PaSSPI: Parallel Semantic Segmentation with Pipeline and Image Slicing

Mao-Siang Chen[*]
siang.cs09@nycu.edu.tw
National Yang-Ming Chiao-Tung University
Hsinchu, Taiwan

Min-Jung Li[†]
a0988282303@gmail.com
National Yang-Ming Chiao-Tung University
Hsinchu, Taiwan

Chu-Chun Chi[‡]
chuchun1231@gmail.com
National Yang-Ming Chiao-Tung University
Hsinchu, Taiwan

1 INTRODUCTION

In recent years, self-driving cars have surged in popularity. However, the ongoing discussions surrounding the security of these systems have raised crucial concerns. In the quest to bolster the safety of self-driving technology, two key elements stand out: ensuring high accuracy and minimizing response time when encountering diverse road scenarios. To address these challenges, one approach we are considering is to focus on a critical component of the self-driving system — semantic segmentation.

2 STATEMENT OF THE PROBLEM

2.1 What is Semantic Segmentation

Semantic segmentation is a basic computer vision task, which aims to label each pixel into a category. The result is a pixel-wise segmentation map of an image, with each pixel labeled into a class or an object. Besides, semantic segmentation models play a critical role in self-driving systems by enabling them to identify regions corresponding to roads, vehicles, pedestrians, landscapes, etc.

2.2 Reasons Of Choosing Semantic Segmentation

The first reason of choosing semantic segmentation is the model's inherent characteristics. Semantic segmentation aims to assign a class label to each pixel in an image, regardless of object boundaries. It doesn't require identifying individual objects or their precise boundaries like instance segmentation does. As a consequence, it might be less affected by challenges related to image slicing. The other reason is that lots of pre-trained semantic segmentation models are based on `torch.nn`, fully supported by packages for pipeline parallelism, which is a preferable choice.

2.3 Problem Statement

In this project, we will explore three distinct methods to optimize the workflow of the semantic segmentation model, which includes utilizing CUDA acceleration, implementing model pipelining, and employing image slicing. Given the paramount importance of achieving a high level of accuracy in this task, our approach will prioritize model acceleration while carefully considering any potential trade-offs in the metrics. To determine the results quality, we will inference and observe the metrics like IoU or Dice Score on several semantic segmentation datasets, for instance, the Cityscape Dataset[4], SYNTHIA Dataset[5], ADE20K[6], etc.

[*]Both authors contributed equally to this research.
[†]Both authors contributed equally to this research.
[‡]Both authors contributed equally to this research.

3 RELATED WORK

3.1 Semantic segmentation

Semantic segmentation is a computer vision task that assigns a class label to every pixel of an image, thus creating detailed maps of different objects and their boundaries within the input image. There have been a lot of projects conducted regarding semantic segmentation tasks. In their research [1], a Flow Alignment Module (FAM) is introduced to effectively and efficiently learn the Semantic Flow between feature maps from adjacent levels. This module facilitates the broadcasting of high-level features to high-resolution features. Finally, achieve 80.4% mIoU on Cityscapes with a frame rate of 26 FPS.

3.2 SegFormer

In the realm of semantic segmentation, SegFormer [2] stands out as an innovative approach that overcomes the limitations of handcrafted and computationally intensive components. Its hierarchical Transformer encoder, featuring Mix Transformer encoders and an efficient self-attention mechanism, captures both local and nonlocal attentions while producing multi-level features. SegFormer's architecture represents a significant advancement in semantic segmentation by demonstrating the effectiveness of a streamlined approach, offering a promising direction for the field. In our project, we leverage the efficiency and robustness of SegFormer and complement its power with parallelization techniques to expedite the process, making it an ideal solution for accelerating segmentation tasks.

3.3 PiPPy

PiPPy[3], short for Pipeline Parallelism in PyTorch, is an open-source project offering a comprehensive toolkit designed to automate parallelism and scaling for PyTorch models. In our project, we are committed to leveraging PiPPy to implement pipeline parallelism due to the automated model parallelism feature it provides. It empowers us to streamline the implementation of parallelism within our models without requiring extensive modifications.

4 PROPOSED APPROACHES

In this section, we will progressively incorporate paralleling at different dimensions into our model in three stages.

4.1 Stage 1. Unparalleled SegFormer

In the initial stage, we will establish a baseline using the unparalleled SegFormer for semantic segmentation. Here, data preprocessing, model inference, and result post-processing will be conducted sequentially, emulating a traditional workflow. Our aim is to set the performance metrics, evaluate the processing time, and assess the quality of segmentation results. This stage provides a benchmark against which we can compare the subsequent paralleling efforts in Stages 2 and 3. Figure 1 shows the architecture of the SegFormer model.

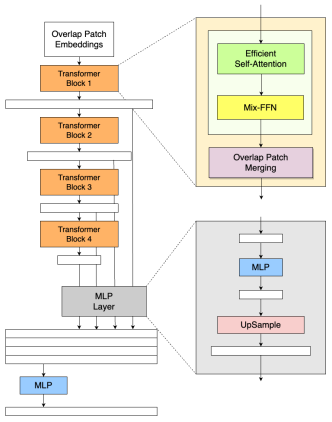

Figure 1: the architecture of SegFormer

4.2 Stage 2. Multi-Threaded Image Slicing

In this phase, the images will be partitioned into several sub-images, and the evaluation step will be paralleled using a multi-threading technique. We aim to conduct performance evaluations on both CPU and CUDA platforms. Finally, evaluate the speed up enhancements and output quality in each. While we expect the image slicing and parallel processing technique to improve the overall speed and reduce memory demands, dividing images into segments may result in information loss at the borders of these sub-images, potentially affecting the overall quality of the segmentation. To address this concern, we plan to implement overlapping segments to maintain context across neighboring sub-images. Finally, We will conduct a comprehensive review of the results obtained from various partitioning strategies to determine the optimal number of partitions for image slicing. Figure 2 demonstrates the framework of partitioning the input image into four sub-images.

4.3 Stage 3. Model Pipelining with PiPPy

Having identified the most efficient method from Stage 2, we will advance to Stage 3, where our focus is on model pipelining. We will construct a model pipeline using the Pippy package, with the selected method from Stage 2 as our foundation. This pipeline will be designed to perform semantic segmentation on both CPU and GPU (CUDA) to harness the benefits of parallel processing and GPU acceleration. Our primary goal is to evaluate the performance, efficiency, and quality of results of this model pipelining approach on both CPU and CUDA, directly comparing the two setups. In addition, we will also make a comparative analysis with the results obtained in Stage 1 to gain a comprehensive understanding of the

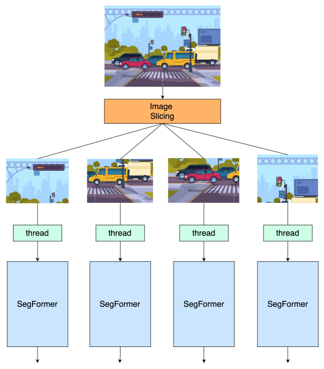

Figure 2: Parallel Image Slicing with Multi-Threading

improvements achieved through parallelization and model pipelining. This stage will provide insights into the optimal configuration for accelerating semantic segmentation tasks using SegFormer.

5 LANGUAGE SELECTION

In our project, we've carefully selected programming languages to optimize our solution. Python is our choice for core modeling tasks due to its versatility and robust machine learning and computer vision libraries. It will serve as the foundation for implementing and fine-tuning the MiT model. In contrast, for paralleling efforts, we've chosen C++, renowned for its performance in critical scenarios, making it ideal for implementing paralleling strategies and achieving high-speed execution.

6 STATEMENT OF EXPECTED RESULTS

6.1 Baseline Establishment with SegFormer

In this stage, we aim to quantify crucial metrics such as IoU, accuracy, precision, recall, and F1 score. This evaluation result will serve as the baseline against the improvements in processing speed and quality in the subsequent stages.

6.2 Multi-Threaded Image Slicing

We anticipate observing enhanced performance, particularly in terms of processing speed, by utilizing parallel processing techniques. For the CPU, we expect a boost in processing speed due to multi-threading. CUDA, on the other hand, is expected to showcase substantial acceleration due to its ability to leverage the parallel processing capabilities of the GPU.
While acknowledging the possible trade-offs on metrics due to slicing, the overlapping strategy is expected to positively influence the correctness and quality of the output. We expect to achieve a balance between enhanced processing speed and the maintenance of high-quality, accurate segmentation.

6.3 Pipelining with PiPPy

In this final stage, we expect the optimized workflow to achieve a better performance as well, and do the comparative analysis accordingly. Pipelining the SegFormer model aims to enhance the overall performance and efficiency in segmentation tasks, while leveraging CUDA for acceleration is also expected to have a significant reduction in processing time.

7 TIMETABLE

Table 1 presents a detailed timetable outlining the various tasks and their respective time periods associated with the project. This schedule serves as a reference for the key milestones and activities conducted throughout the project's duration.

Table 1: Project Timetable

Tasks	Time Period
Set up the development environment	2023/11/01 - 2023/11/02
Collect and prepare the dataset	2023/11/03 - 2023/11/04
Implement the unparalleled SegFormer	2023/11/05 - 2023/11/11
Test and validate the model	2023/11/12 - 2023/11/14
Implement image slicing and multi-threading	2023/11/15 - 2023/11/21
Compare the performance with Stage 1	2023/11/22 - 2023/11/24
Implement model pipelining with pippy	2023/11/25 - 2023/11/30
Compare performance between CPU and CUDA	2023/12/01 - 2023/12/03
Analyze the results from all stages	2023/12/04 - 2023/12/08
Document the code and provide instructions	2023/12/09 - 2023/12/10

REFERENCES

1. Xiangtai Li, Ansheng You, Zhen Zhu, Houlong Zhao, Maoke Yang, Kuiyuan Yang, Yunhai Tong: Semantic Flow for Fast and Accurate Scene Parsing. ECCV (2020)
2. Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, Ping Luo1SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers
3. James Reed, Pavel Belevich, Ke Wen: PiPPy: Pipeline Parallelism for PyTorch. https://github.com/pytorch/PiPPy
4. Cityscape Dataset: https://www.cityscapes-dataset.com/dataset-overview/
5. SYNTHIA Dataset: https://synthia-dataset.net/
6. ADE20K Dataset: https://groups.csail.mit.edu/vision/datasets/ADE20K/