Parallel Image Processing Pipeline Report

Group Members: Tanishque Soni, Mukta Vedpathak, Member 3

October 21, 2024

Abstract

This report presents the implementation and evaluation of a parallel image processing pipeline using three different approaches: pipes, semaphores with shared memory, and locks. We measured the total time taken by each image in each part of the pipeline and verified that pixel data was received as sent. Additionally, the report discusses the ease and difficulty of implementing and debugging each approach.

1 Introduction

The purpose of this assignment was to implement a parallel image processing pipeline that performs three stages:

- **S1:** Smoothening an image and passing the result to the next stage.
- **S2:** Processing the smoothed image and passing it to the final stage.
- **S3:** Applying the final transformation and outputting the image.

Three parallel approaches were used:

1. Communication via **pipes**.
2. Communication via **shared memory** with synchronization using semaphores.
3. Parallelism through **threads** using locks for synchronization.

The table below shows the total processing time for each image in each of these approaches.

2 Completion Times

The following table summarizes the total time taken for processing each image in each of the three parallel approaches.

2.1 Total Times with Pipes

Image ID	Time with Pipes (s)
1	10.4683
2	41.3575
3	83.3955
4	177.7898
5	277.3225
6	479.8914
7	1261.3352

Table 1: Total Processing Times for Each Image Using Pipes

2.2 Total Times with Semaphores

Image ID	Time with Semaphores (s)
1	11.8887
2	37.2273

Table 2: Total Processing Times for Each Image Using Semaphores

2.3 Total Times with Locks

Image ID	Time with Locks (s)
1	35.9014
2	164.847

Table 3: Total Processing Times for Each Image Using Locks

3 Verification Method

To ensure the pixel data was received as sent and in the correct order, we devised the following verification method:

1. A **checksum** was calculated for each image's pixel data before being passed to the next stage.
2. The same checksum was recalculated after each stage to verify that the data integrity and order were maintained.
3. Any mismatch in checksum values was logged as an error, ensuring that the pixels were received in the same order they were sent.

This method proved effective in detecting and preventing data corruption or reordering during the parallel execution.

4 Discussion on Implementation and Debugging

This section discusses the relative ease and difficulty of implementing and debugging each of the approaches:

4.1 Pipes

Implementing pipes was straightforward, and they worked well for transferring pixel data between processes. However, debugging involved monitoring pipe using the print statements to check if we are able to achieve parallelism or not.

4.2 Semaphores with Shared Memory

This approach required careful synchronization using semaphores to manage shared memory access between processes. While int this method the data exchange was faster, debugging was challenging due to potential race conditions, the need for proper semaphore initialization and cleanup, check if the image is read properly from the shared memory and is written to the shared memory, only the critical section is protected such that no two processes access the critical section at the same time.

4.3 Locks with Threads

Implementing locks in a multithreaded environment allowed for efficient in-process communication without the overhead of creating separate processes. However, debugging required tools to detect the data race condition, proper initialization of the threads for the particular processes(S1, S2, S3), only the critical section is protected and no two threads access the critical section at the same time.

4.4 Summary of Implementation Difficulty

- **Pipes:** Easy to implement but required careful buffer management and initialization.
- **Semaphores:** Provided efficient communication but required more effort in synchronization and checking critical conditions like datarace, producer-consumer problem etc.
- **Locks:** Debugging thread safety was more complex and checking the conditions for datarace and producer-consumer problem was the difficult part.

5 Conclusion

This assignment provided insight into the use of parallel processing techniques for image transformation. Each method had its strengths and weaknesses, and the choice of method depends on the specific requirements of the application, such as ease of implementation or speed of execution. The checksum verification method proved effective in maintaining data integrity across all implementations.