



**HUMBER**

School of Applied Technology

## MENG 3540 – Parallel Programming

**Final Report:** Jetson Utilization

**Students:** Jabari Lira Leon & Gerald Jacob

**Student #:** N01493372 & N00741068

**Date:** April 21, 2024

**Professor:** Raji Subramanian

## Introduction

In the rapidly evolving landscape of artificial intelligence (AI) and machine learning (ML), deploying robust frameworks on specialized platforms like Jetson Nano, Jetson AGX Xavier, and Jetson TX2 is paramount for developers and researchers alike. To facilitate this, our guide offers comprehensive instructions for installing two of the most prominent frameworks, TensorFlow and PyTorch, tailored specifically for the Jetson Platform.

As AI applications permeate various industries, the demand for efficient deployment solutions on edge devices intensifies. Jetson modules stand at the forefront of this evolution, offering high-performance computing power within a compact form factor. Harnessing the capabilities of TensorFlow and PyTorch on these platforms unlocks a myriad of possibilities, ranging from real-time object detection to natural language processing, and beyond.

Recognizing the versatility and portability offered by containerization, the project introduces specialized containers tailored for Jetson Orin Nano and JetPack 5.1 environments. These containers, including the Machine Learning Container and TensorFlow Container, empower developers to encapsulate their applications and dependencies, ensuring seamless deployment across diverse computing environments.

## Theory

NVIDIA's embedded systems portfolio represents a cutting-edge convergence of AI, robotics, and autonomous technology, catering to a diverse array of industries and applications. At the heart of this ecosystem lies the Jetson Platform, comprising a family of powerful, energy-efficient modules designed to fuel innovation in edge computing.

**Jetson Platform: Powering Intelligent Edge Devices** The Jetson Platform, spearheaded by flagship products like Jetson Nano, Jetson Xavier NX, and Jetson AGX Xavier, offers unparalleled performance and versatility for edge AI applications. These modules boast high-performance GPUs, specialized AI accelerators, and robust software support, enabling developers to deploy complex AI models with ease.

**Deep Learning at the Edge** With the Jetson Platform, developers can harness the full potential of deep learning algorithms directly on edge devices. From image recognition and natural language processing to robotics and autonomous navigation, the Jetson Platform empowers a new generation of intelligent edge devices capable of real-time decision-making.

**Versatile Applications** NVIDIA's embedded systems find applications across a wide spectrum of industries, including manufacturing, healthcare, transportation, and smart cities. Whether it's enhancing production efficiency, enabling medical diagnostics, or revolutionizing autonomous vehicles, Jetson-powered solutions are driving innovation and transforming industries.

**Developer Ecosystem and Tools** NVIDIA fosters a vibrant developer ecosystem with comprehensive tools, libraries, and frameworks tailored for embedded AI development. From

TensorFlow and PyTorch support to specialized SDKs like JetPack, developers have access to a rich toolkit for prototyping, training, and deploying AI applications on Jetson devices.

**Containerization and Deployment** Recognizing the importance of portability and scalability, NVIDIA offers containerization solutions optimized for Jetson devices. By encapsulating applications and dependencies within containers, developers can streamline deployment, facilitate cross-platform compatibility, and accelerate time-to-market for edge AI solutions.

## Implementation

After a successful installation of the software, this screen will show up. The team experienced difficulties in installing the software. The team will just have a theoretical approach in preparing the report.

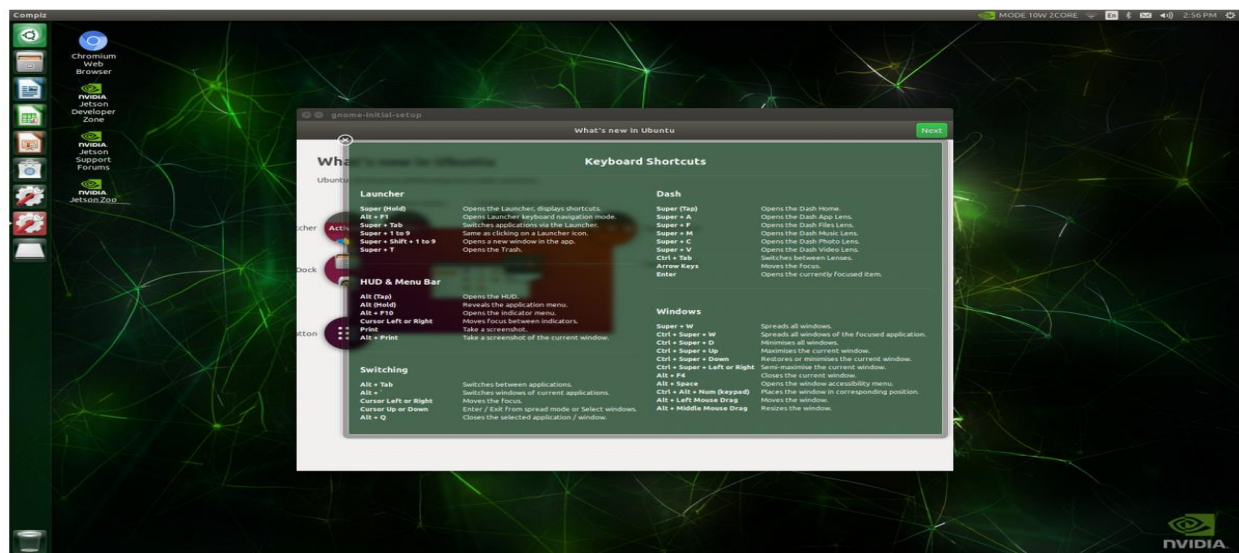


Figure 1: Successful Installation

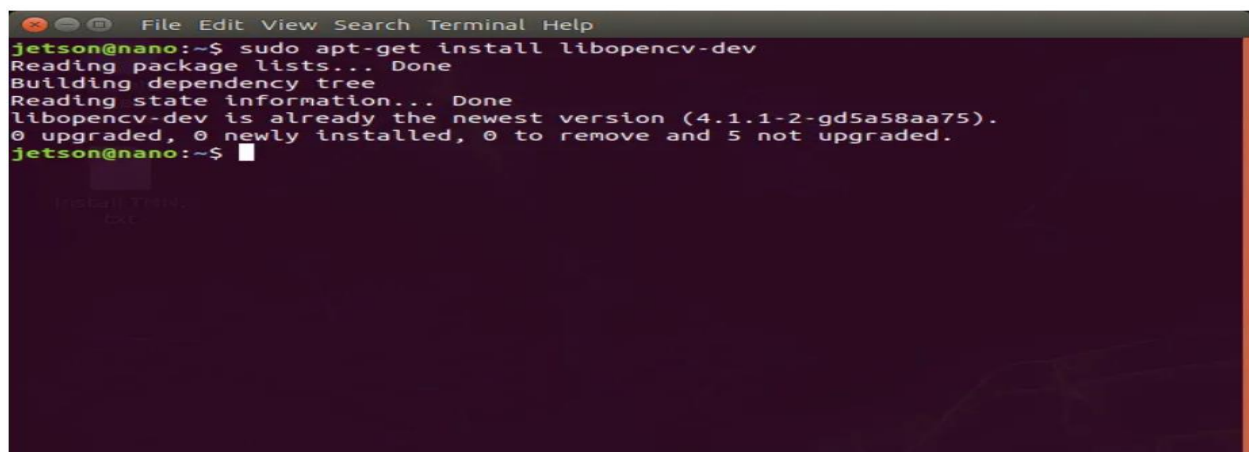
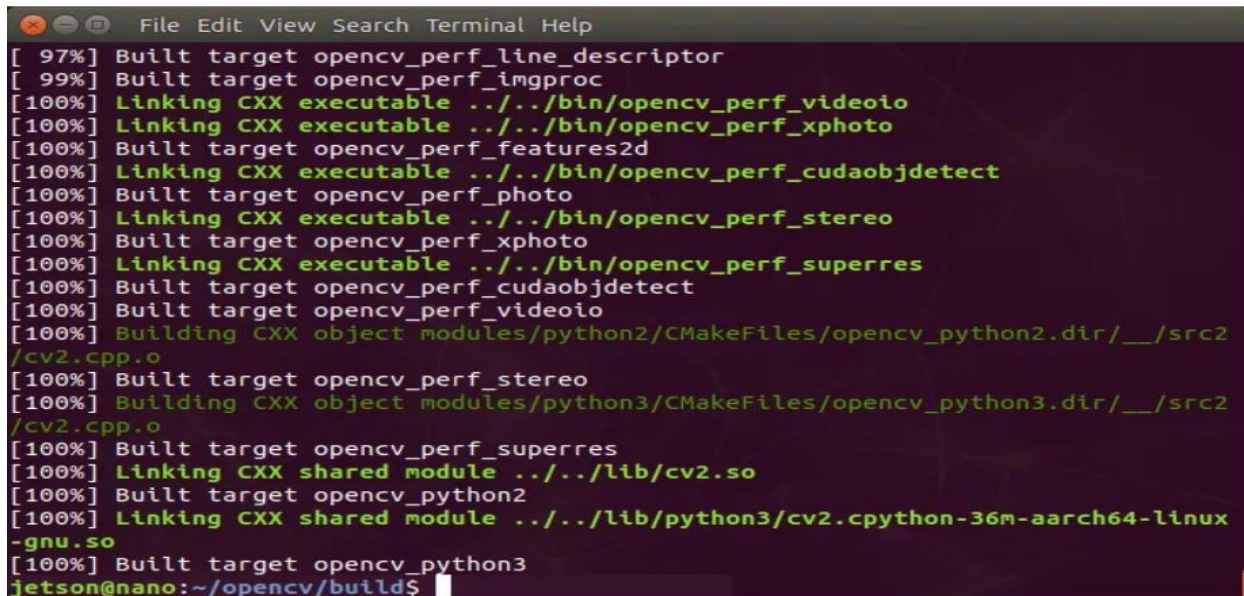


Figure 2: Installing the Software



```
[ 97%] Built target opencv_perf_line_descriptor
[ 99%] Built target opencv_perf_imgproc
[100%] Linking CXX executable ../../bin/opencv_perf_videoio
[100%] Linking CXX executable ../../bin/opencv_perf_xphoto
[100%] Built target opencv_perf_features2d
[100%] Linking CXX executable ../../bin/opencv_perf_cudaobjdetect
[100%] Built target opencv_perf_photo
[100%] Linking CXX executable ../../bin/opencv_perf_stereo
[100%] Built target opencv_perf_xphoto
[100%] Linking CXX executable ../../bin/opencv_perf_superres
[100%] Built target opencv_perf_cudaobjdetect
[100%] Built target opencv_perf_videoio
[100%] Building CXX object modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o
[100%] Built target opencv_perf_stereo
[100%] Building CXX object modules/python3/CMakeFiles/opencv_python3.dir/__/src2/cv2.cpp.o
[100%] Built target opencv_perf_superres
[100%] Linking CXX shared module ../../lib/cv2.so
[100%] Built target opencv_python2
[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-36m-aarch64-linux-gnu.so
[100%] Built target opencv_python3
jetson@nano:~/opencv/build$
```

Figure 3: Installing additional software

## Discussion of Problem Statement

As the demand for efficient object detection in photos continues to surge across various domains such as surveillance, autonomous vehicles, and augmented reality, the need for accelerated processing methodologies becomes increasingly apparent. Traditional sequential algorithms for object detection face challenges in meeting real-time processing requirements and scaling to handle large datasets.

### Challenges:

1. **Computational Intensity:** Object detection algorithms, particularly deep learning-based approaches like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), are computationally intensive, requiring significant processing power for inference.
2. **Real-Time Requirements:** Many applications necessitate real-time object detection capabilities to enable timely decision-making. However, sequential algorithms may struggle to meet the stringent latency requirements of these applications.
3. **Scalability:** With the proliferation of high-resolution images and massive datasets, the scalability of object detection algorithms becomes crucial. Sequential implementations may encounter bottlenecks in processing large volumes of data efficiently.

**Proposed Solution:** To address these challenges, the proposed solution involves leveraging parallel programming paradigms to enhance the performance of object detection algorithms. By harnessing the computational power of multi-core CPUs or specialized hardware accelerators such as GPUs and TPUs, parallel programming techniques can significantly accelerate the inference process and improve scalability.

## Objectives:

1. **Optimized Parallelization:** Develop parallel programming implementations tailored to object detection algorithms, ensuring efficient utilization of available computational resources while maintaining accuracy.
2. **Real-Time Performance:** Achieve real-time or near-real-time object detection capabilities by leveraging parallel processing to reduce inference latency and meet application requirements.
3. **Scalability and Efficiency:** Design parallel algorithms capable of scaling seamlessly across varying hardware configurations and dataset sizes, enabling efficient processing of large-scale image datasets.

**Impact:** Effective parallel programming for object detection in photos has the potential to revolutionize numerous industries reliant on image analysis and computer vision. From enhancing surveillance systems' capabilities to enabling autonomous vehicles' perception systems, the accelerated processing enabled by parallel programming can unlock new possibilities and drive innovation across diverse domains.

## Theoretical Implementation

### 1. Algorithm Selection and Optimization:

- **Algorithm Choice:** Begin by selecting suitable object detection algorithms such as YOLO or SSD, known for their efficiency and accuracy.
- **Algorithm Optimization:** Optimize selected algorithms for parallel execution, identifying parallelizable tasks and minimizing dependencies between them.

### 2. Parallelization Strategies:

- **Data Parallelism:** Divide the dataset into smaller batches and distribute them across available processing units (e.g., CPU cores, GPU threads).
- **Model Parallelism:** Partition the neural network model across multiple processing units, allowing different parts of the model to run concurrently.
- **Pipeline Parallelism:** Decompose the object detection pipeline into sequential stages, each of which can be executed in parallel.

### 3. Implementation Steps:

- **Parallel Framework Selection:** Choose appropriate parallel programming frameworks such as OpenMP, CUDA, or TensorFlow's distributed computing capabilities.
- **Code Refactoring:** Modify the object detection algorithm implementation to incorporate parallelization constructs provided by the selected framework.

- **Task Scheduling:** Implement efficient task scheduling mechanisms to balance workload distribution and minimize idle time across processing units.
- **Memory Management:** Optimize memory usage to minimize data transfer overhead between processing units, leveraging shared memory or asynchronous data loading techniques.
- **Error Handling:** Implement robust error handling mechanisms to detect and recover from parallel execution failures, ensuring the reliability of the system.

#### 4. Performance Evaluation and Optimization:

- **Benchmarking:** Conduct comprehensive performance benchmarking tests to evaluate the speedup achieved through parallel programming compared to sequential execution.
- **Profiling:** Use profiling tools to identify performance bottlenecks and areas for further optimization, such as optimizing memory access patterns or reducing synchronization overhead.
- **Fine-Tuning:** Iteratively fine-tune parallelization parameters and optimization strategies based on performance metrics and application requirements.

#### 5. Real-Time Integration and Deployment:

- **Integration with Applications:** Integrate parallelized object detection algorithms into real-world applications such as surveillance systems, autonomous vehicles, or industrial automation.
- **Deployment Considerations:** Consider deployment constraints such as hardware compatibility, power consumption, and resource availability when deploying parallelized object detection solutions.
- **Continuous Monitoring:** Implement monitoring mechanisms to track system performance and ensure optimal operation over time, allowing for timely adjustments and optimizations as needed.

#### 6. Scalability and Generalization:

- **Scalability Testing:** Validate the scalability of parallelized object detection algorithms across different hardware configurations and dataset sizes.
- **Generalization:** Ensure the generalization of parallelization techniques to other object detection algorithms and computer vision tasks beyond the initial implementation.

#### 7. Documentation and Knowledge Sharing:

- **Documentation:** Document the implementation process, parallelization strategies, and performance optimization techniques for future reference and knowledge sharing.

- **Knowledge Sharing:** Share insights and lessons learned from the implementation process with the broader community through articles, presentations, or open-source contributions.

### Theoretical Skeleton Code

```
#include <iostream>
#include <vector>
#include <omp.h>

// Define the structure for an Image
struct Image {
    // Placeholder for image data
    // Add fields as needed for image representation
};

// Placeholder function for object detection on a single image
void detectObjects(const Image& image) {
    // Placeholder for object detection algorithm
    // Implement object detection algorithm here
}

// Function to perform parallel object detection on a dataset of images
void parallelObjectDetection(const std::vector<Image>& dataset) {
    // Get the number of available CPU cores on the Jetson device
    int numThreads = omp_get_max_threads();

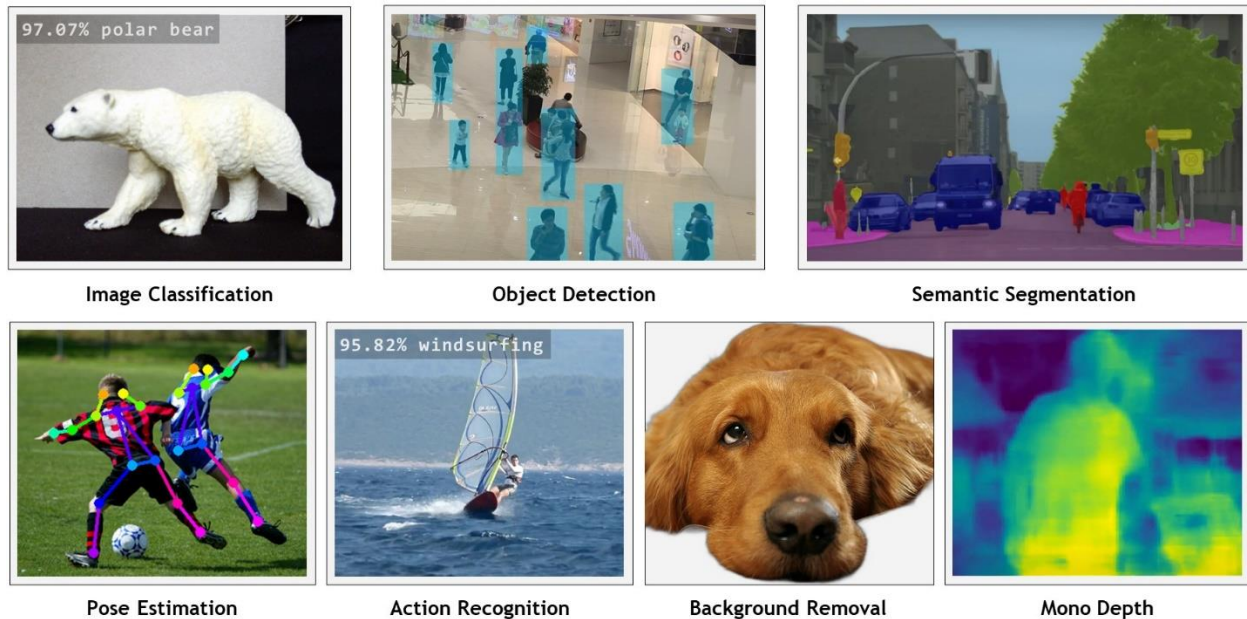
    // Parallelize object detection across multiple images using OpenMP
    #pragma omp parallel for num_threads(numThreads)
    for (int i = 0; i < dataset.size(); ++i) {
        // Call object detection function on each image
        detectObjects(dataset[i]);
        std::cout << "Object detection completed for image " << i <<
std::endl;
    }
}

int main() {
    // Placeholder for loading the dataset of images
    std::vector<Image> dataset; // Assume dataset is populated with images

    // Perform parallel object detection on the dataset
    parallelObjectDetection(dataset);

    return 0;
}
```

## Potential Results



**Figure 4:** Potential Results

The results show how an image can be isolated based on the structure of the program. In the photo above, various potential results can be identified. From image classification, object detection, semantic segmentation, pose estimation, action recognition, background removal, and mono depth.

The application of Jetson to perform parallel programming provides a strong solution on solving real-world problems. The Jetson provides. The team wished that the Jetson worked so that we can implement various solutions and experiment on the system.



## Conclusion

The project has explored the critical role of deploying robust frameworks such as TensorFlow and PyTorch on specialized platforms like Jetson Nano, Jetson AGX Xavier, and Jetson TX2 in the ever-evolving landscape of artificial intelligence (AI) and machine learning (ML). The Jetson Platform stands as a cornerstone, offering unparalleled performance and versatility for edge AI applications.

The versatility and portability offered by containerization have been highlighted, with specialized containers tailored for Jetson Orin Nano and JetPack 5.1 environments, empowering developers to encapsulate their applications and dependencies for seamless deployment across diverse computing environments.

NVIDIA's embedded systems portfolio, epitomized by the Jetson Platform, represents a cutting-edge convergence of AI, robotics, and autonomous technology, catering to a diverse array of industries and applications. From enhancing surveillance systems to revolutionizing autonomous vehicles, Jetson-powered solutions are driving innovation and transforming industries.

The team's theoretical exploration delved into the challenges faced in object detection in photos, including computational intensity, real-time requirements, and scalability issues. To address these challenges, the team proposed leveraging parallel programming paradigms to enhance object detection algorithms, aiming for optimized parallelization, real-time performance, and scalability.

Theoretical implementation strategies were outlined, encompassing algorithm selection and optimization, parallelization strategies, implementation steps, performance evaluation, real-time integration, scalability, and documentation. The potential results demonstrate the diverse applications of parallel programming in image processing, from classification to depth estimation.

Although the theoretical approach laid a solid foundation for implementing parallel programming solutions, the team encountered difficulties in installing the necessary software for practical implementation. Nonetheless, the potential of Jetson in solving real-world problems through parallel programming remains promising, offering robust solutions for diverse applications. With continued exploration and experimentation, Jetson's capabilities in parallel programming can unlock new possibilities and drive innovation across various domains.

## References

Dusty-Nv. (n.d.). *Dusty-NV/Jetson-inference: Hello AI World Guide to deploying deep-learning inference networks and deep vision primitives with TENSORRT and Nvidia Jetson*. GitHub. <https://github.com/dusty-nv/jetson-inference>

*Jetson Developer Tools*. NVIDIA Developer. (n.d.-a). <https://developer.nvidia.com/embedded/develop/tools>

*Jetson Orin Nano Developer Kit Getting Started Guide*. NVIDIA Developer. (n.d.-b). <https://developer.nvidia.com/embedded/learn/get-started-jetson-orin-nano-devkit#next>

*Nvidia embedded systems for next-Gen Autonomous Machines*. NVIDIA. (n.d.). <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/>

Politeik, R. (2023, November 17). *Install opencv on Jetson Orin Nano - Q-engineering*. Q. <https://qengineering.eu/install-opencv-on-orin-nano.html>