kushagra Indurkhya

CS19B1017

Introduction

The aim of this project will be to identify some image processing algorithms and implement alternate versions of them which employ parallel processing to achieve increased throughput, faster computations, and optimal resource utilization.

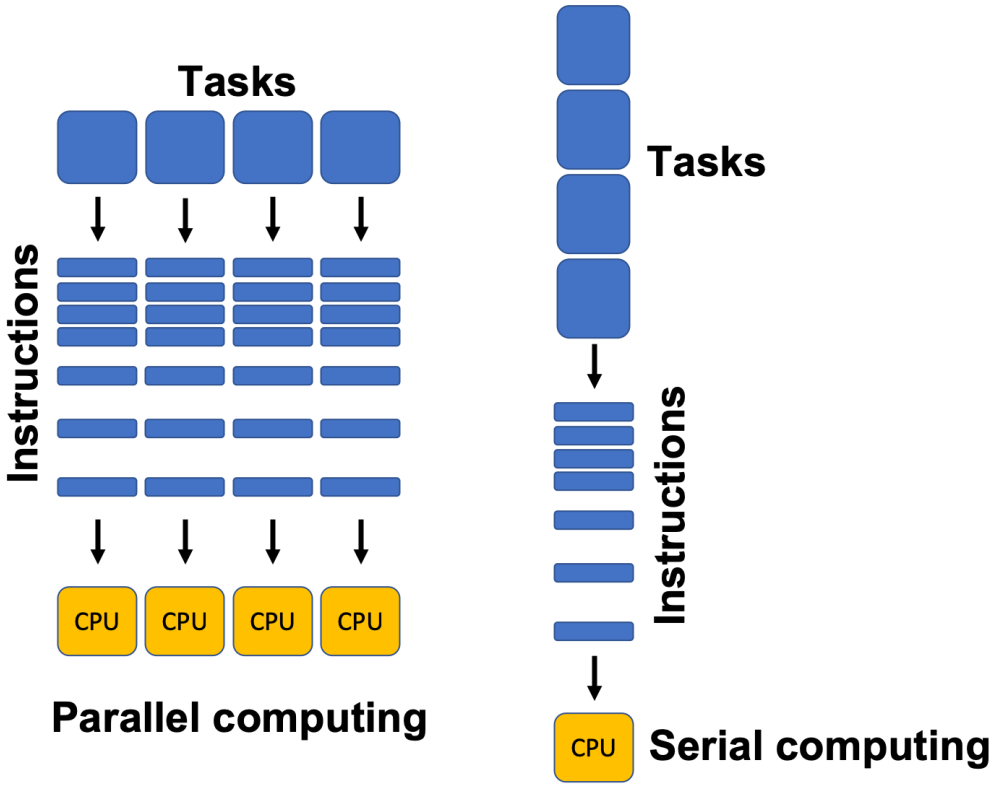
Problem

Many applications need to process a massive amount of image data in a small timeframe, these can be high-performant video editing in which lakhs of frames need to be processed or time-critical computer vision for auto-driving cars, medical imaging applications, etc

Problem with traditional sequential processing on CPU cores is that at a given point of time only one pixel is being processed (Convolution kernel is being applied or arithmetic operations are being performed) so the hardware resources are not being utilized efficiently thus the time taken for the processing of the image to be completed is much more than what it could be by using appropriate algorithmic and hardware modifications.

Proposed Solution

Parallelism of image processing task becomes a key factor for processing a huge raw image data. Parallelization allows scalable and flexible resource management and reduces the time taken in computation.



We can break the problem into tasks for parallel processing in several ways,some of which can be:

- Each pixel is a task
- Each row is a task
- Each Column is a task
- Each Row/Column/Pixel of the Convolution processing step is a task

We have to choose by looking at the tradeoff between overhead of creating a task and the speedup it provides in each of the above mentioned design.

These task can be taken up by GPUs as they were designed for massively parallel processing and many image processing tasks fall into this category. It also often has *thousands of cores instead of just a few dozen like modern CPUs*. The overhead of uploading to the card and downloading to memory is often dwarfed by the speed of processing unless the processing is extremely simple. The GPU cores will often operate on a fragment at a time, which is usually something like a 2x2 pixel area, It's possible to do real-time processing of 4K footage, in many cases, too.

Work Plan

In this section, I have presented a preliminary plan for my project.

Converting Sequential to Parallel

Upon identifying the algorithms that I will try to parallelize, first I would classify them in one of the following paradigms:

- **Pipeline parallelism**: In this sort of processing, long sequences of procedures, or tasks, are parallel
- **Independent Parallelism**: In this sort of parallelism, the tasks don't rely on other tasks. As a result, complete delivery time is considerably reduced.
- **Inter-query and intra-query parallelism**: Transactions are independent. No transaction involves the result of still another transaction to complete. Many CPUs

could be kept active by assigning each job or issue to a separate CPU. This kind of parallelism, employing many split-up, separate queries at once, is called interquery parallelism. To speed up the delivery of an individual large and complex issue, that model decomposes it into smaller problems. It then executes these smaller tasks concurrently by assigning them split-up CPUs.

Then I would try to come up with a parallel implementation of this algorithm, and check my algorithm for the following features of parallel programming:

- **Granularity**: It is defined as the number of basic units and it is classified as Coarse-grained: With few tasks of more intense computing or Fine grain: A enormous number of small parts and less intense computing.
- **Synchronization**: This prevents the overlap of two or more processes.
- **Latency**: This is the time transition of information from request to receipt.
- **Scalability**: It is defined as the ability of an algorithm to maintain its efficiency by increasing the number of processors and the size of the problem in the same proportion.
- **Speedup and efficiency**: These are metrics to assess the quality of parallel implementation.
- **Overheads**: Extra time is needed for the computation.

Implementation

Then I would try to implement this solution on a CPU/GPU:

- For GPU programming, I would try CUDA, a GPU programming language for Nvidia GPU.
- Programming on a GPU will come with its own set of challenges and overheads that will need to be taken into account, for example, Since GPUs do not have direct access to the computer's memory, one should first select the area of sufficient size in the video memory, then copy the data from the random access memory (RAM). Upon completing the calculations, moving the data back to the RAM and cleaning the previously selected video memory area is necessary.

Benchmarking and Comparisons

The matrix by which we can compare the performances can be:

- Time taken by Sequential vs Parallel on CPU (Varying cores or CPU)
- Time taken by Sequential vs Parallel on GPU (Implemented on Nvidia GPU using CUDA)
- Study the core usage and check if our solution is able to achieve maximal CPU/GPU engagement