Accelerated Data Analytics Using Python-Programmed FPGA Hardware

K. R. McCoubrey
Queen's University Belfast, UK
kmccoubrey02@qub.ac.uk

Abstract—**The PYNQ project enables the application of the high-level programming language Python, to be implemented onboard a Field Programmable Gate Array. This high-level adaption enables software developers with limited hardware design experience, to benefit from the FPGA acceleration capabilities. FPGAs are powerful devices with configurable programmable logic, enabling versatile optimization of applications. The addition of high-level access and Python implementation may incur additional problems which are latter studied. Introduction of high-level inclusivity results in optimization at low-levels becomes more difficult. We research viable options for acceleration within the Python code using overlays and low-level options through high-level synthesis. In this research article we explore the methods available for high-level developers to accelerate data analytics on an FPGA. We tailor this acceleration towards the K-Means clustering algorithm for image processing.**

Index Terms—**FPGA, K-Means Clustering, PYNQ, Python, Computer Vision**

I. Introduction

Field Programmable Gate Arrays offer a powerful and efficient deployment for data analysis when combined with an effective hardware configuration and low-level programming. Embedded systems requiring low-power, yet demanding computation and rapidly increasing through fields such as autonomous driving and widespread artificial intelligence solutions. FPGAs are limited by their compatibility of programming techniques. The addition of support for a high-level programming language creates both new opportunities and potential problems, both of which will be explored. One such explored opportunity is the implementation of a K-Means clustering algorithm as a pre-processing stage, for image analysis onboard an FPGA device.

Existing research provides exemplary material proving the power of FPGA hardware combined with an FPGA enabling real-time image processing. Real-time processing is made available by altering the standard Lloyds algorithm for k-means to a histogram-based algorithm using the low-level programming capabilities [1].

This research aims to explore the acceleration of data analytics for image processing on an FPGA, specifically the speedup of K-Means algorithms hardware acceleration. Python is not supported by FPGA boards traditionally due to it's high-level nature, meaning this code stands many layers from machine code itself. Python must first be run through an interpreter program converting the python code into a low-level language such as C. A second program, called a compiler, takes the interpreted code and converts into machine code therefore enabling the processor to run. Python traditionally does not have direct access to hardware resources and configurations necessary for speedup in FPGA algorithms.

Xilinx, the inventor and manufacturer of the FPGA, released in 2016 the PYNQ project. The goal of PYNQ is to enable easier exploitation of the unique benefits of Xilinx devices for designers of embedded systems [2]. Using Python libraries and language, designers can now exploit the benefits of programmable logic and microprocessors in Zynq to build more capable embedded systems. Programmable logic circuits are presented with hardware libraries named *overlays*. Along with overlays, Xilinx produces a High-Level-Synthesis tool (Vivado HLS) enabling hardware acceleration.

The K-Means algorithm is a commonly used method to automatically partition a data set into *k* groups. It proceeds by selecting *k* initial cluster centers and then iteratively refining them. Explained in [3] by:

1. Each instance d_i is assigned to its' closest cluster center
2. Each cluster center C_j is updated to be the mean of its constituent instances.

"It is well known that K-Means clustering is highly sensitive to proper initialization. The classic remedy is to use a seeding procedure […] known as k-means++" [4]. K-Means initializes its beginning cluster centroid points randomly, and algorithmically searches for better centroids through follow-up iterations. K-Means++ by Arthur and Vassilvitskii [5] begins by initializing one centroid and allocates the remaining cluster centroids relative to the first point. Both techniques have advantages and disadvantages regarding precision, speed, and overheads and these will be examined further.

To solve this problem and test various methods, a PYNQ instance is setup enabling Python production on an FPGA. In this scenario, the PYNQ-Z2 board will be used for testing and analysis. Configurations are defined throughout.

II. Literature Review

Initial optimization/acceleration methods have been researched in source [6] regarding the performance of FPGA acceleration using various methods. The author outlines the options for using *high-level synthesis* to address design productivity. HLS enables raising the programming level from the low-level RTL used in languages such as Verilog, to higher level languages of C or C++. Although helpful, Vivado and HLS still requires the expertise of low-level design engineering, restricting its validity of the article topic regarding general purpose software acceleration. Restrictions to productivity are outlined that with HLS tools there remains prohibitive compilation times impeding many mainstream adoptions. A common problem with FPGA accelerations are the methods involved need tailored toward specific FPGA configurations. A solution is produced with the engineering of a programmable course-grained hardware abstraction layer, named an *overlay*. Overlays are deployed to the developer as a programming method, enabling various improvements. Better portability across devices, design reuse, and rapid configuration with orders of magnitude faster than other reconfigurable approaches. The article proceeds to analyze overlay options between time-multiplex and spatially-configured. Overheads are discussed regarding the respective overlay types, but the author does not proceed to conclude which type may or may not suit application implementations for a high-level engineer. Benchmarks are run testing various overlay and hardware configurations, with discussion on hardware performance penalties, and improvements of route time and hardware kernel switching time. The author neglects to conclude the real-time effects this may cause on general purpose applications in which the report is based around. Focusing on portability and mainstream adoption of FPGA based accelerators, there is a clear gap in the analysis. Future testing for K-Means algorithms will need to further explore overlay types and experiment with individual performance effects, specifically for the chosen algorithms. This experimentation will bridge the gap of information identified in the article.

The next publication [7] starts by addressing the target audience of developers who are non-experts in the field of HW description languages or DPR (Dynamic Partial Reconfiguration). The author explains how overlay architectures abstract the complexity and FPGA design challenges whilst maintaining high performance. Evidence is sourced from multiple publications regarding the sustainability of performance using overlays. This makes for easy evaluation of overlay effects on performance for the reader. The article then explores the option of implementing the readily available base overlays, along with partial reconfigurable overlays. Partial configurable overlays aim to make overlays more versatile and to suit better for multiple applications. This may provide a viable option for our acceleration due to its high-level access via Python and ability for hardware configuration. In the related works section, the author explains the PYNQ project basics, and its ability for redistribution via Python package automated installation on other PYNQ systems using 'pip' package manager. This is useful for future reference for further redeployment of PYNQ machines. The author proceeds to explain the ability to produce dynamic partial reconfiguration overlays. This approach benefits from hardware acceleration, while having a shorter reconfiguration time compared to full bitstream overlays due to the inclusion of replacement strategies for the reconfigurable partitions (programmable logic). The author did this by including strategies known from cache memories such as 'least recently used' and 'first in first out'. These methods reduced the resource usage by 40% in preliminary tests. In evaluation of the article, the author has produced a great description of the acceleration possibilities for a high-level developer using Python. Effective strategies have been conveyed and tests ran as evidence. The code used has been released on GitHub which may provide guidance to our custom overlays in future. One gap in the research is the comparison of partial reconfigurable overlays against high-level synthesis, which may provide further understanding of the overheads associated with overlay implementation.

Journal article [8] focusses on the Scikit-learn Python module which integrates a wide range of state-of-the-art machine learning algorithms including the k-means at interest. This module emphasizes primarily on performance and ease of use whilst containing minimal dependencies. Immediately this shows itself as a potential opportunity for our k-means implementation. Since the module is extensively tried and tested, efficient performance and ensuring minimal dependencies is critical when applied to the PYNQ board. Dependencies may lead to potential issues due to the OS configuration with an FPGA, but due to Scikit-learn being only dependent on 'NumPy' and 'SciPy' which are both distributed in the PYNQ official package, this ensures correct functionality and facilitates easy distribution. Further depth of information is discussed regarding individual functions within Scikit-learn and its' corresponding dependencies, providing sufficient confidence on how the library will run on a PYNQ board. The author evaluates Scikit-learn against various machine learning libraries exposed in Python by running benchmarks on a Medalon dataset with multiple algorithms each including k-means. Scikit-learn proves to be superior in elapsed time of task completion. This information is useful for our choice of Python library during implementation. One gap in the test is the lack of a control, or raw coded algorithm as a reference point without overheads of calling a library. Some libraries, such as OpenCV, which includes a K-Means function are not included in this test, lacking a comparison between two of the largest machine learning Python libraries. In conclusion the author has presented sound evidence that Scikit-learn is a capable library and suitable for implementation on the PYNQ board due to its small dependencies but lacks a comparison with OpenCV which will need to be further reviewed.

In conclusion to the literature review, sufficient information has been extracted in relation to accelerating a k-means function. Scikit-learn provides a safe implementation of k-means whilst ensuring efficient performance, but during further research through developers and alternative sources, OpenCV proves more reliable, stable, and industry friendly for image processing applications.The PYNQ project already contains some supporting material for OpenCV designs, therefore may prove more effective for this design. Acceleration will be added through overlays, enabling a Python implementation of hardware configuration. Further hardware acceleration will be explored through partial reconfigurable overlays and high-level synthesis.

III. DEVELOPMENT PLAN

The aim is to research acceleration of data analytics using the python programmed FPGA hardware. The final goal is to utilize the FPGA properties on the PYNQ device, to accelerate image processing, by way of implementing a pre-processing K-Means IP block for streamed image data which is then accessible to the Python frontend. As a new technology, the PYNQ-Z2 board is yet to receive substantial testing on its' limitations regarding python implementations, due to its' rapidly evolving operating system versions and limited hardware resources. Extensive iterative testing is crucial as existing designs and solutions may not be entirely stable between versions. The latest OS version available and used throughout is the PYNQ v2.4 [9]. Initial test and implementations focus on the python development environment with K-Means clustering over image data. The tests will include a function for K-Means clustering provided by the industrial grade 2D image processing OpenCV library. This library is mature and stable, improving the reliability of results. Primary testing will approach from a purely software developer approach to explore any limitations of the python environment. This will read image data from its' microSD non-volatile external memory storage, for alternating K-Means algorithm types (regular vs K-Means++), on various color formats. Upon completion, secondary testing will be conducted to analyze performance of a video capture options. The Linux backend which the PYNQ system is built on, recognizes HDMI as input unlike other system options. This enables testing initiated via OpenCV options, compared to HDMI input options available through the 'base' overlay. The primary acceleration methodology for the PYNQ if through the hardware design synthesis available. Designs will be iteratively testing for standard data transmission and further into image processing, aiming toward real-time analysis.

IV. K-MEANS VS K-MEANS++ OVER IMAGE DATA

Prior to K-Means implementation, a previously working version of HOG human detection had been run on the PYNQ to test potential limitations. The program worked efficiently on regular PC hardware, but unknown errors became apparent when produced on the PYNQ. This may be due to the limited resources available to the board in terms of RAM and overall memory busses. The PYNQ runs with an ARM A9 dual-core processor, with 512MB DDR3 with 16-bit bus at 1050Mbps. Therefore, emphasis on iterative testing of the current PYNQ setup must be made throughout to try and pre-determine any unexpected issues later. Basic python programs were written to handle data, images, and analyse image structure and pixels. The programs ran effectively with no apparent issues beyond a regular software engineers' scope. The next stage is to begin the implementation of k-means over image data. D. Arthur [5] reports observations of k-means++ consistently out-performing regular k-means, in many cases more than 50% faster in execution time. The nature of the k-means++ algorithm having a pre-processing step will initially create extra overheads, but over the large datasets used by Arthur [5] such as the *intrusion* dataset [10] consisting of 494019 point with 35 dimensions, this step greatly reduces the overall computation. Arthur also used large cluster numbers from 10, 25, 50 again enabling greater optimization with the k-means++ pre-processing stage.

RGB Resolution	Clusters	Method	Time(s)
1920x1080	4	K	49.845
1920x1080	4	K++	46.680
355x200	4	K	1.740
355x200	4	K++	1.820
177x100	4	K	0.569
177x100	4	K++	0.570
Greyscale			
1920x1080	4	K	31.448
1920x1080	4	K++	26.967

Table 1: ARM based K-Means Clustering

The results from testing k-means vs k-means++ indicate that processing high definition images such as 1920x1080px image results in very slow performance. With computation timing of 49 and 46 seconds respectively, it is vastly far from achieving real-time processing. K-means++ provides a consistent improvement for the higher resolution image, due to a larger set of datapoints to be processed and the pre-processing stage of cluster initialization becoming more effective with size. Image resolution was resized within the algorithm to then test an approach for speedup. With the reduction of image pixels, the execution time vastly drops, but during testing the k-means++ approach could prove slower. This will be due to the extra computation overhead outweighing its benefit over larger sets, because of the downsizing.
Image colour format is also tested with OpenCV colour conversion taking place prior to k-means analysis from BGR to Greyscale. This effectively reduces the dataset size again for the image as each individual pixel has less data, from three colour values to one. K-Means clustering again improves in execution time over the smaller set.

V. CAPTURE METHOD – OVERLAY VS ARM HANDLING

The linux distribution acting as an operating system for the PYNQ 2.4 image recognizes HDMI ports as input devices. This allows the current OpenCV formatting already tested to recognize a data stream from the HDMI-IN port on the board, without the need for USB interface. The 'base' overlay provided with the PYNQ image also enables video analysis using the default block design and IP provided for HDMI connectivity between both IN and OUT ports, and the ZYNQ7020 chip. Video stream was sent from a laptop at

1280x720 as a screen duplicate. Both methods iterate through 600 frame captures, with each capture waiting for a new frame. Both methods achieved real-time importing of frames and therefore should not provide any bottlenecks with further development into this methodology.

VI. OVERLAY AND IP DESIGN

The final stage is to prepare an IP and block design capable of processing the K-Means clustering algorithm. The initial approach to the problem is to create an IP intercepting the image data between existing blocks of the base overlay. This way, the user still has full control over the extensive video functionality included with the board and not only a clustering block. A software engineer will be less capable of creating full block designs from scratch with the k-means IP, and the passing of data between board components. A compatible IP block capable of dropping in to one data stream is ideal, enabling simple instructions to be followed for implementation onto the base overlay design. Iterative testing for production began small with the creation of a simple adder block design for passing integer data via DMA direct memory access. This adder was nested into a loop to add some minor complexity for timing and resource testing.

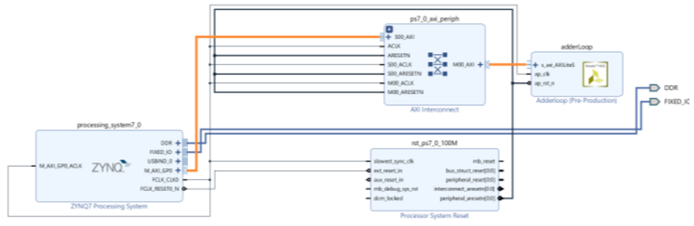
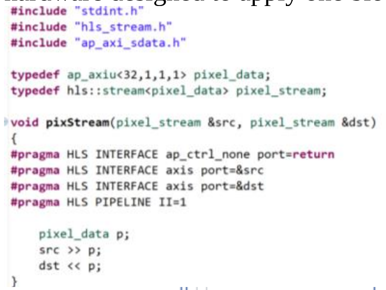
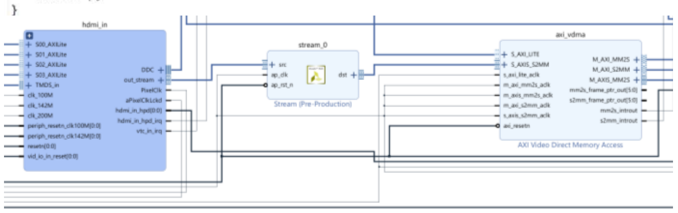


Figure 1 adderLoop block design

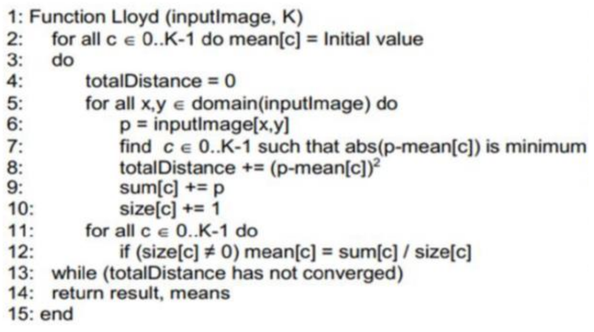
The adderLoop block design ran effectively without issues. To begin analyzing video frames, a simple pixel streaming block was designed and implemented into the base overlay design. This design uses pre-processing to the VDMA video direct memory access within the 'base' overlay, which allows the Python code to handle full video frames in and out of the VDMA block. This solution would be ideal as a drop in processing pipeline, easily allowing an in-experienced hardware designed to apply one block within a data path



```
#include "stdint.h"
#include "hls_stream.h"
#include "ap_axi_sdata.h"

typedef ap_axiu<32,1,1,1> pixel_data;
typedef hls::stream<pixel_data> pixel_stream;

void pixStream(pixel_stream &src, pixel_stream &dst)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE axis port=&src
#pragma HLS INTERFACE axis port=&dst
#pragma HLS PIPELINE II=1

    pixel_data p;
    src >> p;
    dst << p;
}
```



Upon generation of the IP block and placement within the 'base' overlay, the generation of the bitstream to build the files necessary for testing the overlay failed. Extensive time was spent trying to debug the issue of the streaming block, but eventually the fault was found in the official overlay design supplied by the manufacturers. The original was re-downloaded to reduce any chance issues previously cause from editing, but the fault still occurred on a default build with recurring "place_design error" faults. The fault takes approx. 1 hour 30 minutes to occur during the generation of the bitstream, whilst validation of the design was successful. Heavy amounts of time were dedicated to trying to resolve the issue or bypass the problem, but as a software engineer the depth of the issue was too much for the limited time for development. The C++ K-Means code developed was not able to be tested within the design because of the outstanding issue with the supplied 'base' overlay. Another approach had been previously considered, by retrieving the image data from the installed 'base' overlay, storing, and sending the image data via DMA or VMDA in a new overlay design. Having to dynamically reload overlays between processing is unsuitable for effective performance due to loading and driver setup. Data transmission is limited by size with the use of DMA to memory mapped IO, and remains unsuitable data handling of large image resolutions on the PYNQ board.



```
1: Function Lloyd (inputImage, K)
2:    for all c ∈ 0..K-1 do mean[c] = Initial value
3:    do
4:         totalDistance = 0
5:         for all x,y ∈ domain(inputImage) do
6:              p = inputImage[x,y]
7:              find c ∈ 0..K-1 such that abs(p-mean[c]) is minimum
8:              totalDistance += (p-mean[c])$^2$
9:              sum[c] += p
10:             size[c] += 1
11:        for all c ∈ 0..K-1 do
12:             if (size[c] ≠ 0) mean[c] = sum[c] / size[c]
13:   while (totalDistance has not converged)
14:   return result, means
15: end
```

Figure 2 Lloyds' based K-means clustering Algorithm

VII. FUTURE DEVELOPMENT

The K-Means clustering algorithm has scope for improvement to move towards real-time image processing. [1] Displays further acceleration through algorithmic development, enabling improved speeds and a reduced in hardware resources. With a functional solution to the Lloyd's based k-means clustering algorithm, the histogram based solution developed in [1] would be a direct drop-in, assuming the image data was to be handling in greyscale for improved speeds.

VIII. CONCLUSION

Research proves the potential for high performance applications launched on FPGA hardware. For a software developer with strictly a high-level experience background, implementation of unsupervised machine learning image processing designs can be implemented successfully on the PYNQ python front-end, whilst remaining stable for k-means clustering. Alternative may prove volatile under other conditions. For complex acceleration designs to be produced, a developer should consider a standalone block design without the inference of an existing setup. A newer technology such as PYNQ is rapidly evolving, with modifications to the operating system frequently changing, and support for such a device is limited. One should approach the problem from a hardware design perspective with intentions of designing a standalone hardware solution without the inference of any existing solutions. Stability and reproducibility for the PYNQ project remains an great issue for largescale adaption by high-level software developers, with very little support available for the specific interaction for the PYNQ front-end to hardware.

References

[1] T. Deng, D. Crookes, F. M. Siddiqui and R. Woods, "A New Real-Time FPGA-Based Implementation of K-Means Clustering for Images," in *1st International Conference on Intelligent Manufacturing and Internet of Things, IMIOT 2018 and International Conference on Intelligent Computing for Sustainable Energy and Environment, ICSEE 2018*, Chogqing, 2018.

[2] Xilinx, "PYNQ Introduction," Xilinx, 2018. [Online]. Available: https://pynq.readthedocs.io/en/v2.3/. [Accessed 21 11 2018].

[3] K. Wagstaff, C. Cardie, S. Rogers and S. Schrodl, "Constrained K-Means Clustering With Background Knowledge," in *Internation Conference on Machine Learning*, Williamstown, 2001.

[4] O. Bachem, M. Lucic, S. H. Hassani and A. Krause, "Approximate K-Means++ in Sublinear Time," in *AAAi Conference on Artificial Intelligence*, Phoenix, 2016.

[5] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," in *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithm*, New Orleans, 2007.

[6] A. K. Jain, D. L. Maskell and S. A. Fahmy, "Are Course-Grained Overlays Ready for General Purpose Application Acceleration on FPGAs?," in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Auckland, 2016.

[7] B. Janben, P. Zimprich and M. Hubner, "A Dynamic Partial Reconfigurable Overlay Concept for PYNQ," in *27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, 2017.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel and B. Thirion, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[9] Xilinx, "Xilinx Guthub Repository," [Online]. Available: https://github.com/Xilinx/PYNQ/releases. [Accessed 12 April 2019].

[10] *KDD Cup 1999 dataset*, http://kdd.ics.uci.edu/databases/kddcup99/kddcup.html.

[11] S. K. Thakkar, "Dominant colors in an image using k-means clustering," [Online]. Available: https://buzzrobot.com/dominant-colors-in-an-image-using-k-means-clustering-3c7af4622036. [Accessed 3 11 2018].