A Case Study of Accelerating Apache Spark with FPGA

Junjie Hou[1*], Yongxin Zhu[2†*✉], Linghe Kong[3‡], Zhe Wang[4‡], Sen Du[5*], Shijin Song[6*], Tian Huang[7§]
*School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China 200240
†Shanghai Advanced Research Institute, Chinese Academy of Sciences, Shanghai, China 201210
‡Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China 200240
§Department of Physics, University of Cambridge, Cambridge CB2 1TN, UK
zhuyongxin@sari.ac.cn[2✉], {hjj123[1], linghe.kong[3], wang-zhe[4], du_sen[5], Jennifer-song[6]}@sjtu.edu.cn, th523@cam.ac.uk[7]

***Abstract*—Apache Spark is an efficient distributed computing framework for big data processing. It supports in-memory computation of RDDs (Resilient Distributed Dataset) and provides a provision of reusability, fault tolerance, and real-time stream processing. However, the tasks in Spark framework are only performed on CPU. The low degree of parallelism and power inefficiency of CPU may restrict the performance and scalability of the cluster. In order to improve the performance and power dissipation of the data center, heterogeneous accelerators such as FPGA, GPU, MIC (Many Integrated Core) exhibit more efficient performance than the general-purpose processor in big data processing. In this work, we propose a framework to integrate FPGA accelerator into a Spark cluster. We use FPGA to accelerate the Spark tasks developed with Python, and in this way, the main computing load is performed on FPGA instead of CPU. We illustrate the performance of the FPGA based Spark framework with a case study of 2D-FFT algorithm acceleration. The results showed that FPGA based Spark implementation acquires 1.79x speedup than CPU implementation.**

Keywords: Spark, Distributed Computing, Python, FPGA, Heterogeneous Computing, OpenCL, High Performance, Power Efficiency

I. Introduction

As the rapid increasing of data size in recent years, it raises the challenge of processing these huge amount of data in the data center with high performance and acceptable power dissipation. MapReduce model [1] [2] provides an efficient data flow engine to improve the performance of data processing in cluster environment. The popular MapReduce based frameworks such as Apache spark [3], Apache Hadoop [4] are applied in many distributed computing scenarios. Spark is an open-source distributed framework for big data processing, and it supports in-memory computation of RDDs and provides a provision of reusability, fault tolerance, real-time stream processing [5]. Hadoop is mainly used as the distributed file system.

In Spark application, the tasks are only performed on CPU. The low degree of parallelism and power inefficiency may restrict the performance and scalability of the cluster. Heterogeneous accelerators such as FPGA, GPU, MIC exhibit more efficient performance in big data processing than the general-purpose processor. If we integrate these heterogeneous accelerators to the original Spark framework, we can dramatically enhance the performance of the cluster. Due to the customized hardware architecture of FPGA, it exhibits higher energy efficiency than the fixed architecture such as CPU, GPU, and MIC. In this paper we use FPGA as the heterogeneous accelerator.

Spark applications can be developed with Scala, Java, and Python programming language. Python has been quickly gained popularity over the past few years, and it is used from testing microchips to building video games with the PyGame library [6]. In this work, we explore the integration of FPGA accelerator to the original Spark framework to offload the computing tasks which are programmed by Python from CPU to FPGA. The main contributions are as follows.

- The methodology of integrating FPGA accelerator to the Spark framework. We use Xilinx development tool $SDAccel$ and Python class $ctypes$ to address the gaps between FPGA and Python call. Then we utilize RDD method $pipe$ to connect the Spark application with Python function.
- A case study of 2D-FFT algorithm acceleration on

FPGA based Spark framework. We implement 2D-FFT algorithm on FPGA and integrate it to the Spark framework. We acquire about 1.79x speedup than CPU implementation.

The organization of the rest of this paper is as follows. Section II introduces background and related works. Section III presents the methodology of integrating FPGA accelerator to Spark framework. Section IV illustrates the performance of FPGA based Spark framework with a 2D-FFT algorithm acceleration case. We draw a conclusion in Section V.

II. Background and Related Works

A. Apache Spark

Apache Spark is an open-source cluster computing framework for big data processing. The first system was developed by the group of the University of California, Berkeley, and it gained incubator status in Apache very quickly in June 2013 [7]. Spark overtook MapReduce and Hadoop and emerged as the next generation big data processing engine. Spark supports in-memory computing and enables very fast data query operation compared to Hadoop which is the disk-based engine [5]. Spark maintains the fault tolerance and linear scalability of MapReduce, and it extends a data-sharing abstraction called RDD (Resilient Distributed Data sets). Besides, Spark provides rich API in Scala, Java, Python, so it is easier and convenient to program. Spark also provide many high level tools shown in Fig. 1, such as Spark SQL, MLlib library for machine learning, GraphX for graph processing, Spark Streaming for stream processing, and so on [8].

B. FFT Algorithm

FFT (Fast Fourier Transformation) is widely applied in speech processing, graphic processing and software ratio [9], [10], is a high-efficiency implementation of DFT (Discrete Fourier Transform). In DFT, the sequence X(k) (where k = 0, 1, ..., N-1) is calculated by input sequence x[n] (where n = 0, 1, ..., N-1), it is defined in Eq. 1,

$$X(k) = \sum_{n=0}^{N-1} (x[n] W_N^{nk}) \tag{1}$$

phase factor W_N^{nk} is also called $twiddlefactor$, it is defined in Eq. 2.

$$W_N^{nk} = e^{-jnk\frac{2\pi}{N}} \tag{2}$$

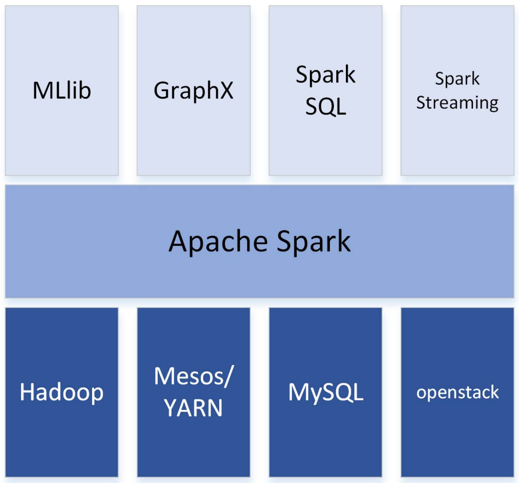

Fig. 1: Apache Spark software stack

FFT is based on decomposing the DFT of N length sequence into successively small DFTs. Algorithm that decompose time series x[n] into successively sub-sequences is called decimation-in-time algorithm. For sequence $X[k]$ decomposition, it is decimation-in-frequency algorithm [11]. In Radix-2 FFT, sequence $x[n]$ is separated into two sequences. One is generated by odd indexes sequence of x[n], and the other is for even. We can write Eq. 1 into Eq. 3, and the complexity of DFT computation is reduced. We only need to compute two DFTs with half complexity. After recursive procedure, the computation reduced to only 2-point. Radix-4/8 FFT is that the essential computation are 4/8-point DFTs. In Fig. 2 we illustrate the signal flow graph of a 16-point Radix-2 FFT example.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{N/2}^{2nk} \tag{3}$$

C. Related Works

In the data center, the heterogeneous accelerators have been used for accelerating the application of cluster. For example, Baidu's Parallel Distributed Deep Learning platform (Paddle) [12]. It integrates GPU and FPGA to accelerate the applications of the cluster. IBM presents the Coherent Accelerator Processor Interface (CAPI) in IBM POWER8. It provides a high bandwidth, low latency path between external devices, the POWER8 core, and the system's open memory architecture [13]. Microsoft has developed a customized FPGA board, Catapult, and placed it into each server in a 1,632-node cluster [14]. Under high load, the throughput of

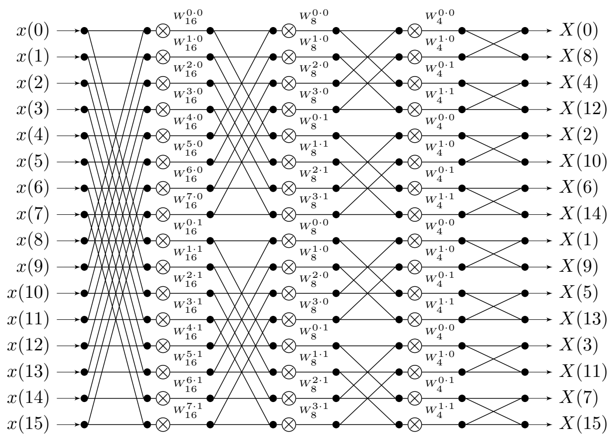

Fig. 2: Signal Flow Graph of a 16-point Radix-2 FFT

each server is improved by a factor of 95%, and while maintaining equivalent throughput, it reduces the tail latency by 29%. However, these works have not provided enough details of integrating FPGA to the cluster. Yu-Ting Chen [15] and Ehsan Ghasemi [16] research on integrating of FPGA to Spark framework in detail. They bridge the gap between JVM and FPGA (Java and C/C++) through JNI. In our work, we use FPGA to accelerate the Spark application developed with Python and propose the method of connecting Spark with Open-CL application which can be used to manipulate diverse processors (such as FPGA, GPU and the other processors which support OpenCL programming) in Spark. We address the gaps between Python and FPGA (C/C++), Spark and the external Python application.

III. FPGA Based Spark Framework

A. Experimental Environment

In this work, the experimental cluster includes 1 master node and 1 slave node as shown in Fig 3. The detailed configurations are listed in Table I. The slave node is equipped with a Xilinx ADM-PCIE-7V3 card. At present, we only have 1 FPGA board available, which limits the size of the cluster. Nevertheless, it does not influence the demonstration of our strategy, which can be similarly applied to the larger size cluster.

B. Methodology of Integrating FPGA to Spark Framework

In Xilinx development environment SDAccel, it provides a way of developing FPGA application through

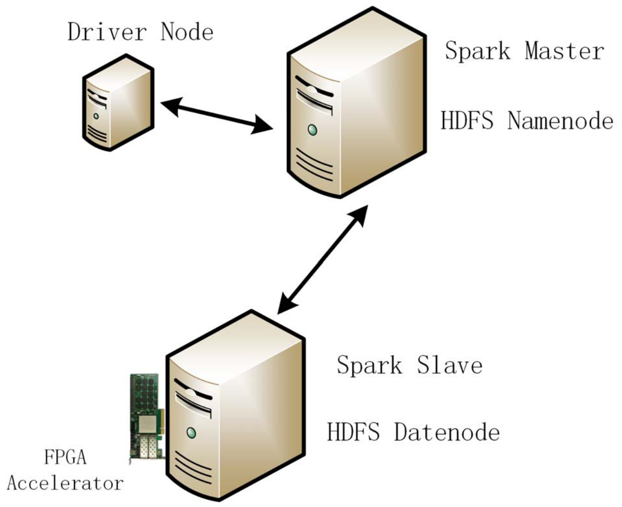

Fig. 3: An Overview of FPGA Based Spark Cluster

TABLE I: Configurations of the Cluster

	CPU	Memory
Master Node	core 2-core i5-6500@3.2GHz	8G
Slave Node	2x Xeon 6-core E5-2620v3@2.40GHz	64G
OS	CentOS 7	
FPGA	ADM-PCIE-7V3 (Virtex 7: XC7VX690T-2)	
Host I/F	PCIe Gen3 x8	
FPGA Tools	Xilinx SDAccel-2017.1	
Other Softwares	Spark-2.1.0, Hadoop-2.7.0, Pyhton-2.7.5	

OpenCL model as shown in Fig. 4. In this way, we can manipulate FPGA (OpenCL kernel) through OpenCL host code (C/C++). In order to invoke the OpenCL host in the other programming languages, we compile the host code into Linux Shared Library (.SO).

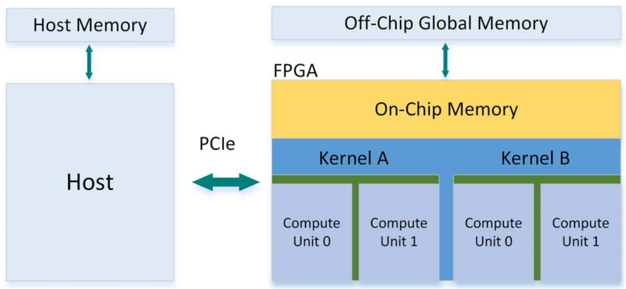

Fig. 4: FPGA Accelerator in OpenCL Model

Spark applications can be developed with Scala, Java, and Python programming language. In our implementation, we use Python language to develop the Spark application. Spark tasks run on JVM (Java Virtual Machine), however, the JVM does not support FPGA devices by default. The first step is to connect the Python code to C/C++ code (OpenCL host code). We use a foreign function library for Python *ctypes* to address this problem. *ctypes* provides C compatible data types, and allows calling functions in DLLs or shared libraries, and can be used to wrap these libraries in pure Python. The official document of *ctypes* library can be referred here [17]. We use *ctypes* in python function to convert Python data type to C/C++ data type and call C/C++ function which manipulates the FPGA, and it is shown in Algorithm 1. In the algorithm, $real\_Py$ and $imag\_Py$ are Python $List$ data type, $real\_C$ and $imag\_C$ are C/C++ $Array$ data type, $libFFT\_Shared.so$ file is the shared library of compiled OpenCL host code.

Algorithm 1 Data Type Conversion and Call of C/C++ function to Perform the FPGA processing in Python function

```
1: /* Convert input of Python data type to C/C++ data type */
2: $real\_C = (ctypes.c\_float * len(real\_Py))(*real\_Py)$
3: $img\_C = (ctypes.c\_float * len(img\_Py))(*img\_Py)$
4:
5: /* Define output of C/C++ data type */
6: $result\_real = (ctypes.c\_float * len(real\_Py))()$
7: $result\_img = (ctypes.c\_float * len(imag\_Py))()$
8:
9: /* Call the C/C++ function to perform the FPGA processing */
10: $fft\_so = ctypes.CDLL("/home/.../libFFT\_Shared.so")$
11: $fft\_so.FFT(real\_C,\ img\_C,$
12: $ctypes.byref(result\_real),\ ctypes.byref(result\_imag))$
```

Through above steps, we can manipulate FPGA in Python code. The next important step is to perform map operation and invoke Python function developed before in Spark. When Spark program is executed, Spark driver segments the RDDs and push out to the Spark Worker(s). The RDD represents an immutable, partitioned collection of elements that can be operated in parallel [3]. RDDs in Spark Worker(s) invoke Python subprocesses using *pipe* method of RDD, which transfers data of RDD to Python function. The *pipe* operation is an RDD method, which allows the developer to process RDD data using external applications. In our scenario, the external application is the python function which invokes the execution of FPGA. The overall methodology of integrating FPGA to Spark framework is shown in Fig 5.

IV. Case study of 2D-FFT Algorithm Acceleration

A. FPGA Implementation of 2D-FFT Algorithm

In the workflow of SKA-SDP (Square Kilometer Array Radio Telescope–Scientific Data Processing), 2D-FFT (Fast Fourier Transform) calculation takes a significant proportion of computation overhead. Thus, in this work, we choose the 2D-FFT acceleration as a case study.

We implement 2D-FFT algorithm on FPGA with OpenCL C in Xilinx SDAccel environment. There are four kinds of algorithms for multidimensional FFT implementation: row-column, vector-radix, nested and polynomial transform [18]. We implement the row-column 2D-FFT in this work, and it is composed of 1D-FFT routines as described in Fig. 6. We compare the single precision raw performance of FPGA (Virtex 7) and CPU (Xeon E5-2620v3) implementation in different scales of 2D-FFT, and the results are presented in Fig. 7. The results showed that the performance of FPGA is inferior to CPU in scales which is smaller than 128 x 128-point 2D-FFT (a single kernel in FPGA implementation). The performance of FPGA is superior to CPU in scales which is larger than or equal to 128 x 128-point. The implementation of 1024 x 1024-point 2D-FFT exceeds the resource limitation of Virtex 7 FPGA. In 512 x 512-point 2D-FFT, the performance of FPGA obtains a speedup about 2.03x than CPU. In the following subsection, we will integrate the FPGA implementation of 2D-FFT to Spark framework and illustrate the performance.

B. Integrating the FPGA Implementation to Spark Framework

We integrate the 2D-FFT implementation to spark framework according to the methodology described before. We compare the performance between FPGA based Spark and CPU implementation of 512 x 512-point 2D-FFT in TABLE II. Compared to the performance of FPGA implementation alone, the performance of FPGA based Spark implementation decreases, and this is caused by RDD generating, computing and collecting operations in Spark. In the larger cluster, the performance of FPGA based Spark implementation will be much better. Nevertheless, in our experimental cluster, we still obtain 1.79x speedup of FPGA based Spark implementation than CPU implementation.

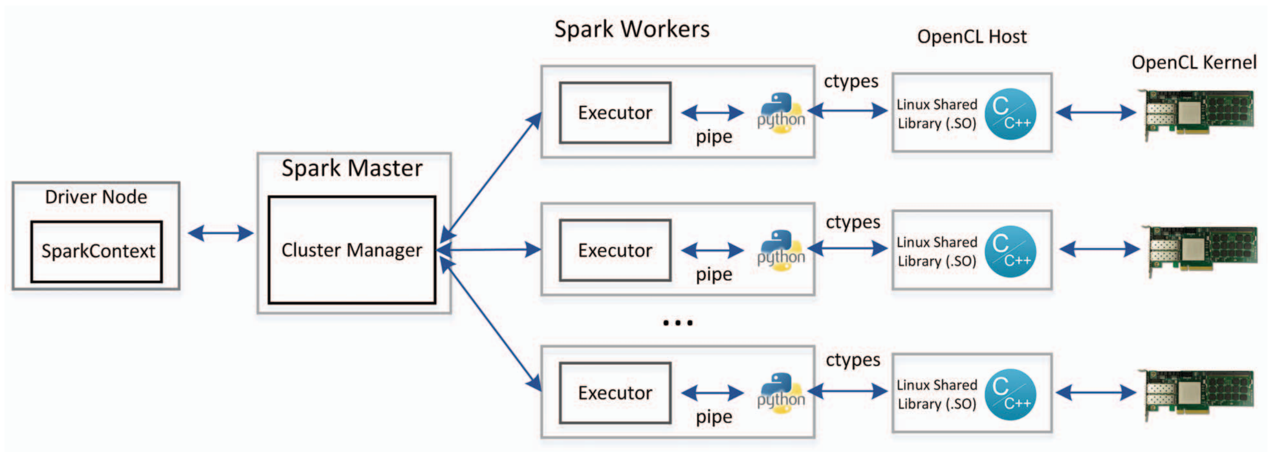

Fig. 5: Overall Methodology of Integrating FPGA to Spark Framework

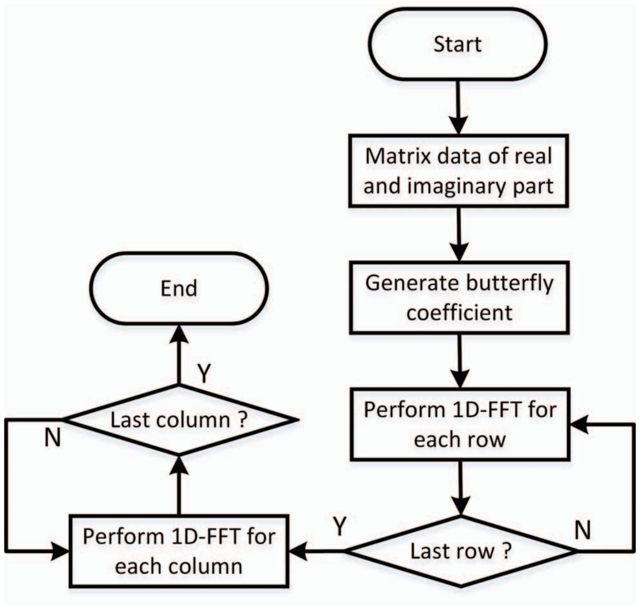
Start
Matrix data of real and imaginary part
Generate butterfly coefficient
Perform 1D-FFT for each row
Last row ?
Y
N
Perform 1D-FFT for each column
Last column ?
Y
N
End

Fig. 6: Algorithm Diagram of Row-Column 2D-FFT

TABLE II: Throughput of FPGA Based Spark and CPU Implementation of 512 x 512-point 2D-FFT (MS/S)

Speedup (1.79x)	FPGA Based Spark	CPU Implementation
Performance	16.31	9.12

V. Conclusions

In this work, we proposed the framework to integrate FPGA accelerator into a Spark cluster. We illustrate the effectiveness of integrating FPGA into Spark framework

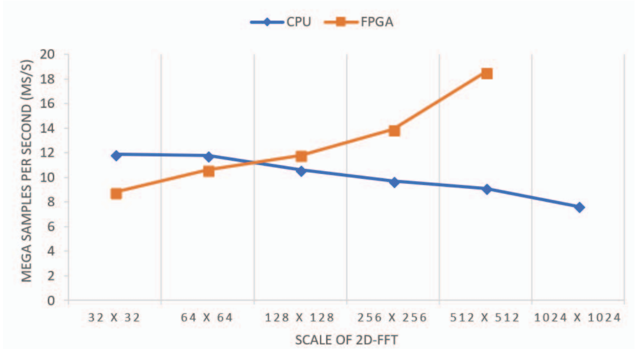

Fig. 7: Raw Performance of FPGA and CPU in Different Scales of 2D-FFT (A Single Kernel in FPGA Implementation)

with a 2D-FFT acceleration case. The results showed that FPGA based Spark implementation of 2D-FFT acquires 1.79x speedup than CPU implementation. In the future work, we will expand the scale of the cluster, and equip each slave node with an FPGA accelerator. Besides, we will optimize the implementation of the other time-consuming algorithms to achieve a better performance in FPGA based Spark implementation.

Acknowledgment

This research is funded by National Key Research & Development Program of China (2016YFE0100600), National Natural Science Foundation (NNSF of China) project (No.61201059).

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Rui Zhao, Zhaopeng Meng, Yan Zheng, Qiangguo Jin, Anbang Ruan, and Hanglun Xie. Somr: Towards a security-oriented mapreduce infrastructure. In *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pages 530–537. IEEE, 2017.

[3] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[5] James G Shanahan and Laing Dai. Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2323–2324. ACM, 2015.

[6] Charles Dierbach. Python as a first programming language. *Journal of Computing Sciences in Colleges*, 29(6):153–154, 2014.

[7] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[8] Abdul Ghaffar Shoro and Tariq Rahim Soomro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.

[9] Berkin Akin, Peter A Milder, Franz Franchetti, and James C Hoe. Memory bandwidth efficient two-dimensional fast fourier transform algorithm and implementation for large problem sizes. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 188–191. IEEE, 2012.

[10] Gokhan Polat, Sitki Ozturk, and Mehmet Yakut. Design and implementation of 256-point radix-4 100 gbit/s fft algorithm into fpga for high-speed applications. *ETRI Journal*, 37(4):667–676, 2015.

[11] Bruno Fernandes and Helena Sarmento. Fpga implementation and testing of a 128 fft for a mb-ofdm receiver. *Analog Integrated Circuits and Signal Processing*, 70(2):241–248, 2012.

[12] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *arXiv preprint arXiv:1412.6632*, 2014.

[13] B Brech, J Rubio, and M Hollinger. Ibm data engine for nosql-power systems edition. *IBM Systems Group, Tech. Rep*, 2015.

[14] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[15] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. When apache spark meets fpgas: a case study for next-generation dna sequencing acceleration. In *The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[16] Ehsan Ghasemi and Paul Chow. Accelerating apache spark with fpgas. *Concurrency and Computation: Practice and Experience*.

[17] ctypes - a foreign function library for python. https://docs.python.org/3/library/ctypes.html.

[18] Pierre Duhamel and Martin Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299, 1990.