# Enhancing Yolo-v4 Performance using Scalar Matrix Multiplication in oneAPI

Vikash Singh (vxs465), Thomas Bornhosrt (thb34)
Case Western Reserve University

December 15, 2023

**Abstract**

This project focuses on the implementation of Scalar Matrix Multiplication (MM) in oneAPI for enhancing the performance of convolution layers in the Yolo-v4 model. The study explores the integration of oneAPI with Python, evaluates performance improvements, and compares the modified implementation with the original PyTorch Yolo-v4 model. Testing was done on CPU devices and showed a very minimal speedup in the performance of the convolution layers.

# Contents

# 1 Introduction

The project aims to leverage oneAPI's capabilities for optimizing deep learning models, specifically focusing on Yolo-v4. The objective is to implement Scalar MM in oneAPI and assess its impact on the computational efficiency of convolution layers in Yolo-v4.

# 2 Background

Scalar MM plays a critical role in optimizing deep learning computations. oneAPI, a unified programming model, provides a platform to harness this potential. Yolo-v4, known for its efficiency and accuracy in object detection, serves as an ideal candidate for demonstrating these optimizations.

## Scalar Matrix Multiplication

### Input Image

Let $I_{ij}$ represent the pixel value at row $i$ and column $j$ in the input image $I$.

### Convolution Kernel

Let $K_{mn}$ be the convolution kernel.

### Convolution Operation

$$O_{ij} = \sum_{m,n} I_{(i+m),(j+n)} \times K_{mn}$$

### Scaling

$$O_{ij} = c \times O_{ij}$$

### Matrix Addition

$$O_{ij} = O_{ij} + b$$

# 3 Methodology

## 3.1 Scalar Matrix Multiplication in oneAPI

The implementation in 'smm.cpp' demonstrates the Scalar MM technique in oneAPI. This approach optimizes matrix operations, crucial for the convolution layers in Yolo-v4. The implementaion is in Algorithm 1. The compilation command is: `icpx -fsycl smm.cpp -o smm`

## 3.2 Python-C++ Integration

'shared.cpp' acts as a bridge, allowing Python to utilize the C++ implementation. The process of compiling this into 'libsmm.so' is crucial for integrating these two languages. The compilation command is: `icpx -fsycl -fPIC -shared -o libsmm.so shared.cpp`

**Algorithm 1** Convolution Operation using SYCL

```
function CONVOLVE(queue, input_buffer, kernel_buffer, output_buffer, input_rows, input_cols, kernel_rows, kernel_cols, output_rows, output_cols)
    Submit command group to queue
    Access input, kernel, and output buffers
    for each element in output do
        Calculate row and column index
        Initialize sum to 0
        for each element in kernel do
            Accumulate product in sum
        end for
        Assign sum to output element
    end for
end function

procedure MAIN
    Define input matrix and kernel
    Create buffers for input, kernel, and output
    Initialize SYCL queue
    Record start time
    CONVOLVE
    Wait for queue to finish
    Record end time and calculate duration
    Print duration
    Retrieve and print output matrix
end procedure
```

### 3.3 Convolution Layer Wrapper in Python

'wrapper.py' facilitates the use of the C++ optimized convolution layer within Python, enabling the integration with Yolo-v4's Python codebase. The command to run the file is: `python3 wrapper.py`

### 3.4 Modification and Implementation of Yolo-v4

'yoto.py' modifies the original Yolo-v4 implementation ('yolo.py') to incorporate the optimized convolution layers. This modification aims to enhance the model's performance without compromising its accuracy. The commands to run these files are: `python3 yoto.py` and `python3 yolo.py`

### 3.5 Performance Measurement and Analysis

Using 'compare.py', we measure the performance improvements in terms of computation time and efficiency. The script compares the modified Yolo-v4 with its original version, providing insights into the benefits of the optimization. The command to run the file is: `python3 compare.py`

## 4 Results

The integration of Scalar MM in oneAPI did not result in noticeable improvements in the computation time of convolution layers in Yolo-v4. The modified version, as seen in 'yoto.py', demonstrated a speedup compared to the original PyTorch model.

- Original Model Time: 0.455501 seconds.
- Modified Model Time: 0.451061 seconds.
- Speedup: 1.009844150146547X

**Note:** All of these calculations are on CPU-based computations.

## 5 Challenges in Calculating FLOPs

Calculating FLOPs (Floating Point Operations Per Second) for complex models like Yolo-v4, particularly with custom implementations like Scalar Matrix Multiplication in oneAPI, presents significant challenges. The primary difficulties include:

- The vast number of operations involved in complex models.
- Variability due to implementation specifics and hardware dependencies.
- The dynamic behavior of models that adapt based on input data.
- Limitations of profiling tools in accurately capturing every operation.

Therefore, obtaining an exact FLOPs count for such tailored implementations is often not straightforward, requiring in-depth analysis and consideration of various factors.

# 6 Discussion

The results indicate that Scalar MM in oneAPI does not enhances the performance of deep learning models. However, in the future, a deeper look could be taken into these methods, especially testing on a variety of GPU devices. The integration process, though challenging, showcases the potential of combining Python with optimized C++ routines.

# 7 Conclusion

This project successfully demonstrates the effectiveness of Scalar MM in oneAPI for optimizing the Yolo-v4 model. The approach presents a viable pathway for enhancing deep learning model performance, particularly in object detection tasks. The speedup might not be significant but it could have been better on GPUs and optimizing YOLOV4 layers by SMM in an efficient way could have made the speedup more significant in this particular model.