Multi-FPGA Accelerator for Computer Vision

Qianshu Li
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
qli456@gatech.edu

Kuo Chih Su
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
ksu48@gatech.edu

Zhenkun Fan
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, USA
zfan87@gatech.edu

Zihan Zheng
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, USA
zzheng345@gatech.edu

***Abstract*—This project aims to develop a communication interface between the programmable logic sides of the ZCU102 and PYNQ-Z2 boards. The project is divided into four stages, each with a specific implementation. Stage one involves exploring board specifications, instantiating Aurora 64B/66B IP for communication, and conducting on-board tests. Stage two focuses on dataflow across FPGAs, bottleneck analysis, and performance estimation. Stage three involves the deployment of a complete neural network on a multi-FPGA system, including the implementation of various methodologies like quantization, pruning, and frame dropping. Finally, stage four involves designing an NoC module for handling complicated data transfer and sharing requests, exploring task-level partition, network topology design, model profiling, and reconfiguration in flight. The project aims to create a robust and efficient multi-FPGA system for neural network implementation.**

***Index Terms*—Multi-FPGA, HLS, Network-on-Chip**

I. Introduction

The rapid growth of artificial intelligence (AI) has led to a significant increase in demand for high-performance computing systems for neural network inference. Traditional computing systems such as CPUs and GPUs have been the primary choices for these applications, but their limited parallelism and high power consumption can lead to performance bottlenecks and reduced efficiency.

To address these challenges, the proposed project aims to develop an efficient multi-FPGA system for neural network inference. FPGAs have emerged as promising candidates for accelerating neural network inference due to their high parallelism, low power consumption, and flexibility. The proposed system will utilize a distributed computation approach, allowing for potentially higher throughput and efficient resource allocation.

The project will explore advanced communication techniques such as Aurora IP and SFP+ interfaces to enable high-speed data transfer between FPGAs. Additionally, the system will leverage techniques such as quantization and pruning to reduce the memory and computation requirements of neural networks and further improve performance.

The proposed multi-FPGA system will also investigate task-level partitioning and network topology design for efficient resource allocation and scheduling. The ultimate goal of this project is to develop an efficient and flexible system for neural network inference that can achieve high performance with low power consumption.

II. Problem Description

This project aims to develop a robust and efficient multi-FPGA system for neural network implementation. Multi-FPGA systems are becoming increasingly popular for accelerating neural network training and inference due to their ability to provide high parallelism and large memory bandwidth. However, implementing neural networks on multi-FPGA systems presents several challenges, including communication between FPGAs, network topology design, and task-level partitioning.

The project seeks to address these challenges by developing a communication interface between the programmable logic sides of the ZCU102 and PYNQ-Z2 boards, implementing consecutive dataflow across boards, deploying a complete neural network on a multi-FPGA system, and designing a NoC module for handling complicated data transfer and sharing requests. By doing so, the project aims to provide a more efficient and scalable solution for neural network implementation on FPGAs, which could significantly reduce the time and cost required for neural network training and inference.

The project will have the following potential applications:

1) Autonomous Driving: Autonomous driving is a complex task that requires real-time processing of a vast amount of data. FPGAs can be used to accelerate the inference of deep neural networks that are commonly used in autonomous driving applications. A multi-FPGA system can enable distributed computation, which can improve the overall performance of the system.
2) Video Processing: FPGAs can be used to accelerate video processing tasks such as object detection and

recognition. A multi-FPGA system can enable distributed computation, which can potentially improve the throughput of the system. The use of advanced communication techniques such as the Aurora IP and SFP+ interfaces can further improve the efficiency of the system.
3) Medical Image Processing: Medical image processing is a computationally intensive task that requires high-performance computing. FPGAs can be used to accelerate the inference of deep neural networks for medical image processing applications such as tumor detection and segmentation. A multi-FPGA system can enable distributed computation, which can potentially improve the throughput of the system.

Overall, this project is important and worth doing because it addresses an emerging need in the field of artificial intelligence for more efficient and scalable solutions for neural network implementation. The project's results could have significant implications for a wide range of applications, from self-driving cars and robotics to healthcare and finance.

III. Related Work

Communication overhead between FPGA boards is a concern in Multi-FPGA implementation. There are several different approaches to improve the communication performance.

The research [2] proposes an innovative approach to address the challenges of large-scale RNN inference and demonstrates the effectiveness of multi-FPGA acceleration with full parallelism levels. The proposed approach has the potential to enable the efficient and scalable inference of large-scale RNNs in various applications, including natural language processing, speech recognition, and machine translation.

The research [4] proposes a custom memory which is based on shared memory communication mechanism. In shared memory mechanism, the implementation that allows simultaneous access to the same location, has various manufacturing difficulties. With a parameterization in FPGA of memory, the memory, which is a N ports writing and N port reading system, for communicating between N processors, enable simultaneously read and write to the memory and each processor can write to the same location simultaneously.

The research [5] introduced PlasticNet, a custom network architecture for interconnected multi-FPGA system which is flexible, reconfigurable and fully compatible with applications designed using a HLS approach. To improve the performance further, the team propose PlasticNet+ [6], the integration of high-speed transceivers into the PlasticNet framework.

The research [7] proposes a multi-FPGA interconnection framework. The proposed framework is fully reconfigurable of packet size, number of lanes per channel, and network interconnection topologies. It also enable the communication among different PEs on the same FPGA board or across different FPGA boards. Part of the design in the research is implemented with C++ and HLS.

IV. Proposed Approach

This project aims to develop an efficient multi-FPGA system for neural network inference. The project proposes to use FPGAs as accelerators for neural network inference, which allows for high performance and flexibility compared to traditional CPU or GPU-based approaches.

The project's approach is different from existing work in several ways. Firstly, the project proposes to use a multi-FPGA system for neural network inference, which allows for distributed computation and potentially higher throughput. Additionally, the project proposes to explore advanced communication techniques such as the use of Aurora IP and SFP+ interfaces for high-speed data transfer between FPGAs.

The project also aims to explore techniques such as quantization and pruning to reduce the memory and computation requirements of neural networks, which can further improve the performance of the system. Finally, the project proposes to explore task-level partitioning and network topology design for efficient resource allocation and scheduling in the multi-FPGA system.

A. Aurora IP

The Aurora IP core is a high-speed serial communication protocol used for data transmission between FPGAs or other devices. It provides a reliable, high-speed, low-latency communication channel between two or more FPGAs in a multi-FPGA system. The Aurora IP core is commonly used in multi-FPGA applications, where it facilitates communication between the FPGAs.

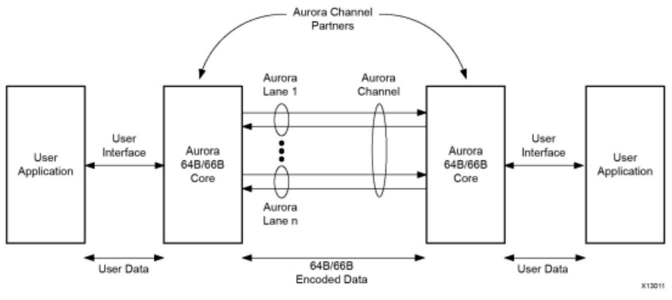

Fig. 1. Aurora 64B/66B Channel Overview

In a multi-FPGA system, the FPGAs typically communicate with each other using a point-to-point communication protocol, such as Aurora, rather than a shared bus architecture. This approach enables higher bandwidth, lower latency, and better scalability than a shared bus architecture, which can become a bottleneck in large multi-FPGA systems.

The Aurora IP core provides several features that make it well-suited for PL to PL communication in multi-FPGA applications:

1) High-speed data transmission: The Aurora IP core supports data rates of up to 12.5 Gbps, providing a high-speed data transmission channel between FPGAs.
2) Error detection and correction: The Aurora IP core uses a 64B/66B encoding scheme [8], which provides built-in

error detection and correction capabilities, ensuring that data is transmitted reliably over the link.

3) Synchronization and alignment: The Aurora IP core includes a synchronization mechanism that ensures that the data transmitted by one FPGA is properly aligned with the data received by the other FPGA, even if there are timing variations or signal distortions on the link.
4) Scalability: The Aurora IP core can be easily scaled to support multiple links between FPGAs, providing a flexible and scalable communication channel for multi-FPGA systems.

B. SFP+ Interface

The SFP+ interface is a hot-pluggable transceiver module that supports data rates of up to 10 Gbps. As shown in Fig.2, the SFP+ interface can be used to connect the ZCU102 board to other devices, such as other FPGAs or network switches, using fiber optic cables or copper cables**??**.

In a multi-FPGA application, the SFP+ interface can be used to provide a high-speed communication channel between FPGAs, enabling them to exchange data at rates of up to 10 Gbps. The SFP+ interface can be used to implement point-to-point links between FPGAs or to connect FPGAs to a network switch for communication with other devices.

The SFP+ interface can be connected to the PL side of the ZCU102 board, allowing the user to implement custom communication protocols and interfaces between FPGAs. This enables the user to develop custom PL to PL communication solutions that meet the specific requirements of their multi-FPGA application.

The SFP+ interface also provides several features that make it well-suited for PL to PL communication in multi-FPGA applications, including:

1) High-speed data transmission: The SFP+ interface supports data rates of up to 10 Gbps, providing a high-speed data transmission channel between FPGAs.
2) Flexible connectivity: The SFP+ interface can be used to connect the ZCU102 board to a wide range of other devices, including other FPGAs and network switches, providing flexibility in designing multi-FPGA systems.
3) Low-latency communication: The SFP+ interface provides a low-latency communication channel between FPGAs, enabling real-time data processing and communication in multi-FPGA applications.

By utilizing the Aurora IP core and SFP+ interface for PL to PL communication in multi-FPGA applications, designers can take advantage of its high-speed, reliable, and scalable communication capabilities, enabling them to build larger and more complex multi-FPGA systems with improved performance and reliability.

Overall, the project's approach is unique in its combination of distributed computation, advanced communication techniques, and resource-efficient neural network techniques to develop an efficient and flexible multi-FPGA system for neural network inference.

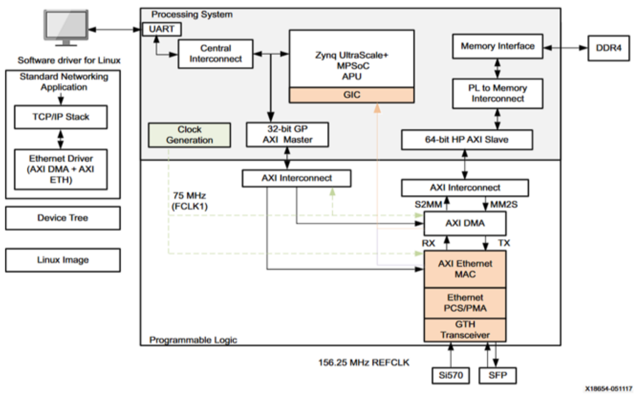

Fig. 2. Aurora 64B/66B Channel Overview

V. Aurora Cross Board Demo

In this work Aurora act as a data stream line between GTH transceivers exchanging AXI4 stream data. 64b/66b encoding is applied in the protocol to guarantee DC balance during transmission. However, whether the clocking across board with 64b/66b decoding will cause performance degradation remains to be seen.

Our first goal is to utilize the GTH transceiver on PL to accomplish BRAM-BRAM communication to enable PL-PL communication. A preliminary study is first conducted on HLS software level, then verified on board to justify that it behaves as expected.

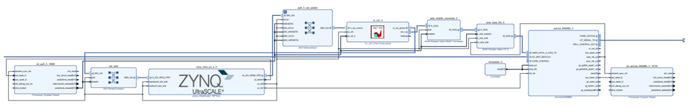

Fig. 3. Receiver Diagram in Vivado

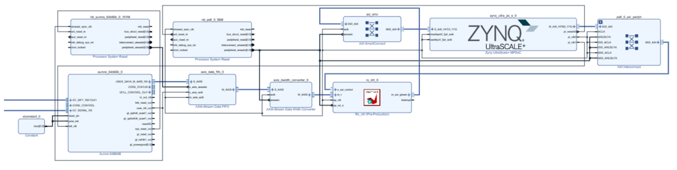

Fig. 4. Transmitter Diagram in Vivado

The block design of the data transmission and data receiving parts are shown in Fig. 3 and Fig. 5. For the sake of demonstration, we built a pair of dummy functions for transmitting and receiving data. For functions arguments, HLS stream is a must for using Aurora to channel data out and into the module, where the datatype is set as 32bits AXI stream.

Besides the dummy functions themselves, some additional parts needs to be added to properly establish the transmitter.

Due to the bit width mismatch between our integer type operand and the data width used by Aurora, which is 64 bit, a data width converter is needed. Also, the PL block frequency is set in 100MHz while Aurora works at 156.25MHz. The frequency of Aurora reference clock is set by default and our guess for that this has to agree with SFP port we used for communication. Since the two modules works across time domain, we put an asynchronous FIFO there.

The receiver is just the counterpart of the transmitter module. It is designed and tested likewise. Aurora block, FIFO, data converter are chained in a reversed manner to receive data from the SFP port. Since the dummy reading block is designed to work at the same frequency as the transmitting block, we simply mirrored the block design to ensure its behavior is correct.

One broad is deployed with a transmitter module plus aurora IP while another is loaded with a receiver module and aurora. To elaborate, two streams are responsible for in and out respectively using AXI port, and the control signals are wrapped with AXI_LITE. As such, test data is sent to the receiver board and stored locally. After transmission, the sent data is verified to assert the correctness of the sent array, which equivalently states the viability of cross-board communication.

We generated the bitstream files of the aforementioned two modules and placed them on two ZCU boards respectively. 1000 predefined integers were transmitted from one to another and could be correctly be printed out. On either of boards, the zynq PS is only responsible for generating control signals for AXILite. We thereby demonstrated the functional correctness of PL to PL transmission to enable dataflow from board to board bypassing PS. This offer potential high speed communication and could be a vital component to deploying large neural nets on different boards.

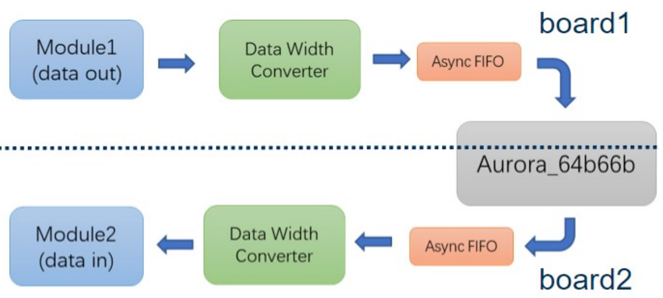

Fig. 5. Block Design Demo

VI. Neural Network Layer Demo

To extend our demo to accommodate neural nets(NN), we applied tiled convolution layer we finished as transmitter, and an ranged batch normalization(RBN) layer as receiver on another board. It is a common practice to apply batch normalization layers after convolution layers in many NN architectures. As they facilitate faster training and convergence. The RBN we applied here is a modified version of L2BN. The main difference is that the normalization factor here is obtained through the range of output feature map and a constant determined by batch size. RBN therefore avoids the hardware unfriendly exponential operations. For this project, since we are only simulating inference, the algorithm is much simpler. Since the mean and variance is computed as running mean and running variance during training, we assume that the values are stored on chip and directly apply them during inference. The operation is therefore simplified as an affine transformation.

We replaced our dummy transmitter with tiled convolution layer, dummy receiver with RBN layer to establish our demo for distributed NN implementation. Besides modules variation, data conversion is another issue we faced. Tiled convolution layer uses fixed point number that takes 16 bits to represent. We accordingly used AXI stream data type with data width changed to 16. But direct type casting between AXI stream and AP_INT is proven problematic. Hence, we has to perform bit wise operation for each 16 bit operand, which is costly but so far our only solution for guaranteeing functional correctness.

For on board testing, we first deploy the two bitstream files on two different boards. Two kernels are started and halt before the transmit/receive commands. Receiver module is activated first, as it will continuously reads data from stream until a predefined exiting condition is triggered. Then the transmitter block is set as active and starts streaming the output feature map through Aurora. The RBN keeps receiving data, processing them and then storing them in local memory. The transmitted data is again proven right, indicating correct behavior of our proposed model.

Convolution and Batchnorm layers are connected in a sequential manner. But this cross board connection method could also be used breadth-wise, for example in a Resnet/Wideresnet like network. Moreover, dividing a single huge layer, for example, large fully connected layers into multiple board is possible. But correct arithmetic operation sometimes need all the parameters for the layer. This may cause parameter duplication and therefore degrade the efficiency of using multiple FPGA for an NN application. How to tackle that problem would require more co-design techniques between algorithm and hardware implementation.

VII. Next step

From experiments we have conducted so far, we could say that the functionality of our proposed cross board PL to PL transmission is correct. Our next step involves to measure the performance of utilizing Aurora for communication. This could be extended further to quantify its bandwidth and efficiency.

Since we have our receiver and transmitter deployed in two separate boards, the main problem we face comes from synchronizing two clock domains for a unified timing measurement. Ideally, the best way for testing is to use a loopback header on the SFP port and measure the protocol overhead on a single board. But that requires to reconfigure Aurora into duplex operating mode. We've been doing the project with Aurora in simplex mode. Moreover, there is no loopback connector available. So this proposal is dropped.

Our next approach is trying to synchronize two ZCU boards where data is transmitted from one to another. We tried to setup ntpserver on Ubuntu. Although the setup was successful, the precision is not high enough to support timing measurement. This approach is dropped as well.

The workaround we are planning is described in the following. In short, we are going to test the latency on a single board, with the simplex Aurora and two SFP ports. The transmitter and receiver module would be deployed on the same board, each with an individual Aurora block handling data streaming out/in. The two aurora ports are mapped to different SFP ports available on the board. When activated, data stream out from transmitter aurora port and directly comes back into the same board to the receiver. Since the two modules now shares the same clock signal, measurement should be available. We had this proposal clarified in figure 4.

Some potential problems lies in triggering modules when there are two modules on a single board. We have seen issues when placing receiving modules, namely aurora, and RBN on the same board being a problem during earlier experiments. Some detailed study maybe needed here.

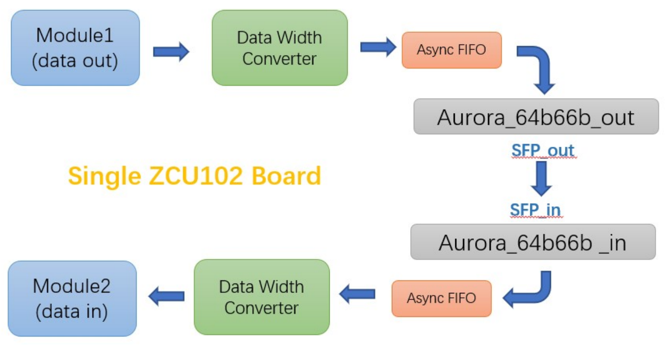

Fig. 6. Block Design Demo

Another test object concerns with HLS streaming. In our early stage of design implementation, we are treating the dataflow as a system with two nodes. In this simplified demo, each node act as either transmitter or receiver. Therefore, we could treat data streaming as an easy implementation with the help of HLS packaged AXI stream data type. But the design for a complete large scale neural network will rise to a more complex scenario and cannot be treated with this trivial manner.
Assume in a scenario where we have multiple boards, each with the assignment of different layers of a large neural net. If we wrote the HLS design in the previous manner, this sequential execution will be conducted in an unpipelined manner. The components staying idle means huge resources and power waste. When all computations are designed in a single board, HLS synthesis infers pipelining automatically. But since we are considering multiple boards, we cannot leverage such convenience. How to achieve pipelining with batches of feature maps under streaming data is a difficulty and also a huge opportunity for performance improvement.
To kick start, we need also figure out how to enable this streaming feature in our c++ testbench file. After proving the functional correctness, we then proceed to HLS and vivado design implementation.

VIII. Conclusion

We conducted a preliminary study on Multiple FPGA boards' BRAM to BRAM communication. The key part in this project falls in utilizing Aurora protocol properly for data transmission. Some additional parts were added in the block design to solve cross time domain of different frequency. Moreover, streaming data with type casting is proven a hindrance and solved through bit level manipulation. With the transmitter and receiver designs deployed on two ZCU102 boards respectively, we were able to observe the expected data transmission behavior. We expect this study could be extend to bigger level for deploying large applications on multiple resource constrained FPGA boards. And could provide hints to the multi-board systems or on edge designs.

IX. Subsessions

1) Stage One: Communication Interface between PL sides of boards (done)
    a) Explore board specifications of ZCU102 and PYNQ-Z2.
    b) Instantiate Aurora 64B/66B IP to enable communication.
    c) Performance evaluation.
2) Stage Two: Dataflow across FPGAs (done)
    a) Implement and test consecutive dataflow across boards.
    b) Integer data layers implementation with the above dataflow.
3) Stage Three: Applying Neural Network layers as demo (done)
    a) Implement and test consecutive dataflow across boards.
    b) Evaluate on board behavior with fixed point data
4) Stage Four: Explore scalability, explore pipelining under streaming across multiple boards (TBD)
    a) Measure the performance on a single board aiming to quantify protocol overhead.
    b) Verify the functional correctness of multiple cascading operations. The data here should be streaming and different stages of operations should be conducted in a pipelined manner.
    c) Test multiple stages of execution on board

References

[1] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. ACM Trans. Reconfigurable Technol. Syst. 11, 3, Article 16 (September 2018), 23 pages.

[2] D. Kwon, S. Hur, H. Jang, E. Nurvitadhi and J. Kim, ”Scalable Multi-FPGA Acceleration for Large RNNs with Full Parallelism Levels,” 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218528.

[3] Aurora 64B/66B Protocol Specification, https://docs.xilinx.com/v/u/en-US/aurora_64b66b_protocol_spec_sp011, October 1, 2014

[4] E. Strollo and A. Trifiletti, ”A shared memory, parameterized and configurable in FPGA, for use in multiprocessor systems,” 2016 MIXDES - 23rd International Conference Mixed Design of Integrated Circuits and Systems, Lodz, Poland, 2016, pp. 443-447, doi: 10.1109/MIXDES.2016.7529783.

[5] C. Salazar-García et al., ”PlasticNet: A low latency flexible network architecture for interconnected multi-FPGA systems,” 2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA), San Jose, Costa Rica, 2020, pp. 1-4, doi: 10.1109/PRIME-LA47693.2020.9062749.

[6] C. Salazar-García, R. García-Ramírez, R. Rímolo-Donadío, C. Strydis and A. Chacón-Rodríguez, ”PlasticNet+: Extending Multi-FPGA Interconnect Architecture via Gigabit Transceivers,” 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Korea, 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401058.

[7] C. Salazar-García et al., ”A custom interconnection multi-FPGA framework for distributed processing applications,” 2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI), Porto Alegre, Brazil, 2022, pp. 1-6, doi: 10.1109/SBCCI55532.2022.9893238.

[8] ”Aurora 64B/66B v12.0 LogiCORE IP Product Guide”, https://docs.xilinx.com/r/en-US/pg074-aurora-64b66b

[9] ”Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit”, https://www.xilinx.com/support/documents/boards_and_kits/zcu102/ug 1182-zcu102-eval-bd.pdf

[10] Yang Zhijie, Wang Lei, Luo Li, Li Shiming, Guo Shasha, and Wang Shuquan, ”Bactran: A Hardware Batch Normalization Implementation for CNN Training Engine”, IEEE EMBEDDED SYSTEMS LETTERS, VOL. 13, NO. 1, MARCH 2021

[11] Jiangmiao Pang,Linlu Qiu,Xia Li, Haofeng Chen, Qi Li, Trevor Darrell, Fisher Yu, ”Quasi-Dense Similarity Learning for Multiple Object Tracking”, arXiv:2006.06664v4 [cs.CV] 8 Sep 2021